PLOS | COMPUTATIONAL BIOLOGY

RESEARCH ARTICLE

Efficient Coalescent Simulation and Genealogical Analysis for Large Sample Sizes

Jerome Kelleher[1]*, Alison M Etheridge[2], Gilean McVean[1,2,3]

1 Wellcome Trust Centre for Human Genetics, University of Oxford, Oxford, United Kingdom, **2** Department of Statistics, University of Oxford, Oxford, United Kingdom, **3** Li Ka Shing Centre for Health Information and Discovery, University of Oxford, Oxford, United Kingdom

* jerome.kelleher@well.ox.ac.uk

OPEN ACCESS

Citation: Kelleher J, Etheridge AM, McVean G (2016) Efficient Coalescent Simulation and Genealogical Analysis for Large Sample Sizes. PLoS Comput Biol 12(5): e1004842. doi:10.1371/journal.pcbi.1004842

Editor: Yun S. Song, UC Berkeley, UNITED STATES

Received: December 10, 2015

Accepted: March 2, 2016

Published: May 4, 2016

Data Availability Statement: All experiments in this paper are fully reproducible using code available at https://github.com/jeromekelleher/msprime-paper.

Funding: This work was supported by Wellcome Trust core award 090532/Z/09/Z to the Wellcome Trust Centre for Human Genetics, Wellcome Trust grant 100956/Z/13/Z to GM, and EPSRC grants EP/I01361X/1, EP/I013091/1 and EP/K034316/1 to AME. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing Interests: The authors have declared that no competing interests exist.

Abstract

A central challenge in the analysis of genetic variation is to provide realistic genome simulation across millions of samples. Present day coalescent simulations do not scale well, or use approximations that fail to capture important long-range linkage properties. Analysing the results of simulations also presents a substantial challenge, as current methods to store genealogies consume a great deal of space, are slow to parse and do not take advantage of shared structure in correlated trees. We solve these problems by introducing sparse trees and coalescence records as the key units of genealogical analysis. Using these tools, exact simulation of the coalescent with recombination for chromosome-sized regions over hundreds of thousands of samples is possible, and substantially faster than present-day approximate methods. We can also analyse the results orders of magnitude more quickly than with existing methods.

Author Summary

Our understanding of the distribution of genetic variation in natural populations has been driven by mathematical models of the underlying biological and demographic processes. A key strength of such coalescent models is that they enable efficient simulation of data we might see under a variety of evolutionary scenarios. However, current methods are not well suited to simulating genome-scale data sets on hundreds of thousands of samples, which is essential if we are to understand the data generated by population-scale sequencing projects. Similarly, processing the results of large simulations also presents researchers with a major challenge, as it can take many days just to read the data files. In this paper we solve these problems by introducing a new way to represent information about the ancestral process. This new representation leads to huge gains in simulation speed and storage efficiency so that large simulations complete in minutes and the output files can be processed in seconds.

Introduction

The coalescent process [1, 2] underlies much of modern population genetics and is fundamental to our understanding of molecular evolution. The coalescent describes the ancestry of a sample of n genes in the absence of recombination, selection, population structure and other complicating factors. The model has proved to be highly extensible, and these and many other complexities required to model real populations have successfully been incorporated [3]. Simulation has played a key role in coalescent theory since its beginnings [2], partly due to the ease with which it can be simulated: for a sample of n genes, we require only $O(n)$ time and space to simulate a genealogy [4].

Soon after the single locus coalescent was derived, Hudson defined an algorithm to simulate the coalescent with recombination [5]. However, after some early successes in characterising this process [6, 7] little progress was made because of the complex distribution of blocks of ancestral material among ancestors. Some years after Hudson's pioneering work, the study of recombination in the coalescent was recast in the framework of the Ancestral Recombination Graph [8, 9]. In the ARG, nodes are events (either recombination or common ancestor) and the edges are ancestral chromosomes. A recombination event results in a single ancestral chromosome splitting into two chromosomes, and a common ancestor event results in two chromosomes merging into a common ancestor. Analytically, the ARG is a considerable simplification of Hudson's earlier work as it models all recombination events that occurred in the history of a sample and not just those that can potentially affect the genealogies. Many important results have been derived using this framework, one of which is particularly significant for our purposes here. Ethier and Griffiths [10] proved that the expected number of recombination events back to the Grand MRCA of a sample of n individuals grows like e^{ρ} as $\rho \to \infty$, where ρ is the population scaled recombination rate. In this paper we consider a diploid model in which we have a sequence of m discrete sites that are indexed from zero. Recombination occurs between adjacent sites at rate r per generation, and therefore $\rho = 4N_e\, r(m-1)$. The Ethier and Griffiths result implies that the time required to simulate an ARG grows exponentially with the sequence length, and we can only ever hope to simulate ARGs for the shortest of sequences.

This result, coupled with the observed poor scaling of coalescent simulators such as the seminal `ms` program [11] seems to imply that simulating the coalescent with recombination over chromosome scales is hopeless, and researchers have therefore sought alternatives. The sequentially Markov coalescent (SMC) approximation [12, 13] underlies the majority of present day genome scale simulation [14–16] and inference methods [17–19]. The SMC simplifies the process of simulating genealogies by assuming that each marginal tree depends only on its immediate predecessor as we move from left-to-right across the sequence. As a consequence, the time required to simulate genealogies scales linearly with increasing sequence length. In practice, SMC based simulators such as `MaCS` [14] and `scrm` [16] are many times faster than `ms`.

The SMC has disadvantages, however. Firstly, the SMC discards all long range linkage information and therefore can be a poor approximation when modelling features such as the length of admixture blocks [20]. Improving the accuracy of the SMC can also be difficult. For example, `MaCS` has a parameter to increase the number of previous trees on which a marginal tree can depend. Counter-intuitively, increasing this parameter beyond a certain limit results in a *worse* approximation to the coalescent with recombination [16]. (The `scrm` simulator provides a similar parameter that does not exhibit this unfortunate behaviour, however.) Incorporating complexities such as population structure [21], intra-codon recombination [22] and inversions [23] is non-trivial and can be substantially more complex than the corresponding modification to the exact coalescent model. Also, while SMC based methods scale well in terms of increasing sequence length, currently available simulators do not scale well in terms of sample size.

We solve these problems by introducing sparse trees and coalescence records as the fundamental units of genealogical analysis. By creating a concrete formalisation of the genealogies generated by the coalescent process in terms of an integer vector, we greatly increase the efficiency of simulating the exact coalescent with recombination. In the section **Efficient coalescent simulation**, we discuss how Hudson's classical simulation algorithm can be defined in terms of these sparse trees, and why this leads to substantial gains in terms of the simulation speed and memory usage. We show that our implementation of the exact coalescent, `msprime`, is competitive with approximate simulators for small sample sizes, and is faster than all other simulators for large sample sizes. This is possible because Hudson's algorithm does not traverse the entire ARG, but rather a small subset of it. The ARG contains a large number of nodes that do not affect the genealogies of the sample [24], and Hudson's algorithm saves time by not visiting these nodes. This subset of the ARG (sometimes known as the 'little' ARG) has not been well characterised, which makes analysis of Hudson's algorithm difficult. However, we show some numerical results indicating that the number of nodes in the little ARG may be a quadratic function of the scaled recombination rate ρ rather than an exponential.

Generating simulated data is of little use if the results cannot be processed in an efficient and convenient manner. Currently available methods for storing and processing genealogies perform very poorly on trees with hundreds of thousands of nodes. In the section **Efficient genealogical analysis**, we show how the encoding of the correlated trees output by our simulations leads to an extremely compact method of storing these genealogies. For large simulations, the representation can be thousands of times smaller than the most compact tree serialisation format currently available. Our encoding also leads to very efficient tree processing algorithms; for example, sequential access to trees is several orders of magnitude faster than existing methods.

The advantages of faster and more accurate simulation over huge sample sizes, and the ability to quickly process very large result sets may enable applications that were not previously feasible. In the **Results and Discussion** we conclude by considering some of these applications and other uses of our novel encoding of genealogies. The methods developed in this paper allow us to simulate the coalescent for very large sample sizes, where the underlying assumptions of the model may be violated [25–27]. Addressing these issues is beyond the scope of this work, but we note that the majority of our results can be applied to simulations of any retrospective population model.

Methods

Efficient coalescent simulation

In this section we define our encoding of coalescent genealogies, and show how this leads to very efficient simulations. There are many different simulation packages, and so we begin with a brief review of the state-of-the-art before defining our encoding and analysing the resulting algorithm in the following subsections.

Two basic approaches exist to simulate the coalescent with recombination. The first approach was defined by Hudson [5], and works by applying the effects of recombination and common ancestor events to the ancestors of the sample as we go backwards in time. Events occur at a rate that depends only on the state of the extant ancestors, and so we can generate the waiting times to these events efficiently without considering the intervening generations. This contrasts with time-reversed generation-by-generation methods [28–31] which are more flexible but also considerably less efficient. The first simulation program published based on Hudson's algorithm was `ms` [11]. After this, many programs were published to simulate

various evolutionary complexities not handled by ms, such as selection [32–35], recombination hotspots [36], codon models [37], intra-codon recombination [22] and models of species with a skewed offspring distribution [38]. Others developed user interfaces to facilitate easier analysis [39, 40].

The second fundamental method of simulating the coalescent with recombination is due to Wiuf and Hein [24]. In Wiuf and Hein's algorithm we begin by generating a coalescent tree for the left-most locus and then move across the sequence, updating the genealogy to account for recombination events. This process is considerably more complex than Hudson's algorithm because the relationship between trees as we move across the genome is non-Markovian: each tree depends on all previously generated trees. Because of this complexity, exact simulators based on Wiuf and Hein's algorithm are significantly less efficient than ms [16, 41]. However, Wiuf and Hein's algorithm has provided the basis for the SMC approximation [12, 13], and programs based on this approach [14–16] can simulate long sequences far more efficiently than exact methods such as ms. Very roughly, we can think of Wiuf and Hein's algorithm performing a depth-first traversal of the ARG, and Hudson's algorithm a breadth-first traversal. Neither explore the full ARG, but instead traverse the subset required to contruct all marginal genealogies.

Recently, Hudson's algorithm has been utilised in cosi2 [35], which takes a novel approach to simulating sequences under the coalescent. The majority of simulators first generate genealogies and then throw down mutations in a separate process. In cosi2 these two processes are merged, so that mutations are generated during traversal of the ARG. Instead of associating a partial genealogy with each ancestral segment, cosi2 maps ancestral segments directly to the set of sampled individuals at the leaves of this tree. When a coalescence between two overlapping segments occurs, we then have sufficient information to generate mutations and map them to the affected samples. This strategy, coupled with the use of sophisticated data structures, makes cosi2 many times faster than competing simulators such as msms [34]. The disadvantage of combining the mutation process with ARG traversal, however, is that the underlying genealogies are not available, and cosi2 cannot directly output coalescent trees.

Many reviews are available to compare the various coalescent simulators in terms of their features [42–47]. Little information is available, however, about their relative efficiencies. Hudson's ms is widely regarded as the most efficient implementation of the exact coalescent and is the benchmark against which other programs are measured [13–16, 41, 47]. However, for larger sample sizes and long sequence lengths, msms is much faster than ms. Also, for these larger sequence lengths and sample sizes, ms is unreliable and crashes [15, 47]. Thus, msms is a much more suitable baseline against which to judge performance. The scrm simulator is the most efficient SMC based method currently available [16].

Hudson's algorithm with sparse trees. An oriented tree [48, p. 461] is a sequence of integers $\pi_1\ \pi_2 \ldots$, such that π_u is the parent of node u and u is a root if $\pi_u = 0$. Fig 1 shows some example tree topologies and corresponding integer sequence encodings. Oriented trees provide a concise and efficient method of representing genealogies, and have been used in coalescent simulations of a spatial continuum model [49, 50]. These simulations adopted the convention that the individuals in the sample (leaf nodes) are mapped to the integers $1, \ldots, n$. For every internal node u we have $n < u < 2n$ and (for a binary tree) the root is $2n - 1$. We refer to such trees as dense because the $2n - 2$ non-zero entries of the (binary) tree π occur at $u = 1, \ldots, 2n - 2$. A sparse oriented tree (or more concisely, sparse tree) is an oriented tree π in which the leaf nodes are $1, \ldots, n$ as before, but internal nodes can be any integer $> n$. For example, the oriented trees $\langle 5, 4, 4, 5, 0\rangle$ and $\langle 6, 5, 5, 0, 6, 0\rangle$ are topologically equivalent, but the former is dense and the latter sparse.

In our simulations, ancestral nodes are numbered sequentially from $n + 1$, and a new node is created when a coalescence occurs within one or more of the marginal genealogies. Note that

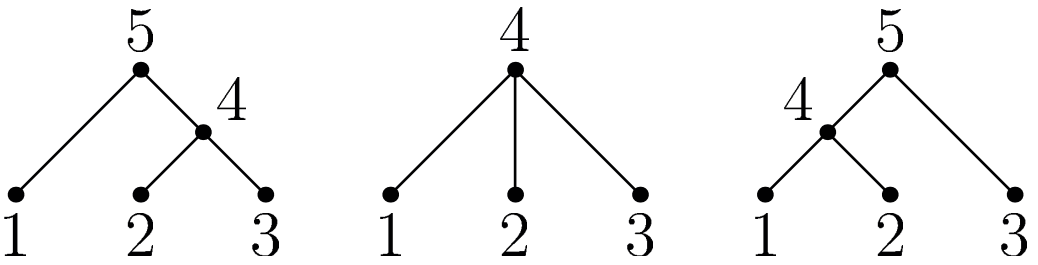

Fig 1. Example oriented trees. From left-to-right, these trees are defined by the sequences ⟨5, 4, 4, 5, 0⟩, ⟨4, 4, 4, 0⟩ and ⟨4, 4, 5, 5, 0⟩, respectively.

doi:10.1371/journal.pcbi.1004842.g001

we make a distinction between common ancestor events and coalescence events throughout. A common ancestor event occurs when two ancestors merge to form a common ancestor. If these ancestors have overlapping ancestral material, then there will also be at least one coalescence event, which is defined as a single contiguous block of sequence coalescing within a common ancestor. In Hudson's algorithm there are many common ancestor events that do not result in coalescence, and it is important to distinguish between them.

Let the tuple (ℓ, r, u) define a segment carrying ancestral material. This segment represents the mapping of the half-closed genomic interval $[\ell, r)$ to the tree node u. Each ancestor a is defined by a set of non-overlapping segments. Initially we have n ancestors, each consisting of a single segment $(0, m, u)$ for $1 \leq u \leq n$. The only other state required by the algorithm is the time t, and the next node w; initially, $t = 0$ and $w = n + 1$.

Let P be the set of ancestors at a given time t. Recombination events happen at rate $\rho L/(m - 1)$ where

$$L = \sum_{a \in P} \left(\max_{(\ell,r,u) \in a} r - \min_{(\ell,r,u) \in a} \ell - 1 \right)$$

is the number of available 'links' that may be broken. (We use a fixed recombination rate here for simplicity, but an arbitrary recombination map can be incorporated without difficulty.) We choose one of the available breakpoints uniformly, and split the ancestry of the individual at that point into two recombinant ancestors. If this breakpoint is at k, we assign all segments with $r \leq k$ to one ancestor and all segments with $\ell \geq k$ to the other. If there is a segment (ℓ, r, u) such that $\ell < k < r$, then k falls within this segment and it is split such that the segment (ℓ, k, u) is assigned to one ancestor and (k, r, u) is assigned to the other.

Common ancestor events occur at rate $|P|(|P| - 1)$. Two ancestors a and b are chosen and their ancestry merged to form their common ancestor. If their segments do not overlap, the set of ancestral segments of the common ancestor is the union of those of a and b. If segments do overlap, we have coalescence events which must be recorded. We define a coalescence event as the merging of two segments over the interval $[\ell, r)$ into a single ancestral segment. In general the coordinates of overlapping segments x and y will not exactly coincide, in which case we create an equivalent set of segments by subdividing into the intersections and 'overhangs'. Suppose then that we have two exactly intersecting segments (ℓ, r, u) and (ℓ, r, v) from a and b respectively; over the interval $[\ell, r)$ the nodes u and v coalesce into a common ancestor, which we associate with the next available node w. We record this information by storing the coalescence record $(\ell, r, w, (u, v), t)$. As we see in the **Generating trees** section, these records provide sufficient information to later recover all marginal trees. After recording this coalescence, we then check if there are any other segments in P that intersect with $[\ell, r)$. If there are, the simulation of this region is not yet complete and we insert the segment (ℓ, r, w) into the ancestor of a

and b. On the other hand, if there is some subset of $[\ell, r)$ such that there is no other segment in P that intersects with it, we know that the marginal tree covering this interval is complete and therefore we do not need to trace its history any further. If any other intervals overlap in a and b, we perform the same operations, and finally update the next available node by incrementing w. In this way, all coalescing intervals within the same ancestor map to the same node w, even if they are disjoint. Conversely, if two disjoint marginal trees contain the same node, we know that this is because multiple segments coalesced simultaneously within the same ancestor.

The algorithm continues generating recombination and common ancestor events at the appropriate rates until P is empty, and all marginal trees are complete. This interpretation of Hudson's algorithm differs from the standard formulations [4, 5, 12] by concretely defining the representation of ancestry and by introducing the idea of coalescence records. We have omitted many important details here in the interest of brevity; see S1 Text for a detailed listing of our implementation of Hudson's algorithm, and S2 Text for an illustration of a complete invocation of the algorithm.

There are several advantages to our sparse tree representation of ancestry. Firstly, we do not need to store partially built trees in memory, and the only state we need to maintain is the set of ancestral segments. This leads to substantial time and memory savings, since we no longer have to copy partially built trees at recombination events or update them during coalescences. We can also actively defragment the segments in memory. For example, suppose that as a result of a common ancestor event we have two segments (ℓ, k, u) and (k, r, u) in an ancestor. We can replace these segments with the equivalent segment (ℓ, r, u). Such defragmentation yields significant time and memory savings.

We have developed an implementation of Hudson's algorithm called `msprime` based on these ideas. This package (written in C and Python) provides an `ms` compatible command line interface along with a Python API, and is freely available under the terms of the GNU GPL at https://pypi.python.org/pypi/msprime. The implementation uses a simple linked-list based representation of ancestral segments, and uses a binary indexed tree [51, 52] to ensure the choice of ancestral segment involved in a recombination event can be done in logarithmic time. The implementation of `msprime` is based on the listings for Hudson's algorithm given in S1 Text, which should provide sufficient detail to make implementation in a variety of languages routine.

Performance analysis. Surprisingly little is known about the complexity of Hudson's algorithm. We do not know, for example, what the expected maximum number of extant ancestors is, nor the distribution of ancestral material among them. The most important unknown value in terms of quantifying the complexity of the algorithm is the expected number of events that must be generated. It is sufficient to consider the recombination events as the number of common ancestor and recombination events is approximately equal [24]. Hudson's algorithm traverses a subset of the ARG as it generates the marginal genealogies in which we are interested, and so we know that the expected number of recombination events we encounter is less than e^{ρ} [10]. This subset of the ARG is sometimes known as the 'little' ARG, but the relationship between the 'big' and little ARGs has not been well characterised.

Fig 2 plots the average number of recombination events generated by Hudson's algorithm for varying sequence lengths and sample sizes. In this plot we also show the results of fitting a quadratic function to the number of recombination events as we increase the scaled recombination rate ρ. The fit is excellent, suggesting that the current upper bound of e^{ρ} is far too pessimistic. Wiuf and Hein [24] previously noted that the observed number of events in Hudson's algorithm was 'subexponential' but did not suggest a quadratic bound. Another point to note is that the rate at which the number of events grows as we increase the sample size is extremely slow, suggesting that Hudson's algorithm should scale well for large sample sizes.

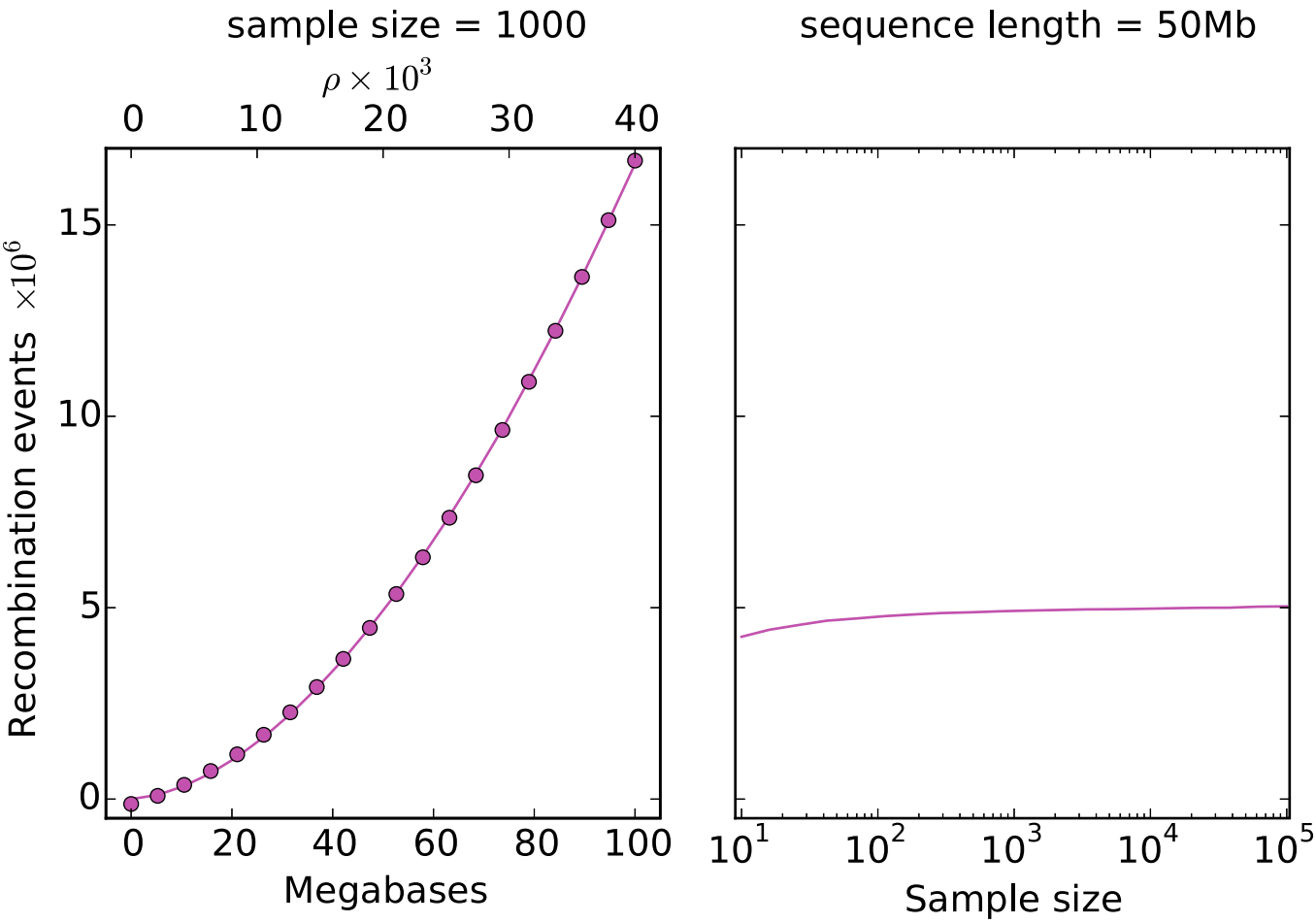

Fig 2. The mean number of recombination events in Hudson's algorithm over 100 replicates for varying sequence length and sample size. In the left panel we fix $n = 1000$ and vary the sequence length. Shown in dots is a quadratic fitted to these data, which has a leading coefficient of 8.4×10^{-3}. In the right panel we fix the sequence length at 50 megabases and vary the sample size.

doi:10.1371/journal.pcbi.1004842.g002

These expectations are borne out well in observations of our implementation of Hudson's algorithm in `msprime`. Fig 3 compares the time required to simulate coalescent trees using a number of simulation packages. As we increase the sequence length in the left-hand panel, the running time of `msprime` increases faster than linearly, but at quite a slow rate. `msprime` is faster than the SMC approximations (`MaCS` and `scrm`) until ρ is roughly 20000, and the difference is minor for sequence lengths greater than this. `msprime` is far faster than `msms`, the only other exact simulator in the comparison (we did not include `ms` in these comparisons as it was too slow and is unreliable for large sample sizes). As we increase the sample size in the right-hand panel, we can see that `msprime` is far faster than any other simulator. Two versions of `msprime` are shown in these plots: one outputting Newick trees (to ensure that the comparison with other simulators is fair), and another that outputs directly in `msprime`'s native format. Conversion to Newick is an expensive process, particularly for larger sample sizes. When we eliminate this bottleneck, simulation time grows at quite a slow, approximately linear rate. The memory usage of `msprime` is also modest, with the simulations in Fig 3 requiring less than a gigabyte of RAM. S3 Fig shows that the mean number of recombination breakpoints (i.e., the number of recombination events within ancestral material) output by all

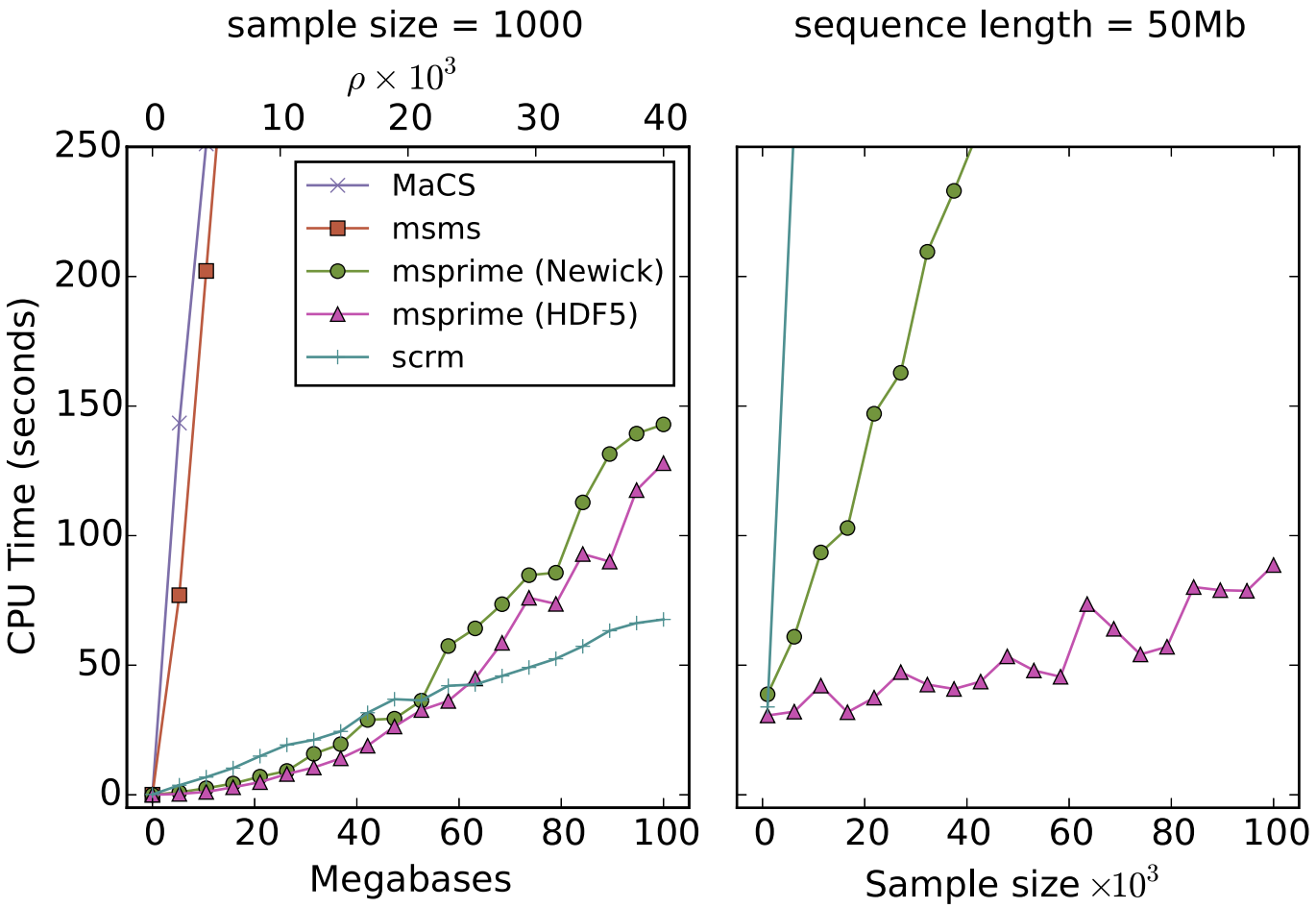

Fig 3. Comparison of the average running time over 100 replicates for various coalescent simulators with varying sequence length and sample size. `msms` [34] is the most efficient published simulator based on Hudson's algorithm that can output genealogies. `MaCS` [14] is a popular SMC based simulator, and `scrm` [16] is the most efficient sequential simulator currently available. Both `MaCS` and `scrm` were run in SMC′ mode. Two results are shown for `msprime`; one outputting Newick trees and another outputting the native HDF5 based format.

doi:10.1371/journal.pcbi.1004842.g003

these simulators is identical, and matches Hudson and Kaplan's prediction [6] very well, giving us some confidence in the correctness of the results. S4 Fig shows the relative performance of `msprime` and `scrm` for a small sample size, and also shows the effect of increasing the size of `scrm`'s sliding window.

We are often interested in the haplotypes that result from imposing a mutation process onto genealogies as well as the genealogies themselves. S1 Fig compares the time required to generate haplotypes using `scrm`, `msprime` and `cosi2`. Simulation times are similar in all three for a fixed sample size of 1000 and increasing sequence length. For increasing sample sizes, both `cosi2` and `msprime` are substantially faster than `scrm`. However, `msprime` is significantly faster than `cosi2` (and uses less memory; see S2 Fig), particularly when we remove the large overhead of outputting the haplotypes in text form.

Performance statistics were measured on Intel Xeon E5-2680 processors running Debian 8.2. All code required to run comparisons and generate plots is available at https://github.com/jeromekelleher/msprime-paper.

Efficient genealogical analysis

There has been much recent interest in the problem of representing large scale genetic data in formats that facilitate efficient access and calculation of statistics [53–55]. The use of 'succinct' data structures, which are highly compressed but also allow for efficient queries is becoming essential: the scale of the data available to researchers is so large that naive methods simply no longer work.

Although genealogies are fundamental to biology, there has been little attention to the problem of encoding trees in a form that facilitates efficient computation. The majority of research has focused on the accurate interchange of tree structures and associated metadata. The most common format for exchanging tree data is the Newick format [56], which although ill-defined [57] has become the de-facto standard. Newick is based on the correspondence of tree structures with nested parentheses, and is a concise method of expressing tree topologies. Because of this recursive structure, specific extensions to the syntax are required to associate information with tree nodes [58, 59]. XML based formats [57, 60] are much more flexible, but tend to require substantially more storage space than Newick [57]. Various extensions to Newick have been proposed to incorporate more general graph structures [61–64], as well as a GraphML extension to encode ARGs directly [65]. Because Newick stores branch lengths rather than node times, numerical precision issues also arise when summing over many short branches [65].

General purpose Bioinformatics toolkits such as BioPerl [66] and BioPython [67] provide basic tools to import trees in the various formats. More specific tree processing libraries such as DendroPy [68], ETE [69], and APE [70] provide more sophisticated tools such as visualisation and tree comparison algorithms. None of these libraries are designed to handle large collections of correlated trees, and cannot make use of the shared structure within a sequence of correlated genealogies. The methods employed rarely scale well to trees containing hundreds of thousands of nodes.

In this section we introduce a new representation of the correlated trees output by a coalescent simulation using coalescence records. In the **Tree sequences** subsection we discuss this structure and show how it compares in practice to existing approaches in terms of storage size. Then, the **Generating trees** subsection presents an algorithm to sequentially generate the marginal genealogies from a tree sequence, which we compare with existing Newick-based methods. Finally, in the **Counting leaves** subsection we show how the algorithm to sequentially visit trees can be extended to efficiently maintain the counts of leaves from a specific subset, and show how this can be applied in a calculation commonly used in genome wide association studies.

Tree sequences. As described earlier, the output of our formulation of Hudson's algorithm is a list of coalescence records. Each coalescence record is a tuple (ℓ, r, u, c, t) describing the coalescence of a list of child nodes c into the parent u at time t over the half-closed genomic interval $[\ell, r)$. (Because only binary trees are possible in the standard coalescent, we assume the child node list c is a 2-tuple (c_1, c_2) throughout. However, arbitrary numbers of children can be accommodated without difficulty to support common ancestor events in which more than two lineages merge [71–74]) We refer to this set of records as a *tree sequence*, as it is a compact encoding of the set of correlated trees representing the genealogies of a sample. Fig 4 shows an illustration of the tree sequence output by an example simulation (see S2 Text for a full trace of this simulation).

The tree sequence provides a concise method of representing the correlated genealogies generated by coalescent simulations because it stores node assignments shared across adjacent trees exactly once. Consider node 7 in Fig 4. This node is shared in the first two marginal trees, and in both cases it has two children, 1 and 6. Even though the node spans two marginal trees, the node assignment is represented in one coalescence record (0, 7, 7, (1, 6), 0.170).

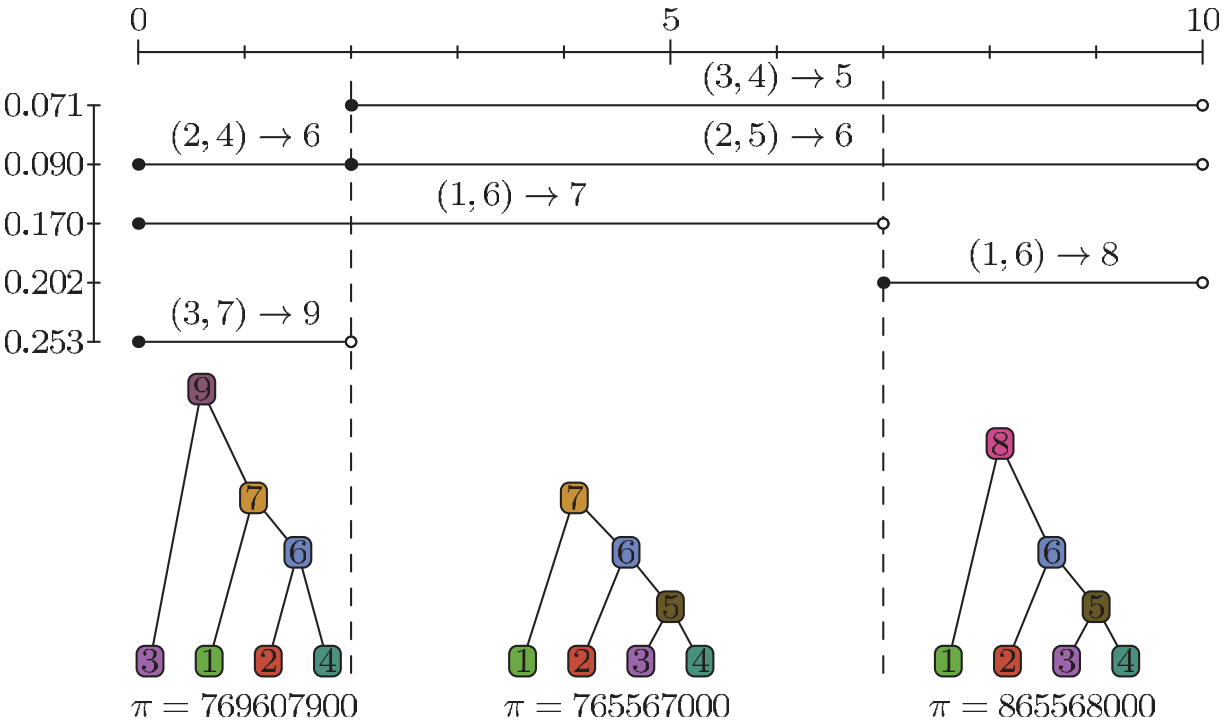

Fig 4. Coalescence records and corresponding marginal trees. The *x*-axis represents genomic coordinates, and *y*-axis represents time (with the present at the top). Each line segment in the top section of the figure represents a coalescence record; e.g., the first segment corresponds to the coalescence record (2, 10, 5, (3, 4), 0.071). The lower section of the figure shows the corresponding trees in pictorial and sparse tree form. We have omitted commas and brackets from this sequence representation for compactness.

doi:10.1371/journal.pcbi.1004842.g004

Importantly, this holds true even though the subtree beneath 6 is different in these trees. Thus, any assignment of a pair of children to a given parent that is shared across adjacent trees will be represented by exactly one coalescence record.

Coalescence records provide a full history of the coalescence events that occurred in our simulation. (Recall that we distinguish between common ancestor events, which may or may not result in marginal coalescences, and coalescence events which are defined as a single contiguous block of genome merging within a common ancestor.) The effects of recombination events are also stored indirectly in this representation in the form of the left and right coordinate of each record. For every distinct coordinate between 0 and m, there must have been at least one recombination event that occurred at that breakpoint. However, there is no direct information about the times of these recombination events, and many recombinations will happen that leave no trace in the set of coalescence records. For example, if we have a recombination event that splits the ancestry of a given lineage, and this is immediately followed by a common ancestor event involving these two lineages, there will be no record of this pair of events.

On the other hand, if we consider the records in order of their left and right coordinates we can also see them as defining the way in which we transform the marginal genealogies as we move across a chromosome. Because many adjacent sites may share the same genealogy, we need only consider the coordinates of our records in order to recover the distinct genealogies and the coordinate ranges over which they are defined. To obtain the marginal tree covering the interval [0,2), for example, we simply find all records with left coordinate equal to 0 and apply these to the empty sparse tree π. To subsequently obtain the tree corresponding to the interval [2, 7) we first remove the records that do not apply over this interval, which must have right

coordinate equal to 2. In the example, this corresponds to removing the assignments $(2, 4) \rightarrow 6$ and $(3, 7) \rightarrow 9$. Having removed the 'stale' records that do not cover the current interval, we must now apply the new records that have left coordinate 2. In this case, we have two node assignments $(3, 4) \rightarrow 5$ and $(2, 5) \rightarrow 6$, and applying these changes to the current tree completes the transformation of the first marginal tree into the second.

There is an important point here. As we moved from left-to-right across the simulated chromosome we transitioned from one marginal tree to the next by removing and applying only two records. Crucially, modifying the nodes that were affected by this transition did not result in a relabelling of any nodes that were not affected. As Wiuf and Hein [24,75] showed, the effect of a recombination at a given point in the sequence is to cut the branch above some node in the tree to the left of this point, and reattach it within another branch. This process is known as a subtree-prune-and-regraft [76, 77] and requires a maximum of three records to express in our tree sequence formulation.

Prune-and-regraft operations that do not affect the root require three records, as illustrated in Fig 5. Two other possibilities exist for how the current tree can be edited as we move along the sequence. The first case is when we have a prune and regraft that involves a change in root node; this requires only two records and is illustrated in the first transition in Fig 4. The other case that can arise from a single recombination event is a simple root change in which the only difference between the adjacent trees is the time of the MRCA. This requires one record, and is illustrated in the second transition in Fig 4. These three possibilities are closely related to the three classes of subtree-prune-and-regraft identified by Song [76, 77].

Knowing the maximum number of records arising from a single recombination event provides us with a useful bound on the expected number of records in a tree sequence. Because the expected number of recombination events within ancestral material is approximately $\rho \log n$ [6, 24] we know that the expected number of tree transitions is $\rho \log n$. The number of records we require for these tree transitions is then clearly $\leq 3\rho \log n$. We also require $n - 1$ records to describe the first tree in the sequence, and so the total number of records is $\leq n + 3\rho \log n - 1$.

Storing a tree sequence as a set of coalescence records therefore requires $O(n + \rho \log n)$ space, whereas any representation that stores each tree separately (such as Newick) must require $O(n\rho \log n)$ space. This difference is substantial in practice. As an example of a practical simulation of the sort currently being undertaken, we repeated the simulation run by Layer et al. [54], in which we simulate a 100 megabase region with a recombination rate of 10^{-3} per base per $4N_e$ generations for a sample of 100,000 individuals. This simulation required approximately 6 minutes and 850MB of RAM to run using `msprime`; the original simulation reportedly required over 4 weeks using `MaCS` on similar hardware.

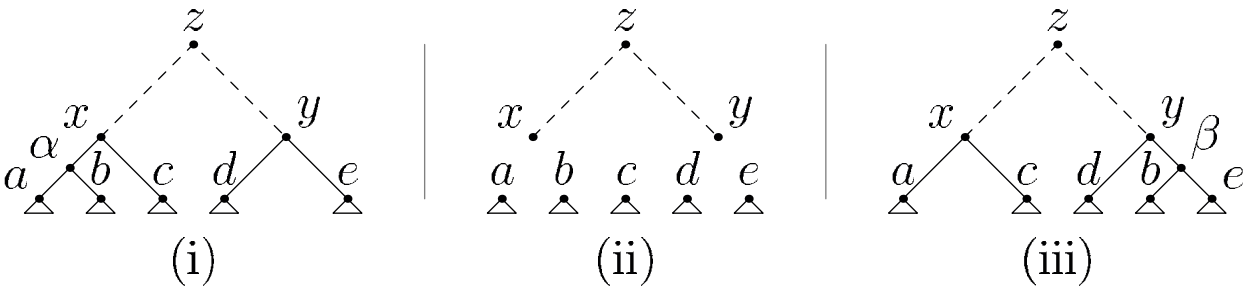

Fig 5. A prune and regraft not involving the root requires three records. (i) We begin with two subtrees rooted at x and y, and we wish to prune the subtree rooted at b and graft it in the branch joining e to y. (ii) We remove the assignments $(a, b) \rightarrow \alpha$, $(\alpha, c) \rightarrow x$ and $(d, e) \rightarrow y$. After this operation, the subtrees $a, \ldots, e$ are disconnected from the main tree. The main trunk the tree rooted at z is unaffected, as are the subtrees below $a, \ldots, e$. (iii) We add the records $(a, c) \rightarrow x$, $(b, e) \rightarrow \beta$ and $(d, \beta) \rightarrow y$, completing the transition.

doi:10.1371/journal.pcbi.1004842.g005

Outputting the results as coalescence records in a simple tab-delimited text format resulted in a 173MB file (52MB when gzip compressed). In contrast, writing the trees out in Newick form required around 3.5TB of space. Because plain text is a poor format for storing structured numerical data [78], `msprime` provides a tree sequence storage file format based of the HDF5 standard [79]. Using this storage format, the file size is reduced to 88MB (41MB using the transparent zlib compression provided by the HDF5 library).

To compare the efficiency of storing correlated trees as coalescence records with the TreeZip compression algorithm [80] we output the first 1000 trees in Newick format, resulting in a 3.2GB text file (1.1GB gzip compressed). The TreeZip compression algorithm required 10 hours to run and resulted in an 882MB file (83MB gzip compressed). Unfortunately, it was not feasible to run TreeZip on all 3.5TB of the Newick data, but we can see that with only around 0.1% of the input data, the compressed representation is already larger than the simple text output of the entire tree sequence when expressed as coalescence records.

Associating mutation information with a tree sequence is straightforward. For example, to represent a mutation that occurs on the branch that joins node 7 to node 9 at site 1 in Fig 4, we simply record the tuple (7, 1). (Infinite sites mutations can be readily accommodated by assuming that the coordinate space is continuous rather than discrete.) Because only the associated node and position of each mutation needs to be stored, this results in a very concise representation of the full genealogical history and mutational state of a sample. Repeating the simulation above with a scaled mutation rate of 10^{-3} per unit of sequence length per $4N_e$ generations resulted in 1.2 million infinite sites mutations. The total size of the HDF5 representation of the tree sequence and mutations was 102MB (49MB using HDF5's zlib compression). In contrast, the text-based haplotype strings consumed 113GB (9.7GB gzip compressed). Converting to text haplotypes required roughly 9 minutes and 14GB of RAM.

The PBWT [53] represents binary haplotype data in a format that is both highly compressed and enables efficient pattern matching algorithms. We converted the mutation data above into PBWT form, which required 22MB of storage. Thus, the PBWT is a more compact representation of a set of haplotypes than the tree sequence. However, the PBWT does not contain any genealogical data, and therefore contains less information than the tree sequence.

Generating trees. Coalescence records provide a very compact means of encoding correlated genealogies. Compressed representations of data usually come at the cost of increased decompression effort when we wish to access the information. In contrast, we can recover the marginal trees from a set of coalescence records orders of magnitude more quickly than is possible using existing methods. In this section we define the basic algorithm required to sequentially generate these marginal genealogies.

For algorithms involving tree sequences it is useful to regard the set of coalescence records as a table and to index the columns independently (see S2 Text for the table corresponding to Fig 4). Therefore define a tree sequence T as a tuple of vectors $T = (\mathbf{l}, \mathbf{r}, \mathbf{u}, \mathbf{c}, \mathbf{t})$, such that for each index $1 \leq j \leq M$, $(\mathbf{l}_j, \mathbf{r}_j, \mathbf{u}_j, \mathbf{c}_j, \mathbf{t}_j)$ corresponds to one coalescence record output by Hudson's algorithm, and there are M records in total. It is also useful to impose an ordering among the children at a node, and so we assert that $\mathbf{c}_{j,1} < \mathbf{c}_{j,2}$ for all $1 \leq j \leq M$.

If we wish to obtain the tree for a given site x we simply find the $n - 1$ records that intersect with this point and build the tree by applying these records. We begin by setting $\pi_j \leftarrow 0$ for $1 \leq j \leq \max(\mathbf{u})$, and then set $\pi_{\mathbf{c}_{j,1}} \leftarrow \mathbf{u}_j$ and $\pi_{\mathbf{c}_{j,2}} \leftarrow \mathbf{u}_j$ for all j such that $\mathbf{l}_j \leq x < \mathbf{r}_j$. Spatial indexing structures such as the segment tree [81] allow us to find all k segments out of a set of N that intersect with a given point in $O(k + \log N)$ time. Therefore, since the expected number of records is $O(n + \rho \log n)$ as shown in the previous subsection, the overall complexity of generating a single tree is $O(n + \log(n + \rho \log n))$.

A common requirement is to sequentially visit all trees in a tree sequence in left-to-right order. One possible way to do this would be to find all of the distinct left coordinates in the $\mathbf{l}$ vector and apply the process outlined above. However, adjacent trees are highly correlated and share much of their structure, and so this approach would be quite wasteful. A more efficient approach is given in Algorithm T below. For this algorithm we require two 'index vectors' $\mathcal{I}$ and $\mathcal{O}$ which give the indexes of the records in the order in which they are inserted and removed, respectively. Records are applied in order of nondecreasing left coordinate and increasing time, and records are removed in nondecreasing order of right coordinate and decreasing time. That is, for every pair of indexes j and k such that $1 \leq j < k \leq M$ we have either $\mathbf{l}_{\mathcal{I}_j} < \mathbf{l}_{\mathcal{I}_k}$ or $\mathbf{l}_{\mathcal{I}_j} = \mathbf{l}_{\mathcal{I}_k}$ and $\mathbf{t}_{\mathcal{I}_j} < \mathbf{t}_{\mathcal{I}_k}$; and similarly, either $\mathbf{r}_{\mathcal{O}_j} < \mathbf{r}_{\mathcal{O}_k}$ or $\mathbf{r}_{\mathcal{O}_j} = \mathbf{r}_{\mathcal{O}_k}$ and $\mathbf{t}_{\mathcal{O}_j} > \mathbf{t}_{\mathcal{O}_k}$. We assume that these index vectors have been pre-calculated below.

Algorithm T. (*Generate trees*). Sequentially visit the sparse trees π in a tree sequence $T = (\mathbf{l}, \mathbf{r}, \mathbf{u}, \mathbf{c}, \mathbf{t})$ with M records.

T1. [Initialisation.] Set $\pi_j \leftarrow 0$ for $1 \leq j \leq \max(\mathbf{u})$. Then set $j \leftarrow 1$, $k \leftarrow 1$ and $x \leftarrow 0$.

T2. [Insert record.] Set $h \leftarrow \mathcal{I}_j$, $\pi_{\mathbf{c}_{h,1}} \leftarrow \pi_{\mathbf{c}_{h,2}} \leftarrow \mathbf{u}_h$, and $j \leftarrow j + 1$. If $j \leq M$ and $\mathbf{l}_{\mathcal{I}_j} = x$, go to T2.

T3. [Visit tree.] Visit the sparse tree π starting at site x. If $j > M$ terminate the algorithm. Otherwise, set $x \leftarrow \mathbf{l}_{\mathcal{I}_j}$.

T4. [Remove record.] Set $h \leftarrow \mathcal{O}_k$, $\pi_{\mathbf{c}_{h,1}} \leftarrow \pi_{\mathbf{c}_{h,2}} \leftarrow 0$ and $k \leftarrow k + 1$. Then, if $\mathbf{r}_{\mathcal{O}_k} = x$ go to T4; otherwise, go to T2.

Algorithm T sequentially generates all marginal trees in a tree sequence by first applying records to the sparse tree π in step T2 for a given left coordinate. Once this is complete, the tree is made available to client code by 'visiting' it [48, p.281] in T3. After the user has finished processing the current tree, we prepare to move to the next tree by removing all stale records in T4, and then return to T2. The algorithm is very efficient. Because each record is considered exactly once in step T2 and at most once in step T4 the total time required by the algorithm is $O(n + \rho \log n)$. To illustrate this efficiency, we consider the time required to iterate over the trees produced by the large example simulation used throughout this section. Reading in the full tree sequence in `msprime`'s native HDF5 based format and iterating over all 1.1 million trees using the Python API required approximately 3 seconds. In contrast, using the BioPython [67] version 1.64 Newick parser required around 3 seconds *per tree*, leading to an estimated 38 days to iterate over all trees. Similarly, ETE [69] version 2.3.9 required 4.5 seconds per tree, and DendroPy [68] version 4.0.2 required around 14 seconds per tree. Comparing Python Newick parsers to `msprime` may be somewhat misleading, since the majority of `msprime`'s tree processing code is written in C. However, APE [70] version 3.1, which uses a Newick parser written in C, also required around 7 seconds per tree. Thus, using `msprime`'s API we can iterate over this set of trees more than a *million times* faster than any of these alternatives.

Algorithm T generates only the sparse tree π mapping each node to its parent. It is easy to extend this algorithm to include information about the node times, children, start and end coordinates and other information. We have also assumed binary trees here, but it is trivial to extend the algorithm to work with more general trees. When computing statistics across the tree sequence it is often useful to know the specific differences between adjacent trees, as this often allows us to avoid examining the entire tree. This information is directly available in Algorithm T. The tree iteration code in `msprime`'s Python API makes all of this information available, facilitating easy tree traversal in both top-down and bottom-up fashion.

Counting leaves. The previous subsection provides an algorithm to efficiently visit all marginal genealogies in a tree sequence. This algorithm can be easily augmented to maintain summaries of tree properties as we sweep across the sequence. As an example of this, we show how to augment Algorithm T to maintain the counts of the number of leaves from a specific set that are below each internal node. More precisely, given some subset S of our sample, we maintain a vector β such that for any node u, β_u is the number of leaves below u that belong to the set S. This allows us to quickly calculate allele frequencies: since each mutation is associated with a particular node u, $\beta_u/|S|$ is the frequency of the mutation within S. Calculating allele frequencies within specific subsets of the sample has many applications, for example calculating summary statistics such as F_{ST} [82], and association tests in genome wide association studies [83].

Suppose we have a tree sequence T and we wish to generate the sparse trees π as before. We now also wish to generate the vector β, such that β_u gives the number of leaf nodes in the subtree rooted at u that are in the set $S \subseteq \{1, \ldots, n\}$. We assume that the index vectors $\mathcal{I}$ and $\mathcal{O}$ have been precomputed, as before.

Algorithm L. (*Count leaves*). Generate the sparse trees π and leaf counts β for a tree sequence T = (**l,r,u,c,t**) with M records and set of leaves S.

L1. [Initialisation.] Set $\pi_j \leftarrow \beta_j \leftarrow 0$ for $1 \leq j \leq \max(\mathbf{u})$. Set $\beta_j \leftarrow 1$ for each $j \in S$. Then set $j \leftarrow 1$, $k \leftarrow 1$ and $x \leftarrow 0$.

L2. [Insert record.] Set $h \leftarrow \mathcal{I}_j$, $\pi_{\mathbf{c}_{h,1}} \leftarrow \pi_{\mathbf{c}_{h,2}} \leftarrow \mathbf{u}_h$, $b \leftarrow \beta_{\mathbf{c}_{h,1}} + \beta_{\mathbf{c}_{h,2}}$ and $j \leftarrow j + 1$.

L3. [Increment leaf counts.] Set $v \leftarrow \mathbf{u}_h$. Then, while $v \neq 0$, set $\beta_v \leftarrow \beta_v + b$ and $v \leftarrow \pi_v$. Afterwards, if $j \leq M$ and $\mathbf{l}_{\mathcal{I}_j} = x$, go to L2.

L4. [Visit tree.] Visit (π, β). If $j > M$ terminate the algorithm; otherwise, set $x \leftarrow \mathbf{l}_{\mathcal{I}_j}$.

L5. [Remove record.] Set $h \leftarrow \mathcal{O}_k$, $\pi_{\mathbf{c}_{h,1}} \leftarrow \pi_{\mathbf{c}_{h,2}} \leftarrow 0$, $b \leftarrow \beta_{\mathbf{c}_{h,1}} + \beta_{\mathbf{c}_{h,2}}$ and $k \leftarrow k + 1$.

L6. [Decrement leaf counts.] Set $v \leftarrow \mathbf{u}_h$. Then, while $v \neq 0$, set $\beta_v \leftarrow \beta_v - b$ and $v \leftarrow \pi_v$. Afterwards, if $\mathbf{r}_{\mathcal{O}_k} = x$, go to L5; otherwise, go to L2.

Algorithm L works in the same manner as Algorithm T: for each tree transition, we remove the stale records that no longer apply to the genomic interval currently under consideration, and apply all new records that begin at location x. We update the sparse tree π by applying a record in step L2, and then update the leaf count β to account for this new node assignment. In step L3 we propagate the corresponding leaf count gain up to the root, before returning to L2 if necessary. Once we have applied all of the inbound records we then visit the tree by making π and β available to the user in L4. Then, if any more trees remain, we move on by removing the outbound records in steps L5 and L6, updating β to account for the corresponding loss in leaf counts. The correctness of the algorithm depends on the ordering of the index vectors $\mathcal{I}$ and $\mathcal{O}$. Records are always inserted in increasing order of time, and always removed in decreasing order of time within a tree transition. Therefore, for any record in which subtrees rooted at c_1 and c_2 become the children of u, we are guaranteed that these subtrees are complete and that β_{c_1} and β_{c_2} are correct. Removing outbound records in reverse order of time similarly guarantees that the leaf counts within the disconnected subtrees that we create are maintained correctly.

Algorithm L clearly examines each record at most once in steps L2 and L5. Steps L3 and L6 contain loops to propagate leaf counts up the tree, and are therefore not constant time operations. Since coalescent genealogies are asymptotically balanced [84], the expected height of a tree (in terms of the number of nodes) is $\log_2 n$. Therefore, the cost of steps L3 and L6 is $O(\log_2 n)$ per record, leading to a $\log_2 n$ extra cost over Algorithm T. In practical terms, this extra cost is negligible. For example, `msprime` automatically maintains counts for all leaves (and optionally can maintain counts for specific subsets) when doing all tree transitions. The 3 second time

quoted above required to iterate over all 1.1 million trees in the large simulation example includes the cost of maintaining counts for all 10^5 leaves at all internal nodes. To demonstrate this efficiency, we ran a simple genome wide association test, where we split the sample into 50,000 cases and controls. One of the most powerful and popular applications for running such association tests is `plink` [85]. After converting the simulated data to a 29G BED file, the stable version of `plink` (1.07) required 176 minutes to run a simple association test. The development version of `plink` (1.9) required 54 seconds. Using `msprime`'s Python API, the same odds-ratio test required around 10 seconds.

Results and Discussion

The primary contribution of this paper is to introduce a new encoding for the correlated trees resulting from simulations of the coalescent with recombination. This encoding follows on from previous work in which trees are encoded as integer vectors [49, 50], but makes the crucial change that tree vectors are sparse. Using this encoding, the effects of each coalescence event are stored as simple fixed-size records that provide sufficient information to recover all marginal genealogies after the simulation has completed. This approach leads to very large gains in simulation performance over classical simulators such as `ms`, so that the exact simulation of genealogies for the coalescent with recombination over chromosome scales is feasible for the first time. We have presented an implementation based on the sparse tree encoding called `msprime`, which is faster than all other simulators for large sample sizes. This simulator supports the full discrete population structure and demographic event model provided by `ms` along with variable recombination rates. We plan to include populations evolving in continuous space [86–88] and gene conversion [89] in subsequent releases.

Coalescence records also lead to an extremely compact storage format that is several orders of magnitude smaller than the most compact method currently available. Despite this very high level of compression, accessing the genealogical data is very efficient. In an example with 100,000 samples, we saw a roughly 40,000-fold reduction in file size over the Newick tree encoding, and a greater than million-fold decrease in the time required to iterate over the genealogies compared to several popular libraries. This efficiency is gained through very simple algorithms that we have stated rigorously and unambiguously, and also analysed in terms of their computational complexity. Being able to process such large sample sizes is not an idle curiosity; on the contrary, we have a pressing need to work with such datasets. We envisage three immediate uses for our work.

Firstly, sequencing projects are being conducted on an unprecedented scale [90–95], and the storage and analysis of these data pose serious computational challenges. Sophisticated new methods are being developed to organise and analyse information on this immense scale [53–55]. Developers have struggled to generate simulated data on a similar scale [53, 54], as present day simulators perform poorly on these huge sample sizes. Using `msprime`, the time required to generate genome scale data for hundreds of thousands of samples is reduced from weeks to minutes.

Secondly, prospective studies such as UK Biobank [96, 97] are collecting genetic and high-dimensional phenotypic data for hundreds of thousands of samples. The key statistical method to interrogate such data is the genome wide association study (GWAS) [98], and large sample size has been identified as the single most important factor in determining the power of these studies [83]. Simulation plays a key role in GWAS, and typically proceeds by superimposing the disease model of interest on haplotypes obtained via various methods [99]. Because the accurate modelling of linkage disequilibrium is essential in disease genetics [100], recombination must be incorporated. Resampling methods [83, 101–103] generate simulated haplotypes based on an existing reference panel, and provide a good match to observed linkage patterns.

However, there is some bias associated with this process, and there are statistical difficulties when the size of the sample required is larger than the reference panel. Other methods obtain simulated haplotypes from population genetics models via forwards-in-time [104, 105] or coalescent [106, 107] simulations. None of these methods can efficiently handle the huge sample sizes required, however. A simulator for high dimensional phenotype data based on `msprime` could alleviate these performance issues and be a key application for the library.

Thirdly, today's large sample sizes provide us with an unprecedented opportunity to understand the history and geographic structure of our species. Aside from its intrinsic interest, correctly accounting for population stratification is critical for the interpretation of association studies [108, 109], particularly for rare variants [110, 111]. Researchers are seeking to understand fine scale population structure using methods based on principal component analysis [112], admixture fractions [113–115], length of haplotype blocks [116–118] and allele frequencies [119]. To date, it has been challenging to assess the accuracy of these methods, as simulations struggle to match the required sequence lengths and sample sizes. Furthermore, methods based on the SMC approximation [17, 18] have been tested using SMC simulations out of necessity, making it difficult to assess the impact of the approximation on accuracy. Simulations of the exact coalescent with recombination at chromosome scales for large sample sizes and arbitrary demographies will be an invaluable tool for developers of such methods.

As we have demonstrated, the tree sequence structure leads to very efficient algorithms, and allows us to encode simulated data very compactly. We would also wish to encode biological data in this structure so that we can apply these algorithms to analyse real data. However, to do this we must estimate a tree sequence from data, which is a non-trivial task. Nonetheless, there has been much work in this area [120] with several heuristic [121] and more principled approaches that may be adopted [19, 122]. Using the PBWT [53] to find long haplotypes (which will usually correspond to long records) seems like a particularly promising avenue.

Finally, an interesting issue arises when we consider the problem of inferring a tree sequence from data. Suppose we have observed a set of haplotypes resulting from a coalescent simulation with infinite sites mutations occurring at a very high rate. Under these conditions, the underlying tree sequence can be recovered exactly from the data, but the corresponding ARG (i.e., the specific realisation of the ARG that was traversed by Hudson's algorithm) cannot. For example, a recombination may have occurred during the simulation that was immediately followed by a common ancestor event involving the same lineages. These nodes in the ARG can have no effect on the data, and are therefore unobservable. To put this in another way, there is no observable information in an ARG that is not in a tree sequence. Given this representational sufficiency and the storage and processing efficiencies demonstrated in this article, we would argue that a tree sequence is a more natural and powerful representation of observed genetic variation than an ARG.

Supporting Information

S1 Text. Detailed listing of Hudson's algorithm.
(PDF)

S2 Text. Illustration of Hudson's algorithm.
(PDF)

S1 Fig. Comparisons of the running times for various coalescent simulators to generate mutations for varying sequence length and sample size. We use a scaled mutation rate of $\theta = 4N_e\mu = 0.0004$.
(PDF)

S2 Fig. The corresponding maximum memory usages for the simulators in S1 Fig.
(PDF)

S3 Fig. The mean number of recombination breakpoints for the simulations in Fig 3 along with the theoretical prediction (black line). This plot shows that the number of recombination events within ancestral material for these simulations is identical for all simulators and agrees very well with the theoretical value of ρH_{n-1}, where H_n is the nth Harmonic number.
(PDF)

S4 Fig. Comparison of simulation times with `msprime` and `scrm` for a sample size of $n = 20$ with increasing sequence length. Several different approximation levels are shown for `scrm` using the `-l` option. The `-l 500r` option is described as a conservative value giving very good accuracy, and `-l 100r` is recommended as a good compromise between running time and accuracy.
(PDF)

Acknowledgments

We would like to thank Richard Durbin for helpful discussions and insights.

Author Contributions

Conceived and designed the experiments: JK AME GM. Performed the experiments: JK. Analyzed the data: JK. Wrote the paper: JK GM.

References

1. Kingman JFC. The coalescent. Stoch Proc Appl. 1982; 13(3):235–248. doi: 10.1016/0304-4149(82)90011-4
2. Hudson RR. Testing the constant-rate neutral allele model with protein sequence data. Evolution. 1983; 37(1):203–217. doi: 10.2307/2408186
3. Wakeley J. Coalescent theory: an introduction. Englewood, Colorado: Roberts and Company; 2008.
4. Hudson RR. Gene genealogies and the coalescent process. Oxford Surveys in Evolutionary Biology. 1990; 7:1–44.
5. Hudson RR. Properties of a neutral allele model with intragenic recombination. Theor Popul Biol. 1983; 23:183–201. doi: 10.1016/0040-5809(83)90013-8 PMID: 6612631
6. Hudson RR, Kaplan N. Statistical properties of the number of recombination events in the history of a sample of DNA sequences. Genetics. 1985; 111(1):147–164. PMID: 4029609
7. Kaplan N, Hudson RR. The use of sample genealogies for studying a selectively neutral *m*-loci model with recombination. Theor Popul Biol. 1985; 28:382–396. doi: 10.1016/0040-5809(85)90036-X PMID: 4071443
8. Griffiths RC. The two-locus ancestral graph. In: Selected Proceedings of the Sheffield Symposium on Applied Probability. vol. 18; 1991. p. 100–117.
9. Griffiths RC, Marjoram P. An ancestral recombination graph. In: Donnelly P, Tavaré S, editors. Progress in Population Genetics and Human Evolution, IMA Volumes in Mathematics and its Applications. vol. 87. Berlin: Springer-Verlag; 1997. p. 257–270.
10. Ethier SN, Griffiths RC. On the two-locus sampling distribution. J Math Biol. 1990; 29:131–159. doi: 10.1007/BF00168175
11. Hudson RR. Generating samples under a Wright-Fisher neutral model of genetic variation. Bioinformatics. 2002; 18(2):337–338. doi: 10.1093/bioinformatics/18.2.337 PMID: 11847089
12. McVean GAT, Cardin NJ. Approximating the coalescent with recombination. Philos Trans R Soc Lond B Biol Sci. 2005; 360:1387–1393. doi: 10.1098/rstb.2005.1673 PMID: 16048782
13. Marjoram P, Wall JD. Fast "coalescent" simulation. BMC Genet. 2006; 7:16. doi: 10.1186/1471-2156-7-16 PMID: 16539698
14. Chen GK, Marjoram P, Wall JD. Fast and flexible simulation of DNA sequence data. Genome Res. 2009; 19:136–142. doi: 10.1101/gr.083634.108 PMID: 19029539

15. Excoffier L, Foll M. fastsimcoal: a continuous-time coalescent simulator of genomic diversity under arbitrarily complex evolutionary scenarios. Bioinformatics. 2011; 27(9):1332–1334. doi: 10.1093/bioinformatics/btr124 PMID: 21398675

16. Staab PR, Zhu S, Metzler D, Lunter G. scrm: efficiently simulating long sequences using the approximated coalescent with recombination. Bioinformatics. 2014; 31(10):1680–1682. doi: 10.1093/bioinformatics/btu861

17. Li H, Durbin R. Inference of human population history from individual whole-genome sequences. Nature. 2011; 475:493–496. doi: 10.1038/nature10231 PMID: 21753753

18. Schiffels S, Durbin R. Inferring human population size and separation history from multiple genome sequences. Nat Genet. 2014; 46:919–925. doi: 10.1038/ng.3015 PMID: 24952747

19. Rasmussen MD, Hubisz MJ, Gronau I, Siepel A. Genome-wide inference of ancestral recombination graphs. PLoS Genet. 2014; 10(5):e1004342. doi: 10.1371/journal.pgen.1004342 PMID: 24831947

20. Liang M, Nielsen R. The lengths of admixture tracts. Genetics. 2014; 197:953–967. doi: 10.1534/genetics.114.162362 PMID: 24770332

21. Eriksson A, Mahjani B, Mehlig B. Sequential Markov coalescent algorithms for population models with demographic structure. Theor Popul Biol. 2009; 76(2):84–91. doi: 10.1016/j.tpb.2009.05.002 PMID: 19433100

22. Arenas M, Posada D. Coalescent simulation of intracodon recombination. Genetics. 2010; 184 (2):429–437. doi: 10.1534/genetics.109.109736 PMID: 19933876

23. Peischl S, Koch E, Guerrero R, Kirkpatrick M. A sequential coalescent algorithm for chromosomal inversions. Heredity. 2013; 111:200–209. doi: 10.1038/hdy.2013.38 PMID: 23632894

24. Wiuf C, Hein J. Recombination as a point process along sequences. Theor Popul Biol. 1999; 55 (3):248–259. doi: 10.1006/tpbi.1998.1403 PMID: 10366550

25. Wakeley J, Takahashi T. Gene genealogies when the sample size exceeds the effective size of the population. Mol Biol Evol. 2003; 20(2):208–213. doi: 10.1093/molbev/msg024 PMID: 12598687

26. Maruvka YE, Shnerb NM, Bar-Yam Y, Wakeley J. Recovering population parameters from a single gene genealogy: an unbiased estimator of the growth rate. Mol Biol Evol. 2011; 28(5):1617–1631. doi: 10.1093/molbev/msq331 PMID: 21172828

27. Bhaskar A, Clark AG, Song YS. Distortion of genealogical properties when the sample is very large. Proc Natl Acad Sci U S A. 2014; 111(6):2385–2390. doi: 10.1073/pnas.1322709111 PMID: 24469801

28. Excoffier L, Novembre J, Schneider S. SIMCOAL: a general coalescent program for the simulation of molecular data in interconnected populations with arbitrary demography. J Hered. 2000; 91(6):506–509. doi: 10.1093/jhered/91.6.506 PMID: 11218093

29. Laval G, Excoffier L. SIMCOAL 2.0: a program to simulate genomic diversity over large recombining regions in a subdivided population with a complex history. Bioinformatics. 2004; 20(15):2485–2487. doi: 10.1093/bioinformatics/bth264 PMID: 15117750

30. Anderson CN, Ramakrishnan U, Chan YL, Hadly EA. Serial SimCoal: a population genetics model for data from multiple populations and points in time. Bioinformatics. 2005; 21(8):1733–1734. doi: 10.1093/bioinformatics/bti154 PMID: 15564305

31. Liang L, Zöllner S, Abecasis GR. GENOME: a rapid coalescent-based whole genome simulator. Bioinformatics. 2007; 23(12):1565–1567. doi: 10.1093/bioinformatics/btm138 PMID: 17459963

32. Spencer CC, Coop G. SelSim: a program to simulate population genetic data with natural selection and recombination. Bioinformatics. 2004; 20(18):3673–3675. doi: 10.1093/bioinformatics/bth417 PMID: 15271777

33. Teshima KM, Innan H. mbs: modifying Hudson's ms software to generate samples of DNA sequences with a biallelic site under selection. BMC Bioinformatics. 2009; 10(1):166. doi: 10.1186/1471-2105-10-166 PMID: 19480708

34. Ewing G, Hermisson J. MSMS: A coalescent simulation program including recombination, demographic structure, and selection at a single locus. Bioinformatics. 2010; 26(16):2064–2065. doi: 10.1093/bioinformatics/btq322 PMID: 20591904

35. Shlyakhter I, Sabeti PC, Schaffner SF. Cosi2: an efficient simulator of exact and approximate coalescent with selection. Bioinformatics. 2014; 30(23):3427–3429. doi: 10.1093/bioinformatics/btu562 PMID: 25150247

36. Hellenthal G, Stephens M. msHOT: modifying Hudson's ms simulator to incorporate crossover and gene conversion hotspots. Bioinformatics. 2007; 23(4):520–521. doi: 10.1093/bioinformatics/btl622 PMID: 17150995

37. Arenas M, Posada D. Recodon: coalescent simulation of coding DNA sequences with recombination, migration and demography. BMC Bioinformatics. 2007; 8(1):458. doi: 10.1186/1471-2105-8-458 PMID: 18028540

38. Zhu S, Degnan JH, Goldstien SJ, Eldon B. Hybrid-Lambda: simulation of multiple merger and Kingman gene genealogies in species networks and species trees. BMC Bioinformatics. 2015; 16(292).

39. Mailund T, Schierup MH, Pedersen CN, Mechlenborg PJ, Madsen JN, Schauser L. CoaSim: a flexible environment for simulating genetic data under coalescent models. BMC Bioinformatics. 2005; 6 (1):252. doi: 10.1186/1471-2105-6-252 PMID: 16225674

40. Ramos-Onsins SE, Mitchell-Olds T. Mlcoalsim: multilocus coalescent simulations. Evol Bioinform Online. 2007; 3:41. PMID: 19430603

41. Wang Y, Zhou Y, Li L, Chen X, Liu Y, Ma ZM, et al. A new method for modeling coalescent processes with recombination. BMC Bioinformatics. 2014; 15(1):273. doi: 10.1186/1471-2105-15-273 PMID: 25113665

42. Carvajal-Rodríguez A. Simulation of genomes: a review. Curr Genomics. 2008; 9(3):155. doi: 10.2174/138920208784340759 PMID: 19440512

43. Liu Y, Athanasiadis G, Weale ME. A survey of genetic simulation software for population and epidemiological studies. Hum Genomics. 2008; 3(1):79. doi: 10.1186/1479-7364-3-1-79 PMID: 19129092

44. Arenas M. Simulation of molecular data under diverse evolutionary scenarios. PLoS Comput Biol. 2012; 8(5):e1002495. doi: 10.1371/journal.pcbi.1002495 PMID: 22693434

45. Yuan X, Miller DJ, Zhang J, Herrington D, Wang Y. An overview of population genetic data simulation. J Comput Biol. 2012; 19(1):42–54. doi: 10.1089/cmb.2010.0188 PMID: 22149682

46. Hoban S, Bertorelle G, Gaggiotti OE. Computer simulations: tools for population and evolutionary genetics. Nat Rev Genet. 2012; 13(2):110–122. PMID: 22230817

47. Yang T, Deng HW, Niu T. Critical assessment of coalescent simulators in modeling recombination hotspots in genomic sequences. BMC Bioinformatics. 2014; 15:3. doi: 10.1186/1471-2105-15-3 PMID: 24387001

48. Knuth DE. Combinatorial Algorithms, Part 1. vol. 4A of *The Art of Computer Programming*. Upper Saddle River, New Jersey: Addison-Wesley; 2011.

49. Kelleher J, Barton NH, Etheridge AM. Coalescent simulation in continuous space. Bioinformatics. 2013; 29(7):955–956. doi: 10.1093/bioinformatics/btt067 PMID: 23391497

50. Kelleher J, Etheridge AM, Barton NH. Coalecent simulation in continuous space: algorithms for large neighbourhood size. Theor Popul Biol. 2014; 95:13–23. doi: 10.1016/j.tpb.2014.05.001 PMID: 24910324

51. Fenwick PM. A new data structure for cumulative frequency tables. Software: Practice and Experience. 1994; 24:327–336.

52. Fenwick PM. A new data structure for cumulative frequency tables: an improved frequency-to-symbol algorithm. The University of Auckland, Department of Computer Science; 1995. 110.

53. Durbin R. Efficient haplotype matching and storage using the positional Burrows-Wheeler transform (PBWT). Bioinformatics. 2014; 30(9):1266–1272. doi: 10.1093/bioinformatics/btu014 PMID: 24413527

54. Layer RM, Kindlon N, Karczewski KJ, Exome Aggregation Consortium, Quinlan AR. Efficient genotype compression and analysis of large genetic-variation data sets. Nat Methods. 2016; 13(1):63–65. doi: 10.1038/nmeth.3654 PMID: 26550772

55. Li H. BGT: efficient and flexible genotype query across many samples. Bioinformatics. 2016; 32 (4):590–592. doi: 10.1093/bioinformatics/btv613 PMID: 26500154

56. Felsenstein J. PHYLIP—phylogeny inference package (version 3.2). Cladistics. 1989; 5:164–166.

57. Vos RA, Balhoff JP, Caravas JA, Holder MT, Lapp H, Maddison WP, et al. NeXML: rich, extensible, and verifiable representation of comparative data and metadata. Syst Biol. 2012; 61(4):675–689. doi: 10.1093/sysbio/sys025 PMID: 22357728

58. Maddison DR, Swofford DL, Maddison WP. Nexus: an extensible file format for systematic information. Syst Biol. 1997; 46(4):590–621. doi: 10.1093/sysbio/46.4.590 PMID: 11975335

59. Zmasek CM, Eddy SR. ATV: display and manipulation of annotated phylogenetic trees. Bioinformatics. 2001; 17(4):383–384. doi: 10.1093/bioinformatics/17.4.383 PMID: 11301314

60. Han MV, Zmasek CM. phyloXML: XML for evolutionary biology and comparative genomics. BMC Bioinformatics. 2009; 10(356).

61. Morin MM, Moret BME. NetGen: generating phylogenetic networks with diploid hybrids. Bioinformatics. 2006; 22(15):1921–1923. doi: 10.1093/bioinformatics/btl191 PMID: 16717070

62. Buendia P, Narasimhan G. Serial NetEvolve: a flexible utility for generating serially-sampled sequences along a tree or recombinant network. Bioinformatics. 2006; 22(18):2313–2314. doi: 10.1093/bioinformatics/btl387 PMID: 16844708

63. Cardona G, Rosselló F, Valiente G. Extended Newick: it is time for a standard representation of phylogenetic networks. BMC Bioinformatics. 2008; 9:532. doi: 10.1186/1471-2105-9-532 PMID: 19077301

64. Than C, Ruths D, Nakhleh L. PhyloNet: a software package for analyzing and reconstructing reticulate evolutionary relationships. BMC Bioinformatics. 2008; 9:322. doi: 10.1186/1471-2105-9-322 PMID: 18662388

65. McGill JR, Walkup EA, Kuhner MK. GraphML specializations to codify ancestral recombinant graphs. Fron Genet. 2013; 4:146.

66. Stajich JE, Block D, Boulez K, Brenner SE, Chervitz SA, Dagdigian C, et al. The Bioperl Toolkit: Perl Modules for the Life Sciences. Genome Res. 2002; 12(10):1611–1618. doi: 10.1101/gr.361602 PMID: 12368254

67. Cock PJA, Antao T, Chang JT, Chapman BA, Cox CJ, Dalke A, et al. Biopython: freely available Python tools for computational molecular biology and bioinformatics. Bioinformatics. 2009; 25 (11):1422–1423. doi: 10.1093/bioinformatics/btp163 PMID: 19304878

68. Sukumaran J, Holder MT. DendroPy: a Python library for phylogenetic computing. Bioinformatics. 2010; 26(12):1569–1571. doi: 10.1093/bioinformatics/btq228 PMID: 20421198

69. Huerta-Cepas J, Dopazo J, Gabaldón T. ETE: a python environment for tree exploration. BMC Bioinformatics. 2010; 11:24. doi: 10.1186/1471-2105-11-24 PMID: 20070885

70. Paradis E, Claude J, Strimmer K. APE: analyses of phylogenetics and evolution in R language. Bioinformatics. 2004; 20:289–290. doi: 10.1093/bioinformatics/btg412 PMID: 14734327

71. Galtier N, Depaulis F, Barton NH. Detecting bottlenecks and selective sweeps from DNA sequence polymorphism. Genetics. 2000; 155(2):981–987. PMID: 10835415

72. Donnelly P, Kurtz TG. Particle representations for measure-valued population models. Ann Probab. 1999; 27(1):166–205. doi: 10.1214/aop/1022677258

73. Pitman J. Coalescents with multiple collisions. Ann Probab. 1999; 27(4):1870–1902.

74. Sagitov S. The general coalescent with asynchronous mergers of ancestral lines. J Appl Probab. 1999; 36(4):1116–1125. doi: 10.1239/jap/1032374759

75. Wiuf C, Hein J. The ancestry of a sample of sequences subject to recombination. Genetics. 1999; 151 (3):1217–1228. PMID: 10049937

76. Song YS. On the combinatorics of rooted binary phylogenetic trees. Ann Comb. 2003; 7(3):365–379. doi: 10.1007/s00026-003-0192-0

77. Song YS. Properties of subtree-prune-and-regraft operations on totally-ordered phylogenetic trees. Ann Comb. 2006; 10(1):147–163. doi: 10.1007/s00026-006-0279-5

78. Kelleher J, Ness RW, Halligan DL. Processing genome scale tabular data with wormtable. BMC Bioinformatics. 2013; 14:356. doi: 10.1186/1471-2105-14-356 PMID: 24308302

79. The HDF Group. Hierarchical Data Format, version 5; 1997–2015. http://www.hdfgroup.org/HDF5/.

80. Matthews SJ, Sul SJ, Williams TL. A novel approach for compressing phylogenetic trees. In: Borodovsky M, Gogarten J, Przytycka T, Rajasekaran S, editors. Bioinformatics Research and Applications. vol. 6053 of Lecture Notes in Computer Science. Springer Berlin Heidelberg; 2010. p. 113–124.

81. Samet H. The Design and Analysis of Spatial Data Structures. Upper Saddle River, New Jersey: Addison-Wesley; 1989.

82. Charlesworth B, Charlesworth D. Elements of Evolutionary Genetics. Greenwood Village, Colorado: Roberts and Company; 2010.

83. Spencer CC, Su Z, Donnelly P, Marchini J. Designing genome-wide association studies: sample size, power, imputation, and the choice of genotyping chip. PLoS Genet. 2009; 5(5):e1000477. doi: 10.1371/journal.pgen.1000477 PMID: 19492015

84. Li H, Wiehe T. Coalescent tree imbalance and a simple test for selective sweeps based on microsatellite variation. PLoS Comput Biol. 2013; 9(5):e1003060. doi: 10.1371/journal.pcbi.1003060 PMID: 23696722

85. Purcell S, Neale B, Todd-Brown K, Thomas L, Ferreira MAR, Bender D, et al. PLINK: a tool set for whole-genome association and population-based linkage analyses. Am J Hum Genet. 2007; 81 (3):559–575. doi: 10.1086/519795 PMID: 17701901

86. Barton NH, Etheridge AM, Véber A. A new model for evolution in a spatial continuum. Electron J of Probab. 2010; 15:7. doi: 10.1214/EJP.v15-741

87. Barton NH, Kelleher J, Etheridge AM. A new model for extinction and recolonisation in two dimensions: quantifying phylogeography. Evolution. 2010; 64(9):2701–2715. doi: 10.1111/j.1558-5646.2010.01019.x PMID: 20408876

88. Barton NH, Etheridge AM, Véber A. Modelling evolution in a spatial continuum. J Stat Mech. 2013; P01002.

89. Wiuf C, Hein J. The coalescent with gene conversion. Genetics. 2000; 155(1):451–462. PMID: 10790416

90. Genome of the Netherlands Consortium, et al. Whole-genome sequence variation, population structure and demographic history of the Dutch population. Nat Genet. 2014; 46(8):818–825. doi: 10.1038/ng.3021 PMID: 24974849

91. UK10K Consortium, et al. The UK10K project identifies rare variants in health and disease. Nature. 2015; 526(7571):82–90. doi: 10.1038/nature14962 PMID: 26367797

92. 1000 Genomes Project Consortium, et al. A global reference for human genetic variation. Nature. 2015; 526(7571):68–74. doi: 10.1038/nature15393 PMID: 26432245

93. Gudbjartsson DF, Helgason H, Gudjonsson SA, Zink F, Oddson A, Gylfason A, et al. Large-scale whole-genome sequencing of the Icelandic population. Nat Genet. 2015; 47(5):435–444. doi: 10.1038/ng.3247 PMID: 25807286

94. Eisenstein M. Big data: The power of petabytes. Nature. 2015; 527(7576):S2–S4. doi: 10.1038/527S2a PMID: 26536222

95. Stephens ZD, Lee SY, Faghri F, Campbell RH, Zhai C, Efron MJ, et al. Big data: astronomical or genomical? PLoS Biol. 2015; 13(7):e1002195. doi: 10.1371/journal.pbio.1002195 PMID: 26151137

96. Collins R. UK biobank: the need for large prospective epidemiological studies. J Epidemiol Community Health. 2011; 65(1):A37. doi: 10.1136/jech.2011.142976b.2

97. Wain LV, Shrine N, Miller S, Jackson VE, Ntalla I, Artigas MS, et al. Novel insights into the genetics of smoking behaviour, lung function, and chronic obstructive pulmonary disease (UK BiLEVE): a genetic association study in UK Biobank. Lancet Respir Med. 2015; 3(10):769–781. doi: 10.1016/S2213-2600(15)00283-0 PMID: 26423011

98. Manolio TA. Bringing genome-wide association findings into clinical use. Nat Rev Genet. 2013; 14(8):549–558. doi: 10.1038/nrg3523 PMID: 23835440

99. Yang J, Lee SH, Goddard ME, Visscher PM. GCTA: a tool for genome-wide complex trait analysis. Am J Hum Genet. 2011; 88(1):76–82. doi: 10.1016/j.ajhg.2010.11.011 PMID: 21167468

100. Schaffner SF, Foo C, Gabriel S, Reich D, Daly MJ, Altshuler D. Calibrating a coalescent simulation of human genome sequence variation. Genome Res. 2005; 15(11):1576–1583. doi: 10.1101/gr.3709305 PMID: 16251467

101. Marchini J, Howie B, Myers S, McVean G, Donnelly P. A new multipoint method for genome-wide association studies by imputation of genotypes. Nat Genet. 2007; 39(7):906–913. doi: 10.1038/ng2088 PMID: 17572673

102. Li C, Li M. GWAsimulator: a rapid whole-genome simulation program. Bioinformatics. 2008; 24(1):140–142. doi: 10.1093/bioinformatics/btm549 PMID: 18006546

103. Su Z, Marchini J, Donnelly P. HAPGEN2: simulation of multiple disease SNPs. Bioinformatics. 2011; 27(16):2304–2305. doi: 10.1093/bioinformatics/btr341 PMID: 21653516

104. Lohmueller KE, Indap AR, Schmidt S, Boyko AR, Hernandez RD, Hubisz MJ, et al. Proportionally more deleterious genetic variation in European than in African populations. Nature. 2008; 451(7181):994–997. doi: 10.1038/nature06611 PMID: 18288194

105. Lohmueller KE. The impact of population demography and selection on the genetic architecture of complex traits. PLoS Genet. 2014; 10(5):e1004379. doi: 10.1371/journal.pgen.1004379 PMID: 24875776

106. Günther T, Gawenda I, Schmid KJ. phenosim—A software to simulate phenotypes for testing in genome-wide association studies. BMC Bioinformatics. 2011; 12(1):265. doi: 10.1186/1471-2105-12-265 PMID: 21714868

107. Chung RH, Shih CC. SeqSIMLA: a sequence and phenotype simulation tool for complex disease studies. BMC Bioinformatics. 2013; 14(1):199. doi: 10.1186/1471-2105-14-199 PMID: 23782512

108. Marchini J, Cardon LR, Phillips MS, Donnelly P. The effects of human population structure on large genetic association studies. Nat Genet. 2004; 36(5):512–517. doi: 10.1038/ng1337 PMID: 15052271

109. McCarthy MI, Abecasis GR, Cardon LR, Goldstein DB, Little J, Ioannidis JP, et al. Genome-wide association studies for complex traits: consensus, uncertainty and challenges. Nat Rev Genet. 2008; 9(5):356–369. doi: 10.1038/nrg2344 PMID: 18398418

110. Mathieson I, McVean G. Differential confounding of rare and common variants in spatially structured populations. Nat Genet. 2012; 44(3):243–246. doi: 10.1038/ng.1074 PMID: 22306651

111. Mathieson I, McVean G. Demography and the age of rare variants. PLoS Genet. 2014; 10(8): e1004528. doi: 10.1371/journal.pgen.1004528 PMID: 25101869

112. Novembre J, Johnson T, Bryc K, Kutalik Z, Boyko AR, Auton A, et al. Genes mirror geography within Europe. Nature. 2008; 456(7218):98–101. doi: 10.1038/nature07331 PMID: 18758442

113. Alexander DH, Novembre J, Lange K. Fast model-based estimation of ancestry in unrelated individuals. Genome Res. 2009; 19(9):1655–1664. doi: 10.1101/gr.094052.109 PMID: 19648217

114. Lawson DJ, Hellenthal G, Myers S, Falush D. Inference of population structure using dense haplotype data. PLoS Genet. 2012; 8(1):e1002453–e1002453. doi: 10.1371/journal.pgen.1002453 PMID: 22291602

115. Liu Y, Nyunoya T, Leng S, Belinsky SA, Tesfaigzi Y, Bruse S. Softwares and methods for estimating genetic ancestry in human populations. Hum Genomics. 2013; 7(1). doi: 10.1186/1479-7364-7-1

116. Ralph P, Coop G. The geography of recent genetic ancestry across Europe. PLoS Biol. 2013; 11(5): e1001555. doi: 10.1371/journal.pbio.1001555 PMID: 23667324

117. Harris K, Nielsen R. Inferring demographic history from a spectrum of shared haplotype lengths. PLoS Genet. 2013; 9(6):e1003521. doi: 10.1371/journal.pgen.1003521 PMID: 23754952

118. Barton NH, Etheridge AM, Kelleher J, Véber A. Inference in two dimensions: allele frequencies versus lengths of shared sequence blocks. Theor Popul Biol. 2013; 87:105–119. doi: 10.1016/j.tpb.2013.03.001 PMID: 23506734

119. Gutenkunst RN, Hernandez RD, Williamson SH, Bustamante CD. Inferring the joint demographic history of multiple populations from multidimensional SNP frequency data. PLoS Genet. 2009; 5(10): e1000695. doi: 10.1371/journal.pgen.1000695 PMID: 19851460

120. Gusfield D. ReCombinatorics. Cambridge Massachusetts: MIT Press; 2014.

121. Minichiello MJ, Durbin R. Mapping trait loci by use of inferred ancestral recombination graphs. Am J Hum Genet. 2006; 79:910–922. doi: 10.1086/508901 PMID: 17033967

122. O'Fallon BD. ACG: rapid inference of population history from recombining nucleotide sequences. BMC Bioinformatics. 2013; 14(1):40. doi: 10.1186/1471-2105-14-40 PMID: 23379678