Efficient ancestry and mutation simulation with msprime 1.0

Franz Baumdicker 1,† Gertjan Bisschop 2,† Daniel Goldstein 3,24,† Graham Gower 1,3,26,§ Aaron P. Ragsdale 4,† and Jerome Kelleher 5,†
Georgia Tsambos 1,§ Sha Zhu 7,† Bjarki Eldon 8,§ E. Castedo Ellerman 9, Jared G. Galloway 10,11,§ Jere Koskela 10,11,‡
Aniella L. Gladstein 12,13,§ Gregor Gorjanc 14,15,§ Bing Guo 16,§ Ben Jeffery 17,§ Warren W. Kretzschmar 18,‡
Konrad Kohse 19, Michael Matschiner 20,§ Dominic Nelson 21,§ Nathaniel S. Pope 22,§ Consuelo D. Quinto-Cortés 23,‡
Murillo F. Rodrigues 24,§ Kumar Saunack 25,§ Thibaut Sellinger 26,§ Kevin Thornton 27,§ Hugo van Kemenade 28,‡,
Anthony W. Woehns 29,7,24,§ Simon Gravel 30,§ Andrew D. Kern 31,§ Jere Koskela 31,‡,§ Peter L. Ralph 31,7,26,§, and Jerome Kelleher 1,7,∗

1Cluster of Excellence “Controlling Microbes to Fight Infections”, Mathematical and Computational Population Genetics, University of Tübingen, 72076 Tübingen, Germany,
2Institute of Evolutionary Biology, The University of Edinburgh, Edinburgh EH9 3JL, UK,
3Khoury College of Computer Sciences, Northeastern University, Boston, MA 02115, USA,
4Lundbeck GeoGenetics Centre, Gloe Institute, University of Copenhagen, 1350 Copenhagen K, Denmark,
5Department of Integrative Biology, University of Wisconsin-Madison, Madison, WI 53706, USA,
6Melbourne Integrative Genomics, School of Mathematics and Statistics, University of Melbourne, Parkville, VIC 3010, Australia,
7Big Data Institute, Li Ka Shing Centre for Health Information and Discovery, University of Oxford, Oxford OX3 7LF, UK,
8Leibniz Institute for Evolution and Biodiversity Science, Museum für Naturkunde, Berlin 10117, Germany,
9Fresh Pond Research Institute, Cambridge, MA 02140, USA,
10Department of Biology, Institute of Ecology and Evolution, University of Oregon, Eugene, OR 97403-5289, USA,
11Computational Biology Program, Fred Hutchinson Cancer Research Center, Seattle, WA 98102, USA,
12Department of Genetics, University of North Carolina at Chapel Hill, Chapel Hill, NC 27599-7264, USA,
13Embark Veterinary, Inc., Boston, MA 02211, USA,
14The Roslin Institute and Royal (Dick) School of Veterinary Studies, University of Edinburgh, Edinburgh EH25 9RG, UK,
15Institute for Genome Sciences, University of Maryland School of Medicine, Baltimore, MD 21201, USA,
16Center for Hematology and Regenerative Medicine, Karolinska Institute, 141 83 Huddinge, Sweden,
17Natural History Museum, University of Oslo, 0318 Oslo, Norway,
18Department of Human Genetics, McGill University, Montréal, QC H3A 0C7, Canada,
19Department of Entomology, Pennsylvania State University, State College, PA 16802, USA,
20National Laboratory of Genomics for Biodiversity (LANGEBIO), Unit of Advanced Genomics, CINVESTAV, Irapuato, Mexico,
21IIT Bombay, Powai, Mumbai 400 076, India,
22Professorship for Population Genetics, Department of Life Science Systems, Technical University of Munich, 85354 Freising, Germany,
23Department of Ecology and Evolutionary Biology, University of California, Irvine, CA 92697, USA,
24Broad Institute of MIT and Harvard, Cambridge, MA 02142, USA,
25Department of Statistics, University of Warwick, Coventry CV4 7AL, UK, and
26Department of Mathematics, University of Oregon, Eugene, OR 97403-5289, USA
†These authors contributed equally to this work, as joint first authors.
‡Independent Researcher.
§These authors contributed equally to this work, as joint last authors.
∗Corresponding author: Email: jerome.kelleher@bdi.ox.ac.uk

Abstract
Stochastic simulation is a key tool in population genetics, since the models involved are often analytically intractable and simulation is usually the only way of obtaining ground-truth data to evaluate inferences. Because of this, a large number of specialized simulation programs have been developed, each filling a particular niche, but with largely overlapping functionality and a substantial duplication of effort. Here, we introduce msprime version 1.0, which efficiently implements ancestry and mutation simulations based on the succinct tree sequence data structure and the tskit library. We summarize msprime’s many features, and show that its performance is excellent, often many times faster and more memory efficient than specialized alternatives. These high-performance features have been thoroughly tested and validated, and built using a collaborative, open source development model, which reduces duplication of effort and promotes software quality via community engagement.

Keywords: simulation; coalescent; mutations; Ancestral Recombination Graphs

Received: September 22, 2021. Accepted: December 3, 2021
© The Author(s) 2021. Published by Oxford University Press on behalf of Genetics Society of America.
This is an Open Access article distributed under the terms of the Creative Commons Attribution License (https://creativecommons.org/licenses/by/4.0/), which permits unrestricted reuse, distribution, and reproduction in any medium, provided the original work is properly cited.
Introduction

The coalescent process (Kingman 1982a, 1982b; Hudson 1983b; Tajima 1983) models the ancestry of a set of sampled genomes, providing a mathematical description of the genealogical tree that relates the samples to one another. It has proved to be a powerful model, and is now central to population genetics (Hudson 1990; Hein et al. 2004; Wakeley 2008). The coalescent is an efficient framework for population genetic simulation, because it allows us to simulate the genetic ancestry for a sample from an idealized population model, without explicitly representing the population in memory or stepping through the generations. Indeed, Hudson (1983b) independently derived the coalescent in order to efficiently simulate data, and used these simulations to characterize an analytically intractable distribution. This inherent efficiency, and the great utility of simulations for a wide range of purposes, has led to dozens of different tools being developed over the decades (Carvajal-Rodriguez 2008; Liu et al. 2008; Arenas 2012; Hoban et al. 2012; Yuan et al. 2012; Yang et al. 2014; Peng et al. 2015).

Two technological developments of recent years, however, pose major challenges to most existing simulation methods. First, fourth-generation sequencing technologies have made complete chromosome-level assemblies possible (Miga et al. 2020), and high quality assemblies are now available for many species. Thus, modeling genetic variation data as a series of unlinked non-recombining loci is no longer a reasonable approximation, and we must fully account for recombination. However, while a genealogical tree relating n samples in the single-locus coalescent can be simulated in O(n) time (Hudson 1990), the coalescent with recombination is far more complex, and programs such as Hudson’s classical ms (Hudson 2002) can only simulate short segments under the influence of recombination. The second challenge facing simulation methods is that sample sizes in genetic studies have grown very quickly in recent years, enabled by the precipitous fall in genome sequencing costs. Human datasets like the UK Biobank (Bycroft et al. 2018) and gnomAD (Karczewski et al. 2020) now consist of hundreds of thousands of genomes and many other datasets on a similar scale are becoming available (Tanjo et al. 2021). Classical simulators such as ms and even fast approximate methods such as acm (Staab et al. 2015) simply cannot cope with such a large number of samples.

The msprime simulator (Kelleher et al. 2016; Kelleher and Lohse 2020) has greatly increased the scope of coalescent simulations, and it is now straightforward to simulate millions of whole chromosomes for a wide range of organisms. The “succinct tree sequence” data structure (Kelleher et al. 2016, 2018, 2019; Wohns et al. 2021), originally introduced as part of msprime, makes it possible to store such large simulations in a few gigabytes, several orders of magnitude smaller than commonly used formats. The succinct tree sequence has also led to major advances in forwards-time simulation (Haller et al. 2018; Kelleher et al. 2018), ancestry inference (Kelleher et al. 2019; Wohns et al. 2021), and calculation of population genetic statistics (Kelleher et al. 2016; Ralph et al. 2020). Through a rigorous open-source community development process, msprime has gained a large number of features since its introduction, making it a highly efficient and flexible platform for population genetic simulation. This paper marks the release of msprime 1.0. We provide an overview of its extensive features, demonstrate its performance advantages over alternative software, and discuss opportunities for ongoing open-source community-based development.

The efficiency of coalescent simulations depends crucially on the assumption of neutrality, and it is important to note that there are many situations in which this will be a poor approximation of biological reality (Johri et al. 2021). In particular, background selection has been shown to affect genome-wide sequence variation in a wide range of species (Charlesworth et al. 1993, 1995; Charlesworth and Jensen 2021). Thus care must be taken to ensure that the results of purely neutral simulations are appropriate for the question and genomic partition under study. A major strength of msprime, however, is that it can be used in conjunction with forwards-time simulators, enabling the simulation of more realistic models than otherwise possible (Haller et al. 2018; Kelleher et al. 2018).

Results

In the following sections, we describe the main features of msprime 1.0, focusing on the aspects that are either new for this version, or in which our approach differs significantly from classical methods (summarized in Table 1). Where appropriate, we benchmark msprime against other simulators, but the comparisons are illustrative and not intended to be systematic or exhaustive. Please see Kelleher et al. (2016) for a performance comparison of msprime against simulators such as ms, mama, and acm.

User interface

The majority of simulation packages are controlled either through a command-line interface (e.g., Hudson 2002, Kern and Schrider 2016), a text-based input file format (e.g., Guillaume and Rougemont 2006; Excoffier and Foll 2011; Shlyakhter et al. 2014), or a mixture of both. Command-line interfaces make it easy to run simple simulations, but as model complexity and the number of parameters increase, they become difficult to understand and error-prone (Ragsdale et al. 2020; Gower et al. 2022). Specifying parameters through a text file alleviates this problem to a degree, but lacks flexibility, for example, when running simulations with parameters drawn from a distribution. In practice, for any reproducible simulation project, users will write a script to generate the required command lines or input parameter files, invoke the simulation engine, and process the results in some way. This process is cumbersome and labor intensive, and a number of packages have been developed to allow simulations to be run directly in a high-level scripting language (Staab and Metzler 2016; Parobek et al. 2017; Gladstein et al. 2018).

The more recent trend has been to move away from this file and command-line driven approach and to instead provide direct interfaces to the simulation engines via an Application Programming Interface (API) (e.g., Thornton 2014; Kelleher et al. 2016; Becheler et al. 2019; Haller and Messer 2019). The primary interface for msprime is through a thoroughly documented Python API, which has encouraged the development of an ecosystem of downstream tools (Terhorst et al. 2017; Chan et al. 2018; Spence and Song 2019; Adrion et al. 2020a, 2020b; Kammt et al. 2020; McKenzie and Eaton 2020; Montinaro et al. 2020; Rivera-Colon et al. 2021; Terasaki Hart et al. 2021). As well as providing a stable and efficient platform for building downstream applications, msprime’s Python API makes it much easier to build reproducible simulation pipelines, as the entire workflow can be encapsulated in a single script, and package and version dependencies explicitly stated using the pip or conda package managers. For example, the errors made in the influential simulation analysis of Martin et al. (2017) were only detected because the
pipeline could be easily run and reanalyzed (Martin et al. 2020; Ragsdale et al. 2020).

A major change for the msprime 1.0 release is the introduction of a new set of APIs, designed in part to avoid sources of error (see Demography) but also to provide more appropriate defaults while keeping compatibility with existing code. In the new APIs, ancestry and mutation simulation are fully separated (see Figure 1), with the sim_ancestry and sim_mutations functions replacing the legacy simulate function. Among other changes, the new APIs default to discrete genome coordinates and finite sites mutations, making the default settings more realistic and resolving a major source of confusion and error. The previous APIs are fully supported and tested, and will be maintained for the foreseeable future. The map program (a command-line interface to the library) has been extended to include new commands for simulating ancestry and mutations separately. A particularly useful feature is the ability to specify demographic models in Demes format (Gower et al. 2022) from the command line, making simulation of complex demographies straightforward. We also provide an ms-compatible command-line interface to support existing workflows.

### Tree sequences

One of the key reasons for msprime’s substantial performance advantage over other simulators (Kelleher et al. 2016) is its use of the “succinct tree sequence” data structure to represent simulation results. The succinct tree sequence (usually abbreviated to “tree sequence”) was introduced by Kelleher et al. (2016) to concisely encode genetic ancestry and sequence variation and was originally implemented as part of msprime. We subsequently extracted the core tree sequence functionality from msprime to create the tskit library, which provides a large suite of tools for processing genetic ancestry and variation data via APIs in the Python and C languages (Tskit developers 2022). The availability of tskit as a liberally licensed (MIT) open source toolkit has enabled several other projects (e.g., Haller and Messer 2019; Kelleher et al. 2019; Terasaki Hart et al. 2021; Wohns et al. 2021) to take advantage of the same efficient data structures used in msprime, and we hope that many more will follow. While a full discussion of tree sequences and the capabilities of tskit is beyond the scope of this article, we summarize some aspects that are important for simulation.

Let us define a genome as the complete set of genetic material that a child inherits from one parent. Thus, a diploid individual has two (monoploid) genomes, one inherited from each parent. Since each diploid individual lies at the end of two distinct lineages of descent, they will be represented by two places (nodes) in any genealogical tree. In the tree sequence encoding a node therefore corresponds to a single genome, which is associated with its creation time (and other optional information), and recorded in a simple tabular format (Figure 2). Genetic inheritance between genomes (nodes) is defined by edges. An edge consists of a parent node, a child node, and the left and right coordinates of the contiguous chromosomal segment over which the child genome inherited genetic material from the parent genome. Parent and child nodes may correspond to ancestor and descendant genomes separated by many generations. Critically, edges can span multiple trees along the genome (usually referred to as “marginal” trees), and identical node IDs across different trees corresponds to the same ancestral genome. For example, in Figure 2, the branch from node 0 to 4 is present in both marginal trees, and represented by a single edge (the first row in the edge table). This simple device, of explicitly associating tree nodes with specific ancestral genomes and recording the contiguous segments over which parent–child relationships exist, generalizes the original “coalescence records” concept (Kelleher et al. 2016), and is the key to the efficiency of tree sequences (Kelleher et al. 2018, 2019, Ralph et al. 2020). Note that this formulation is fully compatible with the concept of an Ancestral Recombination Graph (ARG) and any ARG topology can be fully and efficiently encoded in the node and edge tables illustrated in Figure 2; see the Ancestral Recombination Graphs section below for more details.

The final output of most population genetic simulations is some representation of sequence variation among the specified samples. For coalescent simulations, we usually have three steps: (1) simulate the genetic ancestry, and optionally output the resulting marginal trees; (2) simulate sequence evolution conditioned on this ancestry by generating mutations (see the Simulating mutations section); and (3) output the resulting nucleotide sequences by percolating the effects of the mutations through the trees. Information about the mutations themselves—e.g., where they have occurred on the trees—is usually not retained or made available for subsequent analysis. In msprime, however, we skip step (3), instead using tskit’s combined data model of ancestry and mutations to represent the simulated sequences. As illustrated in Figure 2, mutations are a fully integrated part of tskit’s tree sequence data model, and genetic variation is encoded by recording sites at which mutations have occurred, and where each mutation at those sites has occurred on the marginal tree. Crucially, the genome sequences themselves are never stored, or indeed directly represented in memory (although tskit can output the variant matrix in various formats, if required). It may at first seem inconvenient to

### Table 1 Major features of msprime 1.0 added since version 0.3.0 (Kelleher et al. 2016)

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Ancestry</td>
<td>SMC, SMC2, Beta- and Dirac-coalescent, discrete time Wright–Fisher, and selective sweep models. Instantaneous bottlenecks. Discrete or continuous genomic coordinates, arbitrary ploidy, gene conversion. Output full ARG with recombination nodes, ARG likelihood calculations. Record full migration history and census events. Improved performance for large numbers of populations. Integration with forward simulators such as SLiM and &quot;fwdpy11 (&quot;recapitation&quot;).</td>
</tr>
<tr>
<td>Demography</td>
<td>Improved interface with integrated metadata and referencing populations by name. Import from Newick species tree, &quot;BEAST (Heled and Drummond 2010), and Demes (Gower et al. 2022). Numerical methods to compute coalescence rates.</td>
</tr>
<tr>
<td>Mutations</td>
<td>JC69, HKY, F84, GTR, BLOSUM62, PAM, infinite alleles, SLiM and general matrix mutation models. Varying rates along the genome, recurrent/back mutations, discrete or continuous genomic coordinates, overlaying multiple layers of mutations, exact times associated with mutations.</td>
</tr>
</tbody>
</table>
The vast genomic datasets produced during the SARS-CoV-2 pandemic have highlighted the advantages of storing genetic variation data using the underlying trees. Turakhia et al. (2021) propose the Mutation Annotated Tree (MAT) format (consisting of a Newick tree and associated mutations in a binary format) and the muttsite program as an efficient way to store and process large viral datasets (McBroome et al. 2021), achieving excellent compression and processing performance. Similarly, phastsim (De Maio et al. 2021) was developed to simulate sequence evolution on such large SARS-CoV-2 phylogenies, and also outputs a Newick tree annotated with mutations (not in MAT format) to avoid the bottleneck of generating and storing the simulated sequences. While these methods illustrate the advantages of the general approach of storing ancestry and mutations rather than sequences, they do not generalize beyond their immediate settings, and no software library support is available.

The software ecosystem built around tskit is stable, mature, and rapidly growing. Simulators such as fmdpyll (Thornton 2014), SLiM (Haller and Messer 2019), stdpopsim (Adrion et al. 2020a), Geonomics (Terasaki Hart et al. 2021), and GSpace (Virgoulay et al. 2021), and inference methods such as tsinfer (Kelleher et al. 2019), tdedate (Wohls et al. 2021), and Relate (Speidel et al. 2019) use either the Python or C APIs to support outputting results in tree sequence format. Tree sequences are stored in an efficient binary file format, and are fully portable across operating systems and processor architectures. The tskit library ensures interoperability between programs by having strict definitions of how the information in each of the tables is interpreted, and stringent checks for the internal consistency of the data model.

Figure 1 Visualization of the separation between ancestry and mutation simulation. (A) The result of an invocation of sim_ancestry is two trees along a 1 kb chunk of genome relating three diploid samples. Each diploid individual consists of two genomes (or nodes), indicated by color. (B) This ancestry is provided as the input to sim_mutations, which adds mutations. Graphics produced using tskit’s draw_svg method.

have only this indirect representation of the genome sequences, but it is extremely powerful. First, the storage space required for simulations is dramatically reduced. For a simulation of n samples with m variant sites, we would require O(mn) space to store the sequence data as a variant matrix. However, if this simulation was of a recombining genome with t trees, then the tskit tree sequence encoding requires O(n + t + m) space, assuming we have O(t) mutations at each site (Kelleher et al. 2016). For large sample sizes, this difference is profound, making it conceivable, for example, to store the genetic ancestry and variation data for the entire human population on a laptop (Kelleher et al. 2019). As well as the huge difference in storage efficiency, it is often far more efficient to compute statistics of the sequence data from the trees and mutations than it is to work with the sequences themselves. For example, computing Tajima’s D from simulated data stored in the tskit format is several orders of magnitude faster than efficient variant matrix libraries for large sample sizes (Ralph et al. 2020).

The standard way of representing simulation data is to render the results in a text format, which must subsequently be parsed and processed as part of some analysis pipeline. For example, ms outputs a set of sequences and can also optionally output the marginal trees along the genome in Newick format, and variants of this approach are used by many simulators. Text files have many advantages, but are slow to process at scale. The ability to efficiently process simulation results is particularly important in simulation-based inference methods such as Approximate Bayesian Computation (ABC) (Beaumont et al. 2002; Csilléry et al. 2010; Wegmann et al. 2010) and machine learning-based approaches (Sheehan and Song 2016; Chan et al. 2018; Schröder and Kern 2018; Flagel et al. 2019; Sanchez et al. 2021). Clearly, simulation efficiency is crucial since the size and number of simulations that can be performed determines the depth to which one can sample from the model and parameter space. Equally important, however, is the efficiency with which the simulation results can be transformed into the specific input required by the inference method. In the case of ABC, this is usually a set of summary statistics of the sequence data, and methods avoid the bottleneck of parsing text-based file formats to compute these statistics by either developing their own simulators (e.g., Cornuet et al. 2008; Lopes et al. 2009) or creating forked versions (i.e., modified copies) of existing simulators (e.g., Thornton and Andolfatto 2006; Hickerson et al. 2007; Pavlidis et al. 2010; Huang et al. 2011; Quinto-Cortés et al. 2018), tightly integrated with the inference method. Modern approaches to ABC such as ABC-RF (Pudlo et al. 2016; Raynal et al. 2019) and ABC-NN (Blum and François 2010; Csilléry et al. 2012) use large numbers of weakly informative statistics, making the need to efficiently compute statistics from simulation results all the more acute. By using the stable APIs and efficient data interchange mechanisms provided by tskit, the results of an msprime simulation can be immediately processed, without format conversion overhead. The tskit library has a rich suite of population genetic statistics and other utilities, and is in many cases orders of magnitude faster than matrix-based methods for large sample sizes (Ralph et al. 2020). Thus,
the combination of msprime and tskit substantially increases the overall efficiency of many simulation analysis pipelines. Classical text-based output formats like ms are inefficient to process, but also lack a great deal of important information about the simulated process. The tree-by-tree topology information output by simulators in Newick format lacks any concept of node identity, and means that we cannot reliably infer information about ancestors from the output. Because Newick stores branch lengths rather than node times, numerical precision issues also arise for large trees (McGill et al. 2013). Numerous forks of simulators have been created to access information not provided in the output. For example, ms has been forked to output information about migrating segments (Rosenzweig et al. 2016), ancestral lineages (Chen and Chen 2013), and ms’s fork msHOT (Hellenthal and Stephens 2007) has in turn been forked to output information on local ancestry (Racimo et al. 2017). All of this information is either directly available by default in msprime, or can be optionally stored via options such as record_migrations or record_full_arg (see Ancestral Recombination Graphs) and can be efficiently and conveniently processed via tskit APIs.

Simulating mutations

Because coalescent simulations are usually concerned with neutral evolution (see the Selective sweeps section, however), the problem of generating synthetic genetic variation can be decomposed into two independent steps: first, simulating genetic ancestry (the trees), then subsequently simulating variation by superimposing mutation processes on those trees (see Figure 1). A number of programs exist to place mutations on trees: for instance, the classical Seq-Gen program (Rambaut and Grassly 1997) supports a range of different models of sequence evolution, and various extensions to the basic models have been proposed (e.g., Cartwright 2005; Fletcher and Yang 2009). Partly for efficiency and partly in the interest of simplicity for users (i.e., to avoid intermediate text format conversions), population genetic simulators have tended to include their own implementations of mutation simulation, with most supporting the infinite sites model (e.g., Hudson 2002) but with several supporting a wide range of different models of sequence evolution (e.g., Mailund et al. 2005; Excoffier and Foll 2011; Virgoulay et al. 2021). Thus, despite the logical separation between the tasks of simulating ancestry and neutral sequence evolution, the two have been conflated in practice.

Part of the reason for this poor record of software reuse and modularity is the lack of standardized file formats, and in particular, the absence of common library infrastructure to abstract the details of interchanging simulation data. Although msprime also supports simulating both ancestry and mutations, the two aspects are functionally independent within the software; both ancestry and mutation simulators are present in msprime for reasons of convenience and history, and could be split into separate packages. The efficient C and Python interfaces for tskit make it straightforward to add further information to an existing file, and because of its efficient data interchange mechanisms, there is no performance penalty for operations being performed in a different software package. Thanks to this interoperability, msprime’s mutation generator can work with any tskit tree sequence, be it simulated using SLiM (Haller and Messer 2019) or fwdpy11 (Thornton 2014), or estimated from real data (Kelleher et al. 2019; Speidel et al. 2019; Wohns et al. 2021). It is a modular component intended to fit into a larger software ecosystem, and is in no way dependent on msprime’s ancestry simulator.

We have greatly extended the sophistication of msprime’s mutation generation engine for version 1.0, achieving near feature-parity with Seq-Gen. We support a large number of mutation models, including the JC69 (Jukes and Cantor 1969), F84 (Felsenstein and Churchill 1996), and GTR (Tavaré 1986) nucleotide models and the BLOSUM62 (Henikoff and Henikoff 1992) and PAM (Dayhoff et al. 1978) amino acid models. Other models, such as the Kimura two and three parameter models (Kimura 1980, 1981), can be defined easily and efficiently in user code by specifying a transition matrix between any number of alleles.
Mutation rates can vary along the genome, and multiple mutation models can be imposed on a tree sequence by overlaying mutations in multiple passes. We have extensively validated the results of mutation simulations against both theoretical expectations and output from Seq-Gen (Rambaut and Grassly 1997) and Pylvolve (Spielman and Wilke 2015).

Simulating mutations in msprime is efficient. Figure 3 shows the time required to generate mutations (using the default JCG9 model) on simulated tree sequences for a variety of mutation rates as we vary the number of samples (Figure 3A) and the sequence length (Figure 3B). For example, the longest running simulation in Figure 3B required less than 2 s to generate an average of 1.5 million mutations over 137,081 trees in a tree sequence with 508,125 lengths (Figure 3B). For example, the longest running simulation in Figure 3B required less than 2 s to generate an average of 1.5 million mutations over 137,081 trees in a tree sequence with 508,125 lengths (Figure 3B). For example, the longest running simulation in Figure 3B required less than 2 s to generate an average of 1.5 million mutations over 137,081 trees in a tree sequence with 508,125 lengths (Figure 3B). For example, the longest running simulation in Figure 3B required less than 2 s to generate an average of 1.5 million mutations over 137,081 trees in a tree sequence with 508,125 lengths (Figure 3B). For example, the longest running simulation in Figure 3B required less than 2 s to generate an average of 1.5 million mutations over 137,081 trees in a tree sequence with 508,125 lengths (Figure 3B). For example, the longest running simulation in Figure 3B required less than 2 s to generate an average of 1.5 million mutations over 137,081 trees in a tree sequence with 508,125 lengths (Figure 3B). For example, the longest running simulation in Figure 3B required less than 2 s to generate an average of 1.5 million mutations over 137,081 trees in a tree sequence with 508,125 lengths (Figure 3B). For example, the longest running simulation in Figure 3B required less than 2 s to generate an average of 1.5 million mutations over 137,081 trees in a tree sequence with 508,125 lengths (Figure 3B). For example, the longest running simulation in Figure 3B required less than 2 s to generate an average of 1.5 million mutations over 137,081 trees in a tree sequence with 508,125 lengths (Figure 3B). For example, the longest running simulation in Figure 3B required less than 2 s to generate an average of 1.5 million mutations over 137,081 trees in a tree sequence with 508,125 lengths (Figure 3B). For example, the longest running simulation in Figure 3B required less than 2 s to generate an average of 1.5 million mutations over 137,081 trees in a tree sequence with 508,125 lengths (Figure 3B). For example, the longest running simulation in Figure 3B required less than 2 s to generate an average of 1.5 million mutations over 137,081 trees in a tree sequence with 508,125 lengths (Figure 3B). For example, the longest running simulation in Figure 3B required less than 2 s to generate an average of 1.5 million mutations over 137,081 trees in a tree sequence with 508,125 lengths (Figure 3B). For example, the longest running simulation in Figure 3B required less than 2 s to generate an average of 1.5 million mutations over 137,081 trees in a tree sequence with 508,125 lengths (Figure 3B). For example, the longest running simulation in Figure 3B required less than 2 s to generate an average of 1.5 million mutations over 137,081 trees in a tree sequence with 508,125 lengths (Figure 3B). For example, the longest running simulation in Figure 3B required less than 2 s to generate an average of 1.5 million mutations over 137,081 trees in a tree sequence with 508,125 lengths (Figure 3B).

In human-like parameter regimes and for large sample sizes, msprime’s implementation of the exact coalescent with recombination comprehensively outperforms all other simulators, including those based on SMC approximations (Kelleher et al. 2016). However, it is important to note that although the implementation of Hudson’s algorithm is very efficient, it is still quadratic in the population scaled recombination rate \( \rho = 4N_{\ell} \lambda \), where \( \ell \) is the length of the genome in units of recombination distance. This is because Hudson’s algorithm tracks recombinations not only in segments ancestral to the sample, but also between ancestral segments. As mentioned above, common ancestor events in which the ancestral material of two lineages is merged only result in coalescences in the marginal trees if their ancestral segments overlap. If there is no overlap, the merged segments represent an ancestral chromosome that is a genetic ancestor of the two lineages, but not the most recent common genetic ancestor at any location along the genome. When this happens, the merged lineage carries “trapped” genetic material that is not ancestral to any samples, but where recombinations can still occur (Wiuf and Hein 1999b). For large \( \rho \), recombination events in trapped ancestral material will dominate, and so we can use this as a proxy for the overall number of events in Hudson’s algorithm. Hein et al. (2004, Equation 5.10) gave

\[
\rho(\rho + 1) \left( \sum_{i=1}^{n-1} \frac{1}{i} \right)^2
\]  

as an upper bound on the number of recombination events within trapped ancestral material for \( n \) samples. As discussed in the Time complexity of Hudson’s algorithm Appendix, the quadratic dependence of simulation running time on \( \rho \) implied by Equation (1) is well supported by observations, and provides a
Figure 4 Comparison of simulation performance using msprime (sim_ancestry), SimBac, and fastSimBac for varying (haploid) sample sizes, and the current estimates for E. coli parameters (Lapierre et al. 2016): a 4.6 Mb genome, \( N_e = 1.8 \times 10^8 \), gene conversion rate of \( 8.9 \times 10^{-11} \) per base and mean tract length of 542. We report (A) the total CPU time and (B) maximum memory usage averaged over five replicates (Intel Xeon E5-2680 CPU). We did not run SimBac beyond first two data points because of the very long running times.

useful means of predicting how long a particular simulation might require.

Gene conversion

Gene conversion is a form of recombination that results in the transfer of a short segment of genetic material, for example between homologous chromosomes (Chen et al. 2007). Since gene conversion impacts much shorter segments than crossover recombination (typically below 1 kb), it affects patterns of linkage disequilibrium differently (Korunes and Noor 2017). Wiuf and Hein (2000) modeled gene conversion in the coalescent via a rate at which gene conversion events are initiated along the genome and a geometrically distributed tract length. In terms of the ancestral process, gene conversion differs from crossover recombination (as described in the previous section) in that it extracts a short tract of ancestry into an independent lineage, rather than splitting ancestry to the left and right of a given breakpoint. We have implemented this model of gene conversion in msprime 1.0, and validated the output against ms and analytical results (Wiuf and Hein 2000).

Gene conversion is particularly useful to model homologous recombination in bacterial evolution, and so we compare the performance of msprime with gene conversion to two specialized bacterial simulators, SimBac (Brown et al. 2016) and fastSimBac (De Maio and Wilson 2017). Figure 4A shows that msprime is far more efficient than both SimBac and the SMC-based approximation fastSimBac. Figure 4B shows that msprime requires somewhat more memory than fastSimBac (as expected since fastSimBac uses a left-to-right SMC approximation), but is still reasonably modest at around 1 GB for a simulation of 500 whole Escherichia coli genomes. However, msprime is currently lacking many of the specialized features required to model bacteria, and so an important avenue for future work is to add features such as circular genomes and bacterial gene transfer (Baumdicker and Pfaffelhuber 2014).

Demography

One of the key applications of population genetic simulations is to generate data for complex demographies. Beyond idealized cases such as stepping-stone or island models, or specialized cases such as isolation-with-migration models, analytical results are rarely possible. Simulation is therefore integral to the development and evaluation of methods for demographic inference. The demography model in msprime is directly derived from the approach used in ms, and supports an arbitrary number of randomly mating populations exchanging migrants at specified rates. A range of demographic events are supported, which allow for varying population sizes and growth rates, changing migration rates over time, as well as population splits, admixtures, and pulse migrations.

A major change for msprime 1.0 is the introduction of the new Demography API, designed to address a design flaw in the msprime 0.1 interface which led to avoidable errors in downstream simulations (Ragsdale et al. 2020). The new API is more user-friendly, providing the ability, for example, to refer to populations by name rather than their integer identifiers. We also provide numerical methods to compute the coalescence rates for two or more lineages which can be inverted to obtain the “inverse instantaneous coalescence rate” of Chikhi et al. (2018). Many popular approaches in population genetics use the distribution of coalescence rates between pairs of lineages to infer effective population sizes over time (Li and Durbin 2011; Sheehan et al. 2013; Schiffs and Durbin 2014) or split times and subsequent migration rates between populations (Wang et al. 2020). These numerical methods provide a valuable ground-truth when evaluating such inference methods, as illustrated by Adrion et al. (2020a).

Instantaneous bottlenecks

A common approach to modeling the effect of demographic history on genealogies is to assume that effective population size \( (N_e) \) changes in discrete steps which define a series of epochs (Griffiths et al. 1994; Marth et al. 2004; Keightley and Eyre-Walker 2007; Li and Durbin 2011). In this setting of piecewise constant \( N_e \) capturing a population bottleneck requires three epochs: \( N_e \) is reduced by some fraction \( b \) at the start of the bottleneck, \( T_{\text{start}} \), and recovers to its initial value at time \( T_{\text{end}} \) (Marth et al. 2004). If bottlenecks are short both on the timescale of coalescence and mutations, there may be little information about the duration of a bottleneck (\( T_{\text{end}} - T_{\text{start}} \)) in sequence data. Thus a simpler, alternative model is to assume that bottlenecks are instantaneous (\( T_{\text{end}} = T_{\text{start}} = 0 \)) and generate a sudden burst of coalescence events (a multiple merger event) in the genealogy. The strength of the bottleneck \( B \) can be thought of as an (imaginary) time period during which coalescence events are collapsed, i.e., there is no growth in genealogical branches during \( B \) and the probability that a single pair of lineages entering the bottleneck coalesces during the bottleneck is \( 1 - e^{-B} \). Although this simple two-parameter model of bottlenecks is attractive and both analytic results and empirical inference methods (Griffiths et al. 1994; Galtier et al. 2000; Birchner et al. 2009; Bunnefeld et al. 2015) have been developed under this model, there has been no software available to simulate data under instantaneous bottleneck histories.

We have implemented instantaneous bottlenecks in msprime 1.0 using a variant of Hudson’s linear time single-locus coalescent algorithm (Hudson 1990), and validated the results by comparing against analytical expectations (Bunnefeld et al. 2015).

Multiple merger coalescents

Kingman’s coalescent assumes that only two ancestral lineages can merge at each merger event. Although this is generally a reasonable approximation, there are certain situations in which the underlying mathematical assumptions are violated; for example, in certain highly fecund organisms (Beckenbach 1994; Hedgcock 1994; Åmason 2004; Hedgcock and Pudovkin 2011; Irwin et al. 2016; Vendrami et al. 2021), where individuals have the ability to produce numbers of offspring on the order of the population size and therefore a few individuals may produce the bulk of the
offspring in any given generation (Hedgecock 1994). These population dynamics violate basic assumptions of the Kingman coalescent, and are better modeled by “multiple-merger” coalescents (Donnelly and Kurtz 1999; Pitman 1999; Sagitov 2001, Schweinsberg 2000; Möhle and Sagitov 2001), in which more than two lineages can merge in a given event. Multiple-merger coalescent processes have also been shown to be relevant for modeling the effects of selection on genealogies (Gillespie 2000; Durrett and Schweinsberg 2004; Desai et al. 2013; Neher and Hallatschek 2013; Schweinsberg 2017).

Although multiple merger coalescents have been of significant theoretical interest for around two decades, there has been little practical software available to simulate these models. Kelleher et al. (2013, 2014) developed packages to simulate a related spatial continuum model (Barton et al. 2010), Zhu et al. (2015) simulate genealogies within a species tree based on a multiple-merger model, and Becherer and Knowles (2020) provide a general method for simulating multiple merger processes as part of the Quetzal framework (Becherer et al. 2019). The Beta-Xi-Sim simulator (Koskela and Berenguer 2019) also includes a number of extensions to the Beta-coalescent. None of these methods work with large genomes, and very little work has been performed on simulating multiple merger processes with recombination.

We have added two multiple merger coalescent models in msprime 1.0, the Beta-coalescent (Schweinsberg 2003), and “Dirac”-coalescent (Birkner et al. 2013a), allowing us to efficiently simulate such models with recombination for the first time. These simulation models have been extensively validated against analytical results from the site frequency spectrum (Birkner et al. 2013b; Blath et al. 2016; Hobolth et al. 2019) as well as more general properties of coalescent processes. See the Multiple merger coalescent model in Appendix for more details and model derivations.

**Ancestral Recombination Graphs**

The ARG was introduced by Griffiths (Griffiths 1991; Griffiths and Marjoram 1997) to represent the stochastic process of the coalescent with recombination as a graph. This formulation is complementary to Hudson’s earlier work (Hudson 1983a), and substantially increased our theoretical understanding of recombination. In Griffiths’ ARG formulation, a realization of the coalescent with recombination is a graph in which vertices represent common ancestor or recombination events, and edges represent lineages. There is the “big” ARG, in which we track lineages arising out of recombinations regardless of whether they carry ancestral material (Ethier and Griffiths 1990), and the “little” ARG in which we only track genetic ancestors. Over time, usage of the term has shifted away from its original definition as a stochastic process, to being interpreted as a representation of a particular genetic ancestry as a graph, without necessarily following the specific details of the Griffiths formulation (e.g., Minichiello and Durbin 2006; Mathieson and Scally 2020). Under the latter interpretation, the tree sequence encoding of genetic ancestry (described above) clearly is an ARG: the nodes and edges define a graph in which edges are annotated with the set of disjoint genomic intervals through which ancestry flows.

For our purposes, an ARG is a realization of the coalescent with recombination, in the Griffiths (little ARG) sense. As described in detail by Kelleher et al. (2016), Hudson’s algorithm works by dynamically traversing a little ARG. The graph is not explicitly represented in memory, but is partially present through the extant lineages and the ancestral material they carry over time. We do not output the graph directly, but rather store the information required to recover the genealogical history as nodes and edges in a tree sequence. This is far more efficient than outputting the simulated ARG in its entirety. For a given scaled recombination rate $\rho$ (setting aside the dependency on the sample size $n$), we know from Equation (1) that the number of nodes in an ARG is $O(\rho^2)$, whereas the size of the tree sequence encoding is $O(\rho)$ (Kelleher et al. 2016). This difference between a quadratic and a linear dependency on $\rho$ is profound, and shows why large simulations cannot output an ARG in practice.

Although by default msprime outputs tree sequences that contain full information about the genealogical trees, their correlation structure along the chromosome, and the ancestral genomes on which coalescences occurred, some information is lost in this mapping down from ARG space to the minimal tree sequence form. In particular, we lose information about ancestral genomes that were common ancestors but in which no coalescences occurred, and also information about the precise time and chromosomal location of recombination events. In most cases, such information is of little relevance as it is in principle unknowable, but there are occasions such as visualization or computing likelihoods in which it is useful. We therefore provide the record_full_arg option in msprime to store a representation of the complete ARG traversed during simulation. This is done by storing extra nodes (marked with specific flags, so they can be easily identified later) and edges in the tree sequence (Figure 5). One situation in which a record of the full ARG is necessary is when we wish to compute likelihoods during inference. The likelihood is a central quantity in evaluating the plausibility of a putative
Selective sweeps

Another elaboration of the standard neutral coalescent with recombination is the addition of selective sweeps (Kaplan et al. 1989; Braverman et al. 1995; Kim and Stephan 2002). Sweeps are modeled by creating a structured population during the sojourn of the beneficial mutation through the population (i.e., the sweep phase) in which lineages may transit between favored and unfavorable backgrounds through recombination. This approach allows for many selective sweep scenarios to be simulated efficiently, including recurrent, partial, and soft selective sweeps. However, this efficiency comes at the cost of flexibility in comparison to forwards in time simulation. Several specialized simulators have been developed to simulate sweeps in the coalescent, including Selsim (Spencer and Coop 2004), mbs (Teshima and Innan 2009), mama (Ewing and Hermisson 2010), coa12 (Shlyakhter et al. 2014), and discoal (Kern and Schrider 2016).

Selective sweeps are implemented in the coalescent as a two-step process: first generating an allele frequency trajectory, and then simulating a structured coalescent process conditioned on that trajectory. Following discoal, we generate sweep trajectories in msprime using a jump process approximation to the conditional diffusion of an allele bound for fixation (Coop and Griffiths 2004), as detailed in the Selective sweeps model in Appendix. Given a randomly generated allele frequency trajectory, the simulation of a sweep works by assigning lineages to two different structured coalescent “labels,” based on whether they carry the beneficial allele. The allele frequency trajectory determines the relative sizes of the “populations” in these labels over time, and therefore the rates at which various events occur. Common ancestor events can then only merge lineages from within a label, but lineages can transfer from one label to the other (i.e., from the advantageous to disadvantageous backgrounds, and vice versa) as a result of recombination events. Once we have reached the end of the simulated trajectory, the sweep is complete, and we remove the structured coalescent labels. Simulation may then resume under any other ancestry model.

Discrete time Wright–Fisher

The coalescent is an idealized model and makes many simplifying assumptions, but it is often surprisingly robust to violations of these assumptions (Wakeley et al. 2012). One situation in which the model does break down is the combination of large sample size and long recombining genomes, where the large number of recombination events in the recent past results in more than the biologically possible 2N ancestors in t diploid generations (Nelson et al. 2020). This pathological behavior results in identity-by-descent, long-range linkage disequilibrium, and ancestry patterns deviating from Wright–Fisher expectations, and the bias grows with larger sample sizes (Wakeley et al. 2012, Bhaskar et al. 2014; Nelson et al. 2020). Precisely this problem occurs when simulating modern human datasets, and we have implemented a Discrete Time Wright–Fisher (DTWF) model in msprime to address the issue. The DTWF simulates backwards in time generation-by-generation so that each gamete has a unique diploid parent, and multiple recombinations within a generation results in crossover events between the same two parental haploid copies. The method is described in detail by Nelson et al. (2020).

Figure 7 shows that msprime simulates the DTWF more quickly and requires substantially less memory than ARgon (Palamara 2016), a specialized DTWF simulator. However, the generation-by-generation approach of the DTWF is less efficient than the coalescent with recombination when the number of lineages is significantly less than the population size (the regime where the coalescent is an accurate approximation), which usually happens in the quite recent past (Bhaskar et al. 2014). We therefore support changing the simulation model during a simulation so that we can run hybrid simulations, as proposed by Bhaskar et al. (2014). Any number of different simulation models...
can be combined, allowing for the flexible choice of simulation scenarios. As the DTWF improves accuracy of genealogical patterns in the recent past, we can simulate the recent history using this model and then switch to the standard coalescent to more efficiently simulate the more ancient history.

Integration with forward simulators
A unique feature of `msprime` is its ability to simulate genetic ancestries by extending an existing partial genetic ancestry. Given a tree sequence that is complete up until time t ago as input (where marginal trees may or may not have fully coalesced), `msprime` can efficiently obtain the segments of ancestral material present at this time, and then run the simulation backwards in time from there. This allows a simulated ancestry to be produced by any number of different processes across disjoint time slices. In practice, this feature is used to "complete" forwards-time ancestry simulations (Kelleher et al. 2018) that may have not fully coalesced. This process ("recapitation") can be orders of magnitude faster than the standard approach of neutral burn-in; see Haller et al. (2018) for more details and examples. This interoperability between simulators, where a partial ancestry simulation produced by SLiM (Haller and Messer 2019) or fwdpy11 (Thornton 2014) can be picked up and completed by another simulator, with complete information retained—at scale—is unprecedented. There may be an opportunity for other forward genetic simulators (e.g., Gaynor et al. 2021) to leverage the tree sequence data format and associated tools.

Development model
`Msprime` has a large number of features, encompassing the functionality of several more specialized simulators while maintaining excellent performance. It is developed by a geographically distributed team of volunteers under an open source community development model, with a strong emphasis on code quality, correctness, good documentation, and inclusive development. As in any large code base, unit tests play a key role in ensuring that new additions behave as expected and `msprime` has an extensive suite. These tests are run automatically on different operating systems on each pull request (where a contributor proposes a code change), using standard Continuous Integration (CI) methodology. Other CI services check for common errors, code formatting issues, and produce reports on the level of test coverage for the proposed change.

Unit tests are vital for ensuring software quality and correctness, but they are usually of little value in assessing the statistical properties of simulations. To validate the correctness of simulation output, we maintain a suite of statistical tests (as of 1.0.0, 217 validation tests). These consist of running many replicate simulations to check the properties of the output against other simulators, and where possible against analytical results. For example, simulations of complex demography are validated against ms, selective sweeps against discoal, and Wright–Fisher simulations against forwards in time simulations in SLiM. This suite of tests is run before every release, to ensure that statistical errors have not been introduced.

More visibly to the end user, we also have a high standard for documentation, with precise, comprehensive, and cross-linked documentation that is automatically built from the code base and served through the website https://tskit.dev. With the goal of lowering the entry barrier to new users, we have invested significant effort in writing examples and introductions, and making common tasks discoverable. We also view contributions to documentation as equally important to the project as writing code or designing methods: what use would it be to write reliable, stable software if no-one used it?

An important goal of `msprime`'s development model is to maximize accessibility for prospective users and contributors, and to encourage diversity in our community. Gender and racial inequality caused by discrimination and marginalization is a major problem across the sciences (Wellenreuther and Otto 2016; Shannon et al. 2019) and in open source software development (Trinkenreich et al. 2021). Within our field, the contribution of women to early computational methods in population genetics was marginalized (Dung et al. 2019), and women continue to be under-represented in computational biology (Bonham and Stefan 2017). The authorship of our paper reflects these trends, with a skew toward men and affiliations in the United States and Europe. We know the importance of creating and strengthening networks to develop and maintain a diverse community of contributors, and we are committed to fostering a supportive and collaborative environment that helps to address these inequalities in our field.

Discussion
The 1.0 release of `msprime` marks a major increase in the breadth of available features and the potential biological realism of simulations. These abilities will allow researchers to perform more robust power analyses, more reliably test new methods, carry out more reliable inferences, and more thoroughly explore the properties of theoretical models. Despite this complexity and generalization, `msprime`'s performance is state-of-the-art and all features are extensively tested and statistically validated. These advances have only been possible thanks to a distributed, collaborative model of software development, and the work of many people.

Even though simulation has long been a vital tool in population genetics, such collaborative software development has historically been uncommon. A huge proliferation of tools have been published (the references here are not exhaustive) and only a small minority of these are actively developed and maintained today. The ecosystem is highly fragmented, with numerous different ways of specifying parameters and representing results, and there are significant software quality issues at all stages. This is unsurprising, since the majority of simulation software development is performed by students, often without formal training in software development. The result resembles Haldane’s sieve for new mutations: many new pieces of software stay permanently on a dusty shelf of supplementary materials, while some of those that prove particularly useful when new (like dominant alleles) are quickly adopted. Although this has produced many good tools and enabled decades of research, it also represents a missed opportunity to invest as a community in shared infrastructure and mentorship in good software development practice.

Scientific software is vital apparatus, and must be engineered to a high quality if we are to trust its results. There is a growing realization across the sciences (e.g., Siepel 2019; Harris et al. 2020; Gardner et al. 2021) that investing in shared community infrastructure produces better results than a proliferation of individually maintained tools, allowing scientists to focus on their specific questions rather than software engineering. `Msprime` 1.0 is the result of such a community process, with features added by motivated users, taking advantage of the established development practices and infrastructure. Software development in a welcoming community, with mentorship by experienced developers, is a useful experience for many users. The skills that
contributors learn can lead to greatly increased productivity in subsequent work (e.g., through more reliable code and better debugging skills). We hope that users who find that features they require are missing will continue to contribute to msprime, leading to a community project that is both high quality and sustainable in the long term.

The succinct tree sequence data structure developed for msprime provides a view of not only genetic variation, but also the genetic ancestry that produced that variation. Recent breakthroughs in methods to infer genetic ancestry in recombining organisms (Rasmussen et al. 2014; Kelleher et al. 2019; Speidel et al. 2019, 2021; Schaefer et al. 2021; Wohns et al. 2021) have made it possible to estimate such ancestry from real data at scale for the first time (Harris 2019; Tang 2019). Given such inferred ancestry, many exciting applications become possible. For example, Osmond and Coop (2021) developed a method to estimate the location of genetic ancestors based on inferred trees, and other uses are sure to follow. Since the inferred genetic ancestry becomes the input for other downstream inferences, it is vitally important that these primary inferences are thoroughly validated. For example, Arenas M. 2012. Simulation of molecular data under diverse evolutionary scenarios. PLoS Comput Biol. 8:e1002495.


Data availability

Msprime is freely available under the terms of the GNU General Public License v3.0, and can be installed from the Python Package Index or the conda-forge (conda-forge community 2015) conda channel. Development is conducted openly on GitHub at https://github.com/tskit-dev/msprime/. The documentation for msprime is available at https://tskit.dev/msprime/docs/. The source code for all the evaluations and figures in this manuscript is available at https://github.com/tskit-dev/msprime-1.0-paper/.

Acknowledgments

We acknowledge the contributions of Ivan Krukov who we consider eligible for authorship, but were unable to contact for approval. We would like to thank Iain Mathieson and Alywyn Scally for helpful comments on the manuscript.

Funding

A.D.K. was supported by US National Institutes of Health awards R01GM117724 and R01HG010774. A.L.G. was supported by US National Institutes of Health award R00HG008696 to Daniel R. Schrider. B.E. was supported by Deutsche Forschungsgemeinschaft grant 273887127 through Priority Programme SPP 1819: Rapid Evolutionary Adaptation (grant STE 325/1-2) to Wolfgang Stephan. B.E. would also like to acknowledge funding through the Icelandic Research Centre (Rannís) through an Icelandic Research Fund Grant of Excellence nr. 185151-051 to Einar Árnason, Katrin Halldórsdóttir, Alison Etheridge, Wolfgang Stephan, and B.E. F.B. is funded by the Deutsche Forschungsgemeinschaft. EXC 2064/1: Project number 39072645 and EXC 2124: Project number 390838134. G.B. and K.L. are supported by an European Research Council starting grant (ModelGenomLand 757648) to K.L. Graham Gower was supported by a Villum Fonden Young Investigator award to Fernando Racimo (project no. 00025300). Gregor Gorfanc is supported by the Chancellor’s Fellowship of the University of Edinburgh and the UK Biotechnology and Biological Sciences Research Council grant to the Roslin Institute BBS/E/D/30002275. Jere Koskela is supported in part by UK Engineering and Physical Sciences Research Council grant (EP/R044732/1). Jerome Kelleher is supported by the Robertson Foundation. P.L.R. was supported by US National Institutes of Health award R01HG010774. S.G. acknowledges funding from the Canada Research Chairs Program, from the Canadian Institutes of Health Research PJT 173300, and from the Canadian Foundation for Innovation.

Conflicts of interest

The authors declare that there is no conflict of interest.

Literature cited


Appendix

Mutation generation

The algorithm that msprime uses to simulate mutations on a tree sequence proceeds in two steps: first, mutations are “placed” on the tree sequence (i.e., sampling their locations in time, along the genome, and on the marginal tree), and then the ancestral and derived alleles of each mutation are generated. All mutation models share the code to place mutations, but choose alleles in different ways.

First, mutations are placed on the tree sequence under an inhomogeneous Poisson model by applying them independently to each edge. If an edge spans a region $[a, b]$ of the genome and connected parent and child nodes with times $s < t$, and the mutation rate locally is $\mu$, then the number of mutations on the edge is Poisson with mean $\mu(t - s)(b - a)$, and each mutation is placed independently at a position chosen uniformly in $[a, b]$ and a time uniformly in $[s, t]$. In a discrete genome, all positions are integers and so more than one mutation may occur at the same position on the same edge. Otherwise (i.e., for an infinite-sites model), positions are rejection sampled to obtain a unique floating-point number. If an edge spans a region of the genome with more than one mutation rate, this is done separately for each sub-region on which the mutation rate is constant. Since each edge is processed independently, the algorithm scales linearly with the number of edges in the tree sequence.

Next, alleles are chosen for each mutation. If the site was not previously mutated, then a new ancestral allele is chosen for the site, according to an input distribution of ancestral state allele probabilities. Then, each mutation on the tree is considered in order: mutations are placed at the maximum mutation rate, and then the ancestral and derived alleles of each mutation are generated. All mutation models share the code to place mutations, but choose alleles in different ways.

In previous versions of msprime, silent mutations were disallowed, and we could have removed them from the output entirely. However, we have chosen to leave them in, so that for instance simulating with the HKY mutation model will result in silent mutations if not all equilibrium frequencies are the same. The presence of silent mutations may at first be surprising but there is a good reason to leave them in: to allow layering of different mutation models. Suppose that we wanted to model the mutation process as a mixture of more than one model, e.g., Jukes–Cantor mutations at rate $\mu_1$ and HKY mutations occur at rate $\mu_2$. Layering multiple calls to `sim_mutations` is allowed, so we could first apply mutations with the J69 model at rate $\mu_1$ and then add more with the HKY model at rate $\mu_2$. However, there is a small statistical problem: suppose that after applying Jukes–Cantor mutations, we have an $A \rightarrow C$ mutation, but then the HKY mutations inserts another mutation in the middle, resulting in $A \rightarrow \ldots \rightarrow C$. If neither mutation model allows silent transitions, then this is clearly not correct, i.e., it is not equivalent to a model that simultaneously applies the two models. (The impact is small, however, as it only affects sites with more than one mutation.) The solution is to make the Jukes–Cantor model state-
Run time scales quadratically with the product of $Ne$ base pairs divided by $10^8$. The black lines are quadratic fits separately in chromosome length in Morgans ($values of NeL$ each panel and sample size. Vertical gray lines show the approximate stdpopsim segments with $Ne$ simulation, for various values of effective population size ($simulations on an Intel i7-6600U CPU. Each point is the run time of one Figure 8 recombination rate in these simulations was $10^{7}$ accurately predict simulation time for a given $Kelleher$ quadratic prediction is well supported by observations (see also $et al.$ 2016, Figure 2). We also see that the dependence on $n$ is quite weak, since increasing sample size 100-fold only increases run time by a factor of 2 or so. However, the log$_n^2$ factor implied by Equation (1) (the sum is a harmonic number and can be approximated by log$n$) is not well supported by observed run times (or numbers of events) except possibly at very large values of $rho$. It therefore appears that a different dependence on $n$ is required to accurately predict simulation time for a given $rho$ and $n$.

Figure 8 is a useful yardstick, allowing us to predict how long simulations should take for a wide range of species. Taking a typical chromosome to be 1 Morgan in length, these plots show, roughly, that simulating chromosome-length samples from a population of thousands of individuals takes seconds, while samples from a population of tens of thousands take minutes. Simulating whole chromosomes for many species is very fast, with 1000 samples of chromosome 1 for Arabidopsis thaliana taking less than a second, and a few minutes for dogs and humans. However, the dependence on $rho$ is quadratic, and if $rho$ is sufficiently large, simulations may not be feasible. For example, the Drosophila melanogaster chromosome 2L is about 23.5 Mb long with an average recombination rate of around 2.4 x $10^{-8}$, so $L = 0.57$, and with $N_e = 1.7 x 10^6$ (Li and Stephan 2006), $NeL$ $\approx 10^6$, so extrapolating the curve in Figure 8B predicts that simulation would require around 177 h for 1000 samples. For such large values of $rho$, we recommend users consider approximate simulations. Since msprime does not currently have efficient implementations of approximate coalescent with recombination models, in these cases, we recommend using SMC-based methods such as acem, particularly if sample sizes are small. In practice, to predict the running time of a given simulation in msprime, we recommend that users measure run time in a series of simulations with short genome lengths and the desired sample size, and then predict run time by fitting a quadratic curve to genome length as in Figure 8. It is important to note that the quadratic curves in the two panels of Figure 8 are different, and predicting the run times of days-long simulations using the timing of seconds-long runs is unlikely to be very accurate.

What about simulations with changing population size? To understand how run time depends on demography, it helps to consider why run time is quadratic in $rho$. At any point in time, msprime must keep track of some number of lineages, each of which contains some number of chunks of genetic material. Common ancestor events reduce the number of lineages, and recombination events increase their number. However, with long genomes, only a small fraction of the common ancestor events will involve overlapping segments of ancestry and lead to coalescence in the marginal trees. Such disjoint segments are often far apart (on average, about distance $L/2$), and so recombine apart again immediately, it is these large numbers of rapid and inconsequential events that lead to the quadratic run time. The maximum number of lineages occurs when the increase and decrease in numbers of lineages due to common ancestor and recombination events balance out. To get an idea of run time we can estimate when this balance occurs. Suppose that the maximum number of lineages is $M$; at this time, the rate of common ancestor events is $M(M-1)/(4N_e)$ and the total rate of recombination is $M\ell$, where $\ell$ is the mean length of genome carried by each lineage (including "trapped" non-ancestral material). At the maximum, coalescence and recombination rates are equal, so a typical segment of ancestry will spend roughly half its time in a lineage with at least one other such segment—and, since such lineages carry at least two segments, at most one-third of the lineages carry long trapped segments of ancestry. Since the maximum number of lineages is reached very quickly (Nelson et al. 2020), this implies that $\ell = L/6$. Setting the rates of recombination and common ancestor events to be equal and solving for $M$, we find that $M$ is roughly equal to $L N_e$. The number of lineages then decreases gradually from this maximum on the coalescent time scale, and therefore over roughly 2N_e generations. Since the total rate of events when the maximum number of lineages is present is roughly $L^2 N_e/6$, then the total number of events is proportional to $(L^2 N_e)^2$—i.e., proportional to $rho^2$.

What does this tell us about run time for simulating time-varying population sizes? Suppose that population size today is $N_t$, while T generations ago it was $N_0$. Does the run time depend more on $4N_0 L$ or $4N_t L^2$? The answer depends on how T compares to $N_1$, if $T/N_1 \ll 1$ then the number of extant lineages remaining after T generations is likely to be substantial, and the algorithm
runtime is primarily determined by \( N_2 \). Conversely, if \( T/N_1 \gg 1 \), then few extant lineages are likely to remain by time \( T \) and run-time depends mainly on \( N_1 \). For instance, in many agricultural species \( N_1 \approx 100 \), while \( N_2 \approx 10^3 \), and the run time will depend critically on \( T \)—in other words, simulation will be quick in a species with a strong domestication bottleneck, and slow otherwise.

### Selective sweeps model

Sweep trajectories are generated in `mapprime` using a jump process approximation to the conditional diffusion of an allele bound for fixation (Coop and Griffiths 2004). The jump process moves back in time following the beneficial allele frequency, \( p \), from some initial frequency (e.g., \( p = 1 \)) back to the origination of the allele at \( p = 1/(2N) \), tracking time in small increments \( \delta t \). Then, given the frequency \( p \) at time \( t \) ago, the frequency \( p' \) at time \( t + \delta t \) ago is given by

\[
p' = \begin{cases} 
p + \mu(p)\delta t + \sqrt{p(1-p)\delta t} & \text{with probability } 1/2 \\
p + \mu(p)\delta t - \sqrt{p(1-p)\delta t} & \text{with probability } 1/2 
\end{cases}
\]

where

\[
\mu(p) = \frac{\exp((1-p)/\tan(h(1-p)))}{2N}.
\]

Here, \( \alpha = 2Ns \) is the fitness advantage in homozygotes. This model assumes genic selection (i.e., that the dominance coefficient \( h = 0.5 \)), but can be generalized straightforwardly to include arbitrary dominance. We can also define trajectories to model neutral alleles and soft selective sweeps, which we plan as future additions to `mapprime`.

### Likelihood calculations

We provide two functions to facilitate likelihood-based inference. Both are implemented only for the simplest case of the standard ARG with a constant population size, and require tree sequences compatible with the `record_full_arg` option as their arguments.

The `mapprime.log_arg_likelihood` (\( ts \), \( r \), \( N \)) function returns the natural logarithm of the sampling probability of the tree sequence \( ts \) under the ARG with per-link, per-generation recombination probability \( r \) and population size \( N \) [e.g., Kuhner et al. 2000, Equation (1)]. Specifically, the function returns the logarithm of

\[
\left( \frac{1}{2N} \right)^{s} \left( \prod_{i \in R} t_i \right) \exp \left( -\sum_{i=1}^{q} \left[ \frac{k_i}{2N} \right] + h_i \right) t_i
\]

where \( t_i \) is the number of generations between the \( (i-1) \)th and \( i \)th event, \( k_i \) is the number of extant ancestors in that interval, \( l \) is the number of links in that interval that would split ancestral material should they recombine, \( q \) is the total number of events in the tree sequence \( ts \), \( q_i \) is the number of coalescences, \( R \) is the set of indices of time intervals which end in a recombination, and \( g_i \) is the corresponding gap: the length of contiguous non-ancestral material around the link at which the recombination in question took place. The gap indicates the number of links (or length of genome in a continuous model) at which a recombination would result in exactly the observed pattern of ancestral material in the ARG. For a continuous model of the genome and a recombination in ancestral material, we set \( g_i = 1 \) and interpret the result as a density.

The `mapprime.unnormalised_log_mutation_likelihood` (\( ts \), \( m \)) function returns the natural logarithm of the probability of the mutations recorded in the tree sequence \( ts \) given the corresponding ancestry, assuming the infinite sites model, up to a normalizing constant which depends on the pattern of mutations, but not on the tree sequence or the per-site, per-generation mutation probability \( m \). Specifically, the function returns the logarithm of

\[
e^{-T/m} \left( \frac{Tm}{2} \right)^{M} \prod_{j=1}^{h} \frac{1}{q_j},
\]

where \( T \) and \( M \) are the total branch length and set of mutations in \( ts \), respectively, \( M \) is the number of mutations, and for a mutation \( j \), \( h_j \) is the total branch length on which \( j \) could have arisen while appearing on all of the leaves of \( ts \) it does, and on no others. Unary nodes on marginal trees arising from the `record_full_arg` option mean that, in general \( h_j \) corresponds to the length of one or more edges.

### Multiple merger coalescent model

Multiple merger coalescents, in which no more than one group of ancestral lineages may merge into a common ancestor, at a given time, are referred to as \( \Lambda \)-coalescents. The rate at which a given group of \( k \) out of a total of \( b \) lineages merges is

\[
\lambda_{b,k} = \int_0^1 x^{k-2}(1-x)^{k-1} \Lambda(dx) + aI_{k=2}, \quad 2 \leq k \leq b,
\]

where \( I_{[A]} := 1 \) if \( A \) holds, and zero otherwise, \( a \geq 0 \) is a constant, and \( \Lambda \) is a finite measure on the unit interval without an atom at zero (Donnelly and Kurtz 1999; Pitman 1999; Sagitov 1999). There is also a larger class of simultaneous multiple merger coalescents involving simultaneous mergers of distinct groups of lineages into several common ancestors (Schweinsberg 2000). These are commonly referred to as \( \Xi \)-coalescents, and often arise from population models incorporating diploidy or more general polyploidy (Birkner et al. 2013a; Blath et al. 2016). To describe a general \( \Xi \)-coalescent, let \( \Lambda \) denote the infinite simplex

\[
\Lambda := \left\{ (x_1, x_2, \ldots) : x_1 \geq x_2 \geq \cdots \geq 0, \sum_{j=1}^{\infty} x_j = 1 \right\}.
\]

The rate of mergers is determined by \( \Xi = \Xi_0 + a\delta_0 \), where \( a \geq 0 \) is a constant, \( \delta_0 \) is the Dirac delta measure, and \( \Xi_0 \) is a finite measure on \( \Lambda \) with no atom at \((0, 0, \ldots)\). For an initial number of blocks \( b \geq 2 \) and \( r \in \{1, 2, \ldots, b-1\} \), let \( k_1 \geq 2, \ldots, k_r \geq 2 \) be the sizes of \( r \) merger events and \( s = b - k_1 - \cdots - k_r \) be the number of blocks not participating in any merger. The rate of each possible set of mergers with sizes \((k_1, \ldots, k_r)\) is

\[
\lambda_{n; k_1, \ldots, k_r} = \sum_{\ell_0 = 0}^{\infty} \sum_{\ell_1=1}^{\ell_0} \cdots \sum_{\ell_r=1}^{\ell_{r-1}} \frac{s^{\ell_0}}{(\ell_0)!} \times \left( 1 - \sum_{j=1}^{\infty} \frac{x_j}{1-x_j} \right)^{s-\ell_0} \frac{1}{\sum_{j=1}^{\infty} x_j} \Xi_0(dx) + aI_{[r=2]},
\]

and the number of such \((k_1, \ldots, k_r)\) mergers is

\[
\mathcal{N}(b; k_1, \ldots, k_r) = \binom{b}{k_1, \ldots, k_r} \frac{1}{\prod_{j=1}^{r} \ell_j!},
\]

where \( \ell_j := \{ i \in \{1, \ldots, r \} : k_i = j \} \) is the number of mergers of size \( j \geq 2 \) (Schweinsberg 2000).
Viewing coalescent processes strictly as mathematical objects, it is clear that the class of $\Xi$-coalescents contains $\Lambda$-coalescents as a specific example in which at most one group of lineages can merge at each time, and the class of $\Lambda$-coalescents contain the Kingman-coalescent as a special case. However, viewed as limits of ancestral processes derived from specific population models, they are not nested. For example, one can obtain $\Lambda$-coalescents from haploid population models incorporating sweepstakes reproduction and high fecundity, and $\Xi$-coalescents for the same models for diploid populations (Birkner et al. 2013a). One should therefore apply the models as appropriate, i.e., $\Lambda$-coalescents to haploid (e.g., mtDNA) data, and $\Xi$-coalescents to diploid or polyploid (e.g., autosomal) data (Blath et al. 2016).

In msprime, we have incorporated two examples of multiple-merger coalescents. One is a diploid extension (Birkner et al. 2013a) of the haploid Moran model adapted to sweepstakes reproduction considered by Eldon and Wakeley (2006). Let $N$ denote the population size, and take $\psi \in (0, 1]$ to be fixed. In every generation, with probability $1 - \psi$, a single individual (picked uniformly at random) perishes. With probability $\psi N$ individuals picked uniformly without replacement perish instead. In either case, a surviving individual picked uniformly at random produces enough offspring to restore the population size back to $N$. Taking $\psi = 1/N^\gamma$ for some $\gamma > 0$, Eldon and Wakeley (2006) obtain $\Lambda$-coalescents for which the $\Lambda$ measure in Equation (2) is a point mass at $\psi$. The simplicity of this model does allow one to obtain some explicit mathematical results (see e.g., Der et al. 2012; Eldon and Freund 2018; Freund 2020; Matuszewski et al. 2018), and the model has also been used to simulate genealogies within phylogenies (Zhu et al. 2015). As well as the haploid model of Eldon and Wakeley (2006), msprime provides the diploid version of Birkner et al. (2013a), in which individuals perish as above, but replacements are generated by sampling a single pair of diploid individuals as parents, with children sampling one chromosome from each parent. Hence, there are four parent chromosomes involved in each reproduction event, which can lead to up to four simultaneous mergers, giving rise to a $\Xi$-coalescent with merger rate

$$\lambda_{\text{Dirac}}^{\text{D})} = \frac{c \psi^2}{2} \left( \frac{1 + c \psi^2}{4} \right)^{s} \frac{4}{\left( \psi \right)} \frac{\sum_{s=2}^{\infty}(s)}{2^{k-1} k} \frac{\left( \psi \right)^{k-1} \left( 1 - \psi \right)^{k-t}}{1 + c \psi^2/4}$$

The interpretation of Equation (3) is that “small” reproduction events in which two lineages merge occur at rate $1/(1 + c \psi^2/4)$, while large reproduction events with the potential to result in simultaneous multiple mergers occur at rate $(c \psi^2/4)/(1 + c \psi^2/4)$.

The other multiple merger coalescent model incorporated in msprime is the haploid population model considered by Schweinsberg (2003), as well as its diploid extension (Birkner et al. 2018). In the haploid version, in each generation of fixed size $N$, individuals produce random numbers of juveniles ($X_1, \ldots, X_N$) independently, each distributed according to a stable law satisfying

$$\lim_{k \to \infty} \text{Ck}^k \mathbb{P}(X \geq k) = 1$$

with index $\alpha > 0$, and where $C > 0$ is a normalizing constant. If the total number of juveniles $S_N := X_1 + \ldots + X_N$ produced in this way is at least $N$, then $N$ juveniles are sampled uniformly at random without replacement to form the next generation. As long as $E[X_i] > 1$, one can show that $S_N < N$ has exponentially small probability in $N$, and does not affect the resulting coalescent as $N \to \infty$ (Schweinsberg 2003). If $x > 2$, the ancestral process converges to the Kingman-coalescent, if $1 \leq x < 2$, the ancestral process converges to a $\Lambda$-coalescent with $\Lambda$ in Equation (2) given by the Beta($2 - x, x$) distribution, i.e.,

$$\Lambda(dx) = \mathbb{1}_{[0 < x \leq 1]} \frac{1}{B(2-x, x)} x^{1-x}(1-x)^{-x+1} dx,$$

where $B(a, b) = \Gamma(a)\Gamma(b)/\Gamma(a + b)$ for $a, b > 0$ is the beta function (Schweinsberg 2003). This model has been adapted to diploid populations by Birkner et al. (2018), and the resulting coalescent is $\Xi$-coalescent with merger rate

$$\lambda_{\text{Beta}}^{\text{M})} = \sum_{s=0}^{\infty}(s) \frac{4}{4 \psi^2} \frac{B(k + \ell - x, s + \ell + x)}{B(2 - x, x)}$$

where $k := k_1 + \cdots + k_r$ (Blath et al. 2016; Birkner et al. 2018). The interpretation of Equation (6) is that the random number of lineages participating in a potential merger is governed by the $\Lambda$-coalescent with rate Equation (5), and all participating lineages are randomly allocated into one of four groups corresponding to the four parental chromosomes, giving rise to up to four simultaneous mergers.

The stable law Equation (4) assumes that individuals can produce arbitrarily large numbers of juveniles. Since juveniles are at least fertilized eggs, it may be desirable to suppose that the number of juveniles surviving to reproductive maturity cannot be arbitrarily large. Hence we also consider an adaptation of the Schweinsberg (2003) model, where the random number of juveniles has a deterministic upper bound $\phi(N)$, and the distribution of the number of juveniles produced by a given parent (or pair of parents in the diploid case) is

$$\mathbb{P}(X = k) = \mathbb{1}_{[1 \leq k \leq \phi(N)]} \frac{\phi(N + 1)^s}{\phi(N + 1)^s - 1} \left( \frac{1}{k^x} - \frac{1}{(k + 1)^x} \right).$$

See Eldon and Stephan (2018) for a related model. One can follow the calculations of Schweinsberg (2003) or Birkner et al. (2018) to show that if $1 < x < 2$ then, recalling that $k = k_1 + \cdots + k_r$, the merger rate is

$$\lambda_{\text{Beta,M}}^{\phi(N)} = \sum_{k=1}^{\phi(N)} \frac{4}{4 \psi^2} \frac{B(M; k + \ell - x, s + \ell + x)}{B(M; 2 - x, x)}$$

where $B(z; a, b) := \int_0^x t^{a-1}(1-t)^{b-1} dt$ for $a, b > 0$ and $0 < z < 1$ is the incomplete beta function, and

$$M := \lim_{N \to \infty} \frac{\phi(N)/N}{\phi(N)/N + \mathbb{E}[X_1]} \in (0, 1]$$

(Chetwynd-Diggle et al. 2022). In other words, the measure $\Lambda$ driving the multiple mergers is of the same form as in Equation (5) with $0 < x \leq M$ in the case $1 < x < 2$ and $\lim_{N \to \infty} \mathbb{P}(X_1) \to 0$ if $x \geq 2$ or $\phi(N)/N \to 0$, then the ancestral process converges to the Kingman-coalescent (Chetwynd-Diggle et al. 2022).