Graph-based algorithms for Laplace transformed coalescence time distributions.

Gertjan Bisschop *
Institute of Evolutionary Biology, University of Edinburgh, EH9 3FL Edinburgh, Scotland

Abstract

Extracting information on the selective and demographic past of populations that is contained in samples of genome sequences requires a description of the distribution of the underlying genealogies. Using the Laplace transform, this distribution can be generated with a simple recursive procedure, regardless of model complexity. Assuming an infinite-sites mutation model, the probability of observing specific configurations of linked variants within small haplotype blocks can be recovered from the Laplace transform of the joint distribution of branch lengths. However, the repeated differentiation required to compute these probabilities has proven to be a serious computational bottleneck in earlier implementations.

Here, I show that the state space diagram can be turned into a computational graph, allowing efficient evaluation of the Laplace transform by means of a graph traversal algorithm. This general algorithm can, for example, be applied to tabulate the likelihoods of mutational configurations in non-recombining blocks. This work provides a crucial speed up for existing composite likelihood approaches that rely on the joint distribution of branch lengths to fit isolation with migration models and estimate the parameters of selective sweeps. The associated software is available as an open-source Python library, `agemo`.

*g.bisschop@sms.ed.ac.uk

1 Introduction

The Laplace transform is often introduced as a formal tool to solve differential equations. Yet in mathematical fields like queuing theory, the integral transform is used to simplify the analysis of the studied probabilistic problems. This is due to two key properties of the Laplace. One, the transform of the distribution of the sum of independent random random variables becomes the product of their respective Laplace transformed marginal distributions. Two, the Laplace transform of a random variable X describing the length of an interval, has a clear probabilistic interpretation. Let a Poisson process with intensity ω mark this interval, then the Laplace transform, $f^*(\omega) = E[e^{-\omega X}]$, is the probability of not observing any marks in the considered interval (Råde, 1972). If we translate this idea to the standard coalescent framework (Kingman, 1982; Hudson, 1983; Tajima, 1983) and let the marking process describe the arrival of mutations, then the Laplace transform of the distribution of the total branch length gives the probability of not observing any mutations along the modelled genealogies. Conversely, the probability of not observing any mutations between two coalescence events gives us the transformed distribution of the total branch length spanned by those two events. This means that, given a state-space graph that describes all possible transitions during the coalescent process (see Fig. 2.1), one can easily write down all associated expressions in the Laplace domain (Lohse et al., 2011). This is not the case for the time domain.

The Laplace transformed description of the distribution of coalescence times has been used to tackle two major problems in population genetics: fitting explicit models of population history (Lohse et al., 2011; Frantz et al., 2014; Bunnefeld et al., 2015; Lohse et al., 2016) as well as estimating sweep parameters (Bisschop et al., 2021). Note that because the Laplace transform can be interpreted as the generating function (GF) of a continuous random variable, this method is often referred to as the GF approach. These studies have used composite likelihood-based approaches that summarize mutational information as counts of the (joint) site frequency spectrum within blocks of a fixed length (Bunnefeld et al., 2015). The associated probabilities are given by a multivariate Poisson distribution mixed over the joint distribution of branch lengths. The Laplace transform is well suited to compute these probabilities because this Poisson distribution can be written as a function of the Laplace transform of the branch length distribution. Specifically, the probability of observing $\mathbf{k}$ mutations along each of the k_i branch types is proportional to the $\mathbf{k}^{th}$ derivative with respect to associated variable in the Laplace domain (Lohse et al., 2011). Previous implementations have always used a computer algebra system (CAS) to compute these higher-order derivatives of the Laplace transform. Such an approach suffers from an explosion in the number of terms due to the product or the Leibniz rule. This computational bottleneck has limited the usability of the framework.

One way to solve the computational bottleneck is to replace the recursive description of the gen-

erating function by matrix manipulations. This has been done using phase-type theory (Hobolth et al., 2019). Phase-type distributions are the result of a mixture or convolution of exponential distributions. The theory provides an alternate way of translating the state transition diagram into a description of the branch length distribution of genealogical trees. One of the major computational advantages of this description is that the matrices that define the distribution are typically preserved up to the point of evaluation (Hobolth et al., 2019). Because computationally efficient algorithms for linear algebra operations already exist, phase-type theory lends itself to efficient implementations. With increasing sample size however, matrix operations become computationally unfeasible. Given the sparsity of most real-world state spaces, graph-based representations of these matrices can alleviate this issue to some extent (Røikjer et al., 2022). However, currently no algorithms have been described to extract information from the joint branch length distribution.

Graphs reveal the relationships between the base entities or nodes. Therefore, the more complex these relationships are, the more connected the graph will be and the more useful a graph-based representation becomes. In machine learning for example, computational graphs representing mathematical equations are used to efficiently calculate derivatives (Güne¸ et al., 2018). What I have implemented here takes inspiration from automatic differentiation in that we will use the state-space graph generated during recursion as a computational graph to avoid both symbolic computation and repetitively evaluating the same expressions.

Here, I present the key algorithms underlying this approach as well as `agemo`, an open-source non-symbolic implementation of the GF framework. The paper is structured as follows. First, I will summarize the description of the GF as a large symbolic expression as defined in Lohse et al. (2011) and show how the GF can be represented more succinctly. Secondly, I lay out the graph traversal algorithm that allows efficient evaluation of the GF. I then show how this algorithm can be used to query the joint distribution of branch lengths and tabulate the probabilities of observing mutation configuration in blocks of non-recombining sequence.

2 Methods

2.1 Recursive description of the branch length distribution in the Laplace domain

Given a sample of $n = \sum_{i=1}^{k} n_i$ uniquely labeled lineages from k distinct populations, we can represent all possible coalescent histories of that sample in a single rooted directed graph (Simonsen and Churchill, 1997). By labelling lineages by the samples they subtend, we can associate each node of the transition diagram with a vector $\Omega = (\Omega_1, ..., \Omega_k)$ uniquely describing that state. Here Ω_i represents all lineages present in deme i. As we move through the graph from the source node,

representing the set of all sampled lineages, to the absorbing state or most recent common ancestor (mrca), we track the movement and coalescence of lineages. Coalescence events reduce the number of lineages in a deme, while events like a population split or migration will move lineages from one deme to another. Note that any state where all but Ω_i are empty and $|\Omega_i| = 1$, can be an absorbing state.

The state space graph as described above can be generated recursively. All state transitions are conditionally independent. They only depend on the lineages in the current state and on the set of competing processes that define how one moves from one state to the next. Along each path through the graph, the time to the mrca is distributed as the sum of the inter-event times. In the Laplace domain, the sum of independent random variables is equal to the product of their respective Laplace transformed distributions. This general property of generating functions turns this graph into more than just a visual aid. The graph is now a description of how to generate the Laplace transform of the branch length distributions. Given the expressions that describe the time to go from one state to the next, the distribution associated with a single path through the graph can be retrieved by multiplying all the expressions associated with the edges found along that path. The entire Laplace transform is then a simple sum of the equations describing all paths through the graph.

Finally, because of the probabilistic interpretation of the Laplace transform, the expression associated with each edge equates to the probability of observing the event of interest before any other event happening at rate ω. In the standard coalescent framework, coalescence events are exponentially distributed with rate $\binom{k}{2}$ when there are k lineages remaining. So in the Laplace domain, the distribution of the waiting time until the next coalescence is given by $\binom{k}{2}/(\binom{k}{2} + \omega)$. To incorporate more than one process with an exponentially distributed waiting time, it suffices to observe that $min(X, Y) \sim exp(\omega_x + \omega_y)$ when both $X \sim exp(\omega_x)$ and $Y \sim exp(\omega_y)$. This means one can incorporate as many events with exponentially distributed waiting times as computationally possible, the probability of observing one of the competing events first, and therefore the Laplace transform of the joint distribution of branch lengths, will still have the same general form.

$$f^\star[\boldsymbol{\omega}] = \frac{l'_i \lambda_i}{\sum_{j=0}^{m} l_j \lambda_j + \sum o_k \omega_k} \qquad \text{with} \sum l'_i = l_i \tag{1}$$

In this equation, we associate a unique dummy variable (ω_k) with each of the branch types along which all of the competing processes (λ_j) happen. Roman letters represent integers that count the number of branches of a particular type (o_i), or the number of ways a certain (coalescence or other) event can modify the current state (l_j). Note that in the case of multiple populations, coalescence rates are given relative to the rate in a reference population, i.e. $\lambda_i = N_{e_i}/N_{e_{ref}}$.

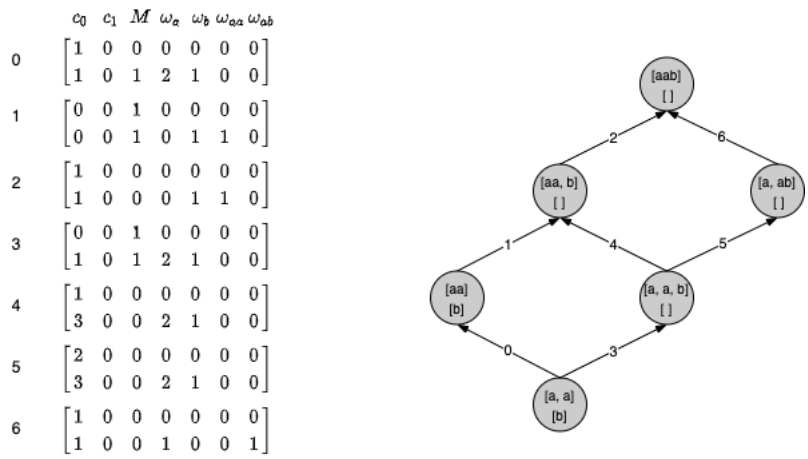

Figure 1: **From coalescent state space to equation array:** coalescent state space graph for two populations A and B with 2 and 1 unphased sample(s), respectively. The demographic model assumes a single mass migration event from population B to A back in time.

Each equation associated with an edge in the state space graph can be encoded as a matrix with two rows containing the integer coefficients (l'_i, $\mathbf{l}$, $\mathbf{o}$). The first row represents the numerator and will only contain a single non-zero value, l'_i. The second row represents the denominator. Storing the equations in this way ensures that we can efficiently substitute in parameter values by taking the dot product with a vector representing a point in parameter space ($\boldsymbol{\lambda}$, $\boldsymbol{\omega}$) once the Laplace transform needs to be evaluated. Also, storing the equation coefficients in matrix form allows us to efficiently perform operations on the equations (see 2.1.1). Therefore, a minimal representation of the GF consists of an array containing all unique equations and an array of lists with equation indices describing all paths through the graph.

2.1.1 Discrete events

For the general description of the GF, we have assumed that all competing processes have exponentially distributed waiting times. As outlined by Lohse et al. (2011), discrete events that only happen once can be included by treating them initially as a competing exponentially distributed process with rate δ. The GF parameterised by the discrete time T of the event can be recovered by taking the inverse transform of the GF divided by its associated dummy variable (δ). This procedure has been used to incorporate population divergence, admixture events, and bottlenecks (Lohse et al., 2011; Bunnefeld et al., 2015), as well as selective sweeps (Bisschop et al., 2021).

Previous implementations have relied on a CAS to obtain an analytic solution for the inverse transform of the GF. However, as long as we limit ourselves to a single discrete event, the GF will always be a sum of the products of factors of the form $f_i(c_i) = \frac{1}{c_i+\delta}$. Using partial fraction expansion, we can formulate a closed-form solution to the inverse Laplace with respect to δ, where T represents the time to the discrete event. Also, having stored all equation coefficients as an array, we can do so in a way that allows for efficient substitution of all parameter values.

$$g(T) = \mathscr{L}^{-1}(\frac{1}{\prod_{i=0}^{k} f_i(\mathbf{x})}) = (-1)^{k+1} \sum_i \frac{e^{-c_i T}}{\prod_{i \neq j}(f_i(\mathbf{x}) - f_j(\mathbf{x}))} \tag{2}$$

Looking at a single path along the graph, only the equations associated with edges leading up to the node representing the discrete event will contain δ. Equations associated with edges past that point can be treated as constants. The resulting inverse of this path will therefore be an expression given by Eq. 2 times the unchanged equations associated with all edges positioned downstream (moving through the graph backwards in time) of the node associated with the discrete event.

2.1.2 Adding in new events

Currently, in addition to coalescence, two types of events have been implemented in the Python library `agemo`: unidirectional migration at a constant rate and population splits (forwards in time). Because of the recursive description, adding in more event types is straightforward and only requires a description of all possible state transitions due to that event given the current configuration of lineages. Note that the library does currently not accommodate events that generate cycles in the graph. This means that bi-directional migration, for example, is not supported.

2.2 Graph traversal algorithm

The one-to-one correspondence between the state space graph and the Laplace transform means the state space graph can be thought of as a computational graph. Evaluating the transform at a single point s in the Laplace domain equates to substituting the value into the expression associated with each edge, followed by multiplying the results along each path and adding the results across all paths. This is the general idea that will be used in the next paragraph.

However, note that in the case of a discrete event and the general form of the inverse (see Eq. 2), the graph needs to be modified slightly so that, ultimately, the nodes of the computational graph again represent the factors of a multiplication. This is achieved by for each path reducing the part of the path leading up to the node associated with the discrete event to a single edge and pairing that edge with the result of Eq. 2. Also note that to simplify the evaluation algorithm, equations are represented by nodes instead of edges (see Fig. 2.2 C).

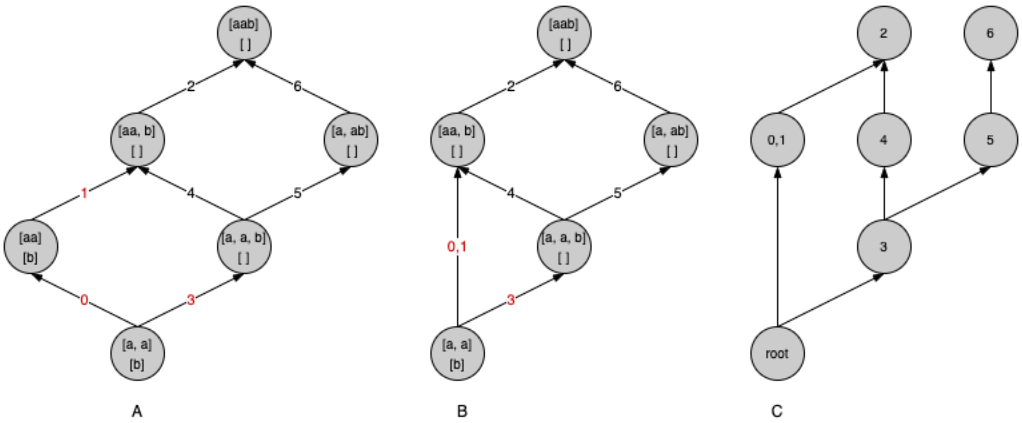

Figure 2: **From coalescent state space to computational graph:** setup identical to Fig. 2.1. A: Unmodified state-space graph. B: collapsed form, grouping all parts of each path that require inverting with respect to the dummy variable associated with the discrete event. The integers in red are the indices of the equations containing δ. C: To simplify the formulated algorithms, the graph gets inverted. The nodes now represent the equations previously associated with the edges.

2.2.1 General algorithm

Given the computational graph (as described in 2.2 and see Fig. 2.2C), a general algorithm to propagate any evaluation of the equations associated with each node is given by Alg. 1. The algorithm relies on the fact that, implicitly, the edges of the graph represent multiplication. The evaluated values associated with each node do not need to be single floats. They can be the coefficients of a generating function, for example, representing the probabilities of seeing particular mutation types (see 2.2.2). In these cases, multiplication and addition operators will need to be defined for propagation. We can then rely on the commutative property to efficiently traverse the graph towards the root. Especially in the case where addition is a less costly operation than multiplication (as is the case for polynomials, see 2.2.2), it will pay off to add the values associated with the children of a node prior to multiplication.

2.2.2 Mutation type probabilities

Given a sample of n lineages, we can distinguish $2^n - 2$ branch types. Along each of these branch types, mutations might occur. Assuming an infinite-site mutation model, the probability of seeing k_i mutations in short blocks of a given length along each of these i branch types can be derived using

Algorithm 1: Propagate values through graph

```
function PROPAGATE;
Input: A (adjacency list of graph), N (evaluated equations for each node), E (topological
       sorting of graph)
foreach parent in E do
    children = A[parent];
    temp = 0;
    if children then
        foreach child in children do
            temp+=N[child];
        end
        if parent not root then
            N[parent] = PRODUCT(temp, N[parent]);
        else
            return temp;
        end
    end
end
```

Algorithm 2: Product of two truncated Taylor series (Neidinger, 2013).

function SERIES_PRODUCT;

Input : Two arrays A and B with same shape

Output: array C of same shape as A and B

foreach *multi-index* $\boldsymbol{k}$ **do**

  $sum = 0$;

  foreach *multi-index* $\boldsymbol{j} \leq$ *multi-index* $\boldsymbol{k}$ **do**

    sum = ADD(sum, $A[\boldsymbol{j}] * B[\boldsymbol{k} - \boldsymbol{j}]$);

  end

  $C[\boldsymbol{k}]$ = sum;

end

return C;

the GF. This is a more general description of the blockwise site frequency spectrum or bSFS as introduced by Bunnefeld et al. (2015). The bSFS only distinguishes mutations along branches with the same number of descendants. By ignoring both phase and root information, the distinguished mutation configurations are essentially instances of the folded (joint) site frequency spectrum for blocks of a fixed length. Each mutation type is defined as a vector of the form $(k_1, ...k_{2^n-2})$ where each entry is a count within the interval $[0, 1, 2, ..., k_i^{max} + 1]$. $k_i^{max} + 1$ is used to group all mutation configurations with more than k_i^{max} mutations. Note that the array of all mutation types with a non-zero probability is a sparse array. Many of the $(\prod_{i=1}^{n-1} k_i^{max} + 2)$ possible mutation types define the presence of mutations along incompatible branch types. Furthermore, because of the symmetries inherent to the coalescent, quite a few mutation types will be equally likely.

The probability of observing mutation configuration $\mathbf{k}$ under a specified model is proportional to a term in a (truncated) Taylor series expansion (see eq. (1) in Lohse et al. (2011) for details). Any naive approach, based on calculating all higher order derivatives using a CAS, will suffer from an explosion in the number of terms due to the Leibniz or product rule when differentiating. Generally, a CAS will fail to take into account the fact that the same partial derivatives of the functions that constitute the expression are computed multiple times. This problem has been well studied for the purpose of automatic differentiation algorithms (Neidinger, 1992, 1995; Griewank et al., 2000; Bettencourt et al., 2019). In fact, it has been shown that a set of recurrence relations can be defined on the coefficients of truncated Taylor series to efficiently compute higher-order derivatives (Neidinger, 2013). Departing from the elementary functions as represented by a computational graph, a complex Taylor series expansion can be performed without recalculating the same derivatives.

Translating this to the graph traversal algorithm outlined above requires us to first obtain all coefficients for a truncated Taylor series of the equation associated with each node in our computational graph. We can then use the algorithm defining the product of two truncated Taylor series (see Alg. (8) in Neidinger (2013) and Alg. 2) to propagate the coefficients of the series associated with each node. Note that adding two truncated Taylor series simply amounts to the pairwise addition of all corresponding coefficients. To obtain the higher-order derivatives needed for the first step, we could use the recurrence relations defined in Neidinger (2013). Note however, that the computational graph representation of the GF we have constructed is not at the level of the elementary functions. Because all equations associated with each of the nodes are well characterized, we can define a closed-form implementation of the derivatives with respect to the distinguished branch types. The equations all fall into one of two categories, depending on whether an inversion step was needed. Given a first-degree multivariate polynomial of the form $f(\mathbf{x}) = \sum c_i x_i + b$, non-inverted equations can be written as $1/f(\mathbf{x})$ (see Eq. 3). Inverted equations on the other hand have building blocks that take on the form of $e^{cf(\mathbf{x})}/(\prod f_i(\mathbf{x}))$. Using Alg. 2 and Eq. 4, we can come up with all partial derivatives for the inverted equations as well. With $s = \sum k_i$ and $\mathbf{x}$ representing the branch

type vector,

$$\frac{\partial f(\mathbf{x})^{-1}}{\partial^{k_i} x_i} = (-1)^s s! \frac{c_i^{k_i}}{f(\mathbf{x})^2} \tag{3}$$

$$\frac{\partial e^{cf(\mathbf{x})}}{\partial^{k_i} x_i} = c^s c_i^{k_i} e^{cf(\mathbf{x})} \tag{4}$$

Note that Alg. 2 contains an explicit `ADD` function. Care needs to be taken when summing (a subset of) the coefficients of a Taylor series: these will be both positive and negative, and as such, catastrophic cancellation might occur, leading to accuracy loss. To counteract this, I implemented the compensated summation algorithm of Ogita-Rump-Oishi (Ogita et al., 2005). The loss of precision is bounded by keeping track of small errors and adjusting the result using the error term. An alternative way of handling this would be to temporarily increase numeric precision at the crucial steps. Lastly, an advantage of using Taylor series coefficients rather than the corresponding derivatives is that the coefficients will always be smaller by a factor $(\sum \boldsymbol{k})!$, leading to less cumulative rounding error (Neidinger, 2013).

3 Results and Discussion

The work presented here constitutes a CAS-independent, open-source implementation of the GF approach. A general outline has been given on how the correspondence between the event state-space graph and the GF can be used to query the distribution of Laplace-transformed coalescence times efficiently. In particular, an algorithm has been laid out to calculate the probability of block-wise mutation configurations by propagating the calculation of series coefficients down the graph of ancestry states. The fact that this automation does not rely on a CAS and that it has been implemented in Python makes `agemo` an ideal back-end for likelihood calculations.

`agemo` relies on `numba` (Lam et al., 2015) just-in-time compilation to speed up the critical parts of the code. Compiling the code using `numba` has a few consequences. Firstly, compilation happens each time the code is run and will require a few seconds. This is generally not an issue given that, for a given model, many points in parameter space will be evaluated. Secondly, some numerical operations are implemented differently in `numba` than in `numpy`. In the case of summation this can lead to a loss of precision and has required the implementation of a compensated sum algorithm. A potentially faster solution would be to temporarily increase machine precision for the evaluation of particular sums. However, this is not possible using `numba` and would therefore require translating part of the code to `C`.

To evaluate performance, I calculated the bSFS for an isolation with migration (IM) model with 2 populations and 2 lineages in each population. Here two populations are descended from a

common ancestral population at some time in the past, and since then unidirectional gene flow is assumed to have happened at a constant rate (Nielsen and Wakeley, 2001). When discarding root and phase information, this leaves just 4 branch types. For each branch type I set $k_{max} = 4$, which means that the final result will contain 4^4 elements. This is the most complex model for which there exists a CAS-based implementation. Note that the original Mathematica implementation (Lohse et al., 2011) can only calculate the bSFS for a simplified IM model with two N_e parameters, meaning that at least two populations must have the same size. An implementation using open source CAS Sagemath (The Sage Developers, 0) takes about 75 s for a $3N_e$ model while `agemo` evaluates a single point in the parameter space in 181 ms. Increasing the sample size to 3 lineages in each population increases the number of nodes in the graph from 76 to 4449. The bSFS now contains 4^7 elements. Run time goes up accordingly to 134 s.

This IM model can be simplified to only include migration. We assume migration has been going on for an infinitely long time. Without any discrete events the graph is now maximally connected. For 2 lineages per population `agemo` takes 5 ms. Table 1 shows how performance scales with an increase in the number of samples per population.

Table 1: Run times migration-only model,
2 populations, unphased and unrooted branchtypes, $\mathbf{k}_{max} = 4$

samples per population	nodes in graph	non-zero entries in bSFS	size bSFS	time
2	30	112	256	5 ms
3	196	1408	16384	480 ms
4	1106	21952	16777216	52.3 s

I also benchmarked the evaluation time against the simulation-based approach as described in Beeravolu et al. (2018). Using `msprime` (Baumdicker et al., 2022), coalescent trees can be simulated under (almost) any demographic model. Without having to simulate mutations, we can calculate the probability of observing each mutation type. Given that mutations on each branch type happen independently, the probability of seeing mutation configuration $(k_1, k_2, ..., k_n)$ is given by the product of n probabilities as given by a Poisson distribution with rate $\theta/2 * t_i$. Here, t_i is the total branch length of branch type i and $\theta = 4N_e\mu$. When averaged across many replicates, the true value will be approximated. Note that particular entries of the bSFS might require fewer/more replicates to get at a good approximation than others (Becquet and Przeworski, 2007; Beeravolu et al., 2018). For the IM-model, with a 1000 replicates one can already approach the true bSFS quite well (Beeravolu et al., 2018). Scaling linearly with the number of replicates, this takes about 450 ms.

For 3 lineages per population run time goes up to 917 ms. With 4 lineages per population, there are 4^{12} entries in the bSFS, making a non-sparse approach prohibitively slow. Using simulations, run times are the same for both the IM and the migration model. Note that I aimed to make the comparison as fair as possible by optimizing the code and compiling the critical parts with `numba`. All calculations have been done on the same MacbookPro (2.2 GHz 6-Core Intel Core i7). Also note that the bottleneck of simulating the bSFS is not the actual simulation itself but the inherent combinatorial explosion of an ever increasing number of mutation configurations with increasing sample size. I attempted to alleviate this by means of sparse matrices, but this came with a speed cost.

This issue is inherent to the way the bSFS is defined and also applies to `agemo`. In part, this is solved by only calculating and storing the values associated with each unique mutation type that has a non-zero probability. However, computing the residual probabilities (observing more than k_{max_i} mutations along each branch type i) in the last step currently still requires us to populate an array of size $\prod k_{max_i} + 2$. Memory usage quickly becomes an issue here, and solving this requires a general sparse array implementation of the existing function. This suggests that the bSFS would benefit from a dedicated sparse-data structure. Ideally, this data structure would also enable us to take advantage of the dependency structure of all higher-order derivatives.

A last inherent limitation to the GF approach is that although we can include discrete events, retrieving the expression parametrized by the time to that discrete event requires us to take an inverse Laplace transform. Unfortunately, translating the mathematical description into a computational graph does not simplify this issue. As discussed, with the inclusion of discrete events the state space graph can no longer be translated into a computational graph without modification. A node must be added to the computational graph for each path leading to a discrete event, thus increasing the number of nodes and decreasing the connectivity of the graph, making a graph-based approach less efficient. `agemo` will therefore always do better in scenarios without discrete events (see Table 1).

As indicated in the Methods section, extending the GF approach to include new event types can easily be done. Because of its recursive nature, it only requires defining a function that describes the impact of the event on the extant lineages. All implemented events can then be combined to define a structured coalescent model. Note however that the current implementation only contains closed-form expressions to efficiently evaluate the GF associated with at most a single discrete event.

The general algorithm outlined here should enable users to query the Laplace transform to extract, for example, topology information, the SFS or the time to the first coalescence event. These functionalities have not been explicitly implemented yet. But can be computed using the described graph and associated expressions. Also, `agemo` was designed with extensibility in mind.

Future work on this library will enable a more diverse range of structured coalescent models as well as the ability to dynamically restrict the graph to those paths that are compatible with a specified topology.

The work described in this paper shows significant similarities with recent progress in phase-type theory (Røikjer et al., 2022). The authors present a general graph-based description of multivariate phase-type distributions and demonstrate the ability of their approach by calculating the SFS for an IM-type model sampling 7 lineages from each population. There are two main advantages of the phase-type theoretic approach. Firstly, incorporating discrete events does not require taking an inverse transform. Second, the paper contains algorithms translating cyclic graphs into an acyclic phase-type distribution, thus taking care of issues associated with, for example, bi-directional migration. On the other hand, `agemo` allows users to take full advantage of the information present within the joint distribution of coalescence times (e.g. bSFS). Including short-range linkage information comes at a computational cost, limiting the applicability to smaller sample sizes. However, previous work has demonstrated that this approach maximizes the information contained in small samples compared to relying on the SFS (Bunnefeld et al., 2015; Bisschop et al., 2021). More importantly however, both frameworks have (independently) combined the same two basic ingredients to efficiently describe coalescent models: a recursive state-space construction and a graph representation for fast evaluation of the represented distributions.

Software

The software is available at http://github.com/LohseLab/agemo. Documentation on how to use `agemo` can be found at https://agemo.readthedocs.io.

acknowledgements

This work was supported by an ERC starting grant (ModelGenomLand, 757648). I would like to thank Konrad Lohse and Derek Setter for constructive comments on the manuscript.

References

Baumdicker, F., Blows, M. W., Goldstein, D., Gower, G., Ragsdale, A. P., Tsambos, G., Zhu, S., Eldon, B., Ellerman, E. C., Galloway, J. G., Gladstein, A. L., Gorjanc, G., Guo, B., Jeffery, B., Kretzschumar, W. W., Lohse, K., Matschiner, M., Nelson, D., Pope, N. S., Quinto-Cortés, C. D., Rodrigues, M. F., Saunack, K., Sellinger, T., Thornton, K., van Kemenade, H., Wohns, A. W.,

Wong, Y., Gravel, S., Kern, A. D., Koskela, J., Ralph, P. L., and Kelleher, J. (2022). Efficient ancestry and mutation simulation with msprime 1.0. *Genetics*, 220(3).

Becquet, C. and Przeworski, M. (2007). A new approach to estimate parameters of speciation models with application to apes. *Genome Research*, 17(10):1505–1519.

Beeravolu, C. R., Hickerson, M. J., Frantz, L. A. F., and Lohse, K. (2018). ABLE: blockwise site frequency spectra for inferring complex population histories and recombination. *Genome Biology*, 19(1):145.

Bettencourt, J., Johnson, M. J., and Duvenaud, B. D. (2019). Taylor-Mode Automatic Differentiation for Higher-Order Derivatives in JAX.

Bisschop, G., Lohse, K., and Setter, D. (2021). Sweeps in time: Leveraging the joint distribution of branch lengths. *Genetics*, 219(2).

Bunnefeld, L., Frantz, L. A., and Lohse, K. (2015). Inferring bottlenecks from genome-wide samples of short sequence blocks. *Genetics*, 201(3):1157–1169.

Frantz, L. A., Madsen, O., Megens, H. J., Groenen, M. A., and Lohse, K. (2014). Testing models of speciation from genome sequences: Divergence and asymmetric admixture in Island South-East Asian Sus species during the Plio-Pleistocene climatic fluctuations. *Molecular Ecology*, 23(22):5566–5574.

Griewank, A., Utke, J., and Walther, A. (2000). Evaluating Higher Derivative Tensors by Forward Propagation of Univariate Taylor Series Source : Mathematics of Computation , Jul ., 2000 , Vol . 69 , No . 231 (Jul ., 2000), pp . 1117- Publi. *Mathematics of Computation*, 69(231):1117–1130.

Güne¸, A., Baydin, G., Pearlmutter, B. A., and Siskind, J. M. (2018). Automatic Differentiation in Machine Learning: a Survey. *Journal of Machine Learning Research*, 18:1–43.

Hobolth, A., Siri-Jégousse, A., and Bladt, M. (2019). Phase-type distributions in population genetics. *Theoretical Population Biology*, 127:16–32.

Hudson, R. R. (1983). Testing the constant-rate neutral allele model with protein sequence data. *Evolution*, 37(1):203–217.

Kingman, J. F. (1982). The coalescent. *Stochastic Processes and their Applications*, 13(3):235–248.

Lam, S. K., Pitrou, A., and Seibert, S. (2015). Numba. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC - LLVM '15*, pages 1–6, New York, New York, USA. ACM Press.

Lohse, K., Chmelik, M., Martin, S. H., and Barton, N. H. (2016). Efficient strategies for calculating blockwise likelihoods under the coalescent. *Genetics*, 202(2):775–786.

Lohse, K., Harrison, R. J., and Barton, N. H. (2011). A general method for calculating likelihoods under the coalescent process. *Genetics*, 189(3):977–987.

Neidinger, R. D. (1992). An Efficient Method for the Numerical Evaluation of Partial Derivatives of Arbitrary Order. *ACM Transactions on Mathematical Software (TOMS)*, 18(2):159–173.

Neidinger, R. D. (1995). Computing multivariable Taylor series to arbitrary order. In *Proceedings of the international conference on Applied programming languages - APL '95*, pages 134–144, New York, New York, USA. ACM Press.

Neidinger, R. D. (2013). Efficient recurrence relations for univariate and multivariate Taylor series coefficients. *Conference Publications*, pages 587–596.

Nielsen, R. and Wakeley, J. (2001). Distinguishing migration from isolation: A Markov chain Monte Carlo approach. *Genetics*, 158(2):885–896.

Ogita, T., Rump, S. M., and Oishi, S. (2005). Accurate sum and dot product. *SIAM Journal on Scientific Computing*, 26(6):1955–1988.

Råde, L. (1972). On the use of generating functions and laplace transforms in applied probability theory. *International Journal of Mathematical Education in Science and Technology*, 3(1):25–33.

Røikjer, T., Hobolth, A., and Munch, K. (2022). Graph-based algorithms for phase-type distributions. *bioRxiv preprint*.

Simonsen, K. L. and Churchill, G. A. (1997). A Markov chain model of coalescence with recombination. *Theoretical Population Biology*, 52(1):43–59.

Tajima, F. (1983). Evolutionary relationship of DNA sequences in finite populations. *Genetics*, 105(2):437–460.

The Sage Developers (0). Sagemath, the sage mathematics software system.