Chapter 8

Ancestral Population Genomics with Jocx, a Coalescent Hidden Markov Model

Jade Yu Cheng and Thomas Mailund

Abstract

Coalescence theory lets us probe the past demographics of present-day genetic samples and much information about the past can be gleaned from variation in rates of coalescence event as we trace genetic lineages back in time. Fewer and fewer lineages will remain, however, so there is a limit to how far back we can explore. Without recombination, we would not be able to explore ancient speciation events because of this—any meaningful species concept would require that individuals of one species are closer related than they are to individuals of another species, once speciation is complete. Recombination, however, opens a window to the deeper past. By scanning along a genomic alignment, we get a sequential variant of the coalescence process as it looked at the time of the speciation. This pattern of coalescence times is fixed at speciation time and does not erode with time; although accumulated mutations and genomic rearrangements will eventually hide the signal, it enables us to glance at events in the past that would not be observable without recombination. So-called coalescence hidden Markov models allow us to exploit this, and in this chapter, we present the tool Jocx that uses a framework of these models to infer demographic parameters in ancient speciation events.

Key words Genome analysis, Coalescence, Hidden Markov models, Population history inference

1 Introduction

Understanding how species form and diverge is a central topic of biology, and by observing emerging species today, we can understand many of the genetic and environmental processes involved. Through such observations, we can understand the underlying forces that drive speciation, but to understand how specific speciation events occurred in the past, and understand the specifics of how existing species formed, we must make the inference from the

The original version of this chapter was revised. The correction to this chapter is available at https://doi.org/10.1007/978-1-0716-0199-0_20

Electronic supplementary material: The online version of this chapter (https://doi.org/10.1007/978-1-0716-0199-0_8) contains supplementary material, which is available to authorized users.

Julien Y. Duthiel (ed.), *Statistical Population Genomics*, Methods in Molecular Biology, vol. 2090, https://doi.org/10.1007/978-1-0716-0199-0_8, © The Author(s) 2020, Corrected Publication 2021

signals these events have left behind. The speciation processes leave genetic “fossils” in the genome of the resulting species, and through what you might call genetic paleontology we can study past events from the signals they left behind.

The main objectives of the methods we describe in this chapter are to infer demographic parameters, Θ , given genetic data, D , through the model likelihood: $L(\Theta | D) = \Pr(D | \Theta)$. Here, we assume that Θ contains information such as effective population sizes, time points where population structure changes (populations split or admix), or migration rates between populations. We can connect data and demographics through coalescence theory [8]. This theory gives us a way to assign probability densities to genealogies; densities that depend on the demographic parameters, $f(G | \Theta)$. Then, if we know the underlying genealogy, we can assign probabilities to observed data using standard algorithms such as Felsenstein’s likelihood recursion [7] and get $\Pr(D | G, \Theta)$. Theoretically, we now simply need to integrate away the nuisance parameter G to get the desired likelihood

$$\mathcal{L}(\Theta | D) = \Pr(D | \Theta) = \int \Pr(D | G, \Theta) f(G | \Theta) dG. \quad (1)$$

In practice, however, the space of all possible genealogies prevents this beyond a small sample size of sequences and for any sizeable length of genetic material. Approximations are needed, and the *sequential Markov coalescent* (see Chapter 1) and *coalescent hidden Markov models* approximate the likelihood in two steps: they assume that sites are independent given the genealogy, i.e.,

$$\Pr(D | G, \Theta) \approx \prod_{i=1}^L \Pr(D_i | G_i, \Theta) \quad (2)$$

where L is the length of the sequence and D_i is the data and G_i the genealogy at site i , and assume that the dependency between genealogies is Markovian:

$$f(G | \Theta) \approx f(G_1 | \Theta) \prod_{i=2}^L f(G_i | G_{i-1}, \Theta). \quad (3)$$

Both assumptions are known to be invalid, but simulation studies indicate that this model captures most important summary statistics from the coalescent [17, 18] and that it can be used to accurately infer parameters in various demographic models [2, 14, 16]. Because of the form the likelihood now has,

$$f(D, G | \Theta) = f(G_1 | \Theta) \prod_{i=2}^L f(G_i | G_{i-1}, \Theta) \prod_{i=1}^L \Pr(D_i | G_i, \Theta), \quad (4)$$

which is the form of a *hidden Markov model*, we can compute the likelihood efficiently using the so-called *Forward* algorithm (see Chapter 3 in Durbin et al. [3]).

This efficiency has permitted us and others (see Chapters 7 and 10) to apply this approximation to the coalescence to infer demographic parameters on whole genome data [1, 9, 11–13, 19, 24, 25, 27] in addition to inferring recombination patterns [20, 21] and scanning for signs of selection [4, 22].

2 Software

We have created a theoretical framework for constructing coalescent hidden Markov models from demographic specifications [2, 14–16] and used it to implement various models in the software package Jcox, available at

<https://github.com/jade-cheng/Jcox.git>

Jcox handles the state space explosion problem of dealing with many sequences by creating hidden Markov models for all *pairs* of sequences and then combining these into a composite likelihood when estimating parameters. In brief, a full analysis looks something like the following. In the remainder of this chapter, we describe in detail how to apply Jcox to sequence data and how to interpret the results.

```
Jcox.py init . iso a.fasta b.fasta
```

It is very important that the verbatim (typewriter font) sections are left exactly as in the input. They contain ascii art that is output from our program.

```
Jcox.py run . iso nm 0.0001 1000 0.1
```

Jcox executes CoalHMMs by specifying a model and an optimizer. It uses sequence alignments in the format of “ziphmm” directories, which is also prepared by Jcox. The program prints to standard output the progression of the estimated parameters and the corresponding log likelihood. The source package contains a set of Python files, and it requires no installation.

2.1 Preparing Data

Jcox takes two or more aligned sequences as input; the number of sequence pairs depends on the CoalHMM model specified for a particular execution. We will discuss CoalHMM model specification later. For example, for inference in a two-population isolation scenario [14], we need a minimum of one pair of aligned sequences, with one sequence from each of the two populations. The input

should be FASTA files with names matching the names of the sequences we will use in the analysis, and since the sequences will be interpreted as aligned, they should all have the same lengths. The preprocessing will skip indels and handle all symbols except A, C, G, and T as the wildcard N.

In the following example, sequence a and sequence b form an alignment. Each sequence may have multiple data segments (e.g., contigs or chromosomes). In the example, we have two segments, 1 and 2. The names for these data segments need to be consistent between the two sequences. In the software we have the data-preparation step and model-inference step. In the data-preparation step, we supply Fasta sequences by providing their file names, e.g., `a.fasta` and `b.fasta`.

```
$ ls
a.fasta b.fasta

$ cat a.fasta | wc -c
1827

$ cat b.fasta | wc -c
1827

$ head *.fasta -n 7
==> a.fasta <==
>1
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaagaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaattaaaaaaaaaaaacaaaaaaaaaaaaa

>2
aaataaaaaaaaaaaaaaaaaaaaaaaaaagacaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaataaaaaaaaaaaaaaaaaaaaaaaaaaaa
==> b.fasta <==
>1
ataaaaaaaaaaaaaaaaaaaaacaaaaaagaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaagaaaaaacaaaaaaaaaaaaaaaaa

>2
aaaaaaaaaacaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaagaaaaaaaaaaaaaaaaaaaaataaaaaaaataaaaaa
```

We use the ZipHMM framework [26] to calculate likelihoods—in previous experiments we have found that ZipHMM gives us one or two orders of magnitude speedup in full genome analyses. To use ZipHMM in Joxc, we must preprocess the

sequence files. The preprocessing step is customized to each demographic model and is done using the

```
$ Jcox.py init
```

command. This command takes a variable number of arguments, depending on how many sequences are needed for the demographic model we intend to use. The first two arguments are the directory in which to put the preprocessed alignment and the demographic model to use. The sequences used for the alignment, the number of which depends on the model, must be provided as the remaining arguments. In the aforementioned two-population isolation scenario, the model `iso`, we need to process two aligned sequences, so the `init` command will take four arguments in total. To create a pairwise alignment for the isolation model, we would execute the following command:

```
$ Jcox.py init . iso a.fasta b.fasta
# Creating directory: ./ziphmm_iso_a_b
# creating uncompressed sequence file
# using output directory "./ziphmm_iso_a_b"
# parsing "a.fasta"
# parsing "b.fasta"
# comparing sequence "1"
# sequence length: 900
# creating "./ziphmm_iso_a_b/1.ziphmm"
# comparing sequence "2"
# sequence length: 900
# creating "./ziphmm_iso_a_b/2.ziphmm"
#Creating5-statealignmentindirectory:./ziphmm_iso_a_b/1.ziphmm
#Creating5-statealignmentindirectory:./ziphmm_iso_a_b/2.ziphmm
```

The result of the `init` command is the directory `ziphmm_iso_a_b` that contains information about the alignment of `a.fasta` and `b.fasta` in a format that ZipHMM can use to efficiently analyze the isolation model. Each Fasta data segment forms its own ZipHMM subdirectory. In the above example, we have two data segments, named 1 and 2, so we have two ZipHMM subdirectories.

```
$ ls
a.fasta  b.fasta  ziphmm_iso_a_b

$ find ziphmm_iso_a_b/
ziphmm_iso_a_b/
ziphmm_iso_a_b/1.ziphmm
```

```

ziphmm_iso_a_b/1.ziphmm/data_structure
ziphmm_iso_a_b/1.ziphmm/nStates2seq
ziphmm_iso_a_b/1.ziphmm/nStates2seq/5.seq
ziphmm_iso_a_b/1.ziphmm/original_sequence
ziphmm_iso_a_b/2.ziphmm
ziphmm_iso_a_b/2.ziphmm/nStates2seq
ziphmm_iso_a_b/2.ziphmm/nStates2seq/5.seq
ziphmm_iso_a_b/2.ziphmm/data_structure
ziphmm_iso_a_b/2.ziphmm/original_sequence

```

The exact structure of this directory is not important to how Jcox is used, but you must preprocess input sequences to match each demographic model you will analyze.

To see the list of all supported models, use the `--help` option. Here `iso` is the two-population two-sequence isolation scenario, shown below.

```

$ Jcox.py ----help
:
ISOLATION MODEL (iso)
  *
  / \ tau
A   B

3 params -> tau, coal_rate, recomb_rate
2 seqs    -> A, B
1 group   -> AB
:

```

For each model, the tool implements, the `--help` command will show an ASCII image of the model, annotated with the parameters of the model and with leaves labelled by populations. Below the image, the parameters are listed in the order they will be output when optimizing the model, followed by the sequences in the order they must be provided to the `init` command when creating the ZipHMM file. Finally, the help lists the pairs of sequences that will be used in the composite likelihood in the list of “groups.” When initializing a sequence alignment, you will get a ZipHMM directory per group.

The two-population isolation demographic model is symmetric, so the order of input Fasta sequences does not matter. This is not always the case. For example, in a three-population admixture model, shown below, the roles the populations take are different. Population C is admixed, and it is formed from ancestral siblings of the two source populations, A and B. The order of input Fasta sequences, therefore, needs to match.

In this model, we have five unknown time points and durations to be estimated, they are three-population isolation time (`iso_time`), two time points where the admixed population merges with each of the two source populations (`buddy23_time_1a`, `buddy23_time_2a`), and finally the duration before all populations find their common ancestry for the first population (`greedy1_time_1a`). The last unknown duration can be calculated: $\text{greedy1_time_2a} = \text{greedy1_time_1a} + \text{buddy23_time_1a} - \text{buddy23_time_2a}$.

```
$ Jcox.py ----help
:
THREE POP ADMIX 2 3 MODEL (admix23)
      *
      / \      greedy1_time_1a
buddy23_time_1a / \ \
      / \_/\      buddy23_time_2a
admix_prop    / <-| \ iso_time
      A      C      B

7 params -> iso_time, buddy23_time_1,
            buddy23_time_2, greedy1_time_1,
            coal_rate, recomb_rate, admix_prop
3 seqs    -> A, B, C
3 groups  -> AC, BC, AB
:
```

When executing the `init` command, the order of the Fasta sequences should match the order of species names in the help command:

```
$ ls
a1.fasta b1.fasta c1.fasta

$ Jcox.py init . admix23 a1.fasta b1.fasta c1.fasta
# Creating directory: ./ziphmm_admix23_a_c
# creating uncompressed sequence file
:

$ ls
a1.fasta b1.fasta c1.fasta
ziphmm_admix23_a_b ziphmm_admix23_a_c ziphmm_admix23_b_c
```

In the two examples above, each population contributes a single sequence to the CoalHMM's construction. Jcox also has models that support two sequences per population.

```

$ Jcox.py ----help
:
THREE POP ADMIX 2 3 MODEL 6 HMM (admixon23-6hmm)

      *
      / \      greedy1_time_1a
buddy23_time_1a / \ \
                / \_\ \      buddy23_time_2a
      admix_prop / <-| \ iso_time
                A1   C1   B1
                A2   C2   B2

7 params -> iso_time,      buddy23_time_1a,
            buddy23_time_2a, greedy1_time_1a,
            coal_rate, recomb_rate, admix_prop
6 seqs  -> A1, A2, B1, B2, C1, C2
6 groups -> A1C1, B1C1, A1B1, A1A2, B1B2, C1C2
:

```

In this example, we have the same admixture demographic model as before but with each population contributing two sequences to form six pairwise alignments, which are then used to construct six HMMs for the inference.

```

$ ls
a1.fasta a2.fasta b1.fasta b2.fasta c1.fasta c2.fasta

$ Jcox.py init . admixon23-6hmm a1.fasta a2.fasta \
$          b1.fasta b2.fasta \
$          c1.fasta c2.fasta
# Creating directory: ./ziphmm_admixon23-6hmm_a1_c1
:

$ ls
a1.fasta b1.fasta c1.fasta
a2.fasta b2.fasta c2.fasta
ziphmm_admixon23-6hmm_a1_a2  ziphmm_admixon23-6hmm_a1_c1  ziphmm_admixon23-6hmm_b1_c1
ziphmm_admixon23-6hmm_a1_b1  ziphmm_admixon23-6hmm_b1_b2  ziphmm_admixon23-6hmm_c1_c2

```

In the two-population isolation model, we have one demographic transition for a pair of samples. That is from a two-population isolation scenario (Fig. 1a) to a single ancestral population scenario (Fig. 1b). In the three-population admix model, we have three kinds of demographic transitions for a pair of samples. They are from a two-population duration (Fig. 1a) to a three-population duration (Fig. 2a), then to another two-population duration (Fig. 2b), finally to a single ancestral population (Fig. 1b). In the three-population duration, only two

Fig. 1 Demographic transition in the two-population isolation model for a pair of samples. Backwards in time, the state space transits from a two-population isolation scenario (a) to a single ancestral population scenario (b)

Fig. 2 Demographic transitions in the three-population admix model for a pair of samples. Backwards in time, the state space transits from a two-population isolation scenario (Fig. 1a) to a three-population scenario (a), then to another two-population scenario (b), and finally to a single ancestral population scenario (Fig. 1b)

populations are allowed to exchange lineages, shown in Fig. 2a as the second and third populations; hence we call this duration *buddy23*. In the second two-population duration, one of the two populations only accepts lineages because it is not involved in the admixture event at the previous state space transition. Since we have one population that never gives lineages during this time, we call this duration *greedy1*.

2.2 Inferring Parameters

To infer parameters, we maximize the model likelihood. Jcox implements three optimization subroutines, Nelder–Mead (NM), genetic algorithms (GA), and particle swarm optimization (PSO). After preparing the ZipHMM directories, user can run the CoalHMM to maximize the likelihood using one of these three algorithms using the `run` command.

```
$ Jcox.py run . iso nm 0.0001 1000 0.1
```

The first argument of this command, like for the `init` command, is the directory where the ZipHMM preprocessed data is found. The next argument is the demographic model. If we preprocessed the ZipHMM data with the `iso` model, we can use `iso` here to fit that model. The third argument is the optimization algorithm, one of `nm`, `ga`, and `pso`.

Following the optimizer option are the initialization values for the optimization. These arguments should match the number and order of parameters given by the `--help` command. In the `iso` model, for example, the parameters are these:

```
$ Jcox.py ----help
:
ISOLATION MODEL (iso)
*
/ \ tau
A   B

3 params -> tau, coal_rate, recomb_rate
2 seqs    -> A, B
1 group   -> AB
:
```

In this model, we infer three parameters: the population split time, `tau`, the coalescent rate, `coal_rate`, and the recombination rate, `recomb_rate`. In this model, populations are assumed to have the same coalescent rate, which is why there is only one parameter for this.

2.2.1 NM

NM was introduced by John Nelder and Roger Mead in 1965 [23] as a technique to minimize a function in a many-dimensional space. This method uses several algorithm coefficients to determine the amount of effect of possible actions.

```
$ Jcox.py run . iso nm 0.0001 1000 0.1
# algorithm          = _NMOptimiser
# timeout            = None
# max_executions     = 1
#
```

```
# 2017-10-11 11:29:08.069462
:
# execution state score param0 param1 param2
0 init -38.2023478685 0.000376954454165 7480.36836670 0.337649514816
1 fmin-in -40.5337262711 0.000385595244114 661.208520686 0.920281817958
1 fmin-cb -40.3804021200 0.000385595244114 694.268946721 0.920281817958
:
1 fmin-cb -37.8927822292 0.000695082517418 200504630.601 32081.6528250
Optimization terminated successfully.
    Current function value: 37.892782
    Iterations: 262
    Function evaluations: 533
1 fmin-out -37.8927822292 0.000695082517418 200504630.601 32081.652825
```

In the output of NM's execution, we have a final report of whether or not the execution was successful together with the optimal solution. It is possible for the optimizer to fail for various reasons, the number of parameters being a major cause of this. If the parameter space is too large, the Nelder–Mead optimizer often fail and one of the other optimizers will do better.

2.2.2 GA

GA was introduced by John Holland in the 1970s [10]. The idea is to encode each solution as a chromosome-like data structure and operate on them through actions analogous to genetic alterations, which usually involves selection, recombination, and mutation. For each type of alteration, various authors have developed different techniques.

```
$ Jcox.py run . iso ga 0.0001 1000 0.1
# algorithm          = _GAOptimiser
# timeout            = None
# elite_count        = 1
# population_size     = 50
# initialization      = UniformInitialisation
# selection           = TournamentSelection
# tournament_ratio    = 0.1
# selection_ratio     = 0.75
# mutation            = GaussianMutation
# point_mutation_ratio = 0.15
# mu                 = 0.0
# sigma              = 0.01
#
# 2017-10-23 10:31:32.821761
#
# param0 = (1.0000000000000016e-05, 0.001)
# param1 = (99.99999999999996, 10000.0)
# param2 = (0.009999999999999995, 1.0)
```

```

#
#
# POPULATION FOR GENERATION 1
# average_fitness = -5.32373335161
# min_fitness      = -10.7962322739
# max_fitness      = -0.613544122419
#
# gen idv    fitness      param0      param1      param2
1   1   -0.61354412  0.00002825  6305.95175380  0.04139445
1   2   -1.38710619  0.00004282  2182.61708962  0.03027973
1   3   -4.45085424  0.00001133  254.73764392  0.01081756
1   4   -9.37092993  0.00067074  116.84983427  0.13757425
1   5  -10.79623227  0.00071728  142.34535478  0.81564586
:
#
# POPULATION FOR GENERATION 2
# average_fitness = -5.83495296756
# min_fitness      = -10.5697879572
# max_fitness      = -0.613544122419
#
# gen idv    fitness      param0      param1      param2
2   1   -0.61354412  0.00002825  6305.95175380  0.04139445
2   2   -0.61382451  0.00002825  6305.95175380  0.13757425
2   3   -6.89850999  0.00002825  116.84983427  0.14110664
2   4  -10.47909826  0.00067074  145.01523656  0.81564586
2   5  -10.56978796  0.00067074  142.34535478  0.81564586
:
:

```

In the output of GA's execution, we have multiple generations of solutions, and multiple solutions per generation. Solutions in each generation are ordered by the fitness, i.e., best solution is at the top. The final solution is, therefore, the first solution in the last generation.

2.2.3 PSO

PSO was introduced by Eberhart and Kennedy in 1995 [5] as an optimization technique relying on stochastic processes, similar to GA. As its name implies, each individual solution mimics a particle in a swarm. Each particle holds a velocity and keeps track of the best positions it has experienced and best position the swarm has experienced. The former encapsulates the social influence, i.e., a force pulling towards the swarm's best. The latter encapsulates the cognitive influence, i.e., a force pulling towards the particle's best. Both forces act on the velocity and drive the particle through a hyperparameter space.

```

$ Jcox.py run . iso pso 0.0001 1000 0.1
# algorithm          = _PSOptimiser
# timeout            = None
# max_iterations      = 50
# particle_count      = 50
# max_initial_velocity = 0.02
# omega              = 0.9
# phi_particle        = 0.3
# phi_swarm           = 0.1
#
# 2017-10-23 10:32:29.123305
#
# param0 = (1.0000000000000016e-05, 0.001)
# param1 = (99.99999999999996, 10000.0)
# param2 = (0.009999999999999995, 1.0)
#
#
# PARTICLES FOR ITERATION 1
# swarm_fitness      = -0.832535308472
# best_average_fitness = -4.40169918533
# best_minimum_fitness = -9.77654933959
# best_maximum_fitness = -0.832535308472
# current_average_fitness = -4.40169918533
# current_minimum_fitness = -9.77654933959
# current_maximum_fitness = -0.832535308472
#
#
#                                best-   best-   best-   best-
# gen idv fitness param0  param1 param2 fitness param0  param1  param2
# 1  0  -0.83  0.000044  4619.31  0.20  -0.83  0.000044  4619.31  0.20
# 1  1  -0.86  0.000048  4502.80  0.26  -0.86  0.000048  4502.80  0.26
# 1  2  -0.89  0.000061  4669.48  0.58  -0.89  0.000061  4669.48  0.58
# 1  3  -1.10  0.000035  2970.77  0.31  -1.10  0.000035  2970.77  0.31
# 1  4  -1.46  0.000057  2148.93  0.15  -1.46  0.000057  2148.93  0.15
#
#
#                                best-   best-   best-   best-
# gen idv fitness param0  param1 param2 fitness param0  param1  param2
# 2  0  -0.81  0.000045  4854.87  0.25  -0.81  0.000045  4854.87  0.25

```

```

2  1  -0.82  0.000040  4622.38  0.21  -0.82  0.000040  4622.38  0.21
2  2  -0.91  0.000064  4599.97  0.59  -0.89  0.000061  4669.48  0.58
2  3  -1.12  0.000038  2917.40  0.29  -1.10  0.000035  2970.77  0.31
2  4  -1.39  0.000058  2308.29  0.14  -1.39  0.000058  2308.29  0.14
:
:
```

In the output of the PSO's execution, we have multiple generations and multiple particles (solutions) per generation. Each particle contains two sets of solutions, the current solution and the best solution that this particle has encountered throughout the PSO's execution. The latter is never worse than the former. Similar to GA, each generation is ordered by the particles' fitness. The final solution is, therefore, the second solution of the first particle in the last generation.

3 Simulation, Execution, and Result Summarization

In this section, we will use a simulation experiment to show how to perform a full analysis and extract the final solution. We will use the software `fastSIMCOAL2` [6] to simulate sequences under given demographic parameters, and we will use the two-population isolation model. All scripts and input files used here can be found in the Companion Material of this book.

We execute the following command to generate variable sites of a two-sequence alignment.

```
$ ./fsc251 -i input.par -n 1
```

The first argument points to a file containing the demographic parameters, shown below. The second argument specified the number of simulations to perform. We need only one pairwise alignment.

```

$ cat input.par
//Number of population samples (demes)
2
//Population effective sizes (number of genes)
12000
12000
//Sample sizes
1
1
//Growth rates: negative growth implies population expansion
0
0
//Number of migration matrices : 0 implies no migration between demes
0
```

```
//historical event: time, source, sink, migrants, ...
1 historical event
10000 0 1 1 2 0 0
//Number of independent loci [chromosome]
1 0
//Per chromosome: Number of linkage blocks
1
//per Block: data type, num loci, rec. rate ...
DNA 8000000 0.00000001 0.00000002 0.33
```

This simulation input file corresponds to the isolation model demography and model parameters. Our goal is to recover these parameters through CoalHMM model-based inference. The `historical event` line contains seven parameters. They are the time of the event (in generations), source population id, destination population id, the proportion of a population that migrated in this event, the new population size of the source population, the new growth rate, and the new migration matrix to use after this event. The last line contains five parameters. They are the type of data, the size of simulated sequence, the recombination rate, and the migration rate.

```
ISOLATION MODEL
*
/ \ Tau
A   B

Tau
= Sim_Time * Sim_Mutation_rate
= 10000 * 0.00000002
= 0.0002

Coal_rate
= 1 / (2 * Sim_Population_size * Sim_Mutation_rate)
= 1 / (2 * 12000 * 0.00000002)
= 2083

Recombination_rate
= Sim_Recombination_rate / Sim_Mutation_rate
= 0.00000001 / 0.00000002
= 0.5
```

The direct output from the simulation program is a directory of the same name as the input file, and in this case this directory contains three files:

```
$ ls input
input_1.arb input_1.simparam input_1_1.arp
```

The first file `input_1.arb` lists the file paths and names of the

generated alignments. The second file `input_1.simpam` records simulation conditions and serves as a log. The last file `input_1_1.arp` contains the variable sites of a sequence alignment. The content of this file is shown below.

```
$ less -S ./input/input_1_1.arp

#Arlequin input file written by the simulation program fastsimcoal2

[Profile]
    Title="A series of simulated samples"
    NbSamples=2

    GenotypicData=0
    GameticPhase=0
    RecessiveData=0
    DataType=DNA
    LocusSeparator=NONE
    MissingData='?'

[Data]
    [[Samples]]

#Number of independent chromosomes: 1
#Total number of polymorphic sites: 10960
# 10960 polymorphic positions on chromosome 1
#414, 1380, 2815, 3855, 4036, 5364, 5772, 5816, ...

#Total number of recombination events: 5381
#Positions of recombination events:
# Chromosome 1
#      3350, 8236, 9270, 10691, 11097, 12316, ...

        SampleName="Sample 1"
        SampleSize=1
        SampleData= {
1_1      1      CCTCGGTGTTGTCAAGGACAGTAACATG...
}

        SampleName="Sample 2"
        SampleSize=1
        SampleData= {
2_1      1      GAATAAAAAAACGTGAATGCAAGTACGAA...
}

[[Structure]]
```



```

StructureName="Simulated data"
NbGroups=1
Group={
  "Sample 1"
  "Sample 2"
}

```

We use the script `arlequin2fasta.py` to convert the Arlequin alignment into Fasta files. Since the Arlequin file contains only the variable sites, we need to specify the total length of the simulated sequence, which should match the simulation parameter in the input file `input.par`, e.g., 8,000,000 in this example.

```
$ ./arlequin2fasta.py input/input_1_1.arp 8000000
```

This creates two Fasta sequences for the pairwise alignment, and they are ready for Jcox's analysis.

```

$ ls
input  input.par

$ ./arlequin2fasta.py ./input/input_1_1.arp 8000000

$ ls
input  input.par
input_1_1-sample_1-1_1.fasta  input_1_1-sample_2-2_1.fasta

```

Analysis using Jcox follows a two-step procedure as described earlier. We first prepare the ZipHMM data directory using the `init` command and then infer parameters using the `run` command. The following commands conduct a full analysis, and it tests all three optimization options using ten independent executions per optimizer.

```

Jcox.py init . iso \
  ./input_1_1-sample_1-1_1.fasta \
  ./input_1_1-sample_2-2_1.fasta
Jcox.py run . iso pso 0.0001 1000 0.1 > pso-0.stdout
Jcox.py run . iso pso 0.0001 1000 0.1 > pso-1.stdout
:
Jcox.py run . iso pso 0.0001 1000 0.1 > pso-9.stdout
Jcox.py run . iso ga 0.0001 1000 0.1 > ga-0.stdout
Jcox.py run . iso ga 0.0001 1000 0.1 > ga-1.stdout
:
Jcox.py run . iso ga 0.0001 1000 0.1 > ga-9.stdout
Jcox.py run . iso nm 0.0001 1000 0.1 > nm-0.stdout
Jcox.py run . iso nm 0.0001 1000 0.1 > nm-1.stdout
:
Jcox.py run . iso nm 0.0001 1000 0.1 > nm-9.stdout

```

Upon completion, we receive ten sets of parameter estimates per optimization method. The format of the standard output, which contains the inference results, is different for each optimization method. We can use the following commands to summarize and plot the outcome. This plotting script is also provided in the Companion Material.

```
tail nm*.stdout -n 1 -q > nm-summary.txt
grep '500 1' ga-*.stdout > ga-summary.txt
grep '500 1' pso-*.stdout > pso-summary.txt
./box-plot-simple.py nm-summary.txt 3 nm-summary.png
./box-plot-simple.py ga-summary.txt 3 ga-summary.png
./box-plot-simple.py pso-summary.txt 3 pso-summary.png
```

The results are shown in Fig. 3. The first command collects the inference results from the NM optimizer. The last line in a NM execution's standard output contains the final estimates. The second two commands collect the inference results from the GA and PSO optimizers. The first solution/particle in the last generation/iteration, which is 500 in this experiment, contains the estimates.

```
$ head *summary.txt
```

```
==> ga-summary.txt <==
```

```
ga-0.stdout: 500 1 -81395.70680891 0.00011837 1815.42025279 0.42354064
ga-1.stdout: 500 1 -81470.10001761 0.00019243 1938.38996498 0.12963492
ga-2.stdout: 500 1 -81424.59984134 0.00021634 1846.60957248 0.19741876
ga-3.stdout: 500 1 -81430.96932585 0.00021685 1886.66976041 0.18309926
ga-4.stdout: 500 1 -81386.45366757 0.00019324 1916.03941578 0.32995308
ga-5.stdout: 500 1 -81463.45628041 0.00004345 1915.25301917 0.23921500
ga-6.stdout: 500 1 -81373.58453032 0.00018669 1968.26116983 0.52133035
ga-7.stdout: 500 1 -81504.94579193 0.00021242 1500.28846236 0.10292456
ga-8.stdout: 500 1 -81374.56618397 0.00019414 2046.25788612 0.52203350
ga-9.stdout: 500 1 -81433.41521075 0.00022051 1876.14477389 0.17886387
```

```
==> nm-summary.txt <==
```

```
1 fmin-out -81373.5832257 0.000186088241216 1966.58533828 0.52229387809
1 fmin-out -81373.5832257 0.000186088706436 1966.58675497 0.52229303544
1 fmin-out -81373.5832257 0.00018608870264 1966.58640056 0.522294017033
1 fmin-out -81373.5832257 0.000186088642201 1966.5864041 0.522295006576
1 fmin-out -81373.5832257 0.000186088168201 1966.58599993 0.522295026609
1 fmin-out -81373.5832257 0.00018608835674 1966.58624347 0.522297122163
1 fmin-out -81373.5832257 0.000186088509117 1966.58601275 0.52229560587
1 fmin-out -81373.5832257 0.000186088949749 1966.58644739 0.522293271654
1 fmin-out -81373.5832257 0.000186088354698 1966.58755713 0.522294573711
1 fmin-out -81373.5832257 0.000186088870812 1966.5853934 0.522294569147
```

(a) Nelder-Mead results.

(b) Genetic Algorithm results.

(c) Particle Swarm results.

Fig. 3 Summary of ten independent simulations and CoalHMM executions on the two-population isolation model using the three optimisation methods. The three columns show parameters speciation time, coalescence rate, and recombination rate, respectively. The simulated values of these parameters are 0.0002 , 2083 , and 0.5 . The number written below each box-plot is the median value of the estimates shown on the y-axis. This median can be used as a point estimate for the parameters

```
==> pso-summary.txt <==
pso-0.stdout: 500 1 -81373.583 0.000186 1966.585 0.522 -81373.583 0.000186
1966.585 0.522
pso-1.stdout: 500 1 -81373.583 0.000186 1966.586 0.522 -81373.583 0.000186
1966.585 0.522
pso-2.stdout: 500 1 -81373.583 0.000186 1966.586 0.522 -81373.583 0.000186
1966.586 0.522
pso-3.stdout: 500 1 -81373.583 0.000186 1966.586 0.522 -81373.583 0.000186
1966.586 0.522
pso-4.stdout: 500 1 -81373.583 0.000186 1966.585 0.522 -81373.583 0.000186
1966.585 0.522
pso-5.stdout: 500 1 -81373.583 0.000186 1966.585 0.522 -81373.583 0.000186
```

```

1966.585 0.522
pso-6.stdout: 500 1 -81373.583 0.000186 1966.586 0.522 -81373.583 0.000186
1966.585 0.522
pso-7.stdout: 500 1 -81373.583 0.000186 1966.586 0.522 -81373.583 0.000186
1966.586 0.522
pso-8.stdout: 500 1 -81373.583 0.000186 1966.585 0.522 -81373.583 0.000186
1966.585 0.522
pso-9.stdout: 500 1 -81373.583 0.000186 1966.586 0.522 -81373.583 0.000186
1966.586 0.522

```

The plotting script simply places these estimates in box plots. The first parameter indicates the summary file to plot. The second parameter indicates the number of parameters that this model has. For the two-population isolation model, we have three parameters. Each particle in PSO contains two sets of results, the local best and swarm best. The second set, swarm's best, should be used. The last parameter specifies the output file's name. At the bottom of each box plot we print the median value of the estimates.

The demographic parameters we use in this experiment are 0.0002, 2083, and 0.5. They are the split time of the two isolated populations, the coalescent rate, and the recombination rate, respectively. These values are roughly recovered by CoalHMM for all the optimizers.

In summary, the following commands conduct a full simulation and estimation data analysis, and it summarizes the final results by creating box plots and printing the median estimate for each parameter.

```

$ ./fsc251 -i input.par -n 1
$ ./arlequin2fasta.py input/input_1_1.arp 8000000
$ ./Jocx.py init . iso \
    ./input_1_1-sample_1-1_1.fasta \
    ./input_1_1-sample_2-2_1.fasta
$ ./Jocx.py run . iso pso 0.0001 1000 0.1 > pso-0.stdout
:
$ grep '500 1' pso-*.stdout > pso-summary.txt
$ ./box-plot-simple.py pso-summary.txt 3 pso-summary.png

```

The first command simulates a pairwise sequence alignment using the fastSIMCOAL2 program. The second command uses a custom script to convert the simulated alignment from the Arlequin format to the Fasta format. The third command prepares the ZipHMM directories using the Fasta sequences. The fourth command executes CoalHMM's model inference and dumps the output to a file. Potentially, multiple independent runs are dispatched and a HPC cluster is involved in this step. The fifth command obtains the inference results from the output file. The number 500 here is the maximum iteration count for this experiment, and

the number 1 indicates the first particle in the last iteration. Finally, the sixth command plots the parameters and presents the median estimates as the final results.

4 Conclusions

We have presented the Jocx tool for estimating parameters in ancestral population genomics. The tool uses a framework of pair-wise coalescent hidden Markov models combined in a composite likelihood to implement various demographic scenarios. A full list of available demography models are available through the tool's help command. Using a simple isolation model, we described an analysis pipeline based on simulating data and then analyzing it using the three different optimizers implement in Jocx. This pipeline is available in the Companion Material associated with this chapter, and serves as a good starting point for getting familiar with Jocx before moving to more involved models.

References

1. Abascal F, Corvelo A, Cruz F, Villanueva-Cañas JL, Vlasova A, Marcet-Houben M, Martínez-Cruz B, Cheng JY, Prieto P, Quesada V, Quilez J, Li G, García F, Rubio-Camarillo M, Frias L, Ribeca P, Capella-Gutiérrez S, Rodríguez JM, Cámara F, Lowy E, Cozzuto L, Erb I, Tress ML, Rodríguez-Ales JL, Ruiz-Orera J, Reverter F, Casas-Marce M, Soriano L, Arango JR, Derdak S, Galán B, Blanc J, Gut M, Lorente-Galdos B, Andrés-Nieto M, López-Otín C, Valencia A, Gut I, García JL, Guigó R, Murphy WJ, Ruiz-Herrera A, Marqués-Bonet T, Roma G, Notredame C, Mailund T, Albà MM, Gabaldón T, Alioto T, Godoy JA (2016) Extreme genomic erosion after recurrent demographic bottlenecks in the highly endangered Iberian lynx. *Genome Biol* 17(1):251
2. Cheng JY, Mailund T (2015) Ancestral population genomics using coalescence hidden Markov models and heuristic optimisation algorithms. *Comput Biol Chem* 57:80–92
3. Durbin R, Eddy SR, Krogh A, Mitchison G (1998) *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge University Press, Cambridge. <http://www.amazon.com/Biological-Sequence-Analysis-Probabilistic-Proteins/dp/0521629713>
4. Dutheil JY, Munch K, Nam K, Mailund T, Schierup MH (2015) Strong selective sweeps on the X chromosome in the human-chimpanzee ancestor explain its low divergence. *PLoS Genet* 11(8):e1005451
5. Eberhart R, Kennedy J (1995) A new optimizer using particle swarm theory. In: *Proceedings of the sixth international symposium on micro machine and human science*, 1995. MHS'95. IEEE, Piscataway, pp 39–43
6. Excoffier L, Dupanloup I, Huerta-Sánchez E, Sousa VC, Foll M (2013) Robust demographic inference from genomic and SNP data. *PLoS Genet* 9(10):e1003905
7. Felsenstein J (1981) Evolutionary trees from DNA sequences: a maximum likelihood approach. *J Mol Evol*. <https://doi.org/10.1007/BF01734359>
8. Hein J, Schierup M, Wiuf C (2004) *Gene genealogies, variation and evolution: a primer in coalescent theory*. Oxford University Press, New York
9. Hobolth A, Dutheil JY, Hawks J, Schierup MH, Mailund T (2011) Incomplete lineage sorting patterns among human, chimpanzee, and orangutan suggest recent orangutan speciation and widespread selection. *Genome Res* 21(3):349–356
10. Holland JH (1992) Genetic algorithms. *Sci Am* 267(1):66–73
11. Jónsson H, Schubert M, Seguin-Orlando A, Ginolhac A, Petersen L, Fumagalli M, Albrechtsen A, Petersen B, Korneliussen TS, Vilstrup JT, Lear T, Myka JL, Lundquist J, Miller DC, Alfarhan AH, Alquraishi SA, Al-Rasheid KAS, Stagegaard J, Strauss G, Bertelsen MF, Sicheritz-Ponten T, Antczak DF, Bailey E, Nielsen R, Willerslev E, Orlando L

- (2014) Speciation with gene flow in equids despite extensive chromosomal plasticity. *PNAS* 111(52):18655–18660
12. Li H, Durbin R (2011) Inference of human population history from individual whole-genome sequences. *Nature* 475(7357):493–496
 13. Locke DP, Hillier LW, Warren WC, Worley KC, Nazareth LV, Muzny DM, Yang SP, Wang Z, Chinwalla AT, Minx P, Mitreva M, Cook L, Delehaunty KD, Fronick C, Schmidt H, Fulton LA, Fulton RS, Nelson JO, Magrini V, Pohl C, Graves TA, Markovic C, Cree A, Dinh HH, Hume J, Kovar CL, Fowler GR, Lunter G, Meader S, Heger A, Ponting CP, Marquès-Bonet T, Alkan C, Chen L, Cheng Z, Kidd JM, Eichler EE, White S, Searle S, Vilella AJ, Chen Y, Flicek P, Ma J, Raney B, Suh B, Burhans R, Herrero J, Haussler D, Faria R, Fernando O, Darré F, Farré D, Gazave E, Oliva M, Navarro A, Roberto R, Capozzi O, Archidiacono N, Della Valle G, Purgato S, Rocchi M, Konkel MK, Walker JA, Ullmer B, Batzer MA, Smit AFA, Hubley R, Casola C, Schrider DR, Hahn MW, Quesada V, Puente XS, Ordoñez GR, López-Otín C, Vinar T, Brejova B, Ratan A, Harris RS, Miller W, Kosiol C, Lawson HA, Taliwal V, Martins AL, Siepel A, Roychoudhury A, Ma X, Degenhardt J, Bustamante CD, Gutenkunst RN, Mailund T, Dutheil JY, Hobolth A, Schierup MH, Ryder OA, Yoshinaga Y, de Jong PJ, Weinstock GM, Rogers J, Mardis ER, Gibbs RA, Wilson RK (2011) Comparative and demographic analysis of orang-utan genomes. *Nature* 469(7331):529–533
 14. Mailund T, Dutheil JY, Hobolth A, Lunter G, Schierup MH (2011) Estimating divergence time and ancestral effective population size of Bornean and Sumatran orangutan subspecies using a coalescent hidden Markov model. *PLoS Genet* 7(3):e1001319
 15. Mailund T, Halager AE, Westergaard M (2012) Using colored petri nets to construct coalescent hidden Markov models: automatic translation from demographic specifications to efficient inference methods. Springer, Berlin, pp 32–50
 16. Mailund T, Halager AE, Westergaard M, Dutheil JY, Munch K, Andersen LN, Lunter G, Prüfer K, Scally A, Hobolth A, Schierup MH (2012) A new isolation with migration model along complete genomes infers very different divergence processes among closely related great ape species. *PLoS Genet* 8(12):e1003125
 17. Marjoram P, Wall JD (2006) Fast “coalescent” simulation. *BMC Genet* 7(1):16
 18. McVean GAT, Cardin NJ (2005) Approximating the coalescent with recombination. *Philos Trans R Soc Lond B Biol Sci* 360(1459):1387–1393
 19. Miller W, Schuster SC, Welch AJ, Ratan A, Bedoya-Reina OC, Zhao F, Kim HL, Burhans RC, Drautz DI, Wittekindt NE, Tomsho LP, Ibarra-Laclette E, Herrera-Estrella L, Peacock E, Farley S, Sage GK, Rode K, Obbard M, Montiel R, Bachmann L, Ingólfsson O, Aars J, Mailund T, Wiig O, Talbot SL, Lindqvist C (2012) Polar and brown bear genomes reveal ancient admixture and demographic footprints of past climate change. *Proc Natl Acad Sci U S A* 109(36):E2382–E2390
 20. Munch K, Mailund T, Dutheil JY, Schierup MH (2014) A fine-scale recombination map of the human-chimpanzee ancestor reveals faster change in humans than in chimpanzees and a strong impact of GC-biased gene conversion. *Genome Res* 24(3):467–474
 21. Munch K, Schierup MH, Mailund T (2014) Unraveling recombination rate evolution using ancestral recombination maps. *BioEssays* 36(9):892–900
 22. Munch K, Nam K, Schierup MH, Mailund T (2016) Selective sweeps across twenty millions years of primate evolution. *Mol Biol Evol* 33(12):3065–3074
 23. Nelder JA, Mead R (1965) A simplex method for function minimization. *Comput J* 7(4):308–313
 24. Prado-Martinez J, Sudmant PH, Kidd JM, Li H, Kelley JL, Lorente-Galdos B, Veeramah KR, Woerner AE, O'Connor TD, Santpere G, Cagan A, Theunert C, Casals F, Laayouni H, Munch K, Hobolth A, Halager AE, Malig M, Hernandez-Rodriguez J, Hernandez-Herraez I, Prüfer K, Pybus M, Johnstone L, Lachmann M, Alkan C, Twigg D, Petit N, Baker C, Hormozdiari F, Fernández-Callejo M, Dabad M, Wilson ML, Stevison L, Campubí C, Carvalho T, Ruiz-Herrera A, Vives L, Mele M, Abello T, Kondova I, Bon-trop RE, Pusey A, Lankester F, Kiyang JA, Bergl RA, Lonsdorf E, Myers S, Ventura M, Gagneux P, Comas D, Siegmund H, Blanc J, Agueda-Calpena L, Gut M, Fulton L, Tishkoff SA, Mullikin JC, Wilson RK, Gut IG, Gonder MK, Ryder OA, Hahn BH, Navarro A, Akey JM, Bertranpetit J, Reich D, Mailund T, Schierup MH, Hvilsom C, Andrés AM, Wall JD, Bustamante CD, Hammer MF, Eichler EE, Marquès-Bonet T (2013) Great ape genetic

- diversity and population history. *Nature* 499 (7459):471–475
25. Prüfer K, Munch K, Hellmann I, Akagi K, Miller JR, Walenz B, Koren S, Sutton G, Kodira C, Winer R, Knight JR, Mullikin JC, Meader SJ, Ponting CP, Lunter G, Higashino S, Hobolth A, Dutheil J, Karakoç E, Alkan C, Sajjadian S, Catacchio CR, Ventura M, Marquès-Bonet T, Eichler EE, André C, Atencia R, Mugisha L, Junhold J, Patterson N, Siebauer M, Good JM, Fischer A, Ptak SE, Lachmann M, Symer DE, Mailund T, Schierup MH, Andrés AM, Kelso J, Pääbo S (2012) The bonobo genome compared with the chimpanzee and human genomes. *Nature* 486(7404):527–531
 26. Sand A, Kristiansen M, Pedersen CNS, Mailund T (2013) zipHMMLib: a highly optimised HMM library exploiting repetitions in the input to speed up the forward algorithm. *BMC Bioinf* 14(1):339
 27. Scally A, Dutheil JY, Hillier LW, Jordan GE, Goodhead I, Herrero J, Hobolth A, Lappalainen T, Mailund T, Marquès-Bonet T, McCarthy S, Montgomery SH, Schwalie PC, Tang YA, Ward MC, Xue Y, Yngvadottir B, Alkan C, Andersen LN, Ayub Q, Ball EV, Beal K, Bradley BJ, Chen Y, Clee CM, Fitzgerald S, Graves TA, Gu Y, Heath P, Heger A, Karakoç E, Kolb-Kokocinski A, Laird GK, Lunter G, Meader S, Mort M, Mullikin JC, Munch K, O'Connor TD, Phillips AD, Prado-Martinez J, Rogers AS, Sajjadian S, Schmidt D, Shaw K, Simpson JT, Stenson PD, Turner DJ, Vigilant L, Vilella AJ, Whitener W, Zhu B, Cooper DN, de Jong P, Dermitzakis ET, Eichler EE, Flicek P, Goldman N, Mundy NI, Ning Z, Odom DT, Ponting CP, Quail MA, Ryder OA, Searle SM, Warren WC, Wilson RK, Schierup MH, Rogers J, Tyler-Smith C, Durbin R (2012) Insights into hominid evolution from the gorilla genome sequence. *Nature* 483 (7388):169–175

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.