The Jaatha HowTo

Lisha Mathew, Paul R. Staab and Dirk Metzler

Version 2.0.1

1 Introduction

Jaatha is a fast composite likelihood method to estimate model parameters of the evolutionary history of (at the moment) two related species or populations. To do so, it uses SNP data from multiple individuals from both species and – optionally but highly recommended – one or more outgroup sequences. This HowTo describes the method and gives an example of using its implementation as an `R` package `jaatha`.

The package itself can be obtained from CRAN using

```
> install.packages('jaatha')
```

or downloaded from http://evol.bio.lmu.de/_statgen/software/jaatha. Jaatha runs on all operating systems supported by `R` – namely Windows, OS X (Mac) and Linux – with the following restriction: On Windows the parallelization and finite sites simulation using seq-gen is currently not working.

A more detailed description of the algorithm can be found in Mathew et˜al. [2012]. Further information about the `R` functions used in this document can be obtained by calling `help()` with the functions name as argument.

Please cite the above paper when using Jaatha in a publication.

2 A demographic model

Before we can apply Jaatha to estimate parameters, we first need to create a model of the evolutionary history of the two species. Jaatha cannot account for the effects of selection, hence we assume a neutral evolution. It can estimate the effects that either "demographic" events – like an expansion of the size of one population or migration between the two populations – or molecular events – like mutation or recombination – have on the genome of the populations. To emphasize that we can not include selection, we refer to a scenario of the evolution of the two species under such events as a "demographic model".

For now, assume that we know that our two species are closely related; hence they must have separated at a certain time in the past. There may still be gene flow ongoing between them to which we will refer to as migration from one population into the other. Hence, we could propose the simple demographic model described in Figure˜1.

To specify that model in `R`, we first need to load `jaatha`.

```
> library(jaatha)
```

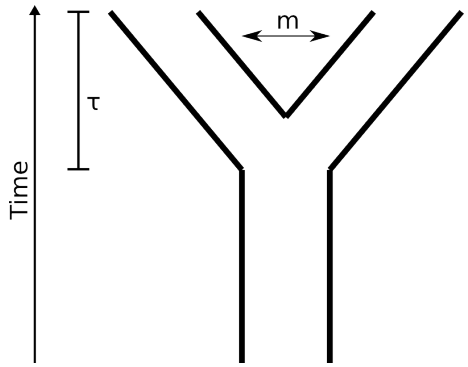

Figure 1: A simple demographic model: The ancestral population splits into two populations τ time units ago, and afterwards individuals migrated from one population to the other with a migration rate M. Mutations are occurring with rate θ and recombination with a known rate ρ.

We can now create an 'empty' demographic model `dm` using the `dm.createDemographicModel()` function:

```
> dm <- dm.createDemographicModel(sample.sizes=c(12,14),
+                                 loci.num=50,
+                                 seq.length=1000)
```

The parameter `sample.sizes` here corresponds to the number of individuals we have sampled from the first population and second population respectively. The second argument `loci.num` states that we are using data from 70 loci while `seq.length` gives the (average) length of each loci[1].

We can now successively add the other assumptions of our model:

```
> dm <- dm.addSpeciationEvent(dm, .1, 5, new.time.point.name='tau')
> dm <- dm.addSymmetricMigration(dm, .01, 5, new.par.name='M')
> dm <- dm.addMutation(dm, 1, 20, new.par.name='theta')
> dm <- dm.addRecombination(dm, fixed=20)
```

The first parameter is the demographic model to which we want to add an assumption/feature. The two following numbers represent the range for the corresponding parameter. The lower border has to be strictly greater the zero, as we are using a logarithmic transformation of the parameter space. The parameters are scaled as in the popular simulation program `ms` [Hudson, 2002] that we use for simulations:

- The parameter for the *speciation* event is the split time τ, which states how many generations ago the split of the population has occurred. As usual in population genetics, it is measured in units of $4N_1$ generations ago, where N_1 is the (diploid) effective population size of the first population.

[1]This is only used when a finite sites model is assumed or if intra-locus recombination is included.

- The parameter for the (symmetric) *migration* is the scaled migration rate M, which is given by $M = 4N_1m$, where m is the fraction of individuals of each population which are replaced by immigrants from the other population each generation.
- The *mutation* parameter θ is $4N_1$ times the neutral mutation rate per locus.
- Finally, the *recombination* parameter ρ is $4N_1$ times the probability of recombination between the ends of the locus per generation.

Keep in mind that a 'good' model – which is one that approximates the real demographic history but is also as simple as possible – is crucial for obtaining meaningful estimates in the end. Jaatha will always try to find the parameters that make the model fit best to your data. If the model does not fit to the data at all, Jaatha will still return estimates, but they will not be meaningful.

3 Theoretical Background

It is important to understand the key concepts behind Jaatha before we can apply it. Like many estimation methods that rely on simulations, Jaatha tries to find the parameters that best fits [2] to your data by simulating artificial data for many different parameter combinations. It uses a learning algorithm to determine how the different parameter values influence the simulated data and uses that knowledge to find the best parameter combination for your data.

You can imagine Jaatha as a method that runs through the parameter space – the space of all possible parameter combinations, in our example a cube with borders from 0.1 to 5, 0.01 to 5 and 1 to 20 – simulating in a small part of the parameter space around the current position (we call this area a *block*). It then searches the new maximum of the current blocks and moves to it, builds a new block around it and so on. The search finally stops when the likelihood cannot be improved anymore or a maximal number of steps has been reached.

To compare the simulated data to the real one, Jaatha uses *summary statistics* of the data. As default, it calculates the Joint Site Frequency Spectrum (JSFS) of the data and further summarizes it by evaluating different sums over the JSFS. Please refer to Naduvilezhath et˜al. [2011] for a detailed description.

4 Importing Your Data

To run Jaatha, you need to calculate the JSFS of your data. To do so, you can use the function `calculateJsfs()`. This function accepts data imported with the `read.dna()` function from the package *ape*. It assumes that you provide a joined, aligned data set with multiple samples from two populations and – optional but highly recommended – one or more outgroup sequences. Additionally, you must provide the numbers of the sequences in the dataset that belong to population one, population two, and the outgroup, respectively.

Please consult the documentation of *ape* in order to get more information about `read.dna()`. For example, the import of the data could look like this:

[2] for Jaatha, the 'best' parameter combination is the one with the highest composite likelihood

```
> library(ape)
> # The path to the data
> sample.file <- system.file('example_fasta_files/sample.fasta',
+                            package='jaatha')
> # Reading the data
> sample.data <- read.dna(sample.file, format='fasta',
+                         as.character=TRUE)
> # Calculating the JSFS
> sample.jsfs <- calculateJsfs(sample.data,
+                              pop1.rows=3:7,
+                              pop2.rows=8:12,
+                              outgroup.row=1:2)
> sample.jsfs
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    0    0    0    0    0
[2,]    1    1    1    0    0    0
[3,]    2    0    1    0    0    0
[4,]    1    1    0    2    0    1
[5,]    1    0    0    0    3    0
[6,]    1    0    0    0    0    0
```

For the purpose of this HowTo, we will use a simulated JSFS, for which we know the real parameters:

```
> # Real parameters: M = 1, tau = 1 and theta = 10
> real.pars <- c(1,1,10)
> # Simulate a JSFS with this parameters
> jsfs <- dm.simSumStats(dm, real.pars)
> # Print the upper left part of the JSFS
> jsfs[1:10, 1:10]
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    0  530  186  146  101   91   63   28   23    24
 [2,]  558   55   25   21   14   21   18   12    5     4
 [3,]  245   32   14   21    9   13   14    6   11    14
 [4,]  111   35   17    9   13    7   13    2   13     8
 [5,]  101   29   16    6    8    9    8    5    3     4
 [6,]   68   15   18    6   16    3    5   10    2     2
 [7,]   39    8    8    4    5    8   10    3    3     2
 [8,]   18    7    4    4    3    5    1    2    3     0
 [9,]   25   10    5    3    5    4    2    4    3     3
[10,]   14   13    5    8    0    3    3    2    5     1
```

5 Running Jaatha

Jaatha is divided into two parts. First we find good starting positions by simulating very coarsely across the entire parameter space. We call this part *initial search*. Afterwards a more thorough *refined search* is performed starting from

the best positions of the first step. Before starting the search, we need to set some options like our demographic model, the summary statistics of the real data, and a seed to ensure repoducibilty:

```
> jaatha <- Jaatha.initialize(dm, jsfs=jsfs, seed=12345)
```

For more options refer to `?Jaatha.initialize` or the Jaatha manual.

5.1 The Initial Search

For the initial search, we divide the parameter space into equally-sized blocks by dividing each of the n parameters ranges into `blocks.per.par` intervals such that we obtain $(\texttt{blocks.per.par})^n$ blocks. Within each block we simulate `sim` data sets with – on a logarithmic scale – uniformly drawn parameter values within each block. To ensure a better sampling of the edges, we simulate in addition data sets for all corner points of each parameter block.

For these data sets we then fit the GLMs and estimate the parameter combination with the maximal score[3]. Each of the blocks provides a single best parameter combination.

In R, the initial search is performed with the command

```
> jaatha <- Jaatha.initialSearch(jaatha,
+                                sim=100,
+                                blocks.per.par=3)

externalTheta set to TRUE for initial search.
*** Starting position is being determined ***
Creating 9 initial blocks ...
*** Block 1  (lowerB: 0.1 0.01  upperB: 0.368 0.079 )
Best parameters:  0.191 0.079 1 | Score: 16658.14

*** Block 2  (lowerB: 0.1 0.079  upperB: 0.368 0.63 )
Best parameters:  0.238 0.36 1 | Score: 16702

*** Block 3  (lowerB: 0.1 0.63  upperB: 0.368 5 )
Best parameters:  0.292 0.63 1 | Score: 16732.24

*** Block 4  (lowerB: 0.368 0.01  upperB: 1.357 0.079 )
Best parameters:  0.368 0.079 1 | Score: 16534.73

*** Block 5  (lowerB: 0.368 0.079  upperB: 1.357 0.63 )
Best parameters:  0.43 0.63 1 | Score: 16787.08

*** Block 6  (lowerB: 0.368 0.63  upperB: 1.357 5 )
Best parameters:  0.759 0.835 1 | Score: 16789.69

*** Block 7  (lowerB: 1.357 0.01  upperB: 5 0.079 )
Best parameters:  1.357 0.079 1 | Score: 15018.95
```

[3]In this phase, Jaatha uses a score instead of the likelihood for computational reasons. The likelihood is proportional to exp(score). The higher the score, the higher the likelihood.

```
*** Block 8  (lowerB: 1.357 0.079  upperB: 5 0.63 )
Best parameters:  1.501 0.63 1 | Score: 16712.76

*** Block 9  (lowerB: 1.357 0.63  upperB: 5 5 )
Best parameters:  1.357 0.826 1 | Score: 16805.35

        score   tau     M theta
 [1,] 16805.35 1.357 0.826    20
 [2,] 16789.69 0.759 0.835    20
 [3,] 16787.08 0.430 0.630    20
 [4,] 16732.24 0.292 0.630    20
 [5,] 16712.76 1.501 0.630    20
 [6,] 16702.00 0.238 0.360    20
 [7,] 16658.14 0.191 0.079    20
 [8,] 16534.73 0.368 0.079    20
 [9,] 15018.95 1.357 0.079    20
```

To visualise the estimates for good stating positions sorted by score, type:

```
> Jaatha.getStartingPoints(jaatha)

        score   tau     M theta
 [1,] 16805.35 1.357 0.826    20
 [2,] 16789.69 0.759 0.835    20
 [3,] 16787.08 0.430 0.630    20
 [4,] 16732.24 0.292 0.630    20
 [5,] 16712.76 1.501 0.630    20
 [6,] 16702.00 0.238 0.360    20
 [7,] 16658.14 0.191 0.079    20
 [8,] 16534.73 0.368 0.079    20
 [9,] 15018.95 1.357 0.079    20
```

Here, there is a big reduction in the scores after the first seven blocks, and a smaller one after the first two. This is suggesting that we either use the first two or the first seven blocks as starting positions for the refined search, depending on how much time we want to spend. For now, we will just use the first two points.

5.2 The Refined Search

Now we can conduct the more thorough refined search described above to improve the likelihood approximations.

```
> jaatha <- Jaatha.refinedSearch(jaatha, best.start.pos=2, sim=100)

*** Search with starting Point in Block 1 of 2 ****
-----------------
Step No 1
Number of blocks kept: 0  / Number of blocks deleted: 0
Best parameters:  1.116 0.899 14.823 | Score: 16437.54
```

```
-----------------
Step No 2
Number of blocks kept: 1  / Number of blocks deleted: 0
Best parameters:  0.918 0.898 12.761 | Score: 16701.71

-----------------
Step No 3
Number of blocks kept: 1  / Number of blocks deleted: 1
Best parameters:  0.797 0.858 10.986 | Score: 16798.07

-----------------
Step No 4
Number of blocks kept: 1  / Number of blocks deleted: 1
Best parameters:  0.97 0.847 10.088 | Score: 16808.11

-----------------
Step No 5
Number of blocks kept: 1  / Number of blocks deleted: 1
Best parameters:  1.179 0.838 9.599 | Score: 16815.65

-----------------
Step No 6
Number of blocks kept: 1  / Number of blocks deleted: 1
Best parameters:  1.282 0.836 9.454 | Score: 16816.43

-----------------
Step No 7
Number of blocks kept: 1  / Number of blocks deleted: 1
Best parameters:  1.247 0.853 9.521 | Score: 16815.16

-----------------
Step No 8
Number of blocks kept: 2  / Number of blocks deleted: 0
Best parameters:  1.212 0.862 9.568 | Score: 16815.54
No sigificant score changes in the last 1 Step(s)

-----------------
Step No 9
Number of blocks kept: 3  / Number of blocks deleted: 0
Best parameters:  1.215 0.86 9.537 | Score: 16815.57
No sigificant score changes in the last 2 Step(s)

-----------------
Step No 10
Number of blocks kept: 4  / Number of blocks deleted: 0
Best parameters:  1.236 0.853 9.51 | Score: 16815.45
No sigificant score changes in the last 3 Step(s)

-----------------
```

```
Step No 11
Number of blocks kept: 5  / Number of blocks deleted: 0
Best parameters:  1.222 0.853 9.539 | Score: 16815.35
No sigificant score changes in the last 4 Step(s)

-----------------
Step No 12
Number of blocks kept: 6  / Number of blocks deleted: 0
Best parameters:  1.202 0.854 9.573 | Score: 16815.43
No sigificant score changes in the last 5 Step(s)

*** Finished search ***
Score has not change much in the last 5 steps.
Seems we have converged.

Calulating log-composite-likelihoods for best estimates:
* Parameter combination 1 of 10
* Parameter combination 2 of 10
* Parameter combination 3 of 10
* Parameter combination 4 of 10
* Parameter combination 5 of 10
* Parameter combination 6 of 10
* Parameter combination 7 of 10
* Parameter combination 8 of 10
* Parameter combination 9 of 10
* Parameter combination 10 of 10

*** Search with starting Point in Block 2 of 2 ****
-----------------
Step No 1
Number of blocks kept: 0  / Number of blocks deleted: 0
Best parameters:  0.624 0.842 14.823 | Score: 16618.06

-----------------
Step No 2
Number of blocks kept: 1  / Number of blocks deleted: 0
Best parameters:  0.513 0.823 12.761 | Score: 16758.76

-----------------
Step No 3
Number of blocks kept: 1  / Number of blocks deleted: 1
Best parameters:  0.624 0.815 10.986 | Score: 16792.55

-----------------
Step No 4
Number of blocks kept: 1  / Number of blocks deleted: 1
Best parameters:  0.759 0.844 10.491 | Score: 16802.23

-----------------
Step No 5
```

```
Number of blocks kept: 1  / Number of blocks deleted: 1
Best parameters:  0.923 0.837 10.124 | Score: 16807.31

-----------------
Step No 6
Number of blocks kept: 1  / Number of blocks deleted: 1
Best parameters:  1.122 0.854 9.666 | Score: 16814.14

-----------------
Step No 7
Number of blocks kept: 1  / Number of blocks deleted: 1
Best parameters:  1.227 0.852 9.53 | Score: 16815.27

-----------------
Step No 8
Number of blocks kept: 1  / Number of blocks deleted: 1
Best parameters:  1.211 0.847 9.616 | Score: 16814.37
No sigificant score changes in the last 1 Step(s)

-----------------
Step No 9
Number of blocks kept: 2  / Number of blocks deleted: 0
Best parameters:  1.196 0.844 9.619 | Score: 16815.33
No sigificant score changes in the last 2 Step(s)

-----------------
Step No 10
Number of blocks kept: 3  / Number of blocks deleted: 0
Best parameters:  1.242 0.843 9.528 | Score: 16816.04
No sigificant score changes in the last 3 Step(s)

-----------------
Step No 11
Number of blocks kept: 4  / Number of blocks deleted: 0
Best parameters:  1.246 0.838 9.51 | Score: 16815.3
No sigificant score changes in the last 4 Step(s)

-----------------
Step No 12
Number of blocks kept: 5  / Number of blocks deleted: 0
Best parameters:  1.236 0.837 9.53 | Score: 16815.43
No sigificant score changes in the last 5 Step(s)

*** Finished search ***
Score has not change much in the last 5 steps.
Seems we have converged.

Calulating log-composite-likelihoods for best estimates:
* Parameter combination 1 of 10
* Parameter combination 2 of 10
```

```
* Parameter combination 3 of 10
* Parameter combination 4 of 10
* Parameter combination 5 of 10
* Parameter combination 6 of 10
* Parameter combination 7 of 10
* Parameter combination 8 of 10
* Parameter combination 9 of 10
* Parameter combination 10 of 10


Best log-composite-likelihood values are:
     log.cl block      tau         M    theta
2 -79.65989     2 1.246475 0.8383166 9.509682
7 -79.73547     2 1.227155 0.8516747 9.530497
8 -79.79661     2 1.211300 0.8471184 9.616227
3 -79.80292     1 1.202239 0.8537882 9.572698
3 -80.07669     2 1.235692 0.8367034 9.530073
```

Hence we perform two independent searches, starting in the `best.start.pos` best starting positions we choose before and according to the general options we choose during initialization, which are stored in the `jaatha` object. In each step, we build a block with `half.block.size` in each direction (on a logarithmic scale) and perform simulations for `sim` random parameter combinations within this block (plus one for very corner). We use this information to estimate the composite maximum likelihood parameters within this block and take this value as new starting position for the next step.

The algorithm stops when the score has not changed more than `epsilon` for five consecutive steps or step `max.steps` is reached. To avoid getting stuck in local maxima, the `weight` option decreases the weight of simulations of previous blocks.

Finally the log composite likelihoods for the best ten parameter combinations are approximated using `sim.final` simulations. These are values printed at the end of the search. This matrix can also be accessed via

```
> likelihoods <- Jaatha.getLikelihoods(jaatha)
> print(likelihoods[1:3, ])

     log.cl block      tau         M    theta
2 -79.65989     2 1.246475 0.8383166 9.509682
7 -79.73547     2 1.227155 0.8516747 9.530497
8 -79.79661     2 1.211300 0.8471184 9.616227
```

6 Parallelization

On Linux and OS X, Jaatha can distribute the simulations on multiple CPU cores. To use this feature, set the `cores` option during initialization. The value of the option specifies the number of cores you want to use. As default, Jaatha will execute 'packages' of 10 simulation on a core in a row to reduce the overhead created by inter-process communication. This is fine for demographic models that can be quickly simulated. However, if you have simulations that takes

multiple seconds and/or many cores available, you may want smaller packages. This can be archived by setting `sim.package.size = 1`.

The use of the `cores` option does affect the seeding system. A run without `cores` will give different results as the identical run with the option set at a value greater than one, even when using the same seed. However, if you use `cores`, the actual number of cores you use does not change the results.

7 Finite sites models

As described in Mathew et˜al. [2012], you can use finite sites mutation models in Jaatha. However, this currently only works on Linux and OS X and requires that seq-gen [Rambaut and Grassly, 1997] is installed on your system. On Debian/Ubuntu can install it running `apt-get install seq-gen` as root. Otherwise download the current version from http://tree.bio.ed.ac.uk/software/seqgen and compile it according to the instruction within the package. Jaatha will search for the seq-gen executable using your PATH variable. If you get an error that it is not able to find it, you can specify the path to the executable using the `Jaatha.setSeqgenExecutable` function.

To create a finite sites model, you must specify a mutation model using the `dm.setMutationModel` function and add an outgroup to your model using `dm.addOutgroup`. Optionally, you can than add a mutation rate heterogeneity to your model using `dm.addMutationRateHeterogenity`.

References

Richard˜R. Hudson. Generating samples under a Wright–Fisher neutral model of genetic variation. *Bioinformatics*, 18(2):337 –338, February 2002. doi: 10.1093/bioinformatics/18.2.337.

Lisha Mathew, Paul˜R. Staab, Laura˜E Rose, and Dirk Metzler. A composite likelihood approach to distinguish gene flow from violations of the infinite-sites model. *Submitted.*, December 2012.

Lisha Naduvilezhath, Laura˜E Rose, and Dirk Metzler. Jaatha: a fast composite likelihood approach to estimate demographic parameters. *Molecular Ecology*, 20(13):2709–2723, July 2011. doi: 10.1111/j.1365-294X.2011.05131.x.

Andrew Rambaut and Nicholas˜C Grassly. Seq-gen: An application for the monte carlo simulation of DNA sequence evolution along phylogenetic trees. *Computer applications in the biosciences : CABIOS*, 13(3):235–238, June 1997. ISSN 1367-4803, 1460-2059. doi: 10.1093/bioinformatics/13.3.235. URL http://bioinformatics.oxfordjournals.org/content/13/3/235.