fastsimcoal

a continuous-time coalescent simulator of genomic diversity under arbitrarily complex evolutionary scenarios

Laurent Excoffier
&
Matthieu Foll

Computational and Molecular Population Genetics lab
Institute of Ecology and Evolution
University of Berne
Baltzerstrasse 6
3012 Berne
Switzerland

Swiss Institute of Bioinformatics
1015 Lausanne, Switzerland

February 2011

1. TABLE OF CONTENTS

2. INTRODUCTION

This manual describes the use of *fastsimcoal*, a program to generate the neutral genomic molecular diversity in current or ancient samples drawn from a population with a complex demographic history. *fastsimcoal* is a completely rewritten version of *simcoal2* (Laval and Excoffier, 2004), a coalescent simulation program implementing a generation by generation approach while *fastsimcoal* is based on a much faster continuous time approximation. Despite a completely new coalescent engine, *fastsimcoal* uses exactly the same input files as simcoal2, and it produces very similar output files.

fastsimcoal typically generates many replicates of random outcome of molecular diversity under a user-defined evolutionary scenario. The evolutionary scenario is defined in an input parameter file (extension .par) and the output diversity is written in *arlequin* project files (extension .arp) that can then be processed with *arlequin* or *arlsumstat* (Excoffier and Lischer, 2010) to get distributions of various summary statistics. Additional options of *fastsimcoal* can be specified on the command line (type "*fastsimcoal -h*" for help on command line options).

fastsimcoal can handle very complex evolutionary scenarios including an arbitrary migration matrix between samples, historical events allowing for population resize, population fusion and fission, admixture events, changes in migration matrix, or changes in population growth rates. The time of sampling can be specified independently for each sample, allowing for serial sampling in the same or in different populations.

Different markers, such as DNA sequences, SNP, STR (microsatellite) or multi-locus allelic data can be generated under a variety of mutation models (e.g. finite- and infinite-site models for DNA sequences, stepwise or generalized stepwise mutation model for STRs data, infinite-allele model for standard multi-allelic data).

fastsimcoal can simulate data in genomic regions with arbitrary recombination rates, thus allowing for recombination hotspots of different intensities at any position. *fastsimcoal* implements a new approximation to the ancestral recombination graph in the form of sequential Markov coalescent allowing it to very quickly generate genetic diversity for >100 Mb genomic segments.

Compiled versions of *fastsimcoal* for Windows, Linux or Mac Os X are available on http://cmpg.unibe.ch/software/fastsimcoal

Since fastsimcoal output is meant to be interfaced with *Arlequin* or *arlsumstat*, the reader may also want to get more information on Arlequin on http://cmpg.unibe.ch/software/arlequin35

3. CHANGES COMPARED TO SIMCOAL2

1. Faster continuous-time coalescent simulations
2. Faster recombination model
3. Serial sampling
4. Generation of DNA sequence data under the infinite-site model
5. Sampling of parameter values from prior distributions
6. Computation of population-specific and joint site frequency spectrum
7. Optional output of all trees under the serial Markov coalescent model of recombination
8. RFLP data cannot be simulated anymore

4. GETTING STARTED

Compiled version of *fastsimcoal* and example files can be downloaded from http://cmpg.unibe.ch/software/fastsimcoal.

The archives include an executable version of *fastsimcoal* for a given platform, the *fastsimcoal pdf* manual, as well as examples *par* files and example template (*tpl*), distribution (*est*), and definition files (*def*) for a veriety of simple evolutionary scenarios and several types of markers, with and without recombination.

INTALLATION

Unzip the archive file to the directory of your choice. A version of *fastsimcoal* should be present (fastsimcoal.exe under windows, or *fastsimcoal* under linux of macos). On a command line, simply type "`fastsimcoal`" and you will have a list of the different command-line options available to run *fastsimcoal*.

In order to access *fastsimcoal* from any directory on your hard disk, put the directory on your path under windows, or put *fastsimcoal* in your `~/bin` directory under linux or Mac OS X.

RUNNING FASTSIMCOAL

There are 3 ways to simulate genetic data with *fastsimcoal*

1) Simulate data under an evolutionary scenario with parameter values defined in an input parameter file

```
fastsimcoal -i test.par -n 100
```

fastsimcoal will use the scenario and the parameter values defined in the parameter file *test.par* and make 100 simulations under this scenario.

2) Simulate data under an evolutionary scenario with parameter values randomly drawn from priors

```
fastsimcoal –t test.tpl –n 10 –e test.est –E 100
```

fastsimcoal will use the scenario defined in the template file *test.tpl* and generate 100 sets of parameter values by randomly drawing these values from the priors defined in the file *test.est*. 10 simulations will be done for each sets of randomly drawn parameter values.

3) Simulate data under an evolutionary scenario with parameter values defined in an external definition file

```
fastsimcoal -t test.tpl -n 100 -f test.def
```

fastsimcoal will use the scenario defined in the template file test.tpl and use the parameter values found in the definition file *test.def*. 100 simulations will be done for each set of predefined parameter values.

Additional descriptions of command line options and input file format can be found in the next chapters

5. STRUCTURE OF INPUT FILES

A SIMPLE UNSUBDIVIDED POPULATION AND DNA SEQUENCES

Let's consider the case of a single population made up of 20000 haploid individuals (or 10000 diploid individuals) where we want to generate diversity along a 10 Kb DNA sequence, with mutation rate $\mu = 2 \times 10^{-8}$ / bp / gen.

The simcoal2-compatible input file *1popDNA.par* describing such a scenario would look like:

1popDNA.par

```
//Number of population samples (demes)
1
//Population effective sizes (number of genes)
20000
//Sample sizes
10
//Growth rates      : negative growth implies population expansion
0
//Number of migration matrices : 0 implies no migration between demes
0
//historical event: time, source, sink, migrants, new size, growth rate, migr. matrix
0  historical event
//Number of independent loci [chromosome]
1 0
//Per chromosome: Number of linkage blocks
1
//per Block: data type, num loci, rec. rate and mut rate + optional parameters
DNA 10000 0.00000 0.00000002 0.33
```

and with the following command line

```
fastsimcoal -i 1popDNA.par -n1
```

fastsimcoal generates the following *Arlequin*-formatted output file, which is located in a directory having the same name as the input file but with the extension *.arp* removed, *1popDNA* in our case:

./1popDNA/1popDNA_1_1.arp

```
#Arlequin input file written by the simulation program fastsimcoal.exe

[Profile]
       Title="A series of simulated samples"
       NbSamples=1

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

[Data]
       [[Samples]]

#Number of independent chromosomes: 1
#Polymorphic positions on chromosome 1
#288, 800, 1367, 1772, 2045, 2191, 2328, 3258, 3385, 3591, 4162, 5176, 5332, 6128,
6224, 6442, 6455, 7846, 8026, 8536

              SampleName="Sample 1"
              SampleSize=10
              SampleData= {
1_1    1      TAAACATATACCTACGACTC
```

```
1_2    1      TAAACATATACCTACGACTC
1_3    1      TTGTACTTTGCCTCCACCTC
1_4    1      TTGTACGTTGCCTCCACCTC
1_5    1      TAAACATATACCTACGACAC
1_6    1      TAAACATATACCTACGACTC
1_7    1      TAGAAATATAACTAAGCATG
1_8    1      GAGTACTTAACGCACACCTC
1_9    1      TTGTACTTTGCCTCCACCTC
1_10   1      TTGTACTTTGCCTCCACCTC

}

[[Structure]]

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

New to *fastsimcoal*, one now only outputs polymorphic sites for DNA sequences and the position of these sites is provided as a comment above the sample definition.

MIGRATION

fastsimcoal can generate data from samples drawn from a subdivided population. For instance the following input file describes an asymmetric 2-deme island model:

2popSTRmigr.par

```
//Number of population samples (demes)
2 samples to simulate :
//Population effective sizes (number of genes)
1000
1000
//Samples sizes
5
5
//Growth rates      : negative growth implies population expansion
0
0
//Number of migration matrices : 0 implies no migration between demes
1
//migration matrix
0.000 0.005
0.001 0.000
//historical event: time, source, sink, migrants, new size, growth rate, migr. matrix
0  historical event
//Number of independent loci [chromosome]
1 0
//Per chromosome: Number of linkage blocks
1
//per Block: data type, num loci, rec. rate and mut rate + optional parameters
MICROSAT  10  0.0000  0.0005 0 0
```

The population sizes are of 1000 genes each, and we want to generate samples of size 5 in each population. One migration matrix is defined, listing the migration rates between the two populations. The migration matrix can be asymmetric, and in the case the entry m_{ij} list the **migration rates backward in time** from population *i* to population *j*. The above-mentioned matrix

```
0.000 0.005
0.001 0.000
```

states that, for each generation backward in time, any gene from population 0 has probability 0.005 to be sent to population 1, and that a gene from population 1 has a probability 0.001 to move to population 0.

Note that if no migration matrix is defined, no migration is assumed between populations.

Coming back to our example input file above, the data to be generated consist here of 10 microsatellite markers that are fully linked on a chromosome, with mutation rate $\mu = 5 \times 10^{-4}$ per generation per locus. A pure stepwise mutation model without range constraints is assumed.

The following command line

```
fastsimcoal -i 2popSTRmigr.par -n1
```

produces the following *Arlequin* output:

2popSTRmigr_1_1.arp

```
#Arlequin input file written by the simulation program fastsimcoal.exe

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

       GenotypicData=0
       GameticPhase=0
       RecessiveData=0
       DataType=MICROSAT
       LocusSeparator=WHITESPACE
       MissingData='?'

[Data]
       [[Samples]]

#Number of independent chromosomes: 1
#Polymorphic positions on chromosome 1
#1, 2, 3, 4, 5, 6, 7, 8, 9, 10

              SampleName="Sample 1"
              SampleSize=5
              SampleData= {
1_1    1       500  499  500  500  499  500  499  500  501  503
1_2    1       498  498  501  502  500  501  498  500  499  499
1_3    1       499  498  500  501  500  501  499  501  502  499
1_4    1       498  498  500  502  500  501  498  501  499  500
1_5    1       498  497  500  503  500  500  498  500  500  499

}
              SampleName="Sample 2"
              SampleSize=5
              SampleData= {
2_1    1       500  499  500  500  499  500  499  500  500  503
2_2    1       498  498  500  501  500  501  498  501  499  499
2_3    1       500  499  500  500  499  500  499  500  500  503
2_4    1       501  499  500  500  499  500  499  500  500  503
2_5    1       499  498  500  501  501  501  499  501  501  499

}

[[Structure]]

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

Note that for microsatellite data, the ancestral allele at each locus is arbitrarily set to have 500 repeats, and that different numbers indicate different number of repeats.

HISTORICAL EVENTS

Historical events can be used to:

- Change the size of a given population
- Change the growth rate of a given population
- Change the migration matrix to be used between population
- Move a fraction of the genes of a given population to another population. This amounts to implementing a (stochastic) admixture or introgression event.
- Move all genes from a population to another population (population). This amounts to fusing two populations into one, looking backward in time, or implementing a population fission looking forward in time.
- One or more of these events at the same time

Note that several events can be defined at different or at the same time in the past, as in the following example file

2popSTRdiv.par

```
//Number of population samples (demes)
2 samples to simulate :
//Population effective sizes (number of genes)
1000
1000
//Samples sizes
5
5
//Growth rates      : negative growth implies population expansion
0
0
//Number of migration matrices : 0 implies no migration between demes
2
//migration matrix
0.000 0.0005
0.0001 0.000
//migration matrix
0.000 0.000
0.000 0.000
//historical event: time, source, sink, migrants, new size, growth rate, migr. matrix
2 historical event
1000 0 0 0 1 0 1
10000 1 0 1 10 0 1
//Number of independent loci [chromosome]
1 0
//Per chromosome: Number of linkage blocks
1
//per Block: data type, num loci, rec. rate and mut rate + optional parameters
MICROSAT  10  0.0000  0.0005 0
```

In this example, we have two migration matrices, and the first migration matrix is used by default until it is changed by a historical event. The second matrix has all entries set to zero, and thus specifies an absence of migrations.

The first historical event

```
1000 0 0 0 1 0 1
```

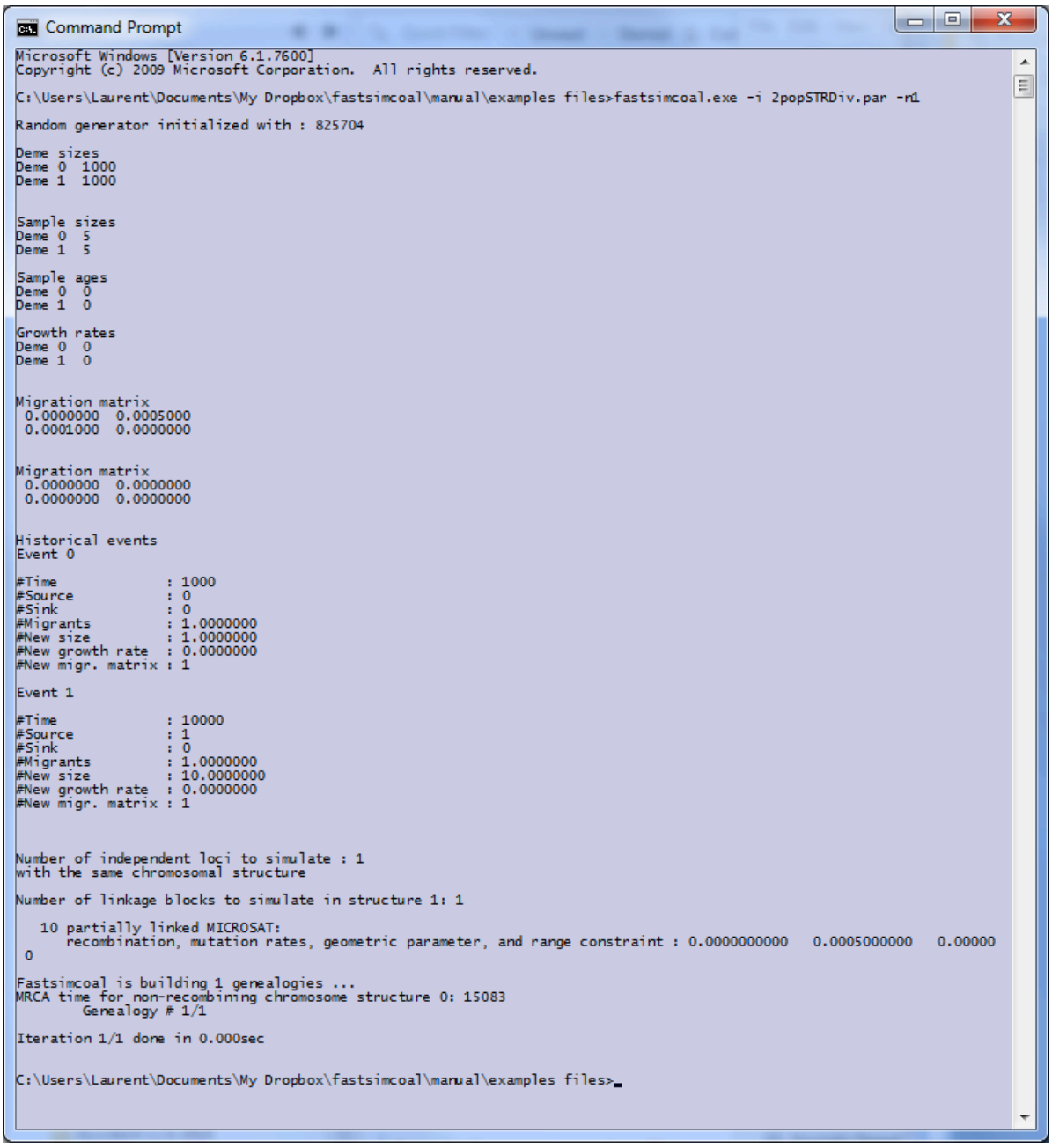
specifies that 1000 generations in the past, we need to change the migration matrix and use migration 1. In doing that it basically stops all migrations between demes.

The second historical event

```
10000 1 0 1 10 0 1
```

says that 10,000 generations in the past all the genes from deme 1 move to deme 0, the size of which is resized by a factor 10 (to 10,000 genes).

You can check that these historical events are correctly understood by fastsimcoal by looking at the console output, which should look like



Alternatively, the simulations conditions are also output in the file *./2popSTRdiv/2popSTRdiv_1.simparam,* located in the directory *2popSTRdiv*, together with the *Arlequin* project *2popSTRdiv_1_1.arp*.

SERIAL SAMPLING

In *fastsimcoal*, it is possible to specify at which time sampling was performed in the past. This is simply achieved by adding sampling time after the specification of the sample size. If no time is specified after the sample size, than sampling at present time is assumed, which also ensures

compatibility with *simcoal2* input files. The use of this new feature is shown in the simple following input file.

1popDNAserial.par

```
//Number of population samples (demes)
3
//Population effective sizes (number of genes)
1000
0
0
//Sample sizes
5
3 500
2 1000
//Growth rates      : negative growth implies population expansion
0
0
0
//Number of migration matrices : 0 implies no migration between demes
0
//historical event: time, source, sink, migrants, new size, growth rate, migr. matrix
2  historical event
500 1 0 1 1 0 0
1000 2 0 1 1 0 0
//Number of independent loci [chromosome]
1 0
//Per chromosome: Number of linkage blocks
1
//per Block: data type, num loci, rec. rate and mut rate + optional parameters
DNA 10000 0.00000 0.00000002 0.33
```

In this scenario, 5 sequences were sampled at the present time, 3 sequences 500 generations ago, and 2 sequences 1000 generations ago. As we consider that these 3 samples come from the same population, we simply use 2 historical events to transfer these ancient DNA sequences to deme 0 at the time of their sampling.

The resulting genealogy can be visualized by asking *fastsimcoal* to output coalescent trees. This is done with the *-T* option on the command line. the following command

```
fastsimcoal -i 1popDNAserial.par -n 2 -T
```

produces tree files in the nexus format, looking like

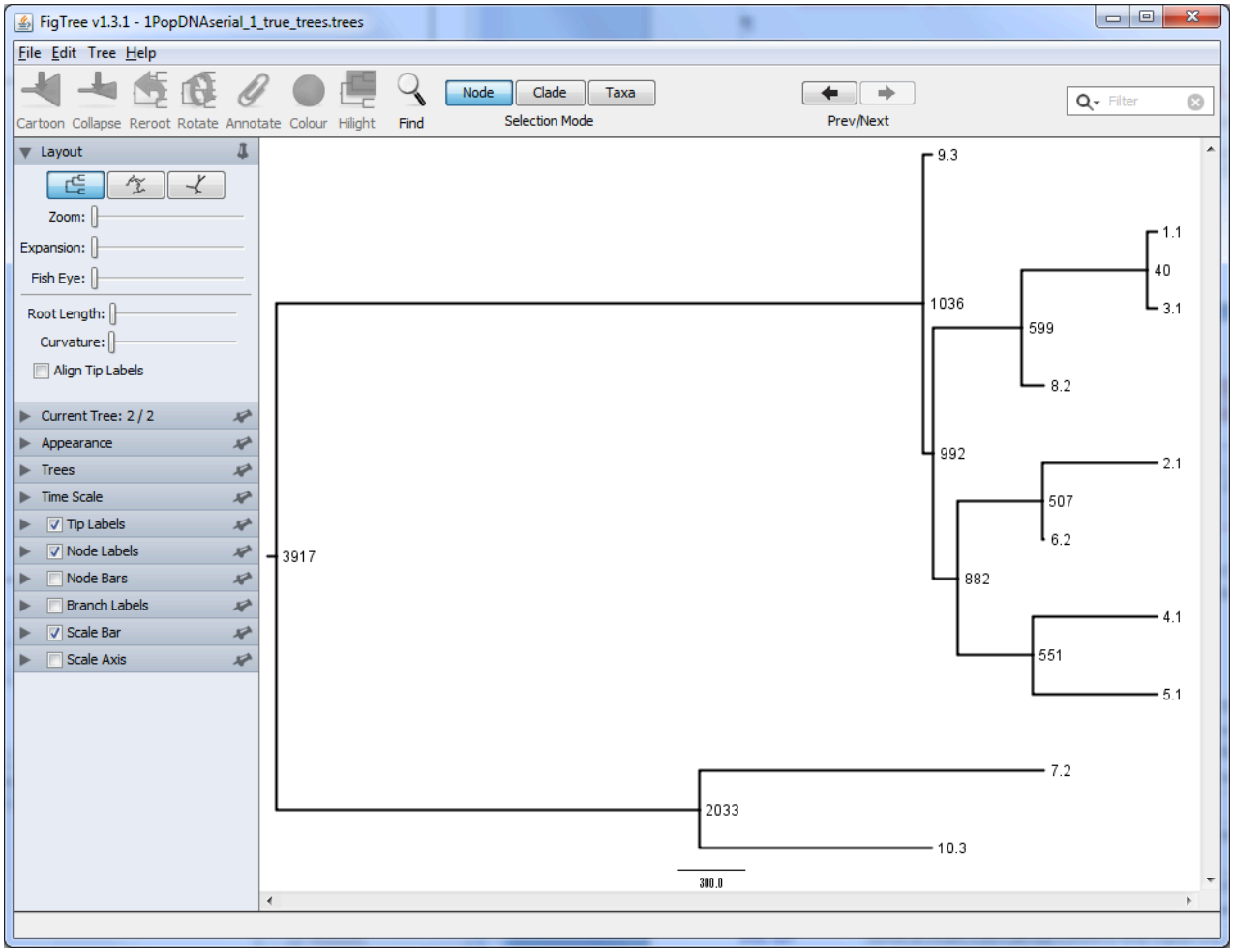
./1PopDNAserial/1PopDNAserial_1_true_trees.trees

```
#NEXUS
begin trees;  [Treefile generated by fastsimcoal.exe (Laurent Excoffier)]

       tree NumGen_tree_1_1_pos_0 = [&U] ((10.3:103, ((4.1:269, 3.1:269):728,
7.2:497):106):1197, ((8.2:580, 9.3:80):852, (6.2:572, (5.1:811, (2.1:55,
1.1:55):756):261):860):368);
       tree NumGen_tree_2_1_pos_0 = [&U] ((9.3:36, (((1.1:40, 3.1:40):559,
8.2:99):393, ((2.1:507, 6.2:7):375, (4.1:551, 5.1:551):331):110):44):2881, (7.2:1533,
10.3:1033):1884);
end;
```

These trees can be conveniently visualized with visualization tools, like *FigTree* (http://tree.bio.ed.ac.uk/software/figtree) freely available for Windows, Linux, or Mac OS X.

our tree of 10 DNA sequences looks like:



We indeed see that the first 5 sequences from deme 0 were sampled at time 0, that the 3 sequences from deme 1 were sampled at time 500, and that the 2 sequences of deme 2 were sampled at time 1000.

Note that it may not be a good idea to record trees for long DNA sequences with recombination, as trees files can become extremely large (>100Mbytes for 10Mbase sequences).

SIMULATION OF SEVERAL CHROMOSOMAL SEGMENTS

It is easy to simulate several chromosomal segments with different types of markers and different recombination or mutation rates.

In the following example files, one simulates 2 independent non-recombining chromosome segments with the same structure. Each chromosome is made up of 4 blocks, each time with a different mutation model.

1PopMultiLocus.par

```
//Number of population samples (demes)
1
//Population effective sizes (number of genes)
10000
//Sample sizes
5
//Growth rates      : negative growth implies population expansion
0
//Number of migration matrices : 0 implies no migration between demes
0
//historical event: time, source, sink, migrants, new size, new growth rate, migr.
matrix
0  historical event0
//Number of independent loci [chromosome]
2 0
//Per chromosome: Number of linkage blocks
4
//per Block: data type, num loci, rec. rate and mut rate + optional parameters
DNA 1000 0 0.0000002 0.33
SNP 3 0 0
MICROSAT 3 0 0.0005 0 0
STANDARD 2 0 0.001
```

The data section of a typical output of this scenario is :

./1PopMultiLocus/1PopMultiLocus_1_1.arp

```
[Data]
       [[Samples]]

#Number of independent chromosomes: 2
#Polymorphic positions on chromosome 1
#40, 255, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008
#Polymorphic positions on chromosome 2
#216, 382, 485, 899, 997, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008

              SampleName="Sample 1"
              SampleSize=5
              SampleData= {
1_1    1      CC 000 499  502  499  5  2       AGTGA 100 501  501  503  10  16
1_2    1      CG 011 501  499  500  1  4       CCGCA 011 500  503  498  9  10
1_3    1      GG 000 498  501  500  6  2       ACGCG 001 498  500  498  6  10
1_4    1      GG 100 499  501  500  6  2       ACGCA 001 500  503  499  10  13
1_5    1      CG 000 500  499  501  2  4       ACGCA 001 498  504  498  8  8

}
```

One can also simulate several chromosomes with completely different structures, like

1PopMultiLocusDiffChrom.par

```
//Number of population samples (demes)
1
//Population effective sizes (number of genes)
10000
//Sample sizes
5
//Growth rates      : negative growth implies population expansion
0
//Number of migration matrices : 0 implies no migration between demes
0
//historical event: time, source, sink, migrants, new size, new growth rate, migr.
matrix
0  historical event0
//Number of independent loci [chromosome]
2 1
//Per chromosome: Number of linkage blocks
2
//per Block: data type, num loci, rec. rate and mut rate + optional parameters
DNA 1000 0 0.0000002 0.33
SNP 3 0 0
//Per chromosome: Number of linkage blocks
2
//per Block: data type, num loci, rec. rate and mut rate + optional parameters
MICROSAT 3 0 0.0005 0 0
STANDARD 2 0 0.001
```

The difference with the previous example is that we explicitly tell *fastsimcoal* to simulate 2 chromosomes with different structures by stating

```
//Number of independent loci [chromosome]
2 1
```

The second number (1) indicates we want to describe different chromosomal structures. Now we need to repeat the block definition for the two structures, while a single block definition was used previously and simply repeated for simulating the two identical chromosomes.

The following output is now produced:

./1PopMultiLocusDiffChrom/1PopMultiLocusDiffChrom_1_1.arp

```
[Data]
       [[Samples]]

#Number of independent chromosomes: 2
#Polymorphic positions on chromosome 1
#85, 105, 169, 277, 372, 470, 629, 635, 702, 934, 960, 998, 1001, 1002, 1003
#Polymorphic positions on chromosome 2
#1, 2, 3, 4, 5

              SampleName="Sample 1"
              SampleSize=5
              SampleData= {
1_1    1      ACCGGCCCACTA 000      502  501  500  19  20
1_2    1      TTATTCGTATTG 111      504  500  501  21  25
1_3    1      ACCGGTCCACTA 000      503  500  501  22  23
1_4    1      TTATGCCTCTCG 111      501  499  506  23  24
1_5    1      TTATGCCTATTG 111      500  498  504  23  21

}
```

RECOMBINATION

Like simcoal2, fastsimcoal can generate molecular diversity along recombining chromosomal segments, and recombination rates between adjacent loci are specified just after the definition of the data type and the number of loci of this type, like in the following example file

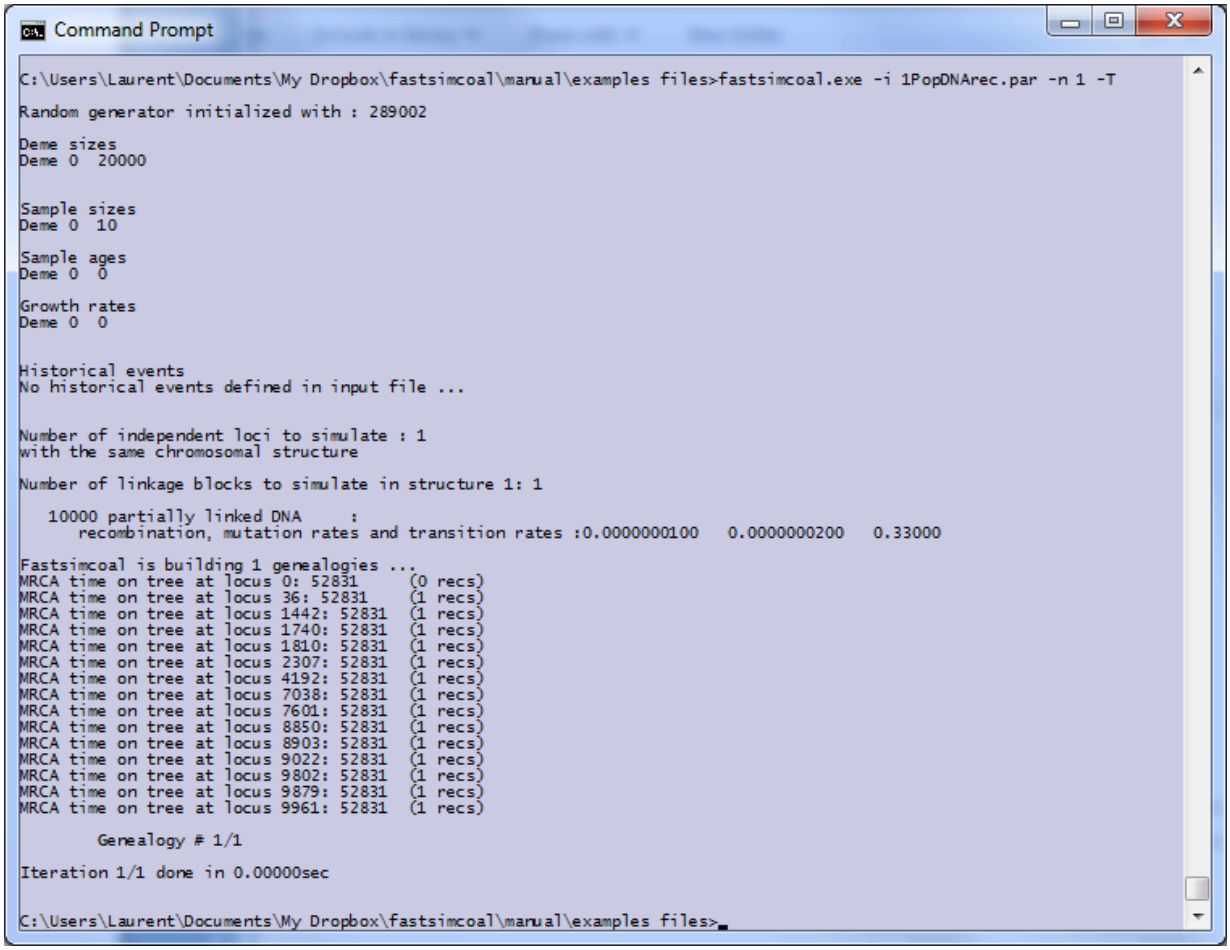
1PopDNArec.par

```
//Number of population samples (demes)
1
//Population effective sizes (number of genes)
20000
//Sample sizes
10
//Growth rates      : negative growth implies population expansion
0
//Number of migration matrices : 0 implies no migration between demes
0
//historical event: time, source, sink, migrants, new size, new growth rate, migr.
matrix
0  historical event
//Number of independent loci [chromosome]
1 0
//Per chromosome: Number of linkage blocks
1
//per Block: data type, num loci, rec. rate and mut rate + optional parameters
DNA 10000 0.00000001 0.00000002 0.33
```

In this case, we want to simulate 10 DNA sequences of 10 Kb with a recombination rate of $r = 10^{-8}$ and mutation rate $\mu = 2 \times 10^{-8}$.

While the syntax to specify recombination is the same as in *simcoal2* , the underlying simulation model is completely different, as we now use a sequential Markov coalescent model (McVean and Cardin, 2005) to simulate recombination. In brief, instead of using the classical ancestral recombination graph for the whole chromosomal segment, we generate a first tree on the left of the segment to be simulated. Then we compute where the next recombination event will occur on the segment, implement one recombination event randomly on the tree, detach the recombining lineage from the left tree, and leave this recombining lineage evolve until it coalesces with one of the lineage on the left tree, potentially changing the topology and the height of the tree. See the Sequential Markov Coalescent section for more details.

In the console, *fastsimcoal* list the position of each recombination breakpoint and the MRCA of the resulting tree, as in



The produced *Arlequin* output file is similar to what would be obtained without recombination, but the nexus tree file now lists all trees produced along the chromosomal segments, as

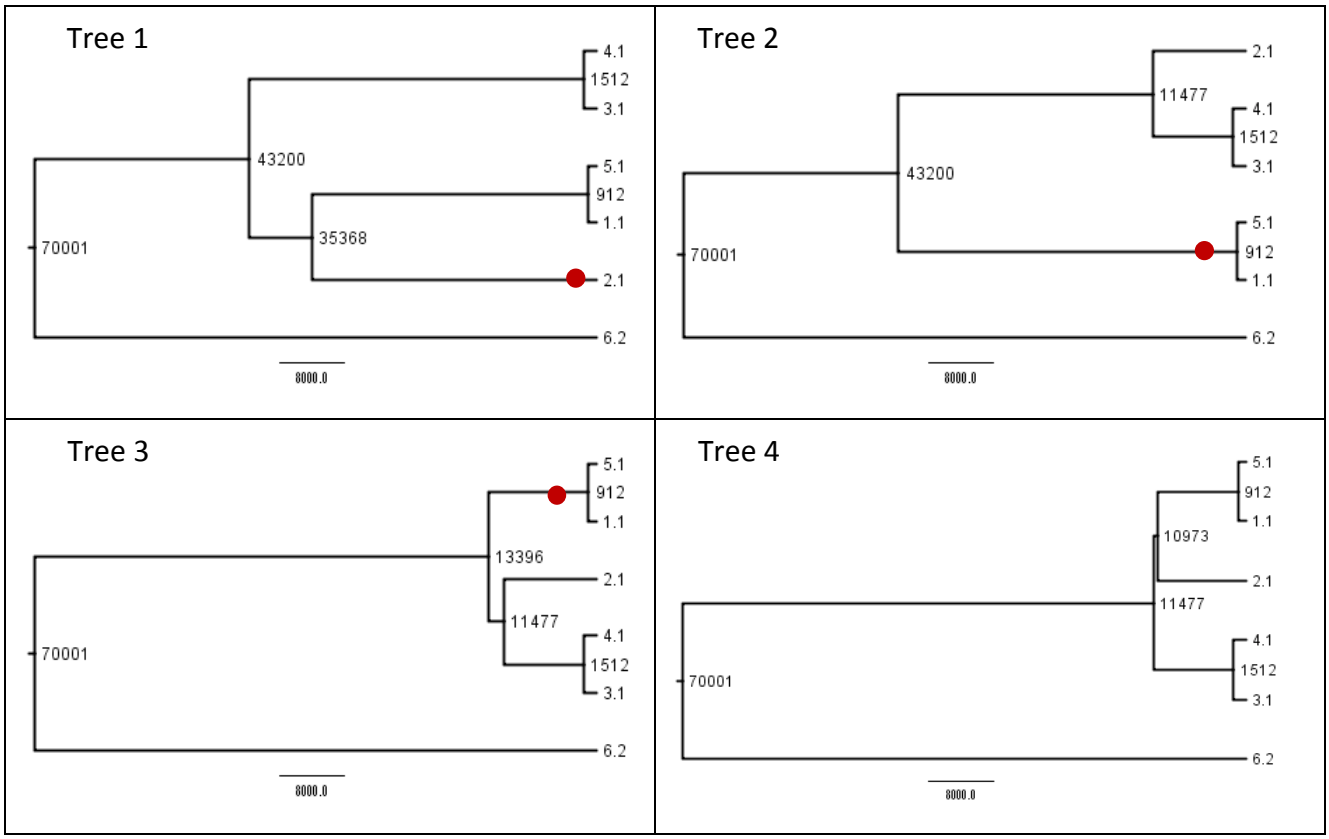
./1PopDNArec/1PopDNArec_1_true_trees.trees

```
#NEXUS
begin trees;  [Treefile generated by fastsimcoal.exe (Laurent Excoffier)]

       tree NumGen_tree_1_1_pos_0 = [&U] ((5.1:3556, (4.1:2838, 8.1:2838):718):49275,
((2.1:707, (6.1:384, 7.1:384):323):36435, ((3.1:3209, 9.1:3209):12690, (1.1:148,
10.1:148):15751):21243):15689);
       tree NumGen_tree_1_2_pos_36 = [&U] ((5.1:3556, (4.1:2838, 8.1:2838):718):49275,
((2.1:707, (6.1:384, 7.1:384):323):26269, ((3.1:3209, 9.1:3209):12690, (1.1:148,
10.1:148):15751):11077):25855);
       tree NumGen_tree_1_3_pos_1442 = [&U] ((5.1:3556, (4.1:2838,
8.1:2838):718):49275, (((3.1:3209, 9.1:3209):4915, (2.1:707, (6.1:384,
7.1:384):323):7417):18852, (1.1:148, 10.1:148):26828):25855);
       tree NumGen_tree_1_4_pos_1740 = [&U] ((5.1:3556, (4.1:2838,
8.1:2838):718):49275, (((3.1:3209, 9.1:3209):4915, (2.1:707, (6.1:384,
7.1:384):323):7417):25747, (1.1:148, 10.1:148):33723):18960);
...
Etc...
...
end;
```

Note that these trees can also all be browsed individually e.g. in *FigTree* to see the changes in topology produced by the recombination events. For instance, using different settings, here are the changes introduced by 3 recombination events having occurred in the subtree connecting

nodes 1 to 5 (approximately located by red dots). Note that additional recombination events fell on other branches of the tree but did not lead to observable changes in tree topology.



INPUT FILE SYNTAX

The syntax of the input file is the same as in *simcoal2*, with a few exceptions.

The input file is divided into the following sections that are each time separated by a comment line. The order of these sections is fixed and cannot be changed.

- Number of populations samples
- Deme sizes
- Sample sizes and sampling times
- Growth rates
- Migration matrices
- Historical events
- Genetic information

NUMBER OF POPULATIONS SAMPLES

This section begins with a comment line. On the second line, the first item should be the number of samples (or demes) to simulate. Additional items on the second line are ignored.

```
//Number of population samples (demes)
1
```

or

```
//Number of population samples (demes)
2 samples to simulate
```

DEME SIZES

This section begins with a comment line, and then has as many lines as demes to be simulated, as mentioned in the previous section.

So if only one sample was defined previously:

```
//Population effective sizes (number of genes)
20000
```

or if, say, three samples were defined previously:

```
//Population effective sizes (number of genes)
20000
10000
100
```

Note that the deme size corresponds here to the number of genes present in a population, which would be the number of individuals for haploid species or to two times the number of individuals for diploid species.

SAMPLES SIZES AND SAMPLING TIMES

This section begins with a comment line, and then lists, for as many samples as defined in the first section, the haploid size of the sample, and optionally the sampling time (the number of generation backward in time when the samples were identified). If the sampling time is omitted, a time of zero is assumed (present sampling).

So the following input

```
//Sample sizes
10
```

is equivalent to

```
//Sample sizes
10 0
```

Other possibilities are:

```
//Sample sizes
10 20
100 1000
```

or

```
//Sample sizes
5
30
0
```

showing in the last case that sample sizes of zero are possible, which just means that no genes were sampled in a given deme at time zero. This additional empty deme may be used in a given evolutionary scenario, for instance as a source population for the currently sampled demes.

GROWTH RATE

This section lists the initial growth rates for all population samples

So the following input

```
//Growth rates     : negative growth implies population expansion
0
```

means that the sampled deme has a stationary population size. Note that growth rates are measured here backward in time, so that negative growth rates imply a forward population expansion. Generally, if the current population size is N_0, and the growth rate is i then the population size *t* generations ago is given by $N_t = N_0 \, e^{rt}$.

In the following example, two populations are shrinking backward in time, implying that the sampled demes went through a recent population expansion

```
//Growth rates     : negative growth implies population expansion
-0.001
-0.025
```

Note that the growth rates can be modified at any time by the use of a historical event.

MIGRATION MATRICES

As usual, this section begins with a comment line, and is followed by a line with the number of migration matrices. If there is no migration between demes , then simply enter 0, like

```
//Number of migration matrices : 0 implies no migration between demes
0
```

If we assume that there are two demes connected by migration, then one could enter for instance the following two migration matrices:

```
//Number of migration matrices : 0 implies no migration between demes
2
//migration matrix
0.000 0.0005
0.0001 0.000
//migration matrix
0.000 0.000
0.000 0.000
```

Here the two migration matrices are put below each other separated by a comment line.

The first migration matrix is asymmetric, with deme 0 sending migrant **backward in time** at rate 0.0005 towards deme 1, while deme1 is sending migrants at a lower rate 0.0001 towards deme 0. Therefore, the non-diagonal entries $\{m_{ij}\}$ of each migration matrix represent the probability for any given lineage to move backward in time from deme *i* to deme *j*. The diagonal entries are ignored. In the above example, the second migration matrix implies an absence of migration. One can switch between migration matrices at any time by means of historical events. Note that by default, the first migration matrix is used at the current time and further back in time until a historical event changes the active migration matrix.

HISTORICAL EVENTS

This section begins with a comment line and is followed by a line specifying the number of historical events. Then, each historical event is defined on a different line. Each historical event is defined by 7 numbers

1) Number of generations *t* before present at which the historical event happened
2) Source deme (the first listed deme has index 0))
3) Sink deme
4) Expected proportion of migrants to move from source to sink. Note that this proportion is not fixed, and that it also represents the probability for each lineage in the source deme to migrate to the sink deme.
5) New size for the **sink** deme, relative to its size at generation *t*.
6) New growth rate for the **sink** deme
7) New migration matrix to be used further back in time.

In the following example, 2 historical events are defined.

```
//historical event: time, source, sink, migrants, new size, growth rate, migr. matrix
2 historical event
1000 0 0 0 1 0 1
10000 1 0 1 10 0 1
```

The first event occurred 1000 generation in the past, and just sets the active migration matrix to matrix 1, whereas matrix 0 had been active until then.

The second event, which occurred 10,000 generations ago, states that all lineages present in deme 1 need to migrate to deme 0. At the same time, the size of deme 1 is increased by a factor 10 and we still use migration matrix 1.

If no historical events are necessary in your simulations just set the number to zero, as

```
//historical event: time, source, sink, migrants, new size, growth rate, migr. matrix
0 historical event
```

GENETIC SETTINGS: CHROMOSOMES, BLOCKS, DATA TYPES, MUTATION, AND RECOMBINATION

GENETIC SETTINGS SUBSECTIONS

The genetic information section has 3 subsections

1) The number of independent **chromosomes** to be simulated and a flag (0, 1) indicating if the different chromosomes have a different (1) or a similar (0) structure. These chromosomes are assumed to be completely unlinked. If the chromosomes have a different structure, the sub-sections 29 and 3) need to be repeated for each chromosome structure.
2) The number of **blocks** to be simulated per chromosome. By block we mean segments of chromosomes that may differ by the type of markers to be simulated, the recombination rate, or the mutation rate. Two consecutive blocks may be of the same type but just differ by the recombination rate, such as for instance simulate a recombination hot spot. Note

that the recombination rate between two blocks is that specified in the first block, which is thus valid both within and between blocks.

3) The properties of genetic data to be simulated per block. In each block the following properties need to be specified, in this order:

   i) **Data type**: DNA, MICROSAT, SNP, or STANDARD

   ii) Number of markers with this data type to be simulated. For DNA, this is the sequence length.

   iii) Recombination rate between adjacent markers (between adjacent nucleotides for DNA).

   iv)-vii) Additional data type-specific properties, like the mutation rate per bp and the transition bias for DNA (see list below).

In the example below, we define genetic data to be simulated in two different types of chromosomes:

```
//Number of independent loci [chromosome]
2 1
//Per chromosome: Number of linkage blocks
2
//per Block: data type, num loci, rec. rate and mut rate + optional parameters
DNA 1000 0 0.0000002 0.33
SNP 3 0 0
//Per chromosome: Number of linkage blocks
2
//per Block: data type, num loci, rec. rate and mut rate + optional parameters
MICROSAT 3 0 0.0005 0 0
STANDARD 2 0 0.001
```

On the first type of chromosome we simulate a DNA sequence of 1000 bp and 3 SNPs. On the second type of chromosome, we simulate 3 microsats under a pure stepwise mutation model and 2 multi-allelic loci under an infinite-allele model. On each chromosome, all markers are fully linked as we do not assume any recombination.

In the next example, we want to simulate two chromosomes with the same structure, and we indicate this with the 0 flag after the number of independent chromosomes to simulate.

```
//Number of independent loci [chromosome]
2 0
//Per chromosome: Number of linkage blocks
2
//per Block: data type, num loci, rec. rate and mut rate + optional parameters
DNA 1000 0 0.0000002 0.33
SNP 3 0 0
```

Note that if a sequence of DNA is found monomorphic in a given population sample (due to a too short tree or a too small mutation rate), a question mark '?' will be output for each individual sequence.

SPECIFIC PARAMETERS FOR DIFFERENT DATA TYPES

The following optional parameters are required for the different types of markers.

	Extra parameters		
Data types	**4th parameter**	**5th parameter**	**6th parameter**
DNA[1]	Mutation rate per bp	Transition rate (fraction of substitutions that are transitions). A value of 0.33 implies no transition bias.	
MICROSAT	Mutation rate per locus.	Value of the geometric parameter for a Generalized Stepwise Mutation (GSM) model. This value represents the proportion of mutations that will change the allele size by more than one step. Values between 0 and 1 are required. A value of 0 is for a strict Stepwise Mutation Model (SMM).	Range constraint (number of different alleles allowed). A value of 0 means no range constraint
SNP	Minimum frequency for the **derived** allele. Note that if this minimum frequency is not possible given the simulated tree at that locus, the derived allele is coded as a 2 instead of a1 in the output file.		
STANDARD	Mutation rate per marker. An infinite allele model is assumed.		

[1] Note that by default, *fastsimcoal* uses a finite site model, implying that short DNA sequences simulated with a high mutation rate can be the target of multiple hits. The command-line option `-I` ensures that an infinite-site model is used.

EXAMPLES:

1. 1 Mb DNA sequence with recombination rate of 1×10^{-8} and mutation rate of 2×10^{-8}, no transition bias

```
DNA 1000000 1E-8 2E-8 0.33
```

2. 10 Kb DNA sequence without recombination and mutation rate of 2×10^{-6}, high transition bias

```
DNA 10000 0 2E-6 0.95
```

3. 20 MICROSAT with recombination rate of 1×10^{-5} between adjacent markers, a mutation rate of 5×10^{-4} per marker, a geometric GSM parameter of 0.1, and a maximum allele size range of 30.

```
MICROSAT 20 0.00001 5E-4 0.1 30
```

   Note that with such a high recombination rate, more than 1 recombination event can occur between adjacent markers, which is adequately taken into account by our modified serial Markov coalescent model (see the description of the SMC' below).

4. 10 SNP sites with variable recombination rates and a minimum derived allele frequency of 0.02.

```
//Number of independent loci [chromosome]
1 0
//Per chromosome: Number of linkage blocks
10
//per Block: data type, num loci, rec. rate + optional parameters
SNP 1 0.00001 0.02
SNP 1 0.001 0.02
SNP 1 0.0002 0.02
SNP 1 0.01 0.02
SNP 1 0 0.02
SNP 1 0.00003 0.02
SNP 1 0.005 0.02
SNP 1 0.00001 0.02
SNP 1 0.002 0.02
SNP 1 0 0.02
```

   With this example, one sees how to simulate SNPs located at fixed recombination distances along a given chromosome segment. Like for MICROSAT before, multiple recombination event between adjacent sites will be handled by our SMC' recombination approximation.

5. 20 STANDARD fully linked markers with a mutation rate of 5×10^{-6} per marker.

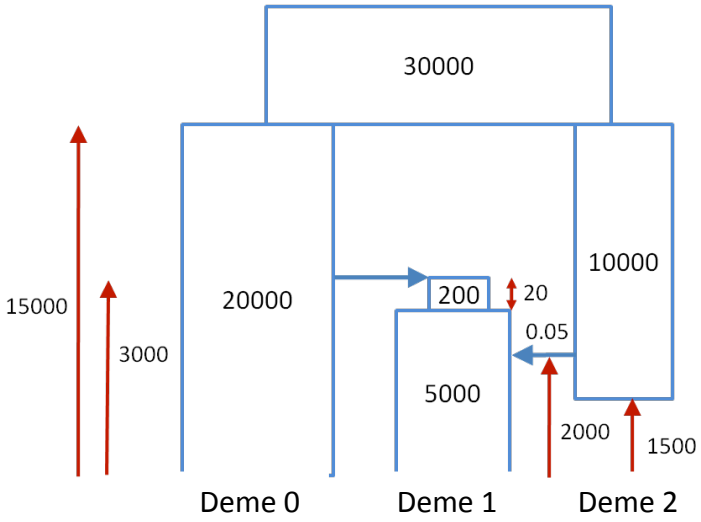
```
STANDARD 20 0 0.000005
```

Note finally that a given chromosome can be simulated for different types of markers, e.g. with a mixture of SNPs and MICROSAT markers:

```
//Number of independent loci [chromosome]
1 0
//Per chromosome: Number of linkage blocks
3
//per Block: data type, num loci, rec. rate + optional parameters
SNP 1 1E-6 0.01
MICROSAT 1 2E-7 5E-4 0 0
SNP 1 0 0.02
```

A RELATIVELY COMPLEX EXAMPLE WITH 3 POPULATIONS, SERIAL SAMPLING, BOTTLENECK, AND INTROGRESSION

Let's assume we are interested in the following relatively complex demographic scenario involving 3 demes (populations).



This scenario assumes that genes from deme 2 were sampled 1500 generations ago, and that 5% of the genes present in deme 1 2000 generations ago actually came from deme 2 (which implies an introgression or admixture event). It also assumes that deme 1 originated from deme 0 2000 ago and went through a bottleneck size of 200 (haploid) individuals during 20 generations before recovering a size of 5000 (haploid) individuals. Finally deme 0 and deme 2 diverged 15000 generations ago from an ancestral population of size 30000, to form tow populations of size 20000 and 10000, respectively.

We can simulate the evolution of 6 10Mb DNA sequences for deme2, and 20 10Mb sequences in both deme 0 and deme 1 under this scenario with the following parameter file *3popDNASFS.par*:

3popDNASFS.par

```
//Number of population samples (demes)
3
//Population effective sizes (number of genes)
20000
5000
10000
//Sample sizes
20
20
6 1500
//Growth rates: negative growth implies population expansion
0
0
0
//Number of migration matrices : 0 implies no migration between demes
0
//historical event: time, source, sink, migrants, new size, new growth rate, migr. matrix
4  historical event
2000 1 2 0.05 1 0 0
2800 1 1 0 0.04 0 0
3000 1 0 1 1 0 0
15000 0 2 1 3 0 0
//Number of independent loci [chromosome]
1 0
//Per chromosome: Number of linkage blocks
1
//per Block: data type, num loci, rec. rate and mut rate + optional parameters
DNA 10000000 0.00000001 0.00000002 0.33
```

6. SAMPLING PARAMETER VALUES FROM PRIOR DISTRIBUTIONS

fastsimcoal has a built-in procedure to sample parameters from prior distributions. The principle is very much the same as that implemented in *ABCToolBox* (Wegmann, et al., 2010), using a template file (**.tpl*) where parameters to be sampled are input as keyword, and an estimation file (**.est*) where parameter distributions are fully specified. Such a way to simulate data is invoked e.g. with the following command line arguments

```
fastsimcoal -t 1popDNArand.tpl -n10 -e 1popDNArand.est -E 1000
```

which tells *fastsimcoal* to use the template file *1popDNArand.tpl* and the estimation *1popDNArand.est* to generate 10 simulations for each of the 1000 sets of randomly drawn parameter values.

TEMPLATE FILE

An example template file reads as follows:

1popDNArand.tpl

```
//Number of population samples (demes)
1
//Population effective sizes (number of genes)
NPOP
//Sample sizes
10
//Growth rates      : negative growth implies population expansion
0
//Number of migration matrices : 0 implies no migration between demes
0
//historical event: time, source, sink, migrants, new size, new growth rate, migr.
matrix
1  historical event
TEXP 0 0 0 RESIZE 0 0
//Number of independent loci [chromosome]
1 0
//Per chromosome: Number of linkage blocks
1
//per Block: data type, num loci, rec. rate and mut rate + optional parameters
DNA 10000 0.00000 MUTRATE 0.33
```

As can be seen above, a template file has exactly the same structure as a parameter file, but some keywords (NPOP, TEXP, RESIZE, and MUTRATE) are present instead of actual parameter values.

Those keywords will be substituted with actual parameter values by *fastsimcoal*, if the distributions of these parameters are found in the estimation file.

Any *unique* keyword can be put in the template file. Note that *fastsimcoal* is not case sensitive, and therefore NPOP and nPop are identical for *fastsimcoal*. By unique, we mean that a keyword cannot be part of another keyword. For instance NPOP and NPOP1 are incompatible as NPOP is included into NPOP1.

ESTIMATION FILE

An estimation file matching the template file *1popDNArand.tpl* would look as follows

1popDNArand.est

```
// Priors and rules file
// *********************

[PARAMETERS]
//#isInt? #name  #dist. #min   #max
//all N are in number of haploid individuals
1  NPOP     logunif     100  100000   output
1  TEXP     logunif     100  5000     output
0  RESIZE   logunif     1e-3   1000   output
0  MUTRATE  unif        1e-7   1e-9   output

[RULES]

[COMPLEX PARAMETERS]

1  ANCSIZE  =  NPOP*RESIZE    output
0  2N       =  2*NPOP         hide
0  THETA    =  2N*MUTRATE     output
```

The estimation file is divided in three main sections

1. The [PARAMETERS] section lists the prior distributions of simple parameters.
   Each parameter can be an *integer* or a *float*, as specified by a first indicator variable.
   Each parameter can either be uniformly or log-uniformly distributed between a minimum and a maximum value that need to be specified.
2. The [RULES] section can include a set of conditions to be met among the simple parameters.
   Rules are typically used to specify that a parameter need to be larger than another one, which could be used for instance to specify that the divergence time between two populations must be larger than the time of a bottleneck. The use of rules will of course modify the priors of the parameters.
3. The [COMPLEX PARAMETERS] lists complex parameters that are obtained as simple operations between any 2 simple or complex parameters or between a parameter and a scalar.
   The following simple operations are possible “+”, “-”, “*”, and */*.

Simple or complex parameters can then be output or hidden in output files

As seen above, comments can be included in the *est* file as double slash, as in C++.

OUTPUT OF SAMPLED PARAMETERS

The sampled parameters are then output in a file having the same name as the template *tpl* file, but with the *.params* extension. Typing the following command-line

```
fastsimcoal -t 1popDNArand.tpl -n 10 -e 1popDNArand.est -E10
```

will draw 10 sets of parameters according to the distributions found in file *1popDNArand.est*. It will then replace the corresponding keywords found in *1popDNArand.tpl* by the values just drawn, and write a *.par* file in the output directory as well as the randomly drawn values in the *.params* file, which should look like:

./1popDNArand/1popDNArand.params

NPOP	TEXP	RESIZE	MUTRATE	ANCSIZE	THETA
299	396	4.1771490	9.76393e-08	1248	5.83883e-05
202	561	28.9307465	9.35616e-09	5844	3.77989e-06
4311	1062	0.0630165	8.50824e-08	271	7.33581e-04
30824	580	0.0046161	7.65865e-08	142	0.0047214
669	188	46.5000659	7.59988e-08	31108	1.01686e-04
118	2132	63.0435917	9.61273e-08	7439	2.26861e-05
49854	722	28.3119282	2.93767e-08	1411462	0.0029291
45727	2870	2.7773714	3.18337e-08	127000	0.0029113
9719	575	15.5669955	6.23843e-08	151295	0.0012126
2021	888	0.0652933	1.12363e-08	131	4.54173e-05

Note that the complex parameter 2N was not output here as it was tagged has hidden in the *.est* file.

Also, the command line "*fastsimcoal –t 1popDNArand.tpl –n 10 –e 1popDNArand.est –E10*" will generate a total of 100 arp file (10 simulations for each of the 10 sets of randomly drawn parameters), to be found in the directory *./1popDNArand*.

7. USING PREDEFINED VALUES FOR A PARTICULAR EVOLUTIONARY MODEL

fastsimcoal allows you to simulate data under a given evolutionary scenario with fixed and predefined values of the model parameters. This is an alternative to the generation of random parameter values seen just above, and it uses pretty much the same syntax on the command line.

For instance,

```
fastsimcoal -t 1popDNArand.tpl -n 5 -f 1popDNA.def
```

will replace the keywords found in the *tpl* file by the values listed in the definition file `1popDNA.def` , and it will perform five simulation for each set of defined values. Note that the *tpl* file mentioned above was used in the previous section, and we show below the content and structure of a definition file.

DEFINITION FILE

A *def* file needs to have a first header line listing the **keywords** associated to each parameter. The keywords present in the template *tpl* file must **all** be listed in the *def* file, but there can be more keywords listed in the *def* file than those listed in the *tpl* file. These additional parameters will simply not be used.

The following lines then list the values of each parameter. For each parameter set, a *par* file will be created by replacing the keywords in the *tpl* file by the corresponding parameter values. In the example below, we list just 10 sets of parameter values.

`1popDNA.def`

NPOP	TEXP	RESIZE	MUTRATE	ANCSIZE	THETA
299	396	4.1771490	9.76393e-08	1248	5.83883e-05
202	561	28.9307465	9.35616e-09	5844	3.77989e-06
4311	1062	0.0630165	8.50824e-08	271	7.33581e-04
30824	580	0.0046161	7.65865e-08	142	0.0047214
669	188	46.5000659	7.59988e-08	31108	1.01686e-04
118	2132	63.0435917	9.61273e-08	7439	2.26861e-05
49854	722	28.3119282	2.93767e-08	1411462	0.0029291
45727	2870	2.7773714	3.18337e-08	127000	0.0029113
9719	575	15.5669955	6.23843e-08	151295	0.0012126
2021	888	0.0652933	1.12363e-08	131	4.54173e-05

Note that this *def* file corresponds to the *params* file we had just generated from the *est* file in the previous section.

The output files (*par* and *arp* files) will be placed in a directory with the same name as the *tpl* file but without the *.tpl* extension.

8. APPENDIX

COMMAND-LINE OPTIONS

As we have already seen, *fastsimcoal* include several command-line flags and options allowing you to parameterize simulations. These options can be listed by simply typing

`fastsimcoal` or `fastsimcoal -h`

We detail the command line options in more detail below. You can use a one-letter (case sensitive) shortcut after a single dash or a longer multi-letter full option after two dashes. The value required after "some options can be separated or not from the option by a white space. For instance, "`-n1`" and "`-n 1`" are both valid.

Shortcut (-)	Full (--)	Description
`-h`	`--help`	Prints a list of all available options
`-i`	`--ifile`	Name of parameter file. Example: `-i 1popDNA.par`
`-n`	`--numsims`	Number of simulations to perform per parameter file or sets of parameter (see –e option) Example: `-n 1000`
`-t`	`--tfile`	Name of template parameter file. The template file must have the extension *.tpl* Example: `-t 1popDNArand.tpl`
`-e`	`--efile`	Name of parameter prior definition file. Parameters are drawn from specified distributions and then substituted into template file. The file must have the extension *.est* . A template file must be provided for this option to work. Example: `-e 1popDNArand.est`
`-E`	`--numest`	Number of sets of parameters to draw from the prior distributions defined in the *est* file Example: `-e 1popDNArand.est -E 100`
`-f`	`--dfile`	Name of a parameter definition file. This file includes a list of sets of parameter values to be substituted in the *tpl* file. Listed parameter values are substituted in the template file to produce a valid parameter file (.par). This is an alternative to the random drawing of parameter values from distributions defined in the *est* file. Example: `-f 1popDNArand.def`
`-F`	`--dFile`	Same as *–f* option, but only it only uses the simple parameters defined in *est* file. Complex parameters are recomputed from the simple parameters. Note that this option must be used together with both the *–t* and the *–e* options. The *est* file is here necessary to define the relationships between the simple parameters.

		Example: `-F 1popDNArand.def`
`-g`	`--genotypic`	Generates *Arlequin* projects outputs (*.*arp*) in genotypic (diploid) format. Without this option, *arp* file are in the haploid format.
`-p`	`--phased`	Specifies that phase is known in *Arlequin arp* output files. Without this options phase is assumed unknown.
`-s`	`--dnatosnp`	Output DNA as SNP data where allele 0 is ancestral and allele 1 is derived.
`-I`	`--inf`	Generates DNA mutations according to an infinite site (IS) mutation model. Under this model, each mutation is supposed to occur at a different but random site on the DNA sequence. Under the IS model, if different mutations are allocated to the same DNA sequence position, they are generated independently, but marked to have occurred at the same position in the *Arlequin* output *arp* file.
`-d`	`--dsfs`	Computes the site frequency spectrum (SFS) for the **derived** alleles for each population sample, for all pairs of samples (joint SFS), and for all populations samples pooled (global SFS). Note that this is only done on SNP data. If your input file is about DNA sequences, then use the *–s* option to convert DNA to SNP data.
`-m`	`--msfs`	Computes the site frequency spectrum (SFS) for the **minor** allele for each population sample, for all pairs of samples (joint SFS), and for all populations samples pooled (global SFS). Note that this is only done on SNP data. If your input file is about DNA sequences, then use the *–s* option to convert DNA to SNP data.
`-H`	`--header`	Generates a header in the site frequency spectrum (SFS) files
`-q`	`--quiet`	Outputs minimal messages to the console instead of detailed messages
`-T`	`--tree`	Output coalescent tree for each non-recombining segment in nexus format. With recombination, a tree is output for each of the segment between recombination breakpoints.
`-k`	`--keep`	Number of simulated polymorphic sites to keep in memory before writing them to temporary files. If this option is not specified, a default value of 10000 used. This option is mostly useful when generating relatively long DNA sequences, with potentially tens of thousands of polymorphic sites. In that case, extended k to a large value will allow you to keep all genetic information in memory and slightly speed up computations. Note that setting up this *k* value to a very large number may imply prohibitive memory requirement for *fastsimcoal*, especially if you have large sample sizes i.e. thousands of individuals). In that case, it is preferable to adjust k, to prevent *fastsimcoal* tu use all your computer memory. In general the default value of *k*= 10000 gives very good results and does not need to be changed. Example: `-k 20000`
`-x`	`--noarloutput`	Does not generate Arlequin output files

SEQUENTIAL MARKOV COALESCENT APPROXIMATION

In this section, we briefly describe the principle of the sequential Markov Coalescent model that is used to approximate the classical ancestral recombination graph (ARG). For more details, please read the original publications (Chen, et al., 2009; Marjoram and Wall, 2006; McVean and Cardin, 2005).

In brief, McVean and Cardin (2005) proposed to approximate the coalescent with recombination over a chromosomal segment based on the ARG by simulating a series of trees that differ each by single recombination events, starting on the left of the segment and moving to the right of the segment.

They thus proposed the following SMC algorithm to simulate coalescent with recombination along a DNA segment of length L:

1) Simulate a normal coalescent tree without recombination on the left hand side of the segment, and set $x = 0$. This initial tree has a total size of $T = T_0$ generations.
2) Select the position of the next recombination event on the segment by drawing a random number $y \sim \text{Exp}(rT)$, where Exp is an exponential distribution, and *r* is the recombination rate per generation between adjacent base pairs.
3) If $x + y < L$, select a position on the current tree where to implement a recombination event, by drawing a uniform number in $1..T$.
4) Detach the branch below the recombining event from the tree, and color in blue the branch between the recombination event and the next interior node.
5) Erase the blue branch.
6) Let the recombining lineage coalesce with the other lineages attached to the tree, and thus reattach the detached tree to the left tree.
7) Record the total length of the new tree T' and set $T = T'$.
8) Let $x += y$
9) Repeat steps 2)-8) until $x + y \geq L$.

The steps 3 to 6 of the SMC algorithm are illustrated below

Illustration of the SMC algorithm

A recombination event ● occurs on a tree

The (blue) branch above the recombination event is erased

The recombining lineage is free to evolve backward in time …

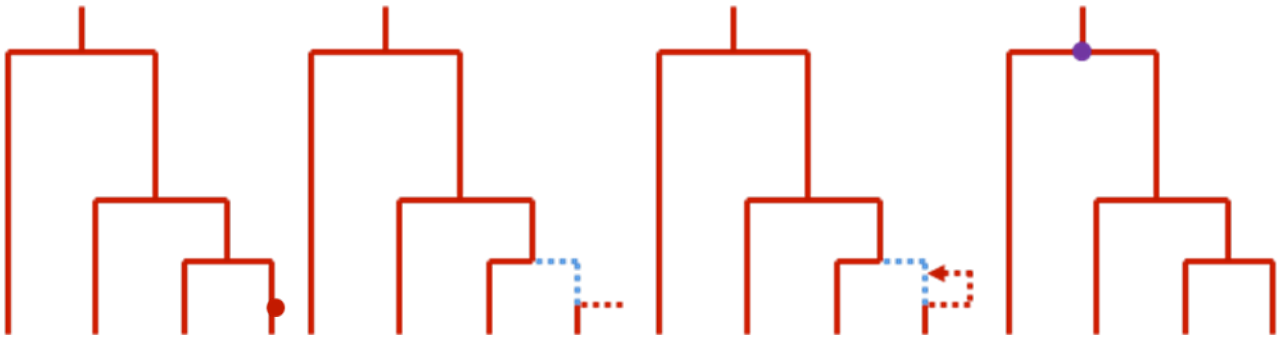
… until it coalesces with one of the lineage on the left tree …

… potentially changing the tree topology and the MRCA of the right tree

Marjoram and Wall (2006) proposed a slight modification of the SMC algorithm consisting in removing step 5 of the SMC algorithm above, and thus keeping the blue branch, and thus leaving the possibility to the recombining lineage to coalesce with it. The blue branches are only erased at the end of the simulation. This SMC' algorithm has been shown to give patterns of linkage disequilibrium more similar to those obtained by *ms* (Marjoram and Wall, 2006), and it is therefore this algorithm that is implemented in *fastsimcoal*.

With SMC', the recombining lineage can indeed coalesce with the blue lineages, and thus lead to a tree identical to the left tree, reflecting the fact that in the ancestral recombination graph, coalescent event can also occur between unsampled lineages.

Illustration of the SMC' algorithm



A recombination event ● occurs on a tree

The blue branch above the recombination event is kept

The recombining lineage is free to evolve backward in time, and can coalesce with the blue branch …

… leading to a tree identical to that on the left.

SITE FREQUENCY SPECTRUM

fastsimcoal can generate the site frequency spectrum for the derived allele (the unfolded site frequency spectrum) of the simulated data sets with the *-s* and *-d* command-line options as follows in the case of the relatively complex 3-population scenario described above:

```
fastsimcoal -i 3PopDNASFS.par -x -s -d -n10 -q
```

The *-s* option is necessary, here because the SFS can only be computed on SNP data, and the *-s* options tells *fastsimcoal* to treat DNA as SNP data (where the 0 allele is ancestral). Note that the *-x* option is there to prevent the output of *arlequin* input files, which would be quite large in this case (about 2.8 Mb for each simulation, with >50,000 polymorphic sites).

fastsimcoal will then generate for each simulation, the population-specific derived site frequency spectrum, all the pairwise joint frequency spectra, as well as the global site frequency spectrum obtained by pooling all demes.

For instance, the 10 SFS of deme 0 look like:

./3PopDNASFS/3PopDNASFS_DAFpop0.obs

```
10 observations
d0_0    d0_1  d0_2  d0_3  d0_4  d0_5  d0_6  d0_7  d0_8  d0_9 d0_10 d0_11 d0_12 d0_13 d0_14 d0_15 d0_16 d0_17 d0_18 d0_19 d0_20
9964672 8054  4046  3032  2190  2118  1548  1431  1358  1138  1096  1010  1020   845   688   803   673   743   668   614  2253
9964237 8407  4203  2833  2675  1904  1587  1443  1241  1093  1073   923   933   917   855   752   837   590   637   618  2242
9963855 8449  4104  3165  2700  2045  1662  1634  1267  1067   904   897   923   930   844   845   808   696   490   556  2159
9964355 8292  4251  3142  2248  1741  1552  1325  1359  1290  1147  1068   938   851   846   768   632   747   626   728  2094
9963918 8582  4285  3110  2391  2081  1506  1475  1279  1119   967   986   929   841   763   797   841   752   651   687  2040
9964418 8409  4483  2844  2241  1896  1580  1407  1307  1109   978  1015   857   821   720   777   697   599   654   647  2541
9965301 7894  4487  3061  2212  1719  1562  1384  1154  1059   962   992   800   799   785   673   715   664   700   571  2506
9963601 8491  4372  3120  2547  2090  1525  1450  1336  1164  1018   877   945   844   741   849   808   768   705   591  2158
9965241 7855  4273  2918  2294  1960  1503  1362  1103  1023   996   965   793   870   810   863   632   804   571   679  2485
9964845 8306  4674  2928  2205  1828  1519  1414  1142  1203  1062  1010   900   882   785   656   654   581   581   552  2273
```

Whereas this is the derived SFS, it has to be noted that the derived allele can have here a frequency of 20/20 since a mutation may have fallen on another portion of the tree leading to lineages sampled in deme 1 or in deme 2. Similarly, we output here the SFS entry 0/20, which just lists the number of monomorphic sites in deme 0.

In the following table we show the first two simulated joint SFS between deme 0 and deme 2

./3PopDNASFS/3PopDNASFS_DAFpop0.obs

10 observations

	d0_0	d0_1	d0_2	d0_3	d0_4	d0_5	d0_6	d0_7	d0_8	d0_9	d0_10	d0_11	d0_12	d0_13	d0_14	d0_15	d0_16	d0_17	d0_18	d0_19	d0_20
d2_0	9952704	7659	3669	2686	1832	1709	1214	1048	960	811	729	689	689	537	389	453	403	386	317	285	1329
d2_1	3558	32	27	25	22	38	19	25	36	21	19	22	21	35	22	23	24	27	22	24	141
d2_2	2359	38	38	29	46	14	28	36	36	26	31	23	33	23	32	23	8	10	28	32	177
d2_3	1341	37	58	29	31	63	27	38	37	28	15	17	31	27	16	38	23	21	43	23	139
d2_4	1059	38	28	8	24	27	43	34	36	38	43	21	25	26	21	13	46	24	17	22	159
d2_5	830	26	15	29	19	33	34	33	28	16	12	19	16	17	12	16	16	15	13	25	171
d2_6	2821	224	211	226	216	234	183	217	225	198	247	219	205	180	196	237	153	260	228	203	137
d2_0	9951575	8022	3824	2506	2278	1569	1249	1100	911	788	696	572	573	525	483	389	404	316	353	287	1160
d2_1	4443	50	28	38	28	44	38	26	25	34	57	40	42	42	32	35	30	21	28	24	183
d2_2	2278	33	45	42	59	30	25	41	29	25	26	17	42	36	34	26	29	26	24	27	205
d2_3	1422	56	19	14	20	19	16	26	26	36	12	25	27	26	37	24	16	8	16	27	170
d2_4	1185	30	29	36	34	44	39	38	35	27	43	35	19	28	27	52	55	24	33	15	220
d2_5	926	30	44	37	41	33	18	39	42	34	65	23	44	35	26	25	21	31	23	25	217
d2_6	2408	186	214	160	215	165	202	173	173	149	174	211	186	225	216	201	282	164	160	213	87

The other SFS are then just listed below in the file `./3PopDNASFS/3PopDNASFS_DAFpop0.obs`

Again, the SFS entries (n_2,j) and (i,n_0) are also listed here as some derived alleles can be fixed in either or both populations if the defining mutation has occurred on some lineages leading exclusively to genes sampled in the third population. Finally, note that SFS cannot be computed on DNA sequences presenting more than 1 million polymorphic sites ,due to prohibitive memory requirements. But for the 3-population model defined above, this would imply the simulation of sequences larger than about 200Mb.

It can be important to realize that the computation of SFS can perfectly be done on simulated data obtained for random values of the parameters, with the use of *tpl* and *est* files, thus allowing one to get the empirical distribution of SFS under a given model. These SFS could then be perfectly use as summary statistics to estimate parameters from an observed SFS under an ABC framework or for model choice (see below).

EXTENSION OF THE SMC' ALGORITHM TO MULTIPLE RECOMBINATION EVENTS

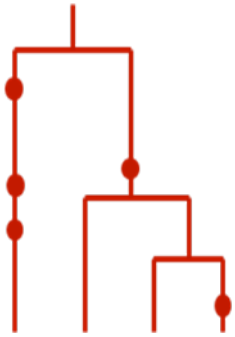
In order to simulate markers located at fixed recombination distances on the chromosome, we have extended the SMC' algorithm to allow for multiple recombination events between these markers.

The new algorithm is as follows:

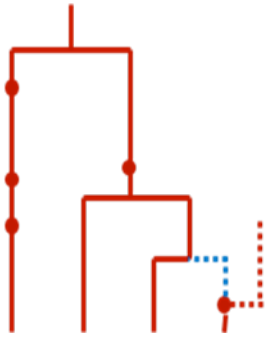
1) Simulate a normal coalescent tree without recombination on the left marker. This initial tree has a total size of $T = T_0$ generations.
2) Draw the number *z* of recombination events to occur between the two markers distant of *r* recombination units as $z \sim \text{Poisson}\,(\lambda = rT)$.
3) Draw *z* random numbers uniformly distributed between 1 and *T* to locate the positions of the *z* recombination events on the tree.
4) Locate the position of the most recent recombination event, and color in blue the branch above this event. All recombination events occurring on a blue branch are temporarily deactivated.
5) Detach the recombination event, and let it evolve until the time of the next active recombination event.

6) If the recombining lineage coalesces with a blue lineage, the lineage is colored back in red and the recombination events above this coalescent event become active again.
7) The next active recombination event is implemented, and we have a new recombining lineage free to evolve and to coalesce either with lineages on the tree or with other detached recombining lineages.
8) Steps 6-7 are repeated until only one lineage remains.
9) Remove all remaining blue lineages to keep a fully red tree.

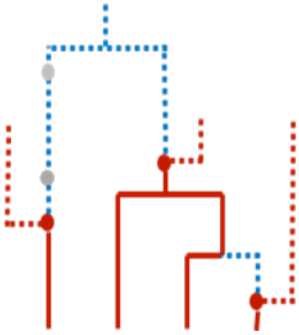
This algorithm is illustrated below



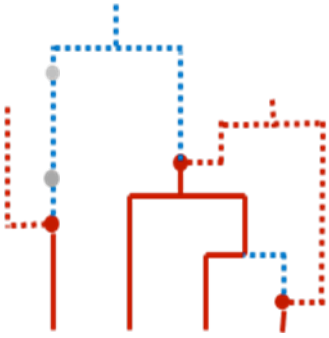
The *z* recombination events are randomly positioned on the tree



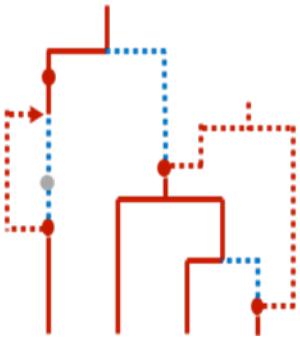
The first recombinant lineage is detached and evolves freely backward in time



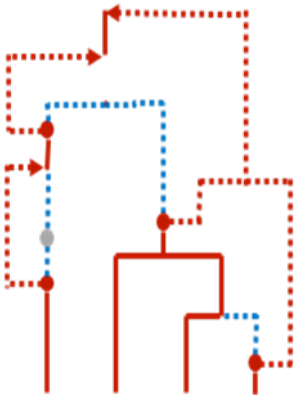
The recombination event on blue branches are deactivated (grayed)



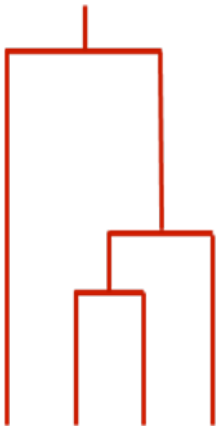
Two recombinant lineages can coalesce



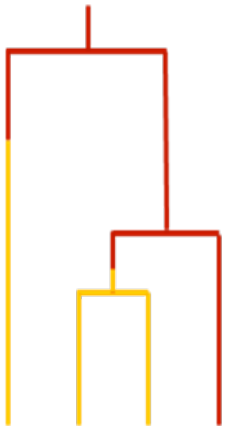
A recombinant lineage coalesces with a blue branch, which reactivates the recombination events above it.



The final coalescent event is reached.



All blue branches are erased to produce the final tree for the right marker.



The shared lineages between the two trees are shown in yellow.

INTEGRATION INTO APPROXIMATE BAYESIAN COMPUTATIONS (ABC)

fastsimcoal can easily be integrated into an ABC framework (see e.g. Beaumont, et al., 2002; Csillery, et al., 2010), using for instance the *ABCToolBox* framework (Wegmann, et al., 2010), which was specially developed to handle simcoal2, and which should be therefore able to accommodate *fastsimcoal* without modification.

ABCToolBox is available on http://cmpg.unibe.ch/software/abctoolbox/ and its manual on

http://cmpg.unibe.ch/software/abctoolbox/ABCtoolbox_manual.pdf.

An alternative, would be to use *fastsimcoal* built-in parameter sampler to generate *params* files and *Arlequin* arp files , which could then be processed by *arlsumstat* (available on http://cmpg.unibe.ch/software/arlequin35/) to produce summary statistics. A file combining these parameters and their associated summary statistics could then be used for ABC estimation, using different software, like

- ABCest3 (http://cmpg.unibe.ch/software/abc/) made by L. Excoffier
- ABCReg (Thornton, 2009) http://www.molpopgen.org/software/abcreg/
- ABCEstimator in ABCToolBox (Wegmann, et al., 2010) http://cmpg.unibe.ch/software/abctoolbox/

As mentioned above, a promising potential use of *fastsimcoal* would be to use the generated site frequency spectra directly as a set of summary statistics to be compared to an observed site frequency spectrum.

RUNNING FASTSIMCOAL ON A CLUSTER

For faster computations *fastsimcoal* can be launched on a Linux cluster.

As an example, the following bash file `fs_scratch.sh` can be submitted on a cluster running the freely available Sun Grid Engine (SGE, http://gridengine.sunsource.net/).

fs_scratch.sh

```
#!/bin/bash

#get local directory
directory=`pwd`

#$ -cwd
# specify resources needed
#$ -l h_cpu=48:00:00
# using a scratch directory, reserving disk space and enough files for simulations
#$ -l scratch=1,scratch_size=100M,scratch_files=5k
#$ -N fs_run
#$ -o fs.out
#$ -e fs.err
#$ -m a
#$ -q all.q

#Copying all files to scratch directory
cp * $TMP
cd $TMP

#Running fastsimcoal
./fastsimcoal -t 1PopDNArand.tpl -n 1 -e 1PopDNArand.est -E 1000 -q

#copying results from scratch to original directory
cp * $directory
cd $directory
```

The simple `qsub` command can be used to launch this bash file on a queue as

```
qsub fs_scratch.sh
```

Note that in *fs_scratch.sh*, *fastsimcoal* is using a scratch directory to write its output. This scratch directory is a temporary directory created by SGE on the node's hard disk, so that *fastsimcoal* does not use the net to transfer files from the node to the master during simulations, which usually increases speed.

To use the scratch file, one needs to specify how much RAM is needed on the disk (*scratch_size*), and how many files will be created (*scratch_files*). These parameters need to be carefully adjusted. Very long DNA sequences may require several Gb per simulations! Also keep in mind that when using *tpl* and *est* files, (*-n* × *-E*) arp files will be created (*1* × *1000* in `fs_scratch.sh`), as well as *-E* par files, and *-E arb* files. In addition, when very long DNA sequences are generated, with many polymorphic sites, *fastsimcoal* creates temporary files in a *./garbage* directory, which is then deleted after the *arp* file has been written. In the garbage directory, *fastsimcoal* will create as many files as the total number of sampled genes (sum of sample sizes) defined in the *par* or the *tpl* file.

Finally, do not forget to make *fastsimcoal* and your bash file executable on your Linux cluster. This is usually done with the *chmod* command as

```
chmod +x fastsimcoal
chmod +x fs_scratch.sh
```

The use of a scratch is not compulsory and a simpler bash file without use of a scratch directory would simply look like:

fs.sh

```
#!/bin/bash
#$ -cwd
# specify resources needed
#$ -l h_cpu=48:00:00
#$ -N fs_run
#$ -o fs.out
#$ -e fs.err
#$ -m a
#$ -q all.q

#Running fastsimcoal
./fastsimcoal -t 1PopDNArand.tpl -n 1 -e 1PopDNArand.est -E 1000 -q
```

Note that a bash file can be created on the fly and qsubmitted from within another bash file. Here is an example, which will submit 10 instances of `fs.h` on 10 different nodes, each one making 1000 simulations from different random parameter values .

```
./launch_fs.sh 10 1PopDNArand 10 1
```

launch_fs.sh

```
#!/bin/bash

#Laurent Excoffier December 2010
#
# The script will launch several instances of fastsimcoal

#Creating a shortcut for fastsimcoal
fs=fastsimcoal

if [ $# -ne 4 ]; then
   echo "Expecting the following values on the command line, in that order"
   echo "   Number of instances to run"
   echo "   Generic name of template file"
   echo "   Number of random parameter to draw"
   echo "   Number of simulations per sets of parameter values"
else
   #Using values from the command line
   numInstances=$1
   genericName=$2
   numEstimates=$3
   numSimsPerEst=$4
   echo "numInstances=$numInstances"
```

```
    echo "genericName=$genericName"
    echo "numEstimates=$numEstimates"
    echo "numSimsPerEst=$numSimsPerEst"
fi

#Directory for job console outputs
msgs=consoleOutputs
mkdir $msgs 2>/dev/null

echo "Launching ${numInstances} instances of $fs"
let COUNT=numInstances
instancesLaunched=0
while [ $COUNT -gt 0 ]; do
  curInst=fs_job${COUNT}.sh
  newDir=${genericName}_res${COUNT}
  mkdir ${newDir} 2>/dev/null
  cp ./$fs ${newDir}/.
  cp ./${genericName}.est ${newDir}/.
  cp ./${genericName}.tpl ${newDir}/.
  cp ./$fs  ${newDir}/.
  cd  ${newDir}
  let instancesLaunched=instancesLaunched+1
  if [ -e ./$fs ] ; then
    (
    echo "#!/bin/bash"
    echo "#$ -cwd"
    echo "# specify resources needed"
    echo "#$ -l h_cpu=48:00:00"
    echo "#$ -N fs${COUNT}_run"
    echo "#$ -o ../$msgs/fs${COUNT}.out"
    echo "#$ -e ../$msgs/fs${COUNT}.err"
    echo "#$ -m a"
    echo "#$ -q all.q"
    echo ""
    echo "chmod +x ./$fs"
    echo "./$fs -t ${genericName}.tpl -n${numSimsPerEst} -e ${genericName}.est -E${numEstimates}
-q"
    echo "rm ./$fs"
    ) > $curInst
    chmod +x $curInst
    qsub ${curInst}
  else
    echo "File $fs not found. Aborting instance $instancesLaunched"
  fi
  cd ..
  let COUNT=COUNT-1
done
```

COMPARATIVE SPEED TESTS

We report below speed tests comparing *fastsimcoal* to *ms* and *MaCS* running all on a Linux cluster made up of 2.6GHz AMD Opterons with 4 GB of RAM and 74 GB HD.

DATA SETS

The following test data sets were used in our comparisons.

	No. of populations	Diploid population size	Migration rate (gen^{-1})	Mutation rate ($gen^{-1} \times bp^{-1}$)	Recombination rate ($gen^{-1} \times bp^{-1}$)	Sample size per population (no. of genes)
1popNoRec	1	12500		2×10^{-8}	0	2000
1popSmallSample	1	12500		2×10^{-8}	1.2×10^{-8}	20
1popLargeSample	1	12500		2×10^{-8}	1.2×10^{-8}	2000
2popNoRec	2	6250	0.001	2×10^{-8}	0	1000
2popSmallSample	2	6250	0.001	2×10^{-8}	1.2×10^{-8}	10
2popLargeSample	2	6250	0.001	2×10^{-8}	1.2×10^{-8}	1000

The population, mutation and recombination parameters correspond to those used by Chen *et al*. (2009), in their comparison of *ms* to *MaCS* in the one population case.

RESULTS

n=2000, no recombination (CPU times in seconds)

Data set	No. of replicates	Sequence length	Program		
			ms	*MaCS*	*fastsimcoal*
1popNoRec	1000	1 Mb	1.1	11.1	9.5
	100	10 Mb	9.6	107.0	72.9
	100	100 Mb	147.9	1319.5	1038.1
2popNoRec	1000	1 Mb	1.2	12.5	9.3
	100	10 Mb	8.9	128.1	71.5
	100	100 Mb	161.2	1513.2	1099.9

Without recombination, *ms* is much faster than the two other programs based on the SMC' approximation, and *fastsimcoal* becomes increasingly faster than *MaCS* with larger recombination rates and with migration.

n=20, recombination (CPU times in seconds)

Data set	No. of replicates	Sequence length	Program		
			ms	*MaCS*	*fastsimcoal*
1popSmallSample	1000	1 Mb	0.344	0.242	0.095
	100	10 Mb	159.246	2.618	0.460
	100	100 Mb	x	26.124	4.364
2popSmallSample	1000	1 Mb	0.378	0.907	0.152
	100	10 Mb	165.507	9.094	1.080
	100	100 Mb	x	97.876	10.559

x : *ms* crashed

For small sample sizes (total *n*=20) and with recombination, the SMC' based programs are becoming much faster than *ms*, which fails to run for 100Mb sequences. For such small sample sizes, *fastsimcoal* is 2.5 to 9.3 times faster than *MaCS*. For *MaCS* and *fastsimcoal*, computing time increases approximately linearly with sequence length, as expected.

n=2000, recombination (CPU times in seconds)

Data set	No. of replicates	Sequence length	Program		
			ms	*MaCS*	*fastsimcoal*
1popLargeSample	1000	1 Mb	3.7	28.1	25.2
	100	10 Mb	x	327.5	235.7
	100	100 Mb	x	3700.8	2635.4
2popLargeSample	1000	1 Mb	3.9	33.3	25.9
	100	10 Mb	x	393.5	240.6
	100	100 Mb	x	4311.1	2684.7

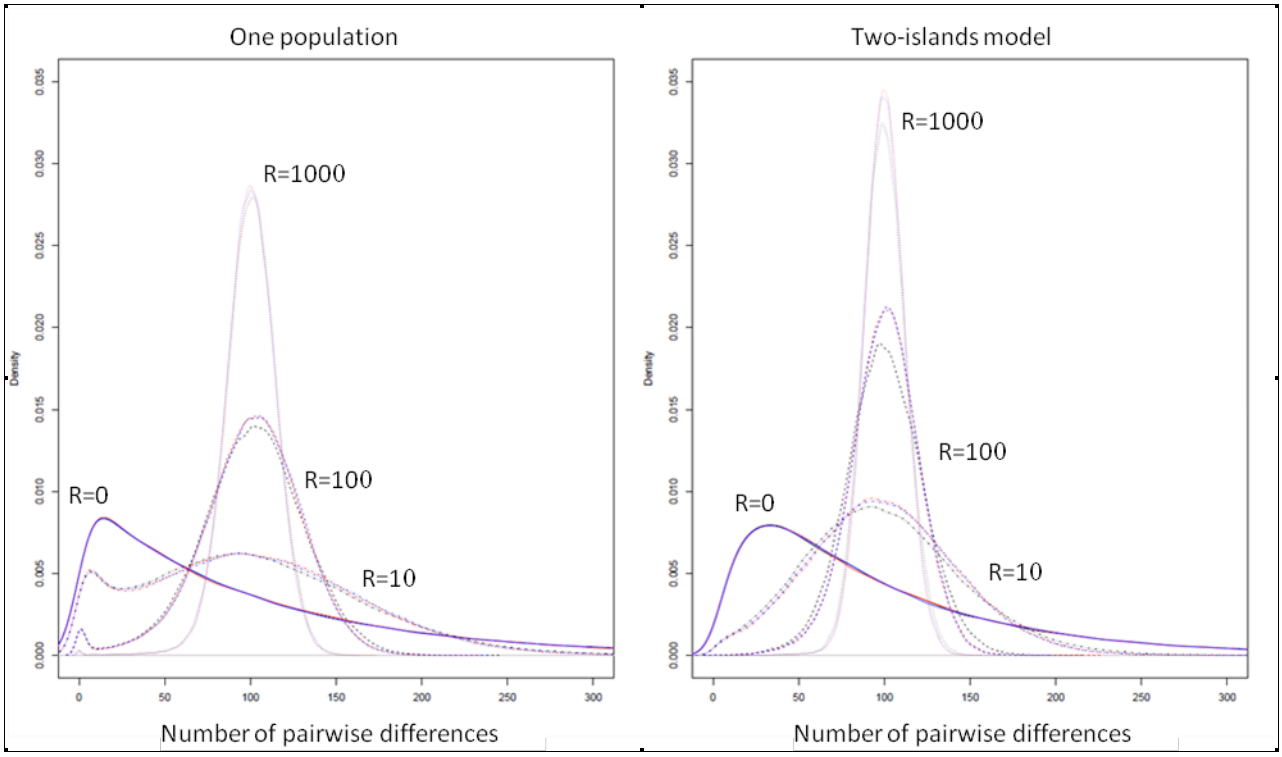
x : *ms* crashed

For large sample sizes (total *n*=2000), *ms* is actually faster than the two other programs for a "small" sequence of 1Mb, but failed to run successfully for longer sequences. For these large sample sizes, *fastsimcoal* is 1.2 to 1.8 times faster than *MaCs*. *fastsimcoal* computing time still increases approximately linearly with sequence length, which is not the case of *MaCS*, which becomes slightly penalized by larger sequences. Note however, that for 1Mb and 10Mb, we used *fastsimcoal* `-k` options allowing it to keep all simulated sites in memory before writing them to the output file, which was not possible for 100Mb sequences, which would use up too much memory.

COMPARATIVE PATTERNS OF SIMULATED MOLECULAR DIVERSITY

NUMBER OF PAIRWISE DIFFERENCES

We report below a comparison of the patterns of diversity within and between populations simulated by *ms*, *MaCs* and *fastsimcoal*.



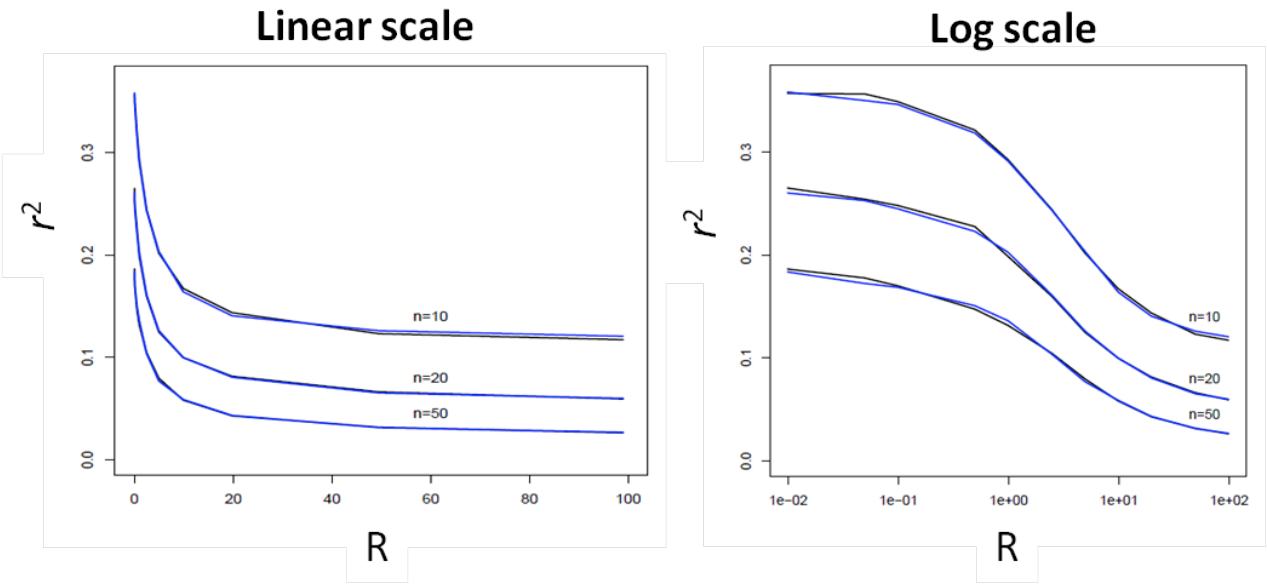
ms results are shown with a black line, *MaCs* with a red line, and *fastsimoal*, with a blue line. In all cases, the empirical distributions of the number of pairwise differences were computed from 100,000 simulations of the coalescent of 2 genes. The 2 genes were drawn from a single population for the one-population case, and were drawn each from a different population in the two-island model case. We used the following population parameters: $\theta = 4N\mu = 100$ for the entire sequence in the one population case; $\theta = 4N_0\mu = 4N_1\mu = 25$ and $M = 4Nm = 0.5$ for the two-island case; and $R = 4Nr$ (where *r* is the total recombination rate between the two ends of the sequence to be simulated) was varied between 0 and 1000, as shown above.

In all cases, *MaCS* and *fastsimcoal* lead to identical distributions, which is expected as they are both based on the same SMC' approximation. In absence of recombination *MaCS* and *fastsimcoal* give also exactly the same distributions as *ms*, but are just running 7-10 times slower, as was seen in the previous section. With very high recombination rates, the SMC-based approximation of *MaCS* and *fastsimcoal* is extremely close to the ancestral recombination graph (ARG) implemented in *ms*, in keeping with previous results (McVean and Cardin, 2005). For "intermediate" recombination rates (*R*=10, *R*=100), some slight differences do emerge between ARG- and SMC-based programs, and these differences are slightly more pronounced in the 2-island model. However, it seems that these differences are much less than differences due to the choice of different demographic, mutation, or recombination parameters.

LINKAGE DISEQUILIBRIUM

Previous studies have shown that patterns of LD were virtually undistinguishable between *ms* and SMC'-based programs (Chen, et al., 2009; Marjoram and Wall, 2006) along DNA sequences.

Here, we report a comparison of the average LD (as measured by r^2) between *simcoal2* and *fastsimcoal* between two SNP markers located at a given recombination distance expressed in *R* units.



The results are based on 20,000 simulations and confirm that average r^2 values are virtually undistinguishable between the two approaches.

9. REFERENCES

Beaumont, M.A., Zhang, W. and Balding, D.J. (2002) Approximate Bayesian computation in population genetics, *Genetics*, **162**, 2025-2035.

Chen, G.K., Marjoram, P. and Wall, J.D. (2009) Fast and flexible simulation of DNA sequence data, *Genome Res*, **19**, 136-142.

Csillery, K., Blum, M.G., Gaggiotti, O.E. and Francois, O. (2010) Approximate Bayesian Computation (ABC) in practice, *Trends Ecol Evol*, **25**, 410-418.

Excoffier, L. and Lischer, H.E.L. (2010) Arlequin suite ver 3.5: a new series of programs to perform population genetics analyses under Linux and Windows, *Molecular Ecology Resources*, **10**, 564-567.

Laval, G. and Excoffier, L. (2004) SIMCOAL 2.0: a program to simulate genomic diversity over large recombining regions in a subdivided population with a complex history, *Bioinformatics*, **20**, 2485-2487.

Marjoram, P. and Wall, J.D. (2006) Fast "coalescent" simulation, *BMC Genet*, **7**, 16.

McVean, G.A. and Cardin, N.J. (2005) Approximating the coalescent with recombination, *Philos Trans R Soc Lond B Biol Sci*, **360**, 1387-1393.

Thornton, K.R. (2009) Automating approximate Bayesian computation by local linear regression, *BMC Genet*, **10**, 35.

Wegmann, D., Leuenberger, C., Neuenschwander, S. and Excoffier, L. (2010) ABCtoolbox: a versatile toolkit for approximate Bayesian computations, *BMC Bioinformatics*, **11**, 116.