Rejector

Software for Population History Inference from Genetic Data
via a Rejection Algorithm

Matthew Jobin

Department of Anthropology
Stanford University
Stanford, CA
94305, USA

mjobin@stanford.edu

http:// www.rejector.org

0. Overview

REJECTOR is a software package comprised of four separate executables designed to act in concert for the purposes of parameter value estimation and comparison of alternate models of population history via rejection-based approximate Bayesian inference (Beaumont, Zhang et al. 2002). This inference method involves the calculation of summary statistics from experimental data, and then the calculation of those same statistics from numerous randomized replications of the data generated from a user-specified model. Comparison of the simulated summary statistics with those generated from the experimental data will indicate the probability that the parameter values used to generate the simulations are close to the real conditions that created the experimental data. Three of the programs were developed from scratch for the package: REJSTATS for summary statistic generation, PRIOR for preparation of randomized Simcoal2 input files, and REJECTOR to control program flow and compare summary statistic values. The fourth executable is a minor modification of the coalescent simulation software Simcoal2 (Laval and Excoffier 2004).

0.1 The Bayesian Paradigm and the Rejection Algorithm

The Bayesian paradigm is a statistical approach that uses probability as a direct measure of uncertainty, thus framing the research question in directly probabilistic terms (Shoemaker, Painter et al. 1999). Bayesian statistics incorporate the use of prior information in the form of distributions of parameters of interest. Statistical inferences generated in this fashion can operate on the entire set of data. These methods evaluate the probability of a given set of data given the parameter values. This method has been used to address questions of population history (Beerli and Felsenstein 1999; Pritchard, Seielstad et al. 1999; Stephens and Donnelly 2000; Beerli and Felsenstein 2001; Wilson and Rannala 2003), but for populations with a complex evolutionary history obtaining estimates can be computationally impractical (Marjoram, Molitor et al. 2003). The use of summary statistics provides the researcher with an efficient method for making estimations, along with the advantage of not having to specify a likelihood function explicitly when writing a simulation (Beaumont 2004). While methods which calculate posterior distributions from the full data set can be very accurate, they are not computationally efficient and do not scale well to the use of multiple loci. Methods based on summary statistics provide the best combination of accuracy and computational practicality (Beaumont, Zhang et al. 2002).

Methods employing the rejection algorithm have been demonstrated to be an efficient way to infer distributions of genetic and demographic parameters using polymorphism data (Macpherson, Ramachandran et al. 2004; Ramakrishnan, Storz et al. 2004). Under this method values for a parameter of interest are simulated from a prior distribution and accepted with a probability proportional to their likelihood. Since likelihoods are difficult to compute directly, a summary statistic is used in place of the full data set, and the candidate parameter is accepted based on the value of the statistic (Tavare, Balding et al. 1997). This method has been extended by the addition of another step, whereby a data set is simulated from the model and a summary statistic is derived from these data (Fu and Li 1997). The summary statistics of the simulated data and the observed data are compared, and the parameter value of interest is accepted if the

simulated statistic is within a tolerance value of the observed (Weiss and von Haeseler 1998).

$$= \frac{(O \quad E)}{E}$$

Multiple iterations of this method create posterior distributions of parameter values whose summary statistics fall within the tolerance value, which can then be used to make inferences about the model used to generate them.

0.2 Multiple Loci and Genetic Inference

While their lack of recombination and uniparental inheritance allow the mitochondrion and the non-recombining portion of the Y chromosome to provide unique insights into genetic history, it is incorrect to assume that the genetic histories of these regions are necessarily identical to the history of the species (Pamilo and Nei 1988; Mountain, Lin et al. 1992), as the number of loci sampled has been shown to have a strong effect on the reliability of population inferences made from gene trees (Mountain and Cavalli-Sforza 1997). Methods of genetic analysis that can incorporate multiple loci and combine disparate types of data (e.g. SNPs, STRs, RFLPs and sequence data) are capable of making inferences closer to the history of the whole genome, by averaging across the effects of each locus in the study.

0.3 The Coalescent

The coalescent is a stochastic process whereby lineages randomly "choose" their parents as they go backwards in time (Kingman 1982). As they are simulated backwards in time, lineages will at random intervals choose the same ancestor, and coalesce, until all lineages have coalesced into one. The coalescent can be used to construct random genealogies for use in testing models of population history, and since it is far more computationally efficient than classical forward-in-time methods (Rosenberg and Nordborg 2002) it is of particular use in simulation studies (Hudson 1990; Excoffier, Novembre et al. 2000; Nordborg 2001; Laval and Excoffier 2004; Anderson, Ramakrishnan et al. 2005; Ramakrishnan, Hadly et al. 2005), wherein summary statistics generated from a model of population history are compared to a statistic generated from observed data in order to test the model.

The rejection algorithm method described above can be coupled with use of the coalescent to generate simulated population histories that provide summary statistics to evaluate (Pritchard, Seielstad et al. 1999). The rejection-sampling method is computationally feasible thanks to use of summary statistics. It enables use of the Bayesian paradigm, with its many advantages, such as the incorporation of prior information and the ability to compare more than two models simultaneously. This method has been demonstrated to be computationally efficient while being capable of providing informative results (Macpherson, Ramachandran et al. 2004). Since each iteration does not depend on the state of other iterations, the work of computation can be

broken up and distributed to multiple computing nodes with relative ease. As stated in a recent review of software for population genetics analysis, approximate Bayesian methods using summary statistics can deal with complex models and be used to estimate parameter values of interest under any model (Excoffier and Heckel 2006). REJECTOR does just that, by implementing an Approximate Bayesian method termed rejection-based approximate Bayesian inference (Beaumont, Zhang et al. 2002).

0.4 Implementation of Rejection-based Approximate Bayesian Inference in REJECTOR

Implementing a rejection-based inference method requires the conversion of the rules of the method into software which can carry out the steps of sampling from prior distributions, simulating data sets and calculating summary statistics. We present here the software package REJECTOR for the purpose of inferring population histories by rejection-based approximate Bayesian inference. REJECTOR performs rejection-based inference given a broad range of genetic data, including DNA sequences, linked and unlinked STRs, linked and unlinked SNPs and linked UEP/STR systems.

The software takes advantage of the extant coalescent simulation program SIMCOAL2 (Laval and Excoffier 2004) by incorporating that program with three other executables that perform rejection algorithm analysis according to the following steps (Pritchard, Seielstad et al. 1999):

1. Calculate summary statistics from a data set.
2. Choose parameters of interest and draw their values from prior distributions.
3. Simulate a sample of the same size as the empirical sample, generating a coalescent tree (Hudson 1990) given the parameter values of step 2.
4. Calculate summary statistics given the data set generated by the simulation in step 3.
5. Record parameter values if all of the summary statistics are within a small tolerance value □ (Weiss and von Haeseler 1998).
6. Repeat step 1.

We have chosen SIMCOAL2 for incorporation into REJECTOR due to its ability to model multiple coalescent and migration events at each generation, and its ability to work with complex demographic scenarios. The authors of SIMCOAL2 felt that the program could be incorporated into an Approximate Bayesian Computation framework (Laval and Excoffier 2004), which REJECTOR accomplishes. All of these three executables have been written in the C++ language, making heavy use of the Standard Template Library containers for speed and generic functionality. We have compiled and tested REJECTOR under Mac OS X, Linux and Windows. REJECTOR can be run on a single machine using one or multiple processor cores, or it can take advantage of Apple's Xgrid technology to run in distributed mode across a network of Macintosh computers.

1. Installation and Quick Start

1.1 Installing under Mac OS X

Please ensure that you have already installed Apple's Developer Tools, which come with your copy of Mac OS X. For OS X 10.5 Leopard, they reside on the same DVD as the operating system install, under "Optional Installs".

When you download REJECTOR from the website (the link resides on the left side of the page), you will receive a file called `rejector.zip`. You can expand this file into the `rejector` folder simply by double-clicking.

Take the folder named `rejector` created by this action, and place it somewhere on your system convenient to you.

You will now need to access the folder from the command line. If you are using a Macintosh, you can find the program that gives you access to the command line in your Utilities folder, which is in your Applications folder. The program is called Terminal. If you double-click it you will see a terminal window appear.

The standard install of Mac OS X will begin with your terminal pointing to your Home folder. To navigate to the rejector folder, use the `cd` and `ls` commands. To list the contents of the folder you are in, type `ls` and hit return:

```
ls
```

To change into a directory inside the one you are in, type:

```
cd <directory-name>
```

To go up to the parent directory of your current folder, type:

```
cd ..
```

Using these you should be able to get to the `rejector` directory. For further tips on command-line navigation, consult any book on UNIX basics, or search online.

Once you are in the rejector directory, you can type `ls` to ensure the contents are there. You should see these directories and files:

`code` -> all of the program files necessary to create the program
`examples` -> a folder full of example input files and associated output files to test
`Makefile` -> the file that controls the building of the software
`scripts` -> PERL, shell, and R scripts for automation and post-processing of data
`rejexample.txt` -> an example input file for initial testing purposes
`rejclean` -> shell script for clearing out intermediate files

The `code` folder contains all of the program files necessary to create the executables. To compile the software, you only need to type one command:

```
make
```

This command will commence compilation of the four executables that comprise REJECTOR. When the compilation is done, four executable files will appear in the rejector directory:

```
prior
rejector
rejstats
simcoalrej2_1_2
```

These are all necessary for the program. DO NOT remove them from the folder unless you want to move ALL FOUR to another directory together. During normal operation of REJECTOR, all four executables will need to be in the same folder, along with your input file.

1.1.1 Using the Provided Universal Mac Binaries

We have provided Mac binaries on the website compiled under Leopard for use with Xgrid simulations involving grids comprised of both Intel and PPC Macs (see pg. 57). Universal binaries are not compiled using `make`; in order to re-compile universal please search on "compile universal binaries" at http://www.apple.com. We will continue to provide up-to-date universal binaries, but unless you intend to run REJECTOR in Xgrid mode using both Intel and PPC Macs, you should use the standard install method above.

1.2 Installing under Linux

Please ensure that your Linux system has the following installed: `g++, gcc, make, perl, unzip`. These are standard for most Linux varieties.

Most graphical interfaces will expand this file into the `rejector` folder simply by double-clicking. If you are using a purely command-line variant of Linux, you will simply need to navigate to the folder containing the `rejector.zip` file and type:

```
unzip rejector.zip
```

The rest is very similar to the Macintosh version above. Download the folder, then consult the Macintosh install section if you need to in order to navigate to the folder and `make` the executables.

NOTE: If, when compiling under Linux, you get run-time errors that seem to complain about being unable to find some version of GLIBC, it is possible your system is not configured correctly. Ensure your Linux machine has `gcc` installed (this is almost always true). You can find out where it is using:

```
locate gcc | less
```

If your system does not have `locate` installed, search manually in `/usr/lib` or `/usr/lib/local/`. Consult a guide on using Linux if still unsure. Once you know where `gcc` lives, you can set a path to it using:

```
setenv LD_RUN_PATH <gcc location>
```

Then try again using `make`.

1.3 Installing under Windows

In order to use REJECTOR in Windows, you will need to install the free Cygwin environment that allows the compilation and execution of Linux-like programs. Visit the Cygwin website:

http://www.cygwin.com

Click the "Install or update now" link in the middle of the page and run the software when requested.

It's easiest to choose to "Install from Internet".

Install Cygwin where you like. The default "C:\cygwin" is a good choice. We will assume that's where you have put it in the following instructions, so if you have not, adjust accordingly. It's best to install for "All users" if possible.

When you see the "Select Packages" window, click the "View" button in the top right corner until it says "Full" next to it. This displays all the packages that can be installed with Cygwin. We only need a few.

Find the entries for the following packages:

```
gcc-core
gcc-g++
make
perl
unzip
```

Click the labels "Skip" next to these entries – they should turn into numbers (e.g. as of this writing 3.4.4-3 for gcc-core, but they will increase over time.). These selections will auto-select other parts of the list – that's normal.

Hit "Next" and let the packages download and install.

A folder named "cygwin" has been created, defaulting to C:\cygwin.

To test the software, from your Start menu find Cygwin->Cygwin Bash Shell. This will start a command-line shell pointing to your Cygwin home directory, which defaults to C:\cygwin\home\<your user name>

Now go to the rejector site at:

http://www.rejector.org

Click on "Rejector Package" at the left and Save it to your Cygwin home directory (as defined above).

From the Cygwin Bash Shell, type

```
ls
```

You should see the file rejector.zip in the listing, as you just placed it there. Now type:

```
unzip rejector.zip
```

to unzip the file, which places the `rejector` directory in the same directory you are in. Now type:

```
cd rejector
```

to enter that directory. If you want, type `ls` again to see the various `rejector` subfolders – `code, docs, examples`, etc. To compile the software, you should then type:

```
make
```

Your computer will spend some time compiling the software – when it is done, the executables `prior.exe, rejector.exe, rejstats.exe, simcoalrej2_1_2.exe` should all have been added to the directory. These are all necessary for the program. DO NOT remove them from the folder unless you want to move ALL FOUR to another directory together. During normal operation of REJECTOR, all four executables will need to be in the same folder, along with your input file.

After testing your installation (see pg. 12), you will be able to look at output with Excel or a similar spreadsheet, prepare data for R using Cygwin's perl installation, and visualize data by downloading the Windows version of R. If you are not familiar with working in Cygwin or other UNIXy environments, pick up an introductory book at your local bookstore or check around online.

NOTE: For all of the examples below, Windows users must add an ".exe" to the end of the executable names under Cygwin. This means:

`prior` becomes `prior.exe`
`rejstats` becomes `rejstats.exe`
`rejector` becomes `rejector.exe`
`simcoalrej2_1_2` becomes `simcoalrej2_1_2.exe`

1.4 Testing your Installation

A single input file, `rejexample.txt`, is included in the top `rejector` directory, with your new executables, so that you can test the software right after compilation. As soon as the software compiles, make sure you are still in the `rejector` directory and type:

```
./rejector –f rejexample.txt –r 10
```

This should run for about a minute. If it completes without an error, the executables are working properly. You can open up the sole output file, `rejexample.rej0.txt`, but with only 10 runs there may not be anything in it. You can clean out the intermediate files by invoking the `rejclean` script:

```
./rejclean
```

To test the other example input files, move the from the `examples` folder into the top-level `rejector` folder. For example:

```
mv examples/handtest.txt  .
```

from the top-level `rejector` folder, this will move the `handtest.txt` folder up to the `rejector` folder. Once there, you can run it by typing:

```
./rejector –f handtest.txt –r <the number of runs you
would like>
```

The general case for most `rejector` runs is:

```
./rejector –f <input file> -r <desired number of runs>
```

1.5 Tutorial

STOP! Before going through this tutorial, make sure you have installed and tested your copy of REJECTOR as described in the sections above.

Now that you have installed REJECTOR and verified that the software is working properly on your system, let's go through an example of how to use REJECTOR to make parameter estimates using genetic information available in a public database.

For this tutorial, we will be using data downloaded from the International HapMap Database (http://www.hapmap.org), a large SNP repository containing human data from several populations. The example files, `genotypes_chrY_CEU_r23a_nr.b36_fwd.txt` and `genotypes_chrY_YRI_r23a_nr.b36_fwd.txt` can be found in the `examples` directory of the `rejector` folder. They were downloaded from HapMap.org with the HapMap Genome Browser (B36) using the function "Download SNP Genotype data" from the Reports and Analysis section of the Browser. Please see http://www.hapmap.org/tutorials.html.en for details on how to use the HapMap Browser.

Note: You will need use of the command line for this tutorial. If you have already installed REJECTOR, you will have used the command line at least once. For Windows, start the Cygwin environment as described on page 10. If using Mac OS X, use Applications:Terminal. On Linux, check some Linux help docs if you are not familiar with the command line.

Step 1 – Convert the HapMap data for use in Rejector

The two HapMap files used as examples in this demonstration contain SNPs from the Y chromosome. Each file contains individuals from a different population – Americans of European descent for `genotypes_chrY_CEU_r23a_nr.b36_fwd.txt, and` Yoruban Africans for `genotypes_chrY_YRI_r23a_nr.b36_fwd.txt`. We will use the provided Perl script `hapmapconvert.pl` to merge the data from the two populations and convert it into a single file that will form the data section of our new REJECTOR input file.

Open a command line window as described in the sections above, then navigate using the `cd` command to your `rejector` directory. If you type `ls` to list the files and directories, you should see a listing similar to that below:

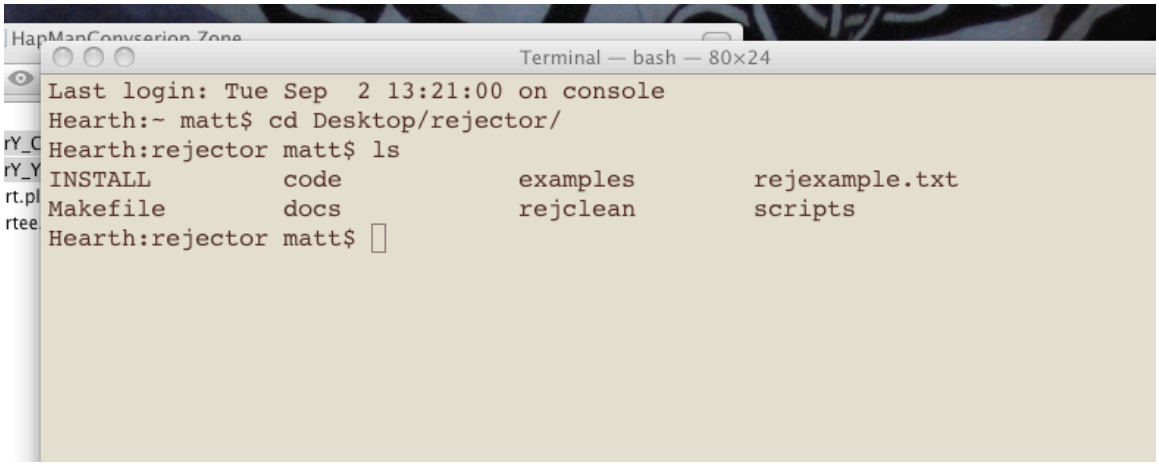


Figure 1. Output of `ls` command in the top `rejector` directory.

This is the place where we will do the work of conversion, leaving a finished, converted REJECTOR input file ready to run in the same directory as the executables. Next type the following command:

```
perl scripts/hapmapconvert.pl
```

If you see a warning message about your system's inability to run Perl, you will not be able to use any of the scripts that come with REJECTOR. Mac and Linux systems should always come with Perl installed – see http://www.perl.org to install a copy in the unlikely event your system does not have it. If you see this message under Cygwin in Windows, go back to the Windows install section on page 10 and reinstall Cygwin from the beginning, making sure you grab the `perl` package with the others.

When the script runs, you will be prompted with:

```
Please type the name of the OUTPUT file you want to
use. It will contain the
data this script will convert from HapMap to Rejector.
```

Let's type the file name `tutorialexample.txt` here. Do so and hit RETURN. Your next prompt will be:

```
Please type in a mutation rate for the SNPs, or hit
return to leave at 1.0e-09
```

Let's just leave the mutation rate for the SNPs as-is for now. Hit RETURN and your next prompt will be:

```
Please type the name of a HapMap genotype file to
convert,
or X to end and output.
```

Now we must enter the directory and name of the first of the two files to convert. Type in this response, then hit RETURN:

```
examples/genotypes_chrY_CEU_r23a_nr.b36_fwd.txt
```

You will see this response:

```
Reading examples/genotypes_chrY_CEU_r23a_nr.b36_fwd.txt

Type in the population identifier for the individuals
in this file to be used in rejector.
```

This prompt asks for a population identifier to be used for all the individuals in this file. Let's keep the CEU identifier from the file name, so type in:

```
CEU
```

and hit RETURN. You will then see the prompt:

```
Number of regions read from
examples/genotypes_chrY_CEU_r23a_nr.b36_fwd.txt: 672

Please type the name of a HapMap genotype file to
convert,
or X to end and output.
```

Now we will enter the second file. Type in:

```
examples/genotypes_chrY_YRI_r23a_nr.b36_fwd.txt
```

and then when prompted give this second file the identifier YRI. You will then again see the prompt:

```
Please type the name of a HapMap genotype file to
convert,
or X to end and output.
```

This time, type `X` and RETURN. The data will be converted and the output placed in the file `tutorialexample.txt`.

Step 2 – Describe a Model

The file you now have, `tutorialexample.txt`, contains converted data ready for use in REJECTOR, but is NOT YET a REJECTOR input file. We only have data at

this point; we have not described the sort of model we wish to investigate, nor any prior distributions for parameters of interest.

Open `tutorialexample.txt` with a text editor. On a Mac, try the included TextEdit app, or try TextWrangler. On Windows, use WordPad or your favorite editor. When you open the file, at the top you should see something like:

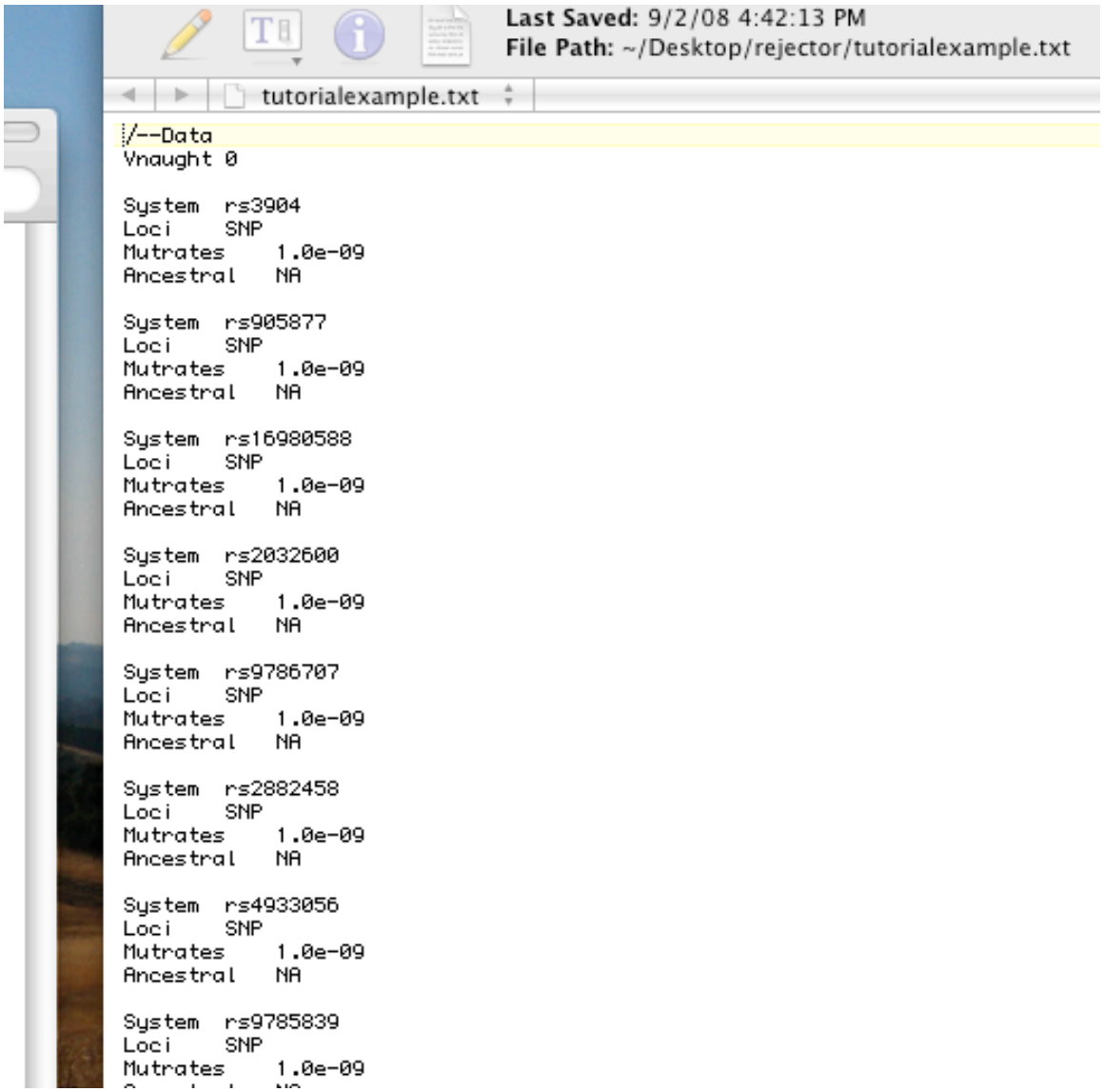


Figure 2. The top portion of the data section of `tutorialexample.txt` before priors have been added, showing the first systems (SNPs).

The above shows the first of a long list of "systems" (genetic sites, in this case SNPs) with associated mutation rates, as added by the script. If you scroll down halfway through the file, you'll see the data itself:

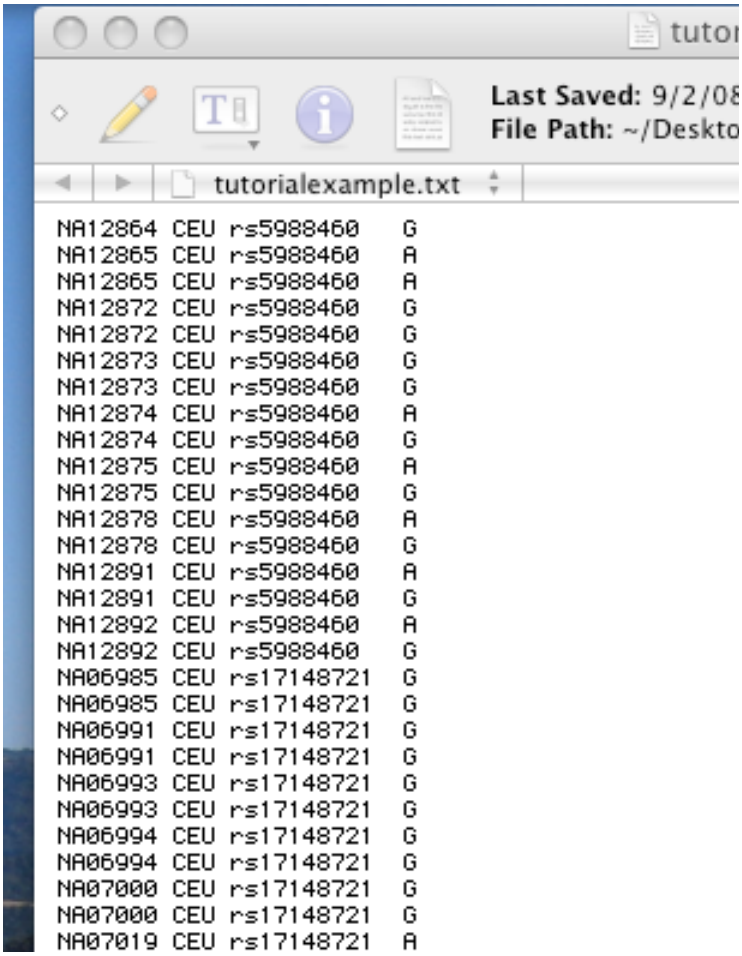


Figure 3. Partway through the data section of `tutorialexample.txt`, showing individual names, population ID, SNP name, and SNP state.

The data consists of four columns, left to right: the name of the individual sampled, the population ID assigned by you to that individual's file in `hapmapconvert.pl`, the name of the SNP and then the state of the SNP. Note each individual has two entries for each SNP, as the HapMap files contained diploid information.

The next thing we will do is describe a model of population history for this data containing prior distributions of parameters of interest. We have input two population IDs into our data, so we must tell REJECTOR to work with EXACTLY two populations. Let us work first on a fairly simple model, one where we only wish to investigate the size of one of the populations – the individuals from the CEU HapMap file. This will be done given a defined time of divergence, size for the second population, no growth in population size, and no migration between the populations once they diverge. This is a simple model, but one which will be illustrative for our purposes here. In other words, our one parameter of interest, and thus the only one which will be given a prior distribution, will be the size of the CEU population.

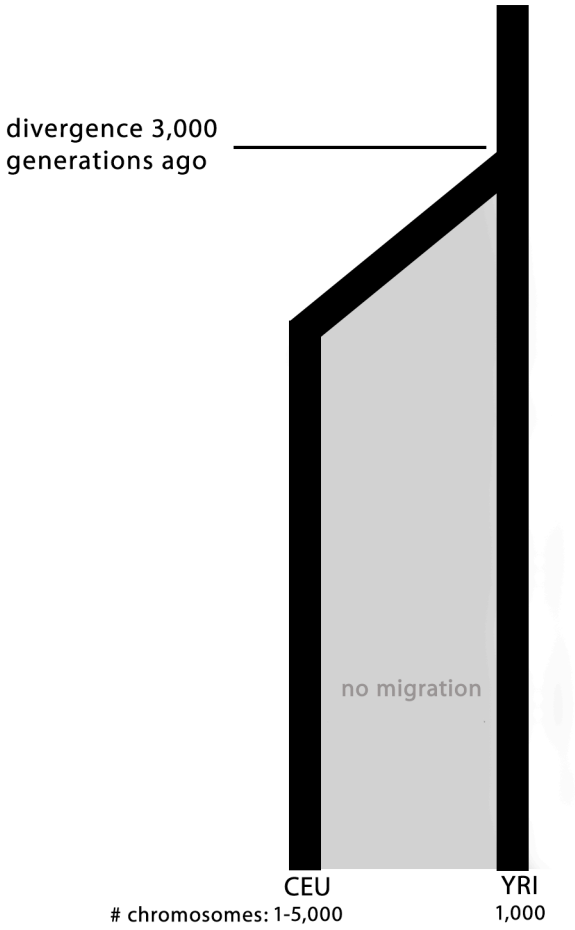


Figure 4. Diagram of the model used in `tutorialexample.txt`.

It is often helpful to begin describing a model using a prior example, in other words opening a previously used model and modifying it, though they can be easily written from scratch. In this case, there is a model ready made. Open up the text file `hapmapmodel.txt` in the `examples` folder using a text editor. It should look like this:

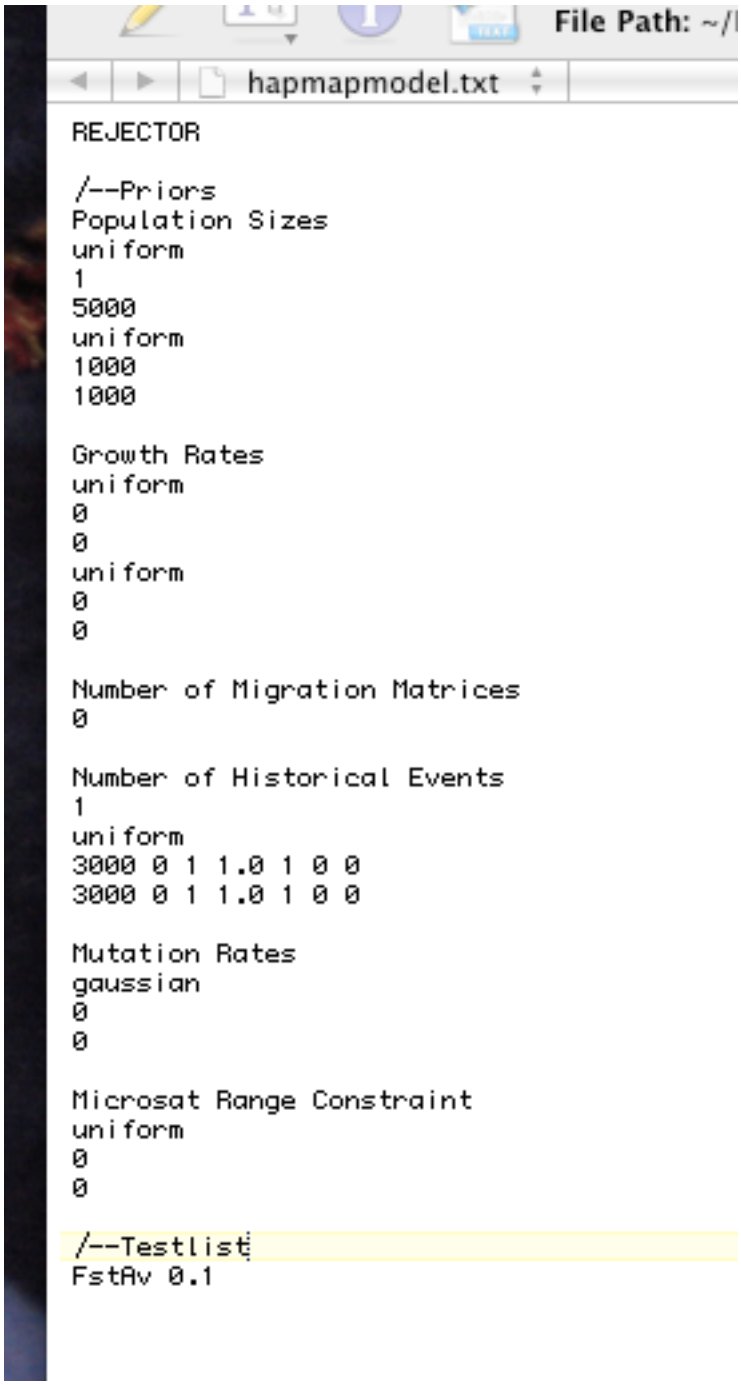


Figure 5. The file `hapmapmodel.txt`, which we will paste to the front of `tutorialexample.txt`.

The text of Figure 5 describes, from top to bottom: the size of the two populations in alphabetical order (CEU, then YRI), the growth rates (also in alphabetical order), the migration matrices (here we are not using any), one historical "event" (in this case the split of the two populations 3,000 generations ago), a (currently dummy) mutation rate modifier, and range constraints on microsatellites in the file (irrelevant here as we are using all SNPs). The last entry, under `/--Testlist`, describes the single summary

statistic we are using, F_{ST} averaged across loci, and the tolerance value, which is the relative amount any simulated iteration may vary in its F_{ST} value from that of the HapMap data and still be accepted into the posterior distribution.

Please see the section on Priors starting on page 29 for detailed information on how priors are constructed, but for now note just a few things. First, you could change the time of divergence by altering both entries of 3,000 in the historical event, say to 4,000. So long as the numbers are the same, every iteration of REJECTOR will use the same time of divergence. Second, if you changed one of the time of divergence values to a different value, you would give REJECTOR a uniform prior distribution – a range – of values from which to randomly sample. This is true of our one prior distribution in the file, the 1 to 5,000 you can see near the top, which describes a uniform prior distribution for the size of the CEU population.

Now open `tutorialexample.txt` in your text editor. This contains the converted data we have just examined. Copy all the text in `hapmapmodel.txt` and paste it at the TOP of `tutorialexample.txt`. Hit RETURN a few times to put some spaces between the line starting with FstAv 0.1 and the line starting with /--Data. When that is done, save `tutorialexample.txt` – you now have a working REJECTOR input file! The top of it should look like this:

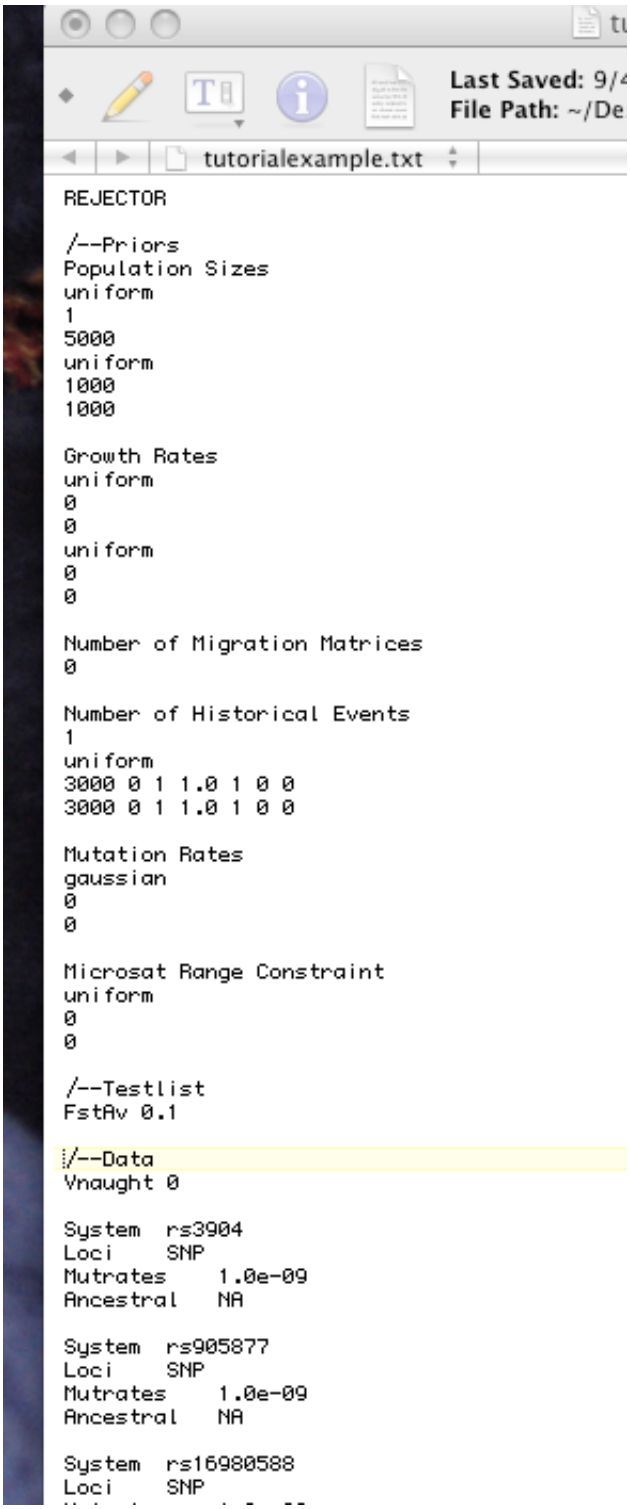


Figure 6. The `tutorialexample.txt` file once the file `hapmapmodel.txt` has been pasted in at its front. This is now a working and ready REJECTOR input file.

Step 3 – Run your model in Rejector

Now that we have created an input file containing data and our model, we can run that file in REJECTOR. Move the file we have been working on – `tutorialexample.txt` – into the top `rejector` folder. It should now be in the same directory as the four executables you created in Sections 1.1 to 1.4. Thus, if you type `ls` at the prompt, you should see something like the following:

```
Hearth:~ matt$ cd Desktop/rejector/
Hearth:rejector matt$ ls
INSTALL                 prior                   scripts
Makefile                rejclean                simcoalrej2_1_2
code                    rejector                tutorialexample.txt
docs                    rejexample.txt
examples                rejstats
Hearth:rejector matt$
```

Figure 7. The results of the `ls` command once the file `tutorialexample.txt` has been moved into the `rejector` directory. Note the presence of the four executables - `prior, rejector, rejstats, simcoalrej2_1_2`.

Once you have confirmed this, it is time to run REJECTOR.

NOTE: It is a good idea to run a file for a few iterations as a test, just to make sure everything works and to gauge how long the program takes per iteration. Start by typing the following command:

```
./rejector –f tutorialexample.txt –r 10
```

The program will start, and after the first iteration completes will count up to 10 iterations, estimating the time remaining:

```
Hearth:rejector matt$ ./rejector -f tutorialexample.txt -r 10
Rejector
Running with a limit of 10 runs.

Completed 1/10 iterations so far. Estimate 1.2 minutes to go.
```

Figure 8. REJECTOR in the process of running, showing estimated time remaining.

On the test system, 10 runs of `tutorialexample.txt` take roughly a minute. Getting the most from REJECTOR requires consideration of the appropriate summary statistics, CPU power available, and time available to run the job. Take a look at the later sections of the guide for detailed descriptions of these factors, including ways to speed the process using multiple CPUs or computers, but for now, let's guess that about 1,000 runs will give us some useful results.

While it is not necessary, its good practice to clean out the intermediate files and results file from the test run. Type `./rejclean` to invoke the script that cleans any

remaining intermediate files, and then delete the results file using `rm tutorialexample.rej0.txt`.

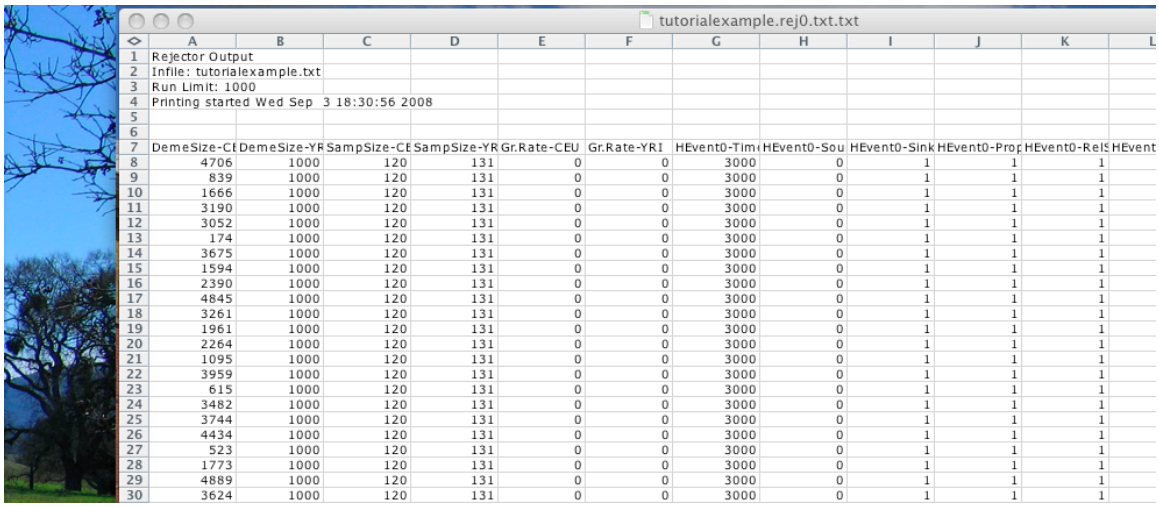
Now for the real thing. Let's make a few changes to the invocation of REJECTOR. We'll increase the number of runs to 1,000 using the `-r` switch, and also add the `-p` switch so that all iterations will print in the output file, regardless of whether they meet the criteria. This is often good for exploratory first runs, just to see where the summary statistic values lie.

```
./rejector -f tutorialexample.txt -r 1000 -p
```

Now it's time for coffee, or the equally pleasing beverage of your choice. REJECTOR is very CPU intensive. It invokes SIMCOAL2 to simulate populations that fall under the model you provide, and then evaluates summary statistics and compares them to the values calculated on the given data. On the test system, 1,000 iterations of `tutorialexample.txt` took a bit over an hour. Of course, if you are using a multi-core computer, as many do these days, only one core will be used, so you can easily use your machine for other tasks without slowing down REJECTOR.

Step 4 – Process results

The 1,000 runs of REJECTOR will be complete when your command line window gives you back control (i.e. you can now type in it again). When the run completes, the results will be saved in `tutorialexample.rej0.txt`. REJECTOR starts a new results file for every 1,000,000 entries – this can be changed by altering the `#define MAX_OUTFILE_SIZE` statement near the top of rejector's main.cpp source file and re-compiling. The next results file would be `tutorialexample.rej1.txt`, then `tutorialexample.rej2.txt` and so on, but since we ran only 1,000 iterations, only `tutorialexample.rej0.txt` will be in your directory. You can directly open this file in a spreadsheet program like Excel. To do this, start Excel, then Open a document. Make sure the dialog is set to "All Documents" to allow text files to open. Once the opening process begins, set the import filter Excel gives you to "Delimited" and then make sure "Tab" is the chosen delimiter. The opened file should look like the one below:



tutorialexample.rej0.txt.txt

	A	B	C	D	E	F	G	H	I	J	K	L
1	Rejector Output											
2	Infile: tutorialexample.txt											
3	Run Limit: 1000											
4	Printing started Wed Sep 3 18:30:56 2008											
5												
6												
7	DemeSize-CE	DemeSize-YR	SampSize-CE	SampSize-YR	Gr.Rate-CEU	Gr.Rate-YRI	HEvent0-Tim	HEvent0-Sou	HEvent0-Sink	HEvent0-Proc	HEvent0-Rel	HEvent
8	4706	1000	120	131	0	0	3000	0	1	1	1	
9	839	1000	120	131	0	0	3000	0	1	1	1	
10	1666	1000	120	131	0	0	3000	0	1	1	1	
11	3190	1000	120	131	0	0	3000	0	1	1	1	
12	3052	1000	120	131	0	0	3000	0	1	1	1	
13	174	1000	120	131	0	0	3000	0	1	1	1	
14	3675	1000	120	131	0	0	3000	0	1	1	1	
15	1594	1000	120	131	0	0	3000	0	1	1	1	
16	2390	1000	120	131	0	0	3000	0	1	1	1	
17	4845	1000	120	131	0	0	3000	0	1	1	1	
18	3261	1000	120	131	0	0	3000	0	1	1	1	
19	1961	1000	120	131	0	0	3000	0	1	1	1	
20	2264	1000	120	131	0	0	3000	0	1	1	1	
21	1095	1000	120	131	0	0	3000	0	1	1	1	
22	3959	1000	120	131	0	0	3000	0	1	1	1	
23	615	1000	120	131	0	0	3000	0	1	1	1	
24	3482	1000	120	131	0	0	3000	0	1	1	1	
25	3744	1000	120	131	0	0	3000	0	1	1	1	
26	4434	1000	120	131	0	0	3000	0	1	1	1	
27	523	1000	120	131	0	0	3000	0	1	1	1	
28	1773	1000	120	131	0	0	3000	0	1	1	1	
29	4889	1000	120	131	0	0	3000	0	1	1	1	
30	3624	1000	120	131	0	0	3000	0	1	1	1	

Figure 9. What the top of `tutorialexample.rej0.txt` should look like in Microsoft Excel.

See Section 4 beginning on page 41 for detailed instructions on examining your results in Excel, but for now simply use Sort under the Data heading in Excel, tell Excel your data has a header row, and sort on the first column marked FstAv(0.1), which should have F's and T's under it. This will sort the data into T's and F's. You can now grab all of the entries under the DemeSize-CEU column – that's the column containing the estimates of the size of the CEU population – that also have a T in the FstAv(0.1) column. This means these entries were accepted within the given tolerance value (0.1) for the summary statistic FstAv (see page 61). You can now do things such as find the average of these values with the Formula Bar in Excel (see Excel's help if you are not familiar with these functions and also see the R `rejsum` command on page 48).

More importantly than just looking at the results or getting an average, and more helpful if you have a lot of results, is visualizing them in a density plot, particularly against the original prior distribution.

To prepare your REJECTOR results for analysis in the statistical package R, we must first prepare them using the provided script `gluedat.pl`. This will put the results in a format readable in R, as well as putting together multiple results files if you have them. While in your top-level `rejector` directory, type:

```
perl scripts/gluedat.pl tutorialexample
```

The resulting file, as shown in the command window, will be `tutorialexample-out.txt`. This is the file we will load and examine in R, containing all the results of this run.

Step 5 – Visualization in R

The R statistics package is free, flexible and powerful. You can find a copy for your system at http://www.r-project.org/. Obtain and install a copy according to the instructions on the R site.

Start R, and once the environment is running, load the provided set of scripts using either the "source" command or its GUI equivalent as shown. Find the file `rejector.r` in the `rejector/scripts` directory and load it:

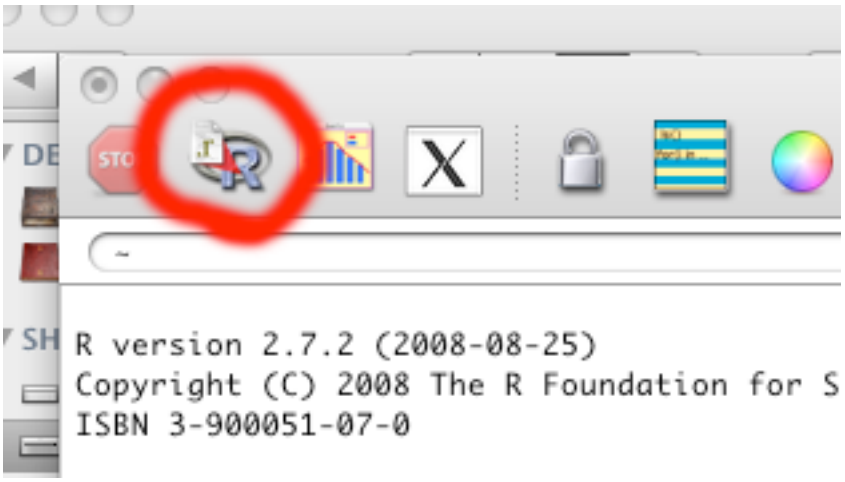


Figure 10. The Source command button in the Mac version of the R GUI.

You can also load `rejector.r` from the R command line using the `source` command directly:

```
source("(location of your rejector
directory)/scripts/rejector.r")
```

Once you have done this, the custom scripts for handling REJECTOR output will be loaded into R. Now use the R GUI's Change Working Directory command (under Misc on the Mac version) and set the working directory to the `rejector` directory where your output file, `tutorialexample-out.txt`, resides.

To look at the column names of your output, use the `rejnames` command:

```
rejnames("tutorialexample-out.txt")
```

The output should be similar to the following:

```
> rejnames("tutorialexample-out.txt")
 [1] "DemeSize.CEU"      "DemeSize.YRI"      "SampSize.CEU"      "SampSize.YRI"      "Gr.Rate.CEU"       "Gr.Rate.YRI"       "HEvent0.Time"      "HEvent0.Source"
 [9] "HEvent0.Sink"      "HEvent0.Prop"      "HEvent0.RelSize"   "HEvent0.NewGrRate" "HEvent0.MMid"      "X"                 "Test.Alpha..."     "FstAv.0.1."
[17] "X.1"               "Test.Alpha....1"   "FstAv.0.1..1"
>
```

Figure 11. Output of the `rejnames` R command, showing the column headers of `tutorialexample-out.txt`.

What we now want to examine is the posterior distribution, or those values of the CEU population size for which the value of FstAv fell within the tolerance value of that calculated from the experimental data taken from HapMap. We will visualize this on the

same figure as the prior distribution to show how the summary statistic and tolerance value constrain the posterior distribution, providing an estimate of the true parameter value given the other parameters of the model.

The command to visualize the distribution is called `rejoned`. It is a complex command, with many options. What we need to know for our current purposes is the following: the name of the file (`tutorialexample-out.txt`), the column number of the summary statistic (the first entry of "FstAv.0.1." or column 16), the column containing the values for our parameter of interest (the "DemeSize.CEU" column, or column 1), and the total number of iterations (1,000, as printed in `tutorialexample.rej0.txt`). In addition, in order to superimpose the prior distribution we selected in our model with the resultant posterior distribution, we must look up what that distribution was in our original input file, `tutorialexample.txt`. There we will find that the prior distribution was uniform, with a minimum of 1 chromosome and a maximum of 5,000 chromosomes. Putting all of this together (see page 49 for more on `rejoned`) gives us the R command:

```
rejoned("tutorialexample-out.txt", 16, 1, 1000,
"uniform", 1, 5000)
```

Running this command in R should produce a pdf file named `tutorialexample-out.pdf` in the same directory. You can open this using Adobe Reader or simply by double-clicking on it using most operating systems. The results should be look similar to that below:

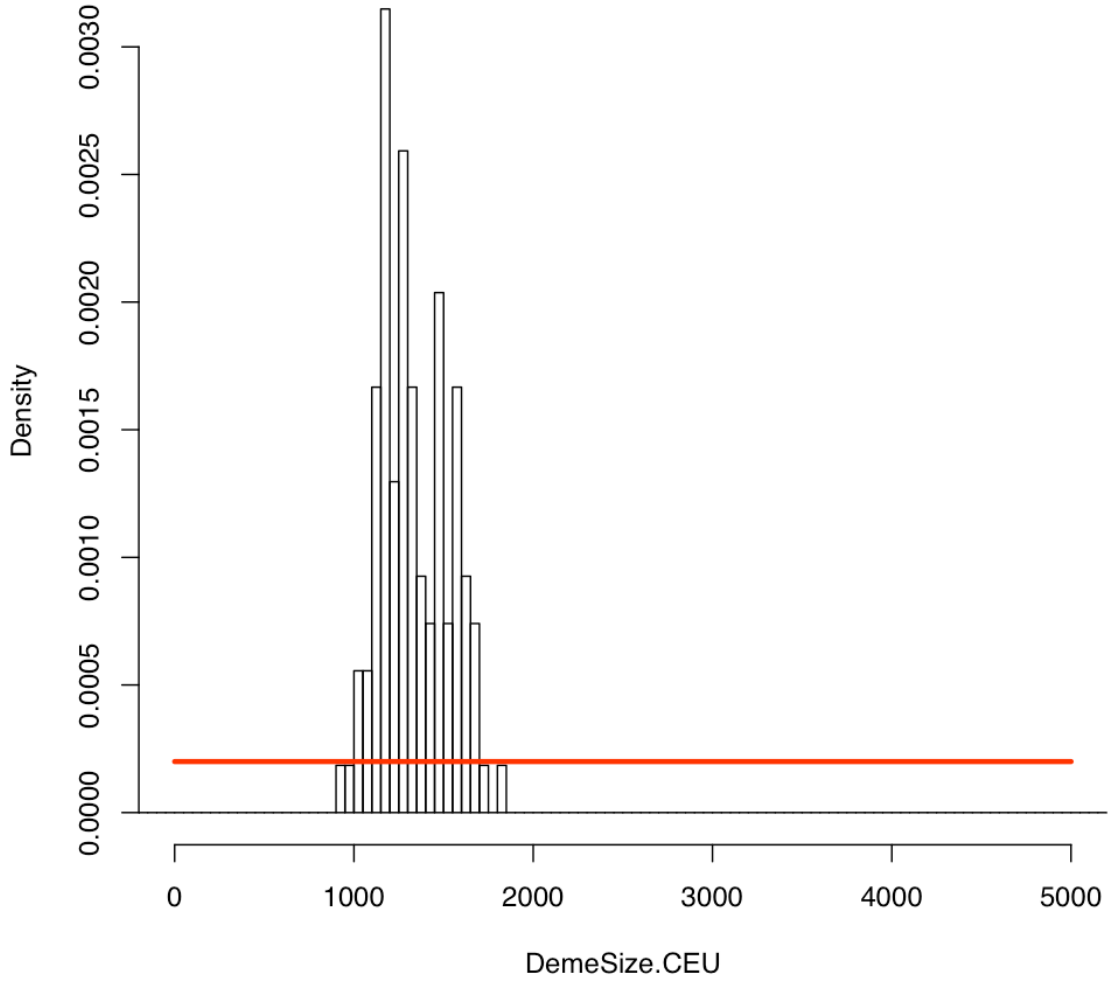


Figure 12. Posterior distribution (histogram) superimposed over prior distribution (solid line) for results of run from `tutorialexample.txt`.

Step 6 - Considering results

Figure 12 superimposes a density plot of the original prior distribution we put into `tutorialexample.txt` with the posterior distribution formed from 1,000 iterations of REJECTOR using the summary statistic FstAv and a tolerance value of 0.1. In other words the histogram in the figure contains all the iterations of REJECTOR for which FstAv was within 0.1, or 10% of the same statistic calculated from the original HapMap data. The prior distribution gave no information but to set the possible sizes of the CEU population to between 1 and 5,000 chromosomes, but as you can see only population sizes between 1,000 and 2,000 were accepted, with a peak just over 1,000. REJECTOR thus indicates that, given all the other parameter values of the model, the population size of the CEU individuals should be in this range.

2. Program Flow

REJECTOR is both the "glue" program that controls the other three and the program that compares the output of the simulated and experimental data via rejection-based approximate Bayesian inference. In a run of REJECTOR, as shown in Figure 13 below, REJSTATS is first called to analyze the experimental data, then PRIOR, SIMCOAL2, and REJSTATS are called in a loop with a number of iterations specified in the initial call to REJECTOR. Each iteration of the loop calls, in sequence, PRIOR to generate the randomized input for SIMCOAL2, then SIMCOAL2 to create the coalescent simulation, then REJSTATS to calculate summary statistics on the simulated data. Finally, REJECTOR reads in the output of simulated data and compares it to the experimental data, keeping track of the simulated data that is within a user-specified tolerance value of the experimental data.

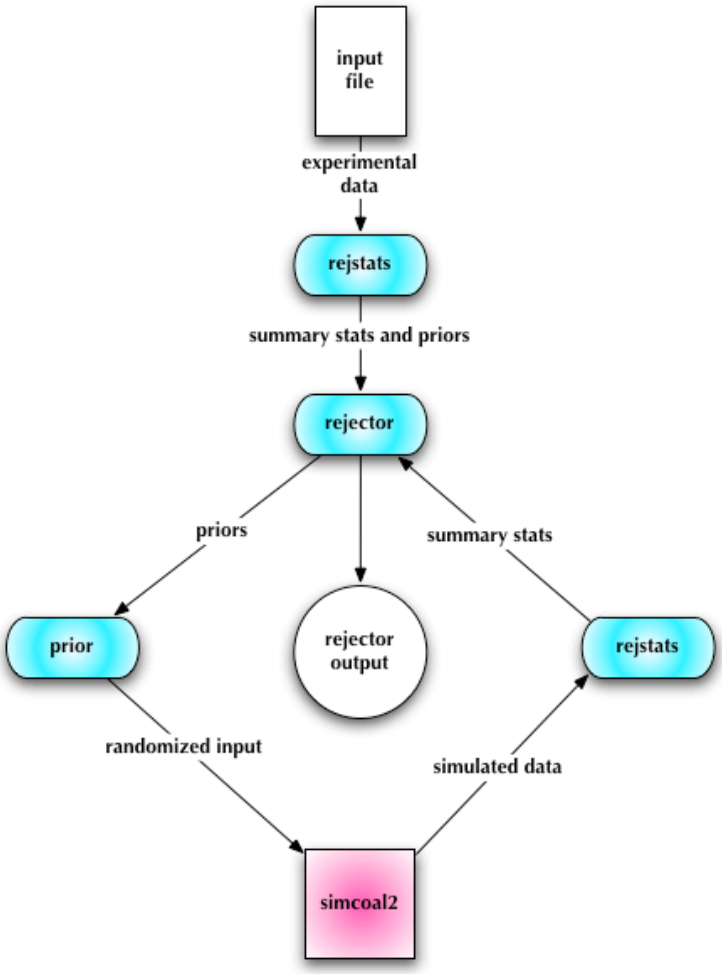


Figure 13: REJECTOR program flow.

3. Structure of Input Files

Important: **Note that to be used in** REJECTOR**, an input file must include a Priors section. It is possible to run** REJSTATS **in standalone mode (see pg. 89) without a Priors section, but not** REJECTOR**.**

REJECTOR input files are tab-delimited text files. An easy way to create and modify them is in Microsoft Excel, or another similar spreadsheet application which can work with text files.

The input file format is quite scalable, allowing large numbers of loci of many different conformations, but REJECTOR must be accurately told how the data is structured to avoid errors. The input file is broken into sections, outlined below. Pieces of an Example file have been included in each section, and then again as one piece on page 39.

3.1 Heading Line

The first line of the infile must be `REJECTOR`, all capitals. Otherwise the program gets all confused.

```
REJECTOR
```

3.2 Priors

The section beginning with `/--Priors` and up until `/--Testlist` defines the priors used in the program. These contain prior information that you can give to REJECTOR in order to influence the posterior distributions it calculates. The essence of REJECTOR is the input of sensible priors (or wide uniform priors if no good priors exist) for parameters of interest, so that you can estimate these parameters with the rejection algorithm.

The priors describe the type of population model you wish to test. The entries are structured using the time-backwards approach common to coalescent simulations – in other words you begin in the present and work backwards in time. The Priors section is structured fairly similarly to the input files for SIMCOAL2, so you can find additional information about them there (http://cmpg.unibe.ch/software/simcoal2/) – one of the biggest differences is that for every numerical entry in a SIMCOAL2 input file, there are two input values for REJECTOR and one word noting the sort of prior distribution these

two values describe. The following is an example Priors section, which will be explained in the sections following:

```
/--Priors
Population Sizes
uniform
100
10000
uniform
100
1000
uniform
100
1000

Growth Rates
uniform
0
0
uniform
0
0
uniform
0
0

Number of Migration Matrices
0

Number of Historical Events
1
uniform
2000 0 1 1.0 1 0 0
15000 0 1 1.0 1 0 0

Mutation Rates
gaussian
0.001
0.0001

Microsat Range Constraint
uniform
5
25
```

Each entry consists of three parts, a parameter that describes the shape of a distribution and two numerical parameters. There are three distribution shapes available in this release of REJECTOR:

• uniform: The two numerical parameters describe the minimum and maximum of a uniform distribution, respectively, between which all values are equally likely to be picked.

• gaussian: The first numerical parameter describes the mean, and the second the standard deviation, of a gaussian distribution.

• gamma: The first numerical parameter describes the scale, and the second the shape, of a gamma distribution.

Please note that REJECTOR will only allow sensible values for any prior value. For example, if you specify a gaussian distribution for the size of a population, REJECTOR will ignore (by resampling until it gets a positive number) random deviates that set the population size to zero or less, as such a value will be nonsensical to SIMCOAL2.

If you want any entry to have a fixed value, in other words to not vary from iteration to iteration, you can do so by giving it a uniform distribution with the minimum and maximum values the same. For example, if you wanted to tell REJECTOR that the size of a population was exactly 1000, you would enter it as below:

```
uniform
1000
1000
```

3.2.1 – Assignments to Population by Alphabetical Order

Before we get into the pieces of the input file, a word about assignment to population. The Population column in the Data section (see pg. 36) defines which population each entry is assigned to. The number of entries for Population Sizes, Sample Sizes, and Growth Rates MUST exactly match the number of populations you have listed in your Data section, as defined in the Population columns of your Data. For example, if in all of your data entries you have listed one of the following names: Adygei, Balochi, Burusho - then you MUST have three entries for Population Sizes, Sample Sizes, and Growth Rates. In addition, if you add a migration matrix, it must be as long and as wide as there are populations in the Data column. Know thine data!

The entries for Population Sizes, Sample Sizes, and Growth Rates are assumed to be in alphanumeric order relative to the Population column of your data, In other words, if in all of your data entries you have listed one of the following names: Adygei, Balochi, Burusho, then the first entries for Population Size will be taken for Adygei, the second for Balochi, and the third for Burusho. The same goes for Sample Sizes and Growth Rates. If we count upward from 0, then the Adygei are Population 0, Balochi are Population 1 and Burusho are Population 2.

3.2.2 – Population Sizes, Sample Sizes, Growth Rates

The entries under Population Sizes, Sample Sizes, and Growth Rates each describes these values for one of populations you have listed in the Data section (see

above for how they are ordered.) The population sizes are the size of the whole population, while Sample Sizes are the individuals the programs tracks during run time.

Growth rates define the exponential parameter *r* in the equation:

$$N(t) = N(0)e^{rt}$$

IMPORTANT: Because we are working *backwards* from the present in REJECTOR, giving a *negative* growth rate means population *expansion*, forward in time.

3.2.3 - Migration Matrices

The example above gave 0 migration matrices, meaning no migration will occur. If you wish to use migration matrices, you can list them in the form shown in the example below:

```
Number of Migration Matrices
2

uniform
3
0.0 0.0 0.01
0.0 0.0 0.01
0.01 0.0 0.02

0.0 0.0 0.03
0.0 0.0 0.03
0.01 0.0 0.05

gaussian
3
0.01 0.01 0.01
0.01 0.01 0.01
0.01 0.01 0.02

0.001 0.001 0.001
0.001 0.001 0.001
0.001 0.001 0.002
```

The first number after the word "Matrices" tells REJECTOR how many matrices to read. For each matrix it then reads the distribution type. The two matrices that follow are matrices of the two numerical parameters for the distribution type – minimum and maximum for uniform, and so on. The matrices are given id numbers starting with 0 for the first one and given to SIMCOAL2 in the order they are read. The first migration matrix (matrix 0) is always the first one used – the others can only be switched to using a historical event (see below).

3.2.4 – Historical Events

Using a historical event, you can set a time (going backwards in generations from the present) when a population diverges, grows or shrinks, or when the simulation

changes migration matrices. Historical events follow the style in the SIMCOAL2 guide (http://cmpg.unibe.ch/software/simcoal2/), save that there are two lines, representing the two sets of numerical parameters for one of the three distribution types. They are of the form:

```
uniform
5000 0 1 1 1 0 0
6000 0 1 1 1 0 0
```

In each line of numbers there are seven entries. This what each of them means:

1st number: When the event occurs, in generations *backwards* from the present.
2nd number: The source population of any migrants.
3rd number: The sink population (destination) of any migrants.
4th number: The proportion of migrants in source going to sink. 1 means all.
5th number: The relative new size of the sink. 1 means same size.
6th number: The new growth rate of the sink.
7th number: The new migration matrix to use.

The entries are all doubled to allow them to become priors, just like the entries for Population Size etc. above. So the example above says:

Between 5000 and 6000 generations ago, on a uniform distribution, all of the lineages from Population 0 move to Population 1 (going *backwards* in time). Population 1 stays the same size prior to this. The new growth rate is 0 and use migration matrix 0.

Bear in mind some parts of a historical event cannot really have priors. The source, sink, and new migration matrices cannot have priors associated with them, as there is no sensible way to compute this.

3.2.5 Historical SNPs

NOTE: This WILL NOT WORK with more than one independent loci or with no recombination. This will ONLY work with the modified version of SIMCOAL2 included with REJECTOR.

NOTE!: The time of the historical SNP chosen by Prior is **requested** of SIMCOAL2, but that node may not be available on the tree. The **actual** generation when the SNP is implemented is reported in the output file under the heading "SC2Time-X", where X is the number of the hisotrical event.

I have made an addition to SIMCOAL2 to allow the user to specify exactly when, and in what deme, a SNP occurs. In this case, the numbers for the historical event mean slightly different things, as follows:

1st number: When the event occurs, in generations *backwards* from the present.
2nd number: The source population of any migrants.
3rd number: MUST BE -1, to denote this as a SNP event.
4th number: Number of the chromosome (ALWAYS 0 for now).
5th number: Number of the locus

6th number: The new growth rate of the sink.
7th number: The new migration matrix to use.

For each SNP to force, you will also need to set the SNP's Mutation rate to -1 to shut of SIMCOAL's normal SNP mutation function.

3.2.6 – Mutation Rates and Microsat Range Constraints

The entry for Mutation rates is no longer used and will be removed in the next release of the software – see the mutation rate listings for each individual locus in the Data section.

Microsat Range Constraints define how many "steps" in size a microsatellite is allowed to take while mutating. This is used to constrain the microsatellite to biologically-realistic behavior. This is usually a uniform distribution, or a uniform distribution with no variation (both values the same). Note that a value of "0" here implies no constraint, so to give no constraints to microsatellite range, use:

```
uniform
0
0
```

3.3 Testlist

The section beginning with /--Testlist allows the user to specify which summary statistics REJECTOR will calculate. You can use any number of summary statistics in the same run of the software. For a description of the summary statistics used, see Calculations (pg. 60).

As of the present release, the following is a list of all calculations REJECTOR will recognize. Note that the spelling and capitalization is exact – an unrecognized name will result in an abort of the program.

```
Heterozygosity
HeterozygosityAv
HeterozygositySub
HeterozygosityAvSub
DeltaMuSquared
DeltaMuSquaredSub
Td
TdSub
Dsw
Beta
BetaAv
```

```
BetaSub
BetaAvSub
MultAlleleLD
MultAlleleLDp
RangeofLocus
SampleSize
SampleSizeSub
MaxLength
MinLength
MeanLength
NumDiffAlleles
CompleteHeterozygotes
CompleteHomozygotes
Haplotypes
Jstatistic
MicrosatVariance
DerivedFraction
NucleotideDiversity
Nei
Fst
```

A tolerance value must be placed next to each summary statistic name. This value is used by REJECTOR to perform rejection algorithm calculations. If you are using REJSTATS as a stand-alone program, any dummy numerical value will be acceptable, as REJSTATS will only be reading the value for the purposes of spacing out the input file correctly. Placing a zero in the alpha column is a good way of reminding yourself that you are using REJSTATS alone.

If you wish to run all calculations, you may simply put the single entry `ALL` in the Testlist section, instead of listing all of the summary statistics. This is done in the example above. The `ALL` name still requires a tolerance value.

Note that some summary statistics require much more computation time than others. `MultAlleleLDp`, for example, often requires a large proportion of the computational resources for every run, as do the two-way statistics D_{sw}, $\partial\mu^2$, and T_D, especially if there are a large number of demes. For many stand-alone runs of REJSTATS, the amount of time taken is not usually long enough for this to be significant, but for long runs of REJECTOR, where REJSTATS is called over and over, consider which statistics you might not need, especially if they are computationally intensive.

3.3.1 Choosing Tolerance Values and Run Times

It is quite reasonable to ask what a good tolerance value might be for any run of REJECTOR. It is, in a sense, the same sort of question one could ask regarding the

common 0.05 value taken to represent significance in classical statistics. There is, somewhat unfortunately, a continuum of possible results, between an infinite tolerance that simply prints random parameter values chosen from your priors and an infinitesimal one that admits nothing into the posterior distribution. In using REJECTOR, keep an exploratory frame of mind. If you intend to run 1,000,000 iterations of a simulation over a week, first run it for 1,000 iterations to get a good guess at how many data points your posterior distribution will have. If there is none, widen the tolerance and try again. If seemingly everything gets though, first re-check the Sensitivity of Summary Statistics section on page 64 to make sure your summary statistic is actually sensitive to the parameter you are estimating, then try again with a reduced tolerance.

In general, bear in mind these rules of thumb:

1. Make sure you have at least one summary statistic that is sensitive to each of the parameters in your model that have prior distributions.

2. Try short test runs of your file – anything up to an hour or two – to roughly gauge how many data points your posterior distribution will contain.

3. Remember that every prior and every summary statistic will provide a new filter overlaying the others, tightening the constraints on acceptance. So long as you do not unlink the summary statistics with the `-s` switch, it is likely that wider tolerances will be needed for multiple summary statistics.

4. When in doubt, try a tolerance of 0.1 and vary from there.

3.4 Data

The Data section contains both a description of the structure of the data and the data itself. Note that the data structure headers must match the actual data or errors will occur. For example, if you describe a system with one SNP and one STR and then give an entry line for the system with one SNP and two STRs the program will read all subsequent data out of order and likely crash, or at least give nonsensical results.

3.4.1 Header

The header part of the Data section begins with the Vnaught entry. This is a constant used with the T_D statistics, explained in the Calculations section (pg. 63) below.

Next is a description of each system in the data. A system is defined in REJECTOR as a group of very closely-linked polymorphisms with no recombination between them. REJECTOR does not model recombination between the loci of a system - it assumes that they are so close together that for simulation purposes the rate of recombination is essentially zero. Systems are assumed to be completely unlinked from each other, for example on different chromosomes.

Each system is given its own entry where it is named, and then its loci are defined in a table, with mutation rates and ancestral states listed below each locus type. These entries are listed in the following lines

3.4.1.1 System – Name of system

A single entry lists the name of the system. Please ensure that this name is also used the in the data list.

3.4.1.2 Loci – List of loci in system

The next line lists, left to right, the loci contained in the system. Please note that the following lines (Mutrates, Ancestral, RecombRt, and FracTs) are vertically organized under this line, with each of those entries referring to the locus listed above (i.e. the first Mutrates entry tells REJECTOR the mutation rate of the locus above it, etc.). You must ensure that all of these lines contain the same number of entries. There are four types of loci recognized by REJECTOR:

• SNPs, or Single Nucleotide Polymorphisms, are mutations at a single nucleotide site. The four alleles possible are A,C, G, and T. Ancestral states for each SNP can be defined by a comma-separated entry in the ancestral row. You can, for example, tell REJECTOR that the ancestral state for a SNP is A, as done for the SNP in system X2 in the example below, or that the ancestral states are T,C as shown in the first SNP of system X1 in the example. If you define any ancestral alleles for a SNP, then REJECTOR assumes that all other alleles from the pool A,C,G and T are descendant. If you do not know the ancestral state for a SNP, putting NA in the entry will ensure that SNP is not included in determining if a sample is ancestral.

• UEPs or Unique Event Polymorphisms, are like SNPs, but are the more general case of any polymorphic locus assumed to happen very slowly, usually only once in the scope of time studied. Examples of UEPs include insertion-deletion polymorphisms and RFLPs. The ancestral state for a UEP works differently than a SNP. If you define an ancestral state for a UEP, REJECTOR assumes that all other alleles are descendant, without knowing in advance what those alleles might be. You can also use NA for a UEPs ancestral state.

• STRs, or Microsatellites, are a series of contiguous repeats of base pairs, for example ATATATATATATAT or TACTACTACTACTAC. They are known to have relatively high mutation rates (Weber and Wong 1993). Many of the summary statistics in REJECTOR specifically target microsatellites, such as $\partial\mu^2$ and T_D. REJECTOR does not make calculations based on ancestral states of STRs, so always put an NA in the ancestral state. REJECTOR uses a Stepwise Mutation Model (SMM) for modeling STR mutation.

• DNA is simply a string of characters. Usually used for long stretches of DNA, e.g. ACTACGATATCTAGTC. WARNING! Placing a DNA locus next to a SNP or UEP will cause Simcoal2 to lump the loci into a single stretch of DNA, which will cause a crash!

3.4.1.3 Mutrates – Mutation rates

The per-locus mutation rate of the locus listed above. You may use scientific notation (i.e. 4.2e-05).

3.4.1.4 Ancestral – Ancestral state

The ancestral state of the locus. Use NA if you do not know the ancestral state. You may also define multiple ancestral states using quotes and a comma. If you want to consider both T and C ancestral, define this entry as "T,C".

3.4.1.5 RecombRt – Recombination Rate) (Optional)

The chance of a recombination event between the locus and the locus listed to its right, per generation. Defaults to 0.

3.4.1.6 FracTs – Transition rate (Optional)

Only used for DNA loci. This is the fraction of substitutions that are transitions. Defaults to 0.33, meaning that the chance of a base turning to any of the other bases is equal. If you use this line, you must enter something under all the loci for the purposes of spacing, but all entries for loci that are not DNA are ignored. Use -1 for other loci types.

3.4.2 Data List

The data list has three column headers for the first three columns. The Tag column is usually the sample or chromosome name. The Population column is used to place the sample in a deem or geographical area. The System column identifies to which system the data belongs. Note that an individual's data is broken down into separate lines for different systems. For example, if the individual HGDP6546 has data for system X1 and for system X2, each system receives its own line, configured to match the structure of each system. The data is NOT appended to a single line.

After the system column, the data is listed out in the order defined by the header for that system. Thus, is System X1 is defined as SNP, STR, SNP, UEP, STR. REJECTOR will read the data in that order in the entry.

3.4.3 Missing Data

REJECTOR allows missing data in an entry. To mark data as missing, use NA. REJECTOR also accepts -1 as missing data, but this is now deprecated. Missing data will not be used in calculations.

3.4.4 Rules for Data Structure

The following rules must be followed in the data section:

- Make sure every entry is for a system listed in the header, and that the data conforms to the structure defined in that header (number of loci, order of loci).
- Make sure you do not put a DNA sequence directly adjacent to a SNP or UEP!

- Within one population and one system, there can be no more than two entries with the same Tag, representing a diploid individual.
- There should be NO spaces in any entry! Avoid use of Tag or Population names like "N. America" (with a space between "N." and "America"). Before you run REJECTOR, go through your input file and remove these spaces. Using a find-and-replace routine in Excel will work quickly, i.e. search for "N. America" and replace with "N.America" (no space) or "N_America".
- Do not leave whitespace at the end of the input file.
- Use NA as an entry for missing data.
- If you want to run two-way statistics (comparing one population to another, e.g. $\partial\mu^2$) make sure there is more than one population in the data.
- When running a subdivided statistic (see Calculations below) be aware that empty populations can be produced if none of the entries in a population match the ancestral or descendant criteria.

3.5 Example infile: handtestguideexample.txt

```
REJECTOR

/--Priors
Population Sizes
uniform
100
10000
uniform
100
1000
uniform
100
1000

Growth Rates
uniform
0
0
uniform
0
0
uniform
0
0

Number of Migration Matrices
0

Number of Historical Events
1
```

```
uniform
2000 0 1 1.0 1 0 0
15000 0 1 1.0 1 0 0

Mutation Rates
gaussian
0.001
0.0001

Microsat Range Constraint
uniform
5
25

/--Testlist
ALL  0.05

/--Data
Vnaught   0

System    X1
Loci SNP  STR  SNP  UEP  STR
Mutrates  1.00E-09  2.80E-04  1.00E-08  1.00E-08  2.80E-04
Ancestral "T,C"     NA   C    L    NA

System    X2
Loci SNP  STR
Mutrates  1.20E-09  3.30E-04
Ancestral A    NA

Tag  PopulationSystem
HGDP01382 Adygei    X1   NA   23   T    L    10000
HGDP01383 Adygei    X1   A    NA   T    L    22
HGDP01383 Adygei    X1   C    -1   C    L    34
HGDP01381 Adygei    X1   A    22   NA   S    22
HGDP01382 Adygei    X1   T    22   T    NA   22
Men2 Am.Ind.   X1   A    80   T    L    80
Yw15 Am.Ind.   X1   C    21   C    L    21
CL132     Am.Ind.   X1   T    24   C    S    24
CL134     Am.Ind.   X1   T    25   T    S    25
BAL248    Balochi   X1   C    22   C    L    22
BAL249    Balochi   X1   C    24   C    L    24
BAL349    Balochi   X1   C    24   C    L    24
HGDP00058 Balochi   X1   C    22   C    S    22
HGDP00060 Balochi   X1   C    20   C    L    20
HGDP00068 Balochi   X1   A    19   C    S    19
HGDP00070 Balochi   X1   A    45   T    L    45
HGDP00072 Balochi   X1   T    21   C    L    21
HGDP01382 Adygei    X2   A    11
HGDP01383 Adygei    X2   A    12
HGDP01383 Adygei    X2   C    10
HGDP01381 Adygei    X2   A    11
HGDP01382 Adygei    X2   A    11
Men2 Am.Ind.   X2   A    11
Yw15 Am.Ind.   X2   C    11
BAL249    Balochi   X2   C    10
BAL349    Balochi   X2   C    10
```

4. Comparisons and Output

REJECTOR takes in the observed data and compares it to each iteration of simulated output. For each statistic, it compares the output entry-by-entry. If all of the entries in the results for one summary statistic are within the tolerance value specified for that statistic, that summary statistic is accepted. Under normal operation, `Rejector` will only print a line of output (thus placing that iteration in the posterior distribution) if all summary statistics are accepted within their specified tolerance. This behavior can be altered using the `-s` switch (see pg. 53).

Output files are saved in the same directory as the input file, in the form `[input file name].rej[number].txt` . The number after the `rej` extension always starts with 0 for the first file. Multiple files will be created with higher numbers if the number of output entries exceeds 50,000, for ease of use with popular spreadsheet programs.

4.1 Header information

The top of any REJECTOR output file lets you know that REJECTOR generated the file, and the name of the input file used, along with the total limit of runs and when the first line of this particular file was printed:

```
Rejector Output
Infile: mHDSandB-miconly-dm2-micsatvar-
wide.txt
Run Limit: 12500
Printing started Mon Nov 20 12:00:01 2006
```

Depending on the numbers of parameters and statistics you have used, this file can grow rather wide, so we have broken up the examples for the rest of the output file below into left-to-right chunks.

4.2 Parameter estimates

The sample parameters below correspond to a three-population test run, with simple names of 1, 84, and 92 for the populations. In this case the deme sizes were allowed to vary, but the growth rates were not. Sample size does not normally vary, but it still printed to remind the user of what the sample sizes were for the input file and that they are constant iteration to iteration. Each row below is one iteration of the software that was accepted within the tolerance value for the statistics used.

DemeSize-1	DemeSize-84	DemeSize-92	SampSize-1	SampSize-84	SampSize-92	Gr.Rate-1	Gr.Rate-84	Gr.Rate 92
3974	2132	3811	205	25	92	0	0	0
356	2575	1561	205	25	92	0	0	0
889	2129	3378	205	25	92	0	0	0
3959	1915	3250	205	25	92	0	0	0
2162	1975	3121	205	25	92	0	0	0

4.3 Migration Matrices

If you used a migration matrix, or multiple such matrices at different points in time in the population history model you used for input, then the values for the migration matrix will be output. Below is a segment of migration matrix output that would appear directly to the right of the example output above:

MM0-0,0	MM0-0,1	MM0-0,2	MM0-1,0
0	0.00276809	0.0017772	0.00544239
0	0.00870264	0.00758221	0.0095526
0	8.15E-05	0.00892863	0.00584986
0	0.000779358	0.0028913	0.00687231
0	0.00226524	0.000645603	0.00798795

Each column is first labeled by which migration matrix it is. In this case this is migration matrix 0, so “MM0”. Recall that you can use multiple matrices, so there could be further listings for MM1, MM2 and so on.

The numbers after each dash can be read as “from” the number before the comma “to” the number after it. Thus MM0-0,1 is the migration rate from Population 0 to Population 1 in that run. The listing 0,1,2… are used here in alphanumeric order of the input populations.

4.4 Historical Events

To the right of the last migration matrix entry are the parameters of each historical event, as outlined on page 32. If, for example, you had a historical event which represents a population divergence, then the accepted divergence times would list under HEvent0-Time below:

HEvent0-Time	HEvent0-Source	HEvent0-Sink	HEvent0-Prop	HEvent0-RelSize	HEvent0-NewGrRate	HEvent0-MMid
100	0	1	0	1	0	1
100	0	1	0	1	0	1
100	0	1	0	1	0	1
100	0	1	0	1	0	1
100	0	1	0	1	0	1

4.5 Summary Statistic Acceptances

To the right of the last historical event, you will find listings for which summary statistics were accepted (within the tolerance value, also called an alpha value) for each printed run. The tolerance value for each statistic is listed next to its name.

REJECTOR can be run in two ways: the default mode only prints a line if all summary statistics are accepted, and thus for each printed line all statistics will display "T" for true. In the mode where lines are printed if any summary statistic is accepted (using the switch `-s` – see pg. 53), some summary statistics may not have been accepted on each line, and thus will print "F"s. An "X" either represents no test, such as for cases like twoway ancestral statistics where one or both demes are found to be empty, or a type of summary statistic excluded from comparison, such as SampleSize summary statistics. You can easily use Excel's Data->Sort functions to sort on T's or F's to pull out the summary statistic results you want. See page 45 below for provided ways to analyze the output file using the R statistics package.

Test(Alpha)->	DeltaMuSquared(0.1)	MicrosatVarianceAv(0.1)
	T	T
	T	T
	F	T
	T	T
	T	F

Next to the true-false result lines you will find another result array. The headings are identical, but the results are not true or false, but are instead the proportion of summary statistics accepted. Here a "-999" means the same thing as an "X" in the true-false results. As you will see by comparison, a "T" for a summary statistic will correspond with a value that is equal to 1, meaning that all entries in that result array were within the alpha value.

4.6 Testing Alternate Hypotheses of Population History

There are a few cases for which REJECTOR's method of parameter estimation may not be able to investigate the features of interest in a population history. For example, prior distributions cannot be set that directly alter tree shape, or how many populations

exist in the present day. In such cases, another method can be used to investigate population histories. Alternate models of population history can be designed by the user, such as the two alternate models below:

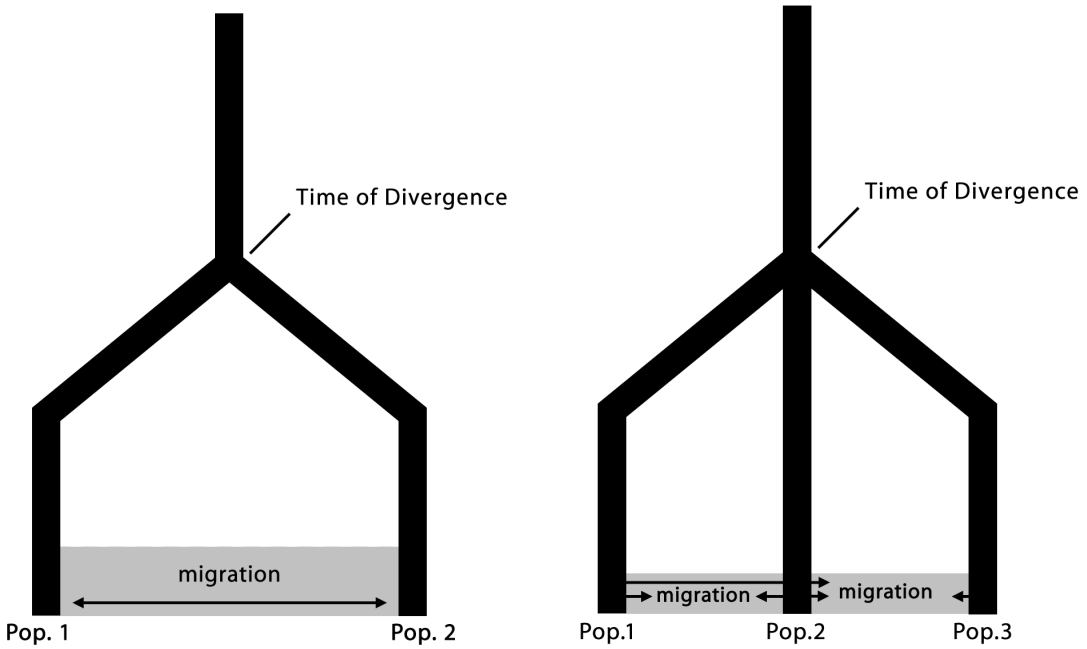


Figure 14. Two alternate models of population history. The data set is divided into two or three present-day populations, and then the alternate input files can each be run for the same number of iterations of REJECTOR. The number of acceptances can then be compared for the summary statistics chosen.

These alternate models can each be run in REJECTOR for a set number of iterations and the number of acceptances can be compared to investigate which of the two models provides summary statistic values closer to that given by the experimental data. This method is most useful when other features of the population history, such as divergence times and migration rates, can be assigned with a reasonable degree of certainty that they are near the true values.

5. Post-Processing and Visualization

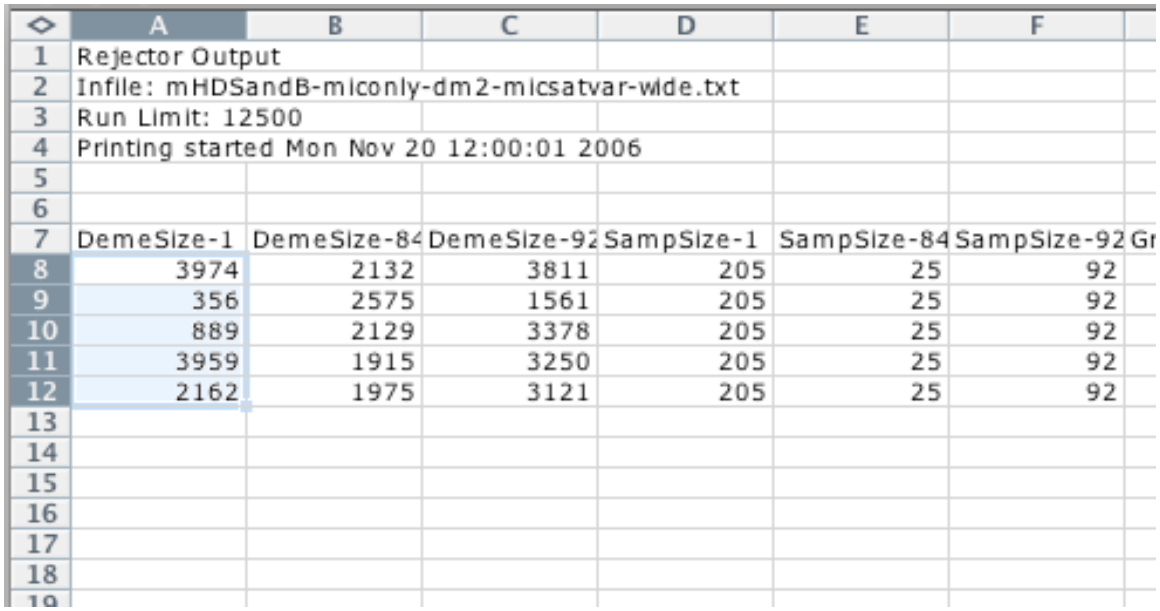
Please see the folder `scripts` in `rejector.zip` for perl scripts and R functions for processing of REJECTOR output. A number of the most useful scripts and functions are outlined below.

5.1 Output files in Excel

This can work for both any one output file or a composite output file put together with gluedat.pl (see pg. 47 below). Open the output file in Excel. To do so you will need to convert it – when prompted by Excel, choose that you would like to import Delimited, and choose by Tabs on the next page. This will give you a nice Excel-converted file to work with. Please bear in mind Excel's limitations in row and column number, depending on the version you are using.

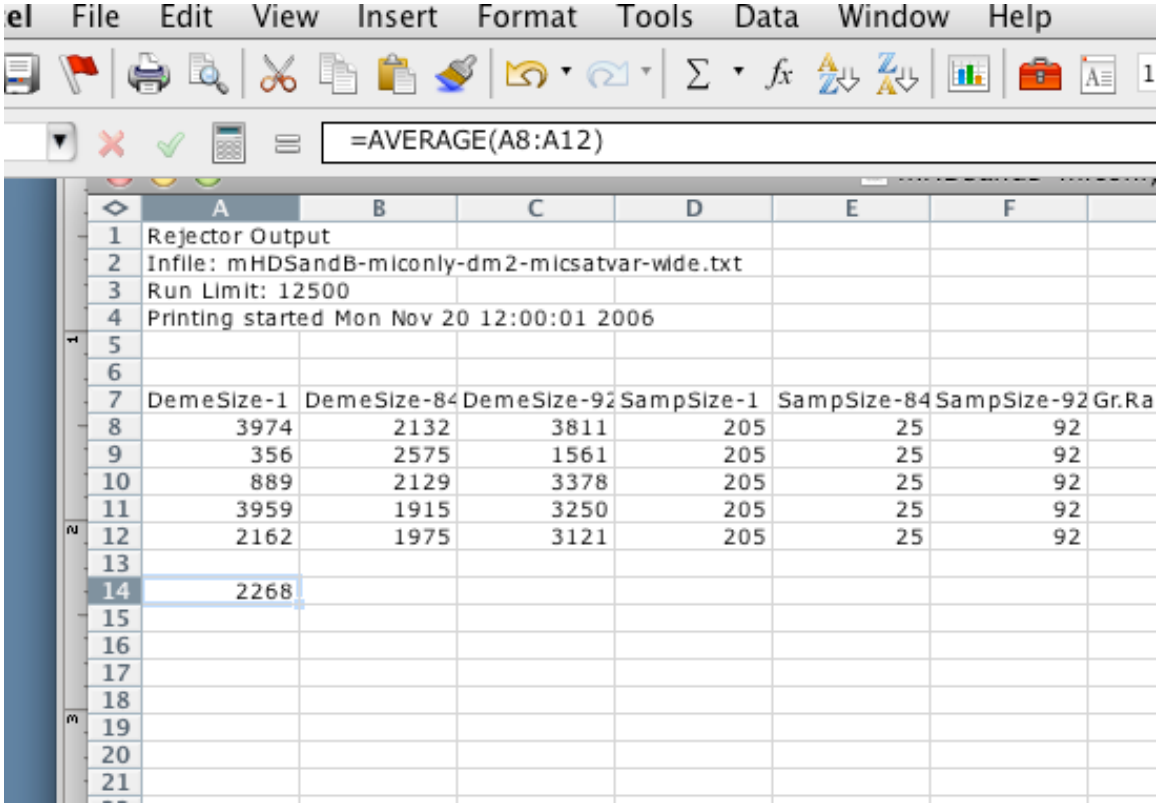
Here is an example of what you can do with an output file in Excel. Say you would like to know about the distribution of DemeSize for your first population, for all those runs accepted for DeltaMuSquared. This is what to do:

1. Select everything below the header line (i.e. the line with DemeSize-1, DemeSize-2 etc.).
2. Figure out what column the DeltaMuSquared summary statistic output is on (where there are T's and F's)
3. Go to the Data column at the top and choose Sort.
4. Sort on the column for DeltaMuSquared. The T's and F's will now be sorted out, and you can choose only those lines which are T for DeltaMuSquared. You may with to delete the F lines, just for clarity, but if so keep a backup of the original!
5. Select the entries under the column for the first DemeSize entry, but only for those entries where DeltaMuSquared is T! (This could be on the top or bottom depending on whether you sort Ascending or Descending.)
6. You can now use Excel's functions to extract useful information from the data. For example, you could take the mean of the column as shown in Figure below.



	A	B	C	D	E	F
1	Rejector Output					
2	Infile: mHDSandB-miconly-dm2-micsatvar-wide.txt					
3	Run Limit: 12500					
4	Printing started Mon Nov 20 12:00:01 2006					
5						
6						
7	DemeSize-1	DemeSize-84	DemeSize-92	SampSize-1	SampSize-84	SampSize-92
8	3974	2132	3811	205	25	92
9	356	2575	1561	205	25	92
10	889	2129	3378	205	25	92
11	3959	1915	3250	205	25	92
12	2162	1975	3121	205	25	92
13						
14						
15						
16						
17						
18						

Figure 15. I have selected all the entries for DemeSize-1 after deleting all entries that are F for DeltaMuSquared.



	A	B	C	D	E	F
1	Rejector Output					
2	Infile: mHDSandB-miconly-dm2-micsatvar-wide.txt					
3	Run Limit: 12500					
4	Printing started Mon Nov 20 12:00:01 2006					
5						
6						
7	DemeSize-1	DemeSize-84	DemeSize-92	SampSize-1	SampSize-84	SampSize-92
8	3974	2132	3811	205	25	92
9	356	2575	1561	205	25	92
10	889	2129	3378	205	25	92
11	3959	1915	3250	205	25	92
12	2162	1975	3121	205	25	92
13						
14	2268					
15						
16						
17						
18						
19						
20						
21						

Figure 16.The Entry A14 above is the mean (average) of the column above it.

5.2 gluedat.pl – Sticking multiple output files together

REJECTOR's default mode creates a series of output files labeled `[input file name].rejX.txt`, where `X` is a number beginning at 0 and counting up. A new output file is made when the last reaches 50,000 accepted iterations. To stick all of these files back together to post-process in R, simply put them all in a folder with the `gluedat.pl` script from the `rejector` scripts folder. The following assumes you have PERL installed on your machine. Mac and Linux users, this is almost a certainty. Windows users, you may want to install PERL from http://www.perl.org/ or include it as part of your cygwin tools.

Once you have all your `rejector` output files (i.e. named `myfile.rej0.txt`, `myfile.rej1.txt`, etc.) together with `gluedat.pl`, go to that directory on the command line and type:

```
perl gluedat.pl myfile
```

Just leave the ".rej0.txt" part off. If your files were really named `myfile.rej0.txt` and so on, it will produce a file named `myfile-out.txt`. This file will contain ALL the data from all the other output files. You can use this file for analysis in Excel as on page 45 above, or in R as below.

5.3 Graphing Results in R using rejector.r

R (http://www.r-project.org/) is a free environment for statistical computing. We have included some R routines in a file called `rejector.r` in the `scripts` directory. Here is how to use it to visualize and analyze your REJECTOR output:

1. Download R from http://www.r-project.org/ and install it. We will be talking about using R for MacOSX here; consult the R documentation for any differences on other systems.
2. Create a composite output file (`myfile-out.txt`) using `gluedat.pl` as shown above. Put `rejector.r` in the same directory.
3. Run R and change the working directory (command-D on Mac, usually under the Misc column at top) to the directory where your outfile resides.
4. Go to File-Source File and choose `rejector.r` or, just type `source("rejector.r")` in the R console.

From here there are a number of functions you can use, each given its own section below. Let us assume from here on that your output file is called

```
myfile-out.txt
```

for simplicity's sake. All of the functions here are done from the R Console once the steps above are carried out.

5.3.1 rejde - Looking at your data

To simply see the R spreadsheet of your loaded data:

```
rejde("myfile-out.txt")
```

5.3.2 rejnames – Identifying your columns

For most of the functions below, you will need to know the number of the columns for both the summary statistics and the parameters you want to work with. Find out how R has numbered them using:

```
rejnames("myfile-out.txt")
```

5.3.3 rejsum – Mean, Variance, Median, Credibility Intervals, Mode

This function will load in your output file, then give you the mean, variance median and 95% credibility interval of the parameters you specify for the summary statistics you specify **- but only those entries where the statistic is T!** The function, and all the functions below it, ONLY calculate for entries where that statistic is marked T, so you don't need to sift out the F and X entries first.

Find the columns for the statistics and parameters you wish to examine as in Section 5.3.2 above. Each output file may be of different sizes based on its model configuration, so do not assume you know the column numbers for different input files! If, for example, you wanted to know the mean, variance, etc. of the parameters in columns 1 through 4 and 7 for accepted summary statistics in columns 51 and 52, you would input the following:

```
rejsum("myfile-out.txt", c(51,52), c(1:4,7))
```

The means, variances, medians and credibility intervals will be saved in `myfile-out-summary.txt`, which will look something like the following:

COL	NAME	MEAN	VAR	MEDIAN	2.5%Int	97.5%Int
DemeSize.1	DeltaMuSquared.0.05.	1007.465	3204.882	1005	902	1121.425
HEvent0.Time	DeltaMuSquared.0.05.	2023.87	15688.23	2022	1781.575	2267.85

The entry under COL defines which parameter is listed in the row. The entry under NAME defines which summary statistic is marked "T" for all the entries used to make the calculations for the row. The 2.5%Int and 97.5%Int columns are the credibility intervals; the true value of the parameter is expected to lie between these values 95% of the time. A new entry will place the mode of the distribution to the right of the 97.5%Int column.

5.3.4 rejoned – One-Dimensional Plots

To make histograms of the distributions of parameters of interest, find the column numbers as in Section 5.3.2 above, and also look up the total number of runs this REJECTOR session completed (This can be found in the header of each original output file). For example, in following the example above, for a 10,000-run REJECTOR session:

```
rejoned("myfile-out.txt", c(51,52), c(1:4,7), 10000)
```

This will produce the file `myfile-out.pdf` in the folder, which you can open to see histograms such as the one below:

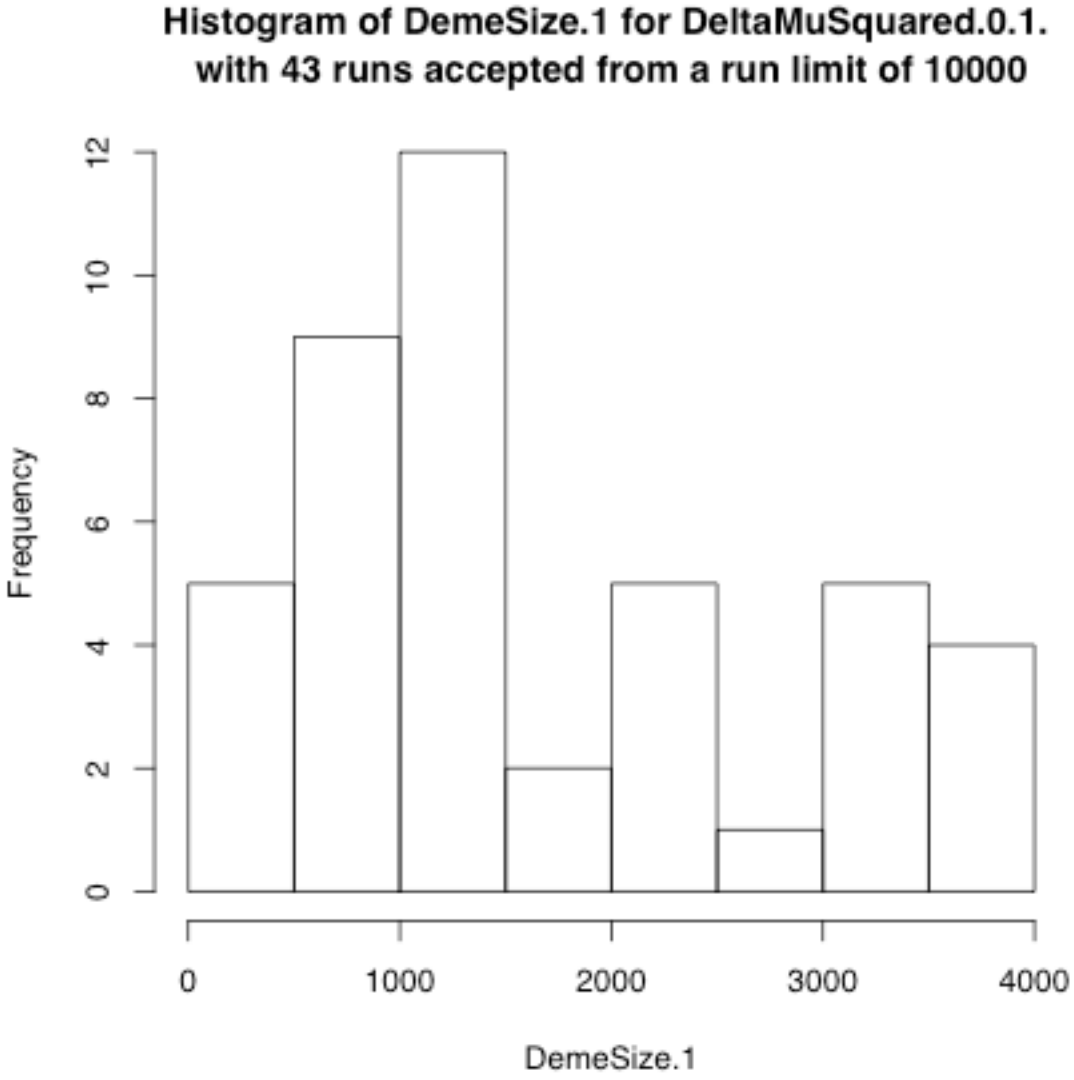


Figure 17. Example histogram generated from a run of Rejection, showing DemeSize.1 (estimates of the size of Population 1).

Adding in the total number of runs here will keep a record of the ratio of acceptances with the histogram, so that you can avoid looking it up later.

To use a density plot instead of a histogram, add the variable `density=TRUE`

```
rejoned("myfile-out.txt", c(51,52), c(1:4,7), 10000,
density=TRUE)
```

Simply typing `rejoned` at the R prompt will provide a full listing of the optional switches and variables for the function. Look at the top of the output, under “usage”. You can place a prior distribution on the figure and place a mark if you know the true parameter value.

5.3.5 rejtwod Two-dimensional plots

The distribution of one parameter can be plotted against another using the standard MASS library in R and the `rejtwod` function. for example, typing:

```
rejtwod("myfile-out.txt", c(51,52), c(1,2), 10000)
```

on the same data as above produced the plot below:

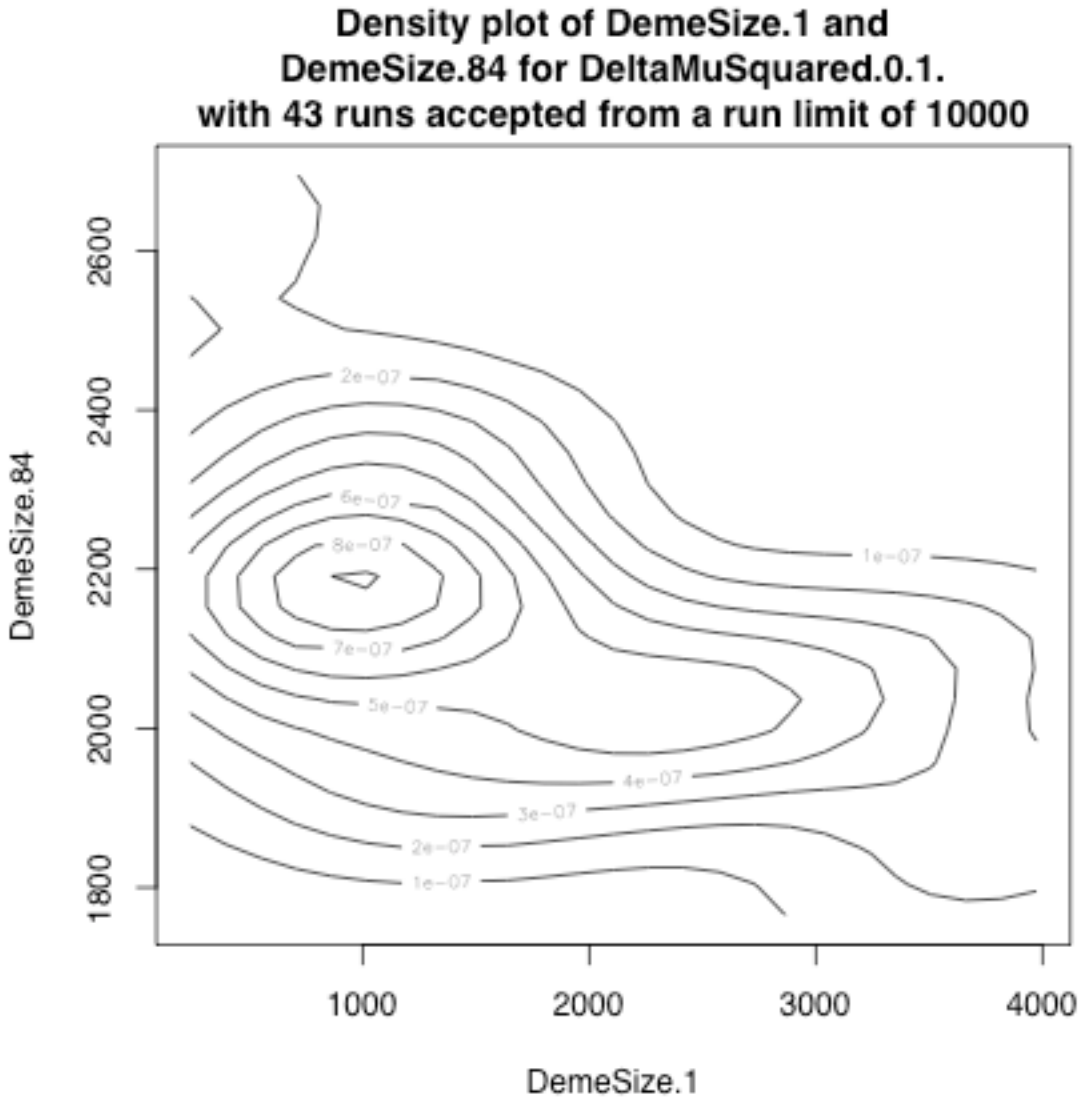


Figure 18. Example two-dimensional contour plot of accepted estimates of DemeSize.1 versus DemeSize.84

Typing `rejtwod` at the R prompt will provide a full listing of the optional switches and variables for the function. Look at the top of the output, under “usage”. You can specify the scale and boundaries for the plot, place prior distributions and place a mark if you know the true parameter values.

5.3.6 Odds and Ends

There are a few alternate versions of the functions listed above. To look them up, simply open `rejector.r` in a text editor and read through it. We have included batch files to allow plotting from multiple files.

6. Ways to Invoke REJECTOR

REJECTOR is a command line program. To use it, open a terminal window (i.e. the Terminal app on the Macintosh) and navigate to the directory where the program resides. Once there, the program can be most simply invoked by typing:

```
./rejector
```

which runs the copy of REJECTOR in that directory. Alternatively, you can place the executable in a directory in your PATH environment variable and invoke by typing

```
rejector
```

which will invoke REJECTOR from any directory.

6.1 The –f and –r switches: Setting the input file and maximum number of iterations

If invoked with the above method, REJECTOR will then prompt you for the name and location of the input file you wish to use. It will look in the directory you were in when you typed the command (i.e. your current working directory) unless you specify a path to the file explicitly.

REJECTOR will then ask for the maximum number of runs. You can also specify the input file on the command line:

```
./rejector –f [rejstats input file] –r [max. runs]
```

An example of such a command is:

```
./rejector –f rejexample.txt –r 100000
```

Note that if the input file is not the same as the current working directory, it must be specified in the command. For example:

```
./rejector –f /usr/local/rejector/rejexample.txt –r
100000
```

6.2 The –s switch: Unlinking summary statistic criteria

If you wish the results of each summary statistic to be "unlinked" (i.e. a line is printed in the output file is at least one summary statistic is accepted) then append the `–s` switch:

```
./rejector –f rejexample.txt –r 100000 –s
```

6.3 The –c switch :Copying outfiles to different directories

If you wish to copy each outfile to a new location after it completes, use the `–c` switch, adding the directory you wish it to be copied to after the switch. This does not yet work with Xgrid runs.

```
./rejector –f rejexample.txt –r 100000 –c
/home/mruserguy/src/
```

6.4 The –n switch: Altering output file numbering

Output files are usually numbered starting from 0. If you wish to change the starting number, use the `–n` switch:

```
./rejector –f rejexample.txt –r 10000 –n 5
```

6.5 The –a switch: Averaging SIMCOAL2 runs

`Rejector` can also be used to average the results of multiple SIMCOAL2 runs. Invoke this function using the `–a` switch. The final parameter specifies the number of SIMCOAL2 runs, which will each have summary statistics calculated using REJSTATS. The averaged outfile is of the form `[infile]-av.out`:

```
./rejector –f rejexample.txt –a 100000
```

6.6 The –p switch: Printing all iterations regardless of result

In the interests of saving space, REJECTOR normally prints output lines only when they meet the criteria for acceptance. If you wish all iterations to be printed, whether or not they are accepted, use the `–p` switch as shown below:

```
./rejector –f rejexample.txt –r 10000 –p
```

6.7 The –m switch: Invoking with Multiple Processor Cores

If you have a computer with multiple cores or processors, you can direct REJECTOR to take advantage of your hardware by using the `–m` switch and specifying the

number of parallel threads you would like the program to use. If, for example, you ran REJECTOR as shown below:

```
./rejector –f rejexample.txt –r 10000 –m 2
```

the software would run using two separate threads, greatly reducing total simulation time. **Note,** however, that there is no benefit to specifying more threads that your computer has processor cores. Check your computer's documentation if you are unsure of how many you have. At present, many computers have 2 cores, but some workstations have 8 or more. Note also that the –m switch only specifies multiple threads on the same computer. For distributed runs on multiple computers (Macs only), see the section on Xgrid runs (pg. 57) below.

6.8 The –ks switch: Keeping statistics values

Invoking using the –ks switch will include the values of the summary statistics in the output file. Note, however, that if you wish to keep ALL summary statistic values, regardless of whether the value is accepted within tolerance, you must also use the –p switch.

6.9 Invoking long REJECTOR runs

Since REJECTOR is run from the command line, it requires an active terminal window to run the program. This means that closing a window running REJECTOR will abort the run prematurely. This can be a problem for very long runs of REJECTOR, particularly for common scenarios where the computer running REJECTOR is a server or dedicated machine but the user's computer is his or her own desktop or laptop machine. Most UNIX-like operating systems, including Mac OS X, allow for a simple solution to this. The `screen` command can be used to create a window, invoke commands, and then "detach" it, allowing it to run as a background process on a remote machine.

To run REJECTOR in such a manner, open a terminal window on the machine you wish to run REJECTOR (either on its own hardware or remotely through `ssh` or something similar). Once you can ensured REJECTOR is properly installed and your input files are ready, invoke the screen command with:

```
screen –R rej
```

This will create a new "screen" or virtual terminal window called rej (you can call it anything you like, "rej" is just a suggestion). You may now invoke your long run of REJECTOR. Once it is running, detach the screen by typing <ctrl>-a and then d. You will be back to the screen you had before the screen command was invoked. You may check that REJECTOR is still running in the background by running `ps`, `top` or a similar process-reporting program. You will likely see `rejstats` and `simcoalrej_212` being invoked in succession as REJECTOR runs through its paces. You may now close this window and leave REJECTOR to do its work.

To check on REJECTOR's status, open a terminal on the machine running REJECTOR and type:

```
screen –r rej
```

substituting the name you had for the screen if you didn't use "rej". If REJECTOR has exited back to a prompt and printed the working time, it is done and you can pick up the results. Otherwise just hit <ctrl>-a and then d again and check back later.

For more information on screen, just open a terminal and type:

```
man screen
```

7. Use with Xgrid (Macs only)

REJECTOR can be used on distributed computing networks using Apple's Xgrid technology (http://www.apple.com/server/macosx/features/xgrid.html). Please note that this option only works with Macintosh computers. An Xgrid run of REJECTOR will seed each iteration of simulation out to a separate computer, allowing multiple computers to simulate and calculate statistics in parallel. The invoking computer (in Xgrid terminology the "client") runs REJECTOR to compare the output and record accepted statistics. For long runs, it is still wise to run REJECTOR using the screen command as shown above.

Before you attempt to run an Xgrid version of REJECTOR, you should do the following:

- learn how Xgrid works using the Xgrid documentation, and by typing xgrid at the command line to read how the commands are invoked
- ensure that you have an Xgrid network that is properly set up and tested, and that you know the hostname and password of the controller
- ensure that your grid is on a fast network, preferably 100 Mbps or higher
- know how many cpu's you can utilize on the grid
- run REJECTOR briefly in single-processor mode to ensure everything works properly and to assess the speed of computation
- when running REJECTOR in xgrid mode, the computer that invoked REJECTOR (the xgrid "client") will be used to run PRIOR and to compare output files. If this machine is an agent on the grid, it is best to configure it to " Accept tasks only when idle" in the xgrid section of the Sharing System Preferences Pane.

Please bear in mind that Xgrid is heavily dependent on network resources, since it pushes executables and data files back and forth across the network. This network overhead can potentially offset the benefit of parallel computation if the number of cpu's you can access is too small or the network is too slow. This can become more acute when the time taken for each run of REJSTATS and SIMCOAL2 is very small, as there might be more time taken moving files than processing data. With test input files that took 1.5 seconds each to run in rejstats, on an 8 cpu network at 100Mbps speed, Xgrid rejector was three times as fast as a single-processor run. Your mileage may vary.

Invocation of REJECTOR using Xgrid requires a few extra parameters. The following is a sample command and to run rejector in command mode:

```
./rejector –f rejexample.txt –r 1000 –x 4 64.23.65.76
p4ssw0rd
```

The parameters are, in order:

- the rejstats-format input file
- the number of runs before termination
- the –x switch to tell REJECTOR to run in Xgrid mode

- the number of cpus to use
- the hostname or ip address of the controller
- the password for the controller

To use with the statistics unlinked (see above), append the `-s` switch as the first argument:

```
./rejector -f rejexample.txt -r 1000 -s -x 4
64.23.65.76 p4ssw0rd
```

When using REJECTOR in Xgrid mode, extra output files will be created in directories under the working directory. They will be of the form `[input file]-[number]`, and will contain the working copies of .arp, .out, .par and batch files used for each cpu active on the thread. Normal operation of REJECTOR will remove these directories after each run. While REJECTOR checks and tries to empty directories it's about to use, it's a good idea to remove any copies of these directories you find before invoking REJECTOR.

NOTE! REJECTOR is more sensitive to failures of the other executables in Xgrid mode. For example, if a computer is assigned a thread with 10 jobs, and any of them should fail for reasons such as a failure to coalesce in SIMCOAL2, the whole thread will abort and a new one will be created. This means that if you have so configured your priors that SIMCOAL2 aborts often, Xgrid jobs will progress very slowly! Most of the time, you can fix this by carefully examining your input file – you are unlikely to want to generate priors such that SIMCOAL2 cannot coalesce the lineages. Check, for example, that all the lineages do end up in the same deme at some point in the past, and that said deme does not continue to grow (backwards in time) or it will be impossible to coalesce!

8. Rejstats and Statistics

REJSTATS is a program that calculates summary statistics on genetic data. It is designed to be run either as a part of the REJECTOR package, or invoked manually as a stand-alone program.

Genetic data is categorized in REJSTATS in three ways. All samples are assigned to a geographic group called population. The data derived from any individual is broken into very-tightly linked regions termed systems. Each system contains a number of loci, which are individual polymorphisms such as Single Nucleotide Polymorphisms or microsatellites. REJSTATS can handle any number of loci in any number of systems for any number of populations, limited only by the computational resources available to the user.

8.1 Calculations

Calculations, or summary statistics, are performed by REJSTATS on the input data. Some calculations are limited to certain types of loci, for example, $\partial\mu^2$ is based on microsatellite lengths, and thus cannot be calculated for SNPs or UEPs. Wherever a calculation cannot be run, the results are given as NA.

The calculations are divided into groups based on the way the data is manipulated:

- Oneway – These calculations are performed on each locus in each population. Example: Heterozygosity.
- OnewayAveraged – These calculations are performed on each locus in each population, and are then averaged for all loci of the same type (SNP, UEP, MICROSAT). Example: HeterozygosityAv.
- Onewaydlist – As a oneway, but a special case where at least part of the statistic is performed across all loci.
- Twoway – These calculations are performed between populations, producing a two-dimensional table of results. Example: DeltaMuSquared.
- Twowayxloci – As a twoway, but operating deme by deme to allow for averaging across loci. Example: Nei.
- PerSystem – These calculations are performed on an entire system. Example: SampleSize.
- PerLocusXPop – As PerSystem, but calculations that are performed across populations. Example: F_{st}.
- TwoLoci – These calculations are performed between loci within the same system. Example: LDChiSquare.

8.2 Calculations on Parts of a Population

Some calculations can be performed on samples within a population that are marked ancestral or descendent. SampleSize, Heterozygosity, HeterozygosityAv, Beta, BetaAv, DeltaMuSquared and Td can be done in this way. These subdivided summary statistics are invoked by adding Sub to the name of the statistic in the Testlist. These summary statistics are output separately from the standard statistic, having the suffix "-ANC" added for the ancestral and "-DEC" for the descendant. Because there can be any number of loci in a system, the rule for determining if a sample in ancestral is as follows:

• *A sample is marked ancestral if at least one of its loci is ancestral and none of its loci are descendant.*

The reverse is true for determining descendant samples. Any locus marked NA for ancestral state is not considered when determining ancestral status. A sample that has no ancestral state information at all is neither ancestral nor descendant.

8.3 List of Summary Statistics

8.3.1 Beta imbalance ratio

The beta index measures the imbalance of variance- and homozygosity-based estimates of θ, and is calculated:

$$(t) = \frac{V(t)}{[1/P_0(t)^2 \quad 1]/2'}$$ (Kimmel, Chakraborty et al. 1998)

8.3.2 CompleteHeterozygotes (PerSystem)

This summary statistic counts the number of samples that are heterozygous at all loci in a system. Note that only diploid samples (two samples per system) are examined.

8.3.3 CompleteHomozygotes (PerSystem)

This summary statistic counts the number of samples that are homozygous at all loci in a system. Note that only diploid samples (two samples per system) are examined.

8.3.4 Delta mu Squared *(Twoway)*

Delta-mu squared, or $\partial\mu^2$, is a measure of the square of the difference of the mean STR length between two populations (Goldstein, Ruiz Linares et al. 1995).

8.3.5 Derived Fraction *(PerSystem)*

A simple calculation of the fraction of derived (descendant) samples over the total samples in a deme.

8.3.6 D_{sw} (Twowayxloci)

D_{sw} is an extension of Nei's minimum genetic distance, weighted by the difference in repeats between alleles, assuming a stepwise mutation model (Shriver, Jin et al. 1995). It takes as its model the stepwise mutation model of STRs. If x_i is the frequency of allele i in population x and x_j is the frequency of allele j in population x, then:

$$d_{xw} = \sum_{i \neq j} x_i x_j \, \delta_{ij}$$

where

$$\delta_{ij} = |i - j|$$

and d_{yw} is the same as d_{xw} for population y, then

$$d_{xyw} = \sum_{i \neq j} x_i y_j \, \delta_{ij}$$

and

$$D_{sw} = d_{xyw} - \frac{(d_{xw} + d_{yw})}{2}$$

For multiple loci, d_{xw}, d_{yw}, and d_{xyw} are averaged before calculating D_{sw}.

8.3.7 F_{st} (PerLocusXPop)

The fixation index F_{st} measures the amount of genetic differentiation between subpopulations. It is calculated by taking the average heterozygosity at each locus (h_s), and then for each locus across populations (h):

$$F_{ST} = \frac{(h - h_s)}{h}$$ (Cavalli-Sforza, Piazza et al. 1994)

8.3.8 Haplotypes (PerSystem)

The number of distinct haplotypes in a system.

8.3.9 Heterozygosity (Oneway, OnewayAveraged)

Heterozygosity is a measure of the allelic variation at a locus:

$$H = \frac{n(1 - \sum p_i^2)}{(n - 1)}$$

where n represents the number of chromosomes in a sampled group and p_i represents the frequency of the ith allele in that group.

8.3.10 Linkage Disequilibrium (Twoloci)

Linkage disequilibrium (LD) is defined as the nonindependence of alleles between two or more loci. The chi-square value calculated here is:

$$\chi^2_T = \sum_{u=1}^{k} \sum_{v=1}^{l} \frac{n\hat{D}_{uv}^2}{\tilde{p}_u \tilde{p}_v} \text{ (Weir 1996) where } D_{uv} = p_{uv} \quad p_u p_v .$$

8.3.11 Linkage Disequilibrium p-value(Twoloci)

The above chi-square value is calculated for the data, then compared against many chi-square values generated by randomizing the two loci while keeping row and column totals the same. The number of random chi-squared values in excess of the experimental value gives a p-value and is thus a test for the statistical significance of the linkage disequilibrium.

8.3.12 Linkage Disequilibrium (r^2)

This is another calculation of linkage disequilibrium that uses the square of the correlation coeffificent, or r^2 (Hill and Robertson 1968).

8.3.13 Maximum Length, Minimum Length, Range, Mean Length *(Oneway)*

For microsatellites, these statistics simply print longest and shortest microsatellite at the locus, the difference between the two, and the mean (average) length.

8.3.14 Microsatellite Variance *(Oneway)*

The variance in length at each microsatellite locus.

8.3.15 Nucleotide Diversity *(Oneway)*

The average number of nucleotide differences per site between two sequences (Nei 1987). Assumes a randomly mating population. Estimated by:

$$\hat{\ } = \frac{n}{n \quad 1} \quad x_i x_j \quad _{ij}$$

where n is the population size, x_i is the population frequency of the ith type of DNA sequence, and $_{ij}$ is the proportion of different nucleotides between the ith and jth types of DNA sequences.

8.3.16 Nei (Twowayxloci)

Nei's minimum genetic distance measures the minimum number of codon differences per locus (Nei 1987). The chance of picking different alleles from the same population *x* is:

$$d_x = \sum_{i \neq j} x_i x_j$$

and d_y is the same for population *y*. Between populations *x* and *y*:

$$d_{xy} = \sum_{i \neq j} x_i y_j$$

For multiple loci d_x, d_y, and d_x are averaged, and the minimum genetic distance is calculated by:

$$D_m = D_{xy(m)} \quad (D_{x(m)} + D_{y(m)})/2$$

8.3.17 Number of Different Alleles *(Oneway)*

A count of the number of different alleles at a locus.

8.3.18 Sample Size *(PerSystem)*

The number of samples at a locus.

8.3.19 T_D (Twoway)

T_D is an estimator of the time of divergence based on microsatellite data that does not assume mutation-drift equilibrium (Zhivotovsky 2001). It requires a knowledge of the variance in the number of repeats at the beginning of population separation, V_0, which can be included in the input file. It is calculated by:

$$T_D = \frac{D_1}{2w} \quad \frac{V_0}{w}$$

where D_1 is the average over loci of the average squared difference between pairs of alleles sampled, one each from the two populations (Goldstein, Ruiz Linares et al. 1995), and *w* is the effective mutation rate.

9 Sensitivity of Summary Statistics

Below are tests of combinations of summary statistics and parameter values using various configurations of data. These sensitivity analyses were conducted using a known model, meaning that the summary statistic values of the "experimental" data were calculated using this model and then used as target values for the sensitivity tests. The model is shown in Figure 19. These analyses test both the sensitivity of various summary statistics to changes in parameter values, and the accuracy of the statistics in recovering the known true parameter value.

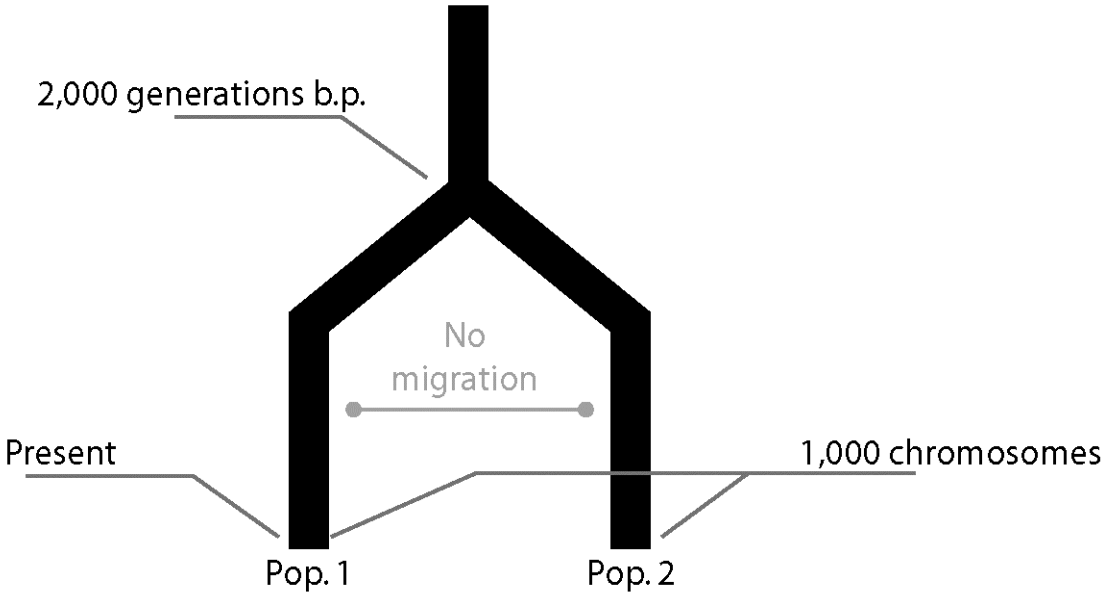


Figure 19. Target population model for all simulations in this chapter save for those with prior distributions for migration rate (Section 9.3).

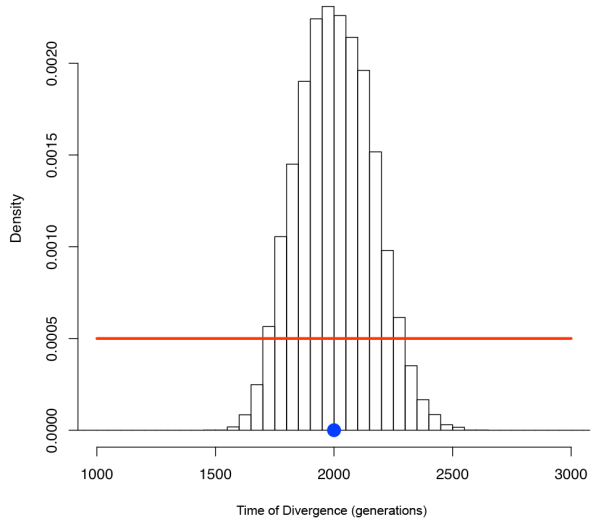
In the case of the one-dimensional histograms below, the solid line represents the prior distribution for the parameter, while the dot is the true parameter value calculated from the model in Figure 19. The accompanying tables list the number of accepted iterations of the simulation as well as the mean, variance, median and 95% credibility interval for the posterior distribution.

9.1 Time of Divergence

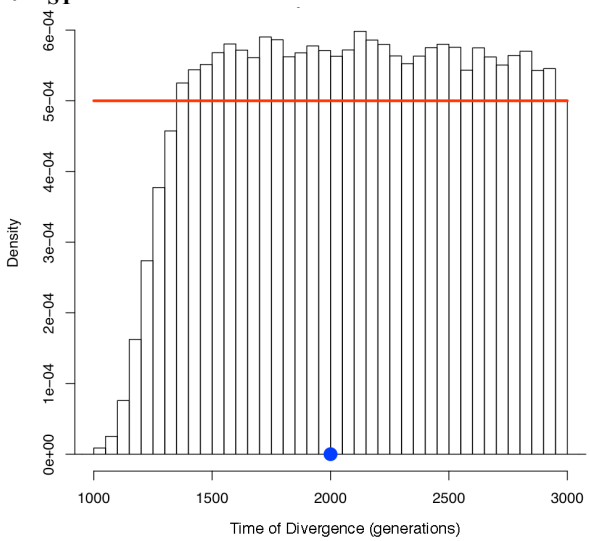
9.1.1 Comparison of Summary Statistics – 1000 STRs

We compared estimates of time of divergence for various summary statistics using 1000 unlinked STRs. The results indicated that the summary statistic μ^2 was most capable of recovering the true value of this parameter.

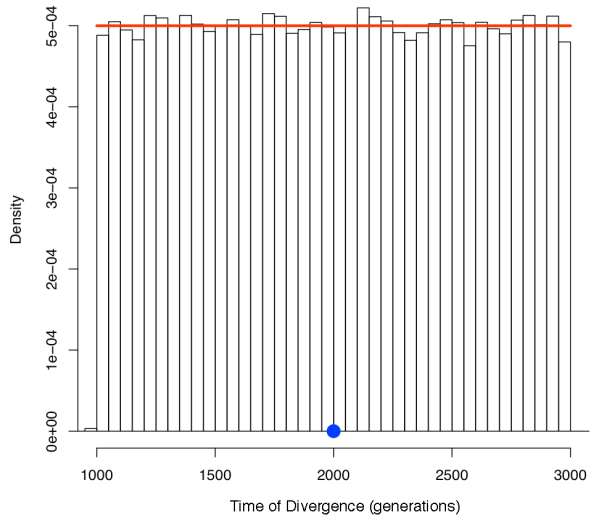
A: μ^2



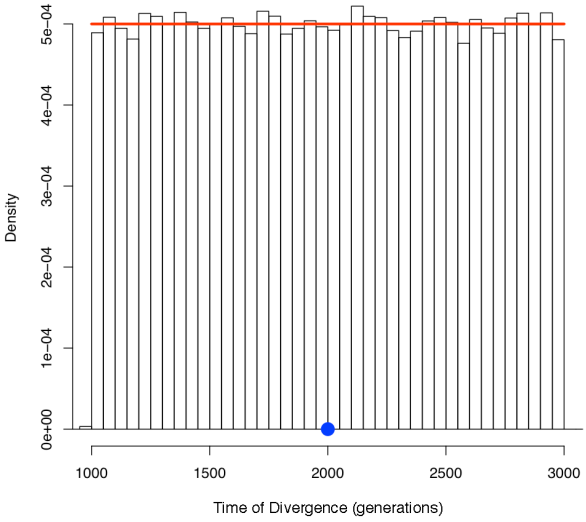
B: F_{ST}



C: Av. Heterozygosity



D: Microsatellite Variance



E: Nei's **F: Number of Different Alleles**

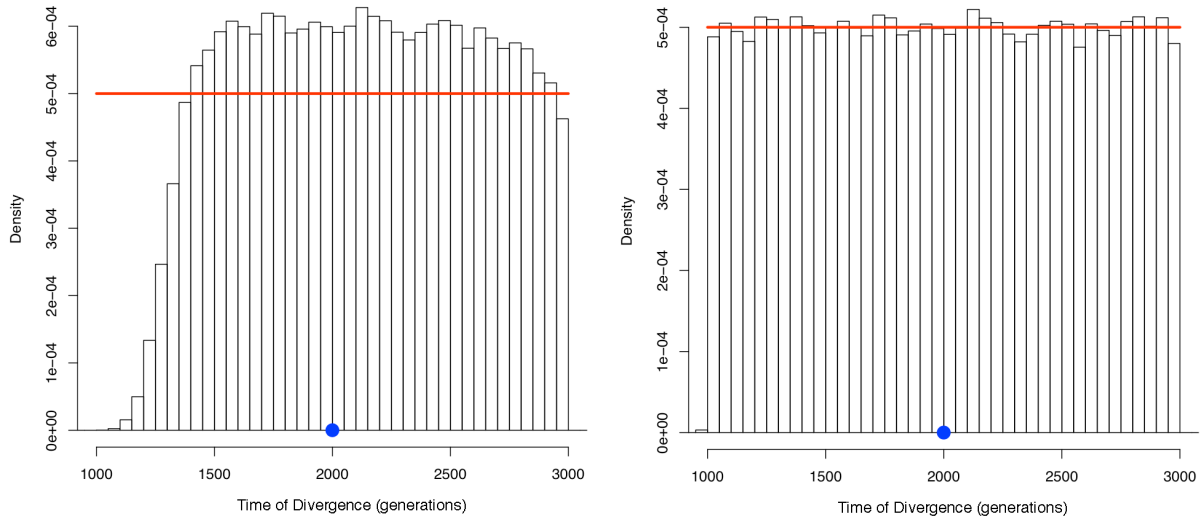


Figure 20. Comparison of summary statistics for estimation of time of divergence. Histograms of estimates of time of divergence for 1000 unlinked STRs, accepted within a tolerance value of 0.1. For all the above there were 100,000 total runs of REJECTOR.

Table 1. Rate of acceptance, and mean, variance, and 95% credibility interval of time of divergence for six summary statistics.

Summary Statistic	No. accepted runs (from 100,000)	Mean time of divergence (gens)	Variance in time of divergence	2.5% Credibility Interval	97.5% Credibility Interval
□μ^2	20079	2007	24818	1719	2320
F_{ST}	87253	2109	260818	1243	2951
Av. Heterozyg.	100000	2000	332836	1051	2948
Microsat. Variance	98005	2000	333054	1051	2948
Nei's	83157	2132	238458	1309	2946
Number of Different Alleles	100000	2000	332836	1051	2948

9.1.2 Effect of Increasing Number of Loci

Increasing the number of loci decreases the variance, as shown in the case of $\Box\mu^2$ averaged across loci for STRs in Figure 21.

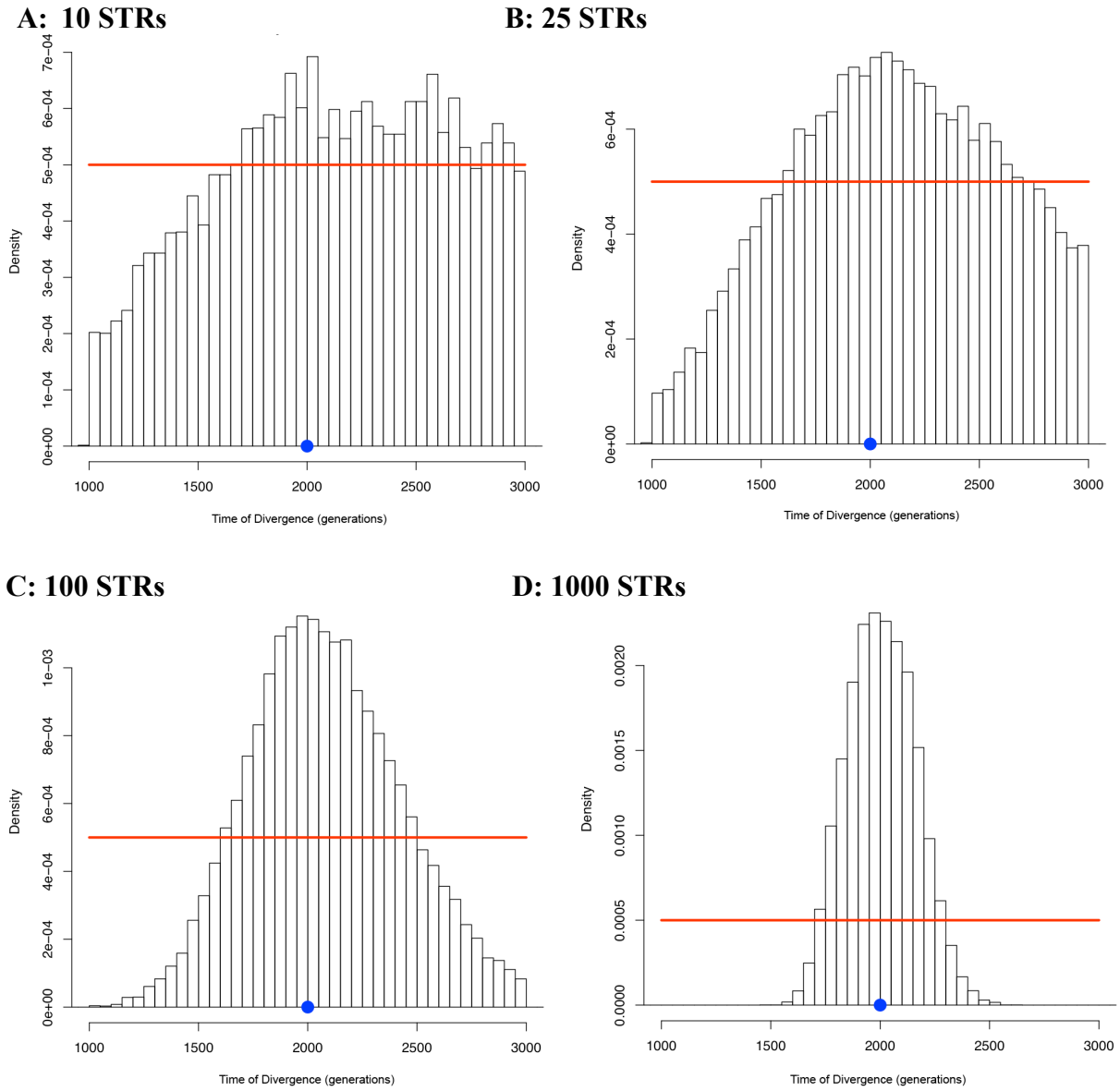


Figure 21. Effects of increasing number of loci. Histograms of divergence time estimates for unlinked STRs, using $\Box\mu^2$ and averaging across loci, accepted within a tolerance value of 0.1. For all the above there were 100,000 total runs of REJECTOR.

As the number of loci increase, the mean approaches the true divergence time of 2,000 generations and the variance of the divergence time estimates decrease markedly:

Table 2. Rate of acceptance, and mean, variance, and 95% credibility interval of time of divergence as a function of number of loci for $\delta\mu^2$. Mean and variance decrease, with an increasing number of loci.

Number of unlinked STRs	No. accepted runs (from 100,000)	Mean time of divergence (gens)	Variance in time of divergence	2.5% Credibility Interval	97.5% Credibility Interval
10	12770	2107	273364	1124	2950
25	16964	2105	227282	1193	2935
100	20407	2082	185551	1451	2792
1000	21234	2007	24786	1719	2319

Similar results were found for Nei's minimum genetic distance and T_D. As $\partial\mu^2/w$ is an estimator of time of divergence, where w represents effective mutation rate (Zhivotovsky 2001), and T_D is calculated from:

$$T_D = \frac{D_1}{2w} \quad \frac{V_0}{w}$$ (see also pg. 63)

where D_1 is the average over loci of the average squared difference between pairs of alleles sampled, the estimates for time of divergence for each statistic should be identical with V_0=0, as shown in Table 3.

Nei's distance showed a markedly higher rate of acceptance, as that statistic varied less with changes to time of divergence and thus had less power to detect changes in that parameter:

Table 3. Rate of acceptance, and mean, variance and 95% credibility interval of time of divergence for 100 unlinked STRs - $\delta\mu^2$, Nei's minimum genetic distance and T_D. Mean divergence time is comparable for all three statistics. Nei's minimum genetic distance is less sensitive to changes of divergence time, and thus has a higher acceptance rate.

Summary Statistic	No. accepted runs (from 100,000)	Mean time of divergence (gens)	Variance in time of divergence	2.5% Credibility Interval	97.5% Credibility Interval
$\delta\mu^2$	20407	2082	185551	1451	2792
Nei's	71880	2081	252520	1172	2941
T_D	20407	2082	185551	1451	2792

9.1.2 Gaussian Prior Distribution for Time of Divergence

We compared estimates of divergence time using 1000 unlinked STRs and the summary statistic $\Box\mu^2$ for two prior distributions. The first prior distribution was uniform, with the mean centered on the true parameter value. The second prior distribution was gaussian, with a mean offset from the true parameter value. Comparable posterior distributions resulted in both cases, with means close to the true parameter value, similar variances and similar rates of acceptance:

A: Uniform, centered on true value **B: Gaussian, offset from true value**

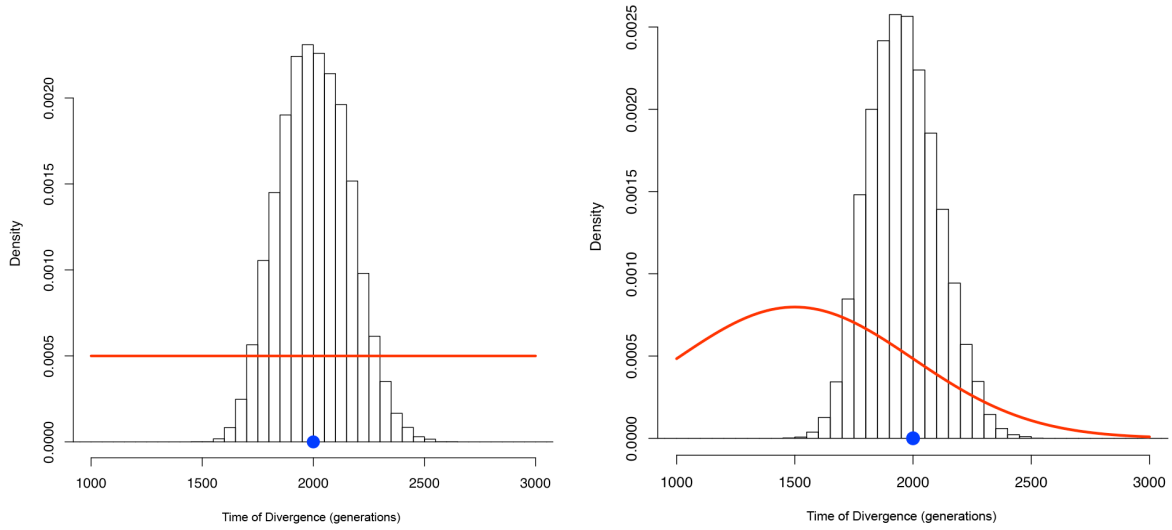


Figure 22. Comparison of uniform and gaussian prior distributions for estimating time of divergence. The mean of the uniform prior distribution is equal to the true parameter value, the mean of the gaussian prior distribution is not. Histograms of divergence time estimates for 1000 unlinked STRs, using $\Box\mu^2$, accepted within a tolerance value of 0.1. For all the above there were 100,000 total runs of REJECTOR.

Table 4. Rate of acceptance, and mean and variance of time of divergence for 100 unlinked STRs, using $\Box\mu^2$. Comparison of a uniform distribution with mean centered on the true parameter value and a gaussian distribution with a mean different than the true parameter value.

Prior Type	No. accepted runs (from 100,000)	Mean time of divergence (gens)	Variance in time of divergence	Prior Mean	True Parameter Value
Uniform	20079	2006	24817	2000	2000
Gaussian	19180	1958	24774	1500	2000

9.1.3 Effects of Decreasing Tolerance

Decreasing the tolerance had the expected effect of decreasing the variance and acceptance rate, as shown with 1,000 unlinked STRs and the summary statistic $\Box\mu^2$ in Figure 23:

A: 1000 STRs - tolerance 0.1 **B: 1000 STRs – tolerance 0.05**

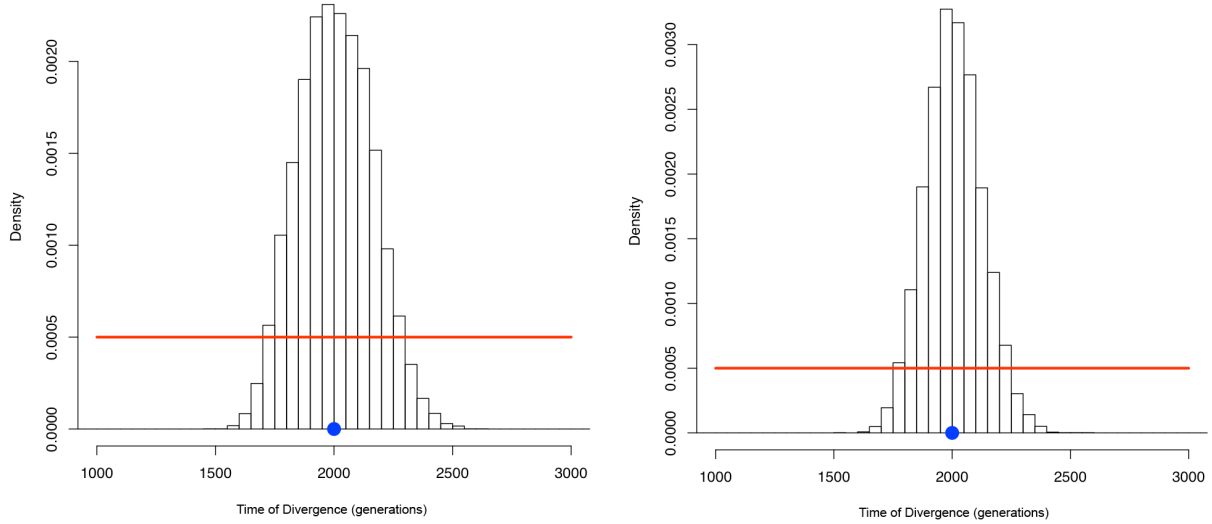


Figure 23. Effects of decreasing tolerance. Histograms of divergence time estimates for 1,000 unlinked STRs for $\Box\mu^2$ averaged across loci. For both of the above there were 100,000 total runs of REJECTOR.

Table 5. Rate of acceptance, and mean, variance and 95% credibility interval of time of divergence as a function of tolerance for $\Box\mu^2$. Variance and acceptance rate drop with a decrease in tolerance. Mean divergence time also decreases with tolerance, approaching the true parameter value of 2,000 generations.

Tolerance	No. accepted runs (from 100,000)	Mean time of divergence (gens)	Variance in time of divergence	2.5% Credibility Interval	97.5% Credibility Interval
0.1	20079	2082	185551	1719	2319
0.05	9890	2007	14499.17	1779	2251

9.1.4 SNPSTRs

SNPSTRs are short segments of DNA containing one or more SNPs and exactly one STR (Mountain, Knight et al. 2002). Here we perform a simple test of 100 unlinked STRs versus 100 unlinked SNPSTRs, performing the SNPSTR simulation SNP-blind, that is to say, with no attention paid to the state of the SNP. This makes the data equivalent, essentially 100 unlinked STRs per simulation. We performed this test to ensure that the software could deal with linked systems correctly.

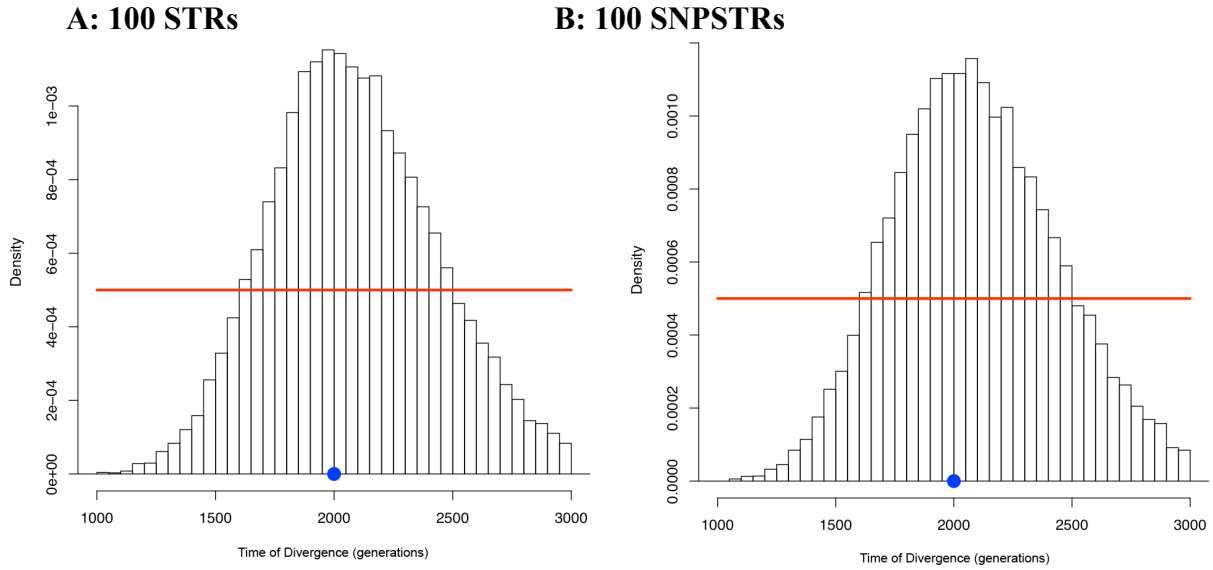


Figure 24. STRs versus SNPSTRs (SNP-blind). Histograms of divergence time for unlinked STRs, using $\Box\mu^2$ and averaging across loci, accepted within a tolerance value of 0.1. For both the above there were 100,000 total runs of REJECTOR.

The object of using linked systems such as SNPSTRs is to take advantage of the increased accuracy in estimation of parameters such as time of divergence that the combination of two or more closely-linked sites with widely differing mutation rates can afford (Ramakrishnan and Mountain 2004). We subdivided populations of 100 unlinked SNPSTRs based on SNP allele into ancestral and descendant populations, and compared estimates of time of divergence using $\Box\mu^2$. Time of divergence estimates for SNPSTRs with the ancestral SNP allele were similar to SNP-blind estimates. The SNPSTRs with the descendant SNP allele tended to underestimate the target parameter value of 2,000 generations. As each iteration of the program creates a new simulated genealogy, the descendent SNP is not present in every deme in all iterations.

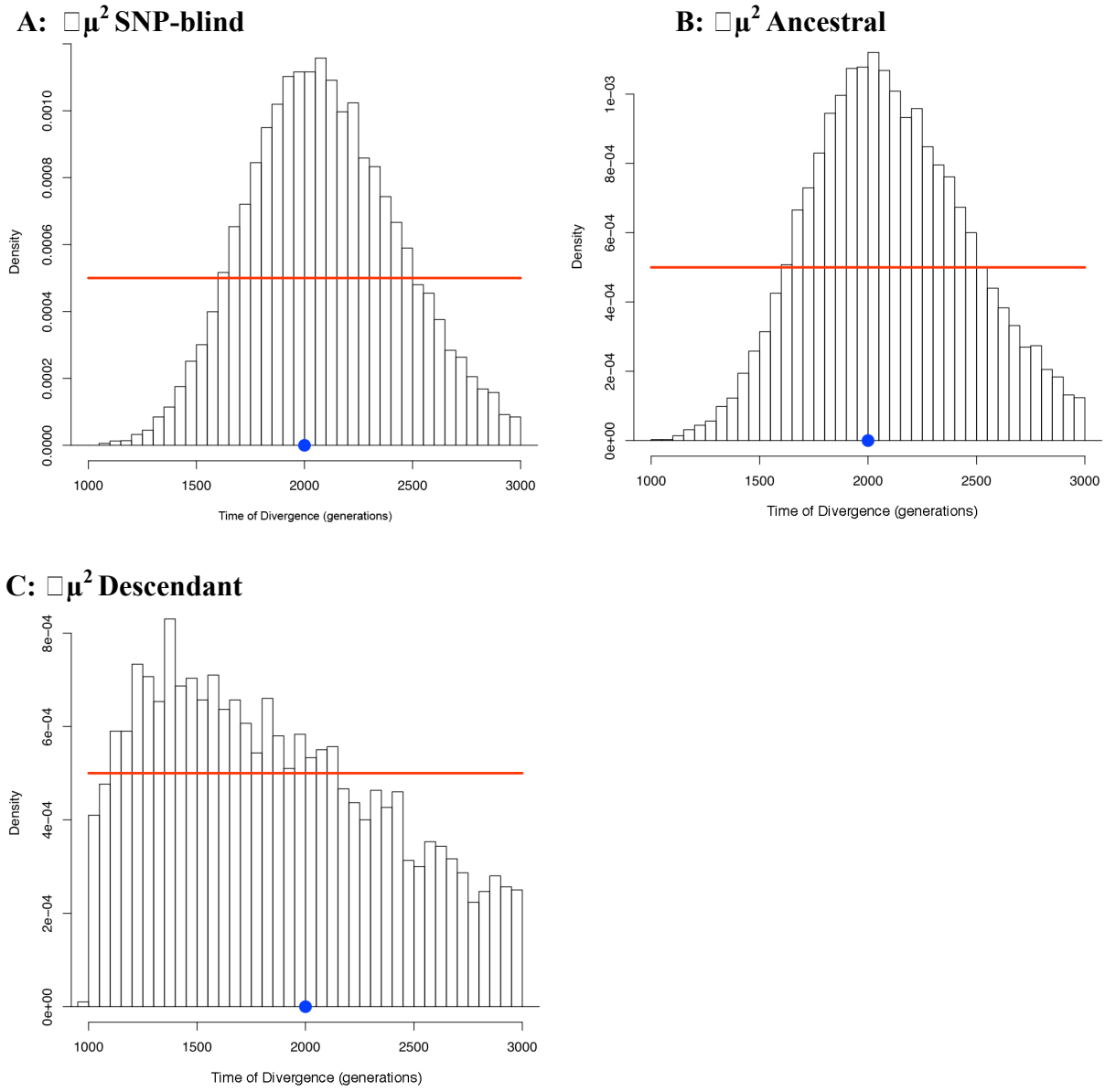


Figure 25. Effects of subdividing SNPSTRs based on SNP allele. Histograms of divergence time estimates for 100 unlinked SNPSTRs, accepted within a tolerance value of 0.1. For all the above there were 100,000 total runs of REJECTOR.

Table 6. Rate of acceptance, and mean, variance and 95% credibility interval of time of divergence for 100 unlinked SNPSTRs using μ^2 – SNP-blind, ancestral and descendant. The descendant sample had a lower mean divergence time and acceptance rate than the ancestral and SNP-blind samples, and a higher variance also apparent from its flatter distribution in Figure 25.

Population	No. accepted runs (from 100,000)	Mean time of divergence (gens)	Variance in time of divergence	2.5% Credibility Interval	97.5% Credibility Interval
SNP-blind	20287	2089	119062	1453	2801
Ancestral	20368	2094	129586	1434	2836
Descendant	5999	1846	276085	1060	2903

9.1.5 Nei's Minimum Genetic Distance – SNPs versus STRs

Variance in the posterior distribution of Nei's minimum standard genetic distance is lower for SNPs than STRs, as Figure 26 demonstrates. There is also a notable difference in the number of acceptances, as shown in Table 7.

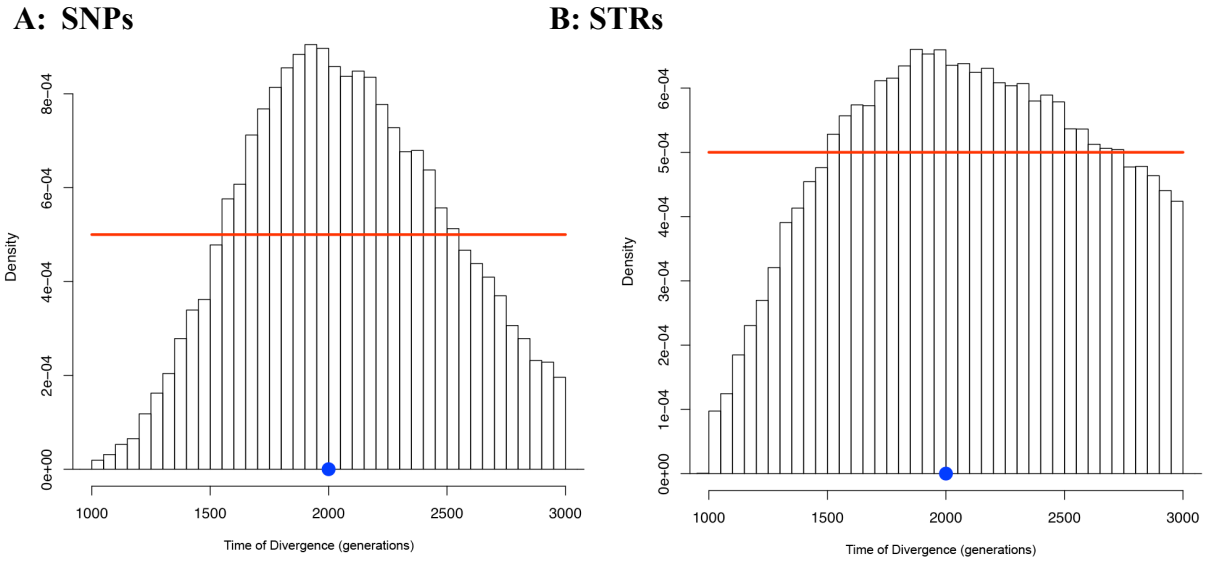


Figure 26. Nei's minimum genetic distance. Histograms of divergence time for 100 unlinked markers, accepted within a tolerance value of 0.1. For all the above there were 100,000 total runs of REJECTOR.

Table 7. Rate of acceptance, and mean, variance and 95% credibility interval of time of divergence for 100 unlinked markers using Nei's minimum genetic distance. Mean divergence time is roughly constant for all three types of locus, but variance and acceptance rate is much higher for STRs than for SNPs.

Marker	Mean time of divergence (gens)	Variance in time of divergence	No. accepted runs (from 100,000)	2.5% Credibility Interval	97.5% Credibility Interval
SNPs	2071	171981	23059	1314	2883
STRs	2081	252520	71880	1172	2792

9.1.6 Effects of Linkage Between Loci

We compared estimates of time of divergence using $\Box\mu^2$ and Nei's minimum genetic distance for 100 unlinked STRs and for 100 SNPs and STRs all fully linked to one another in a single non-recombining block. The fully-linked configuration tended to overestimate the true parameter value of 2,000 generations, as well as showing a higher variance, for $\Box\mu^2$. For Nei's minimum genetic distance there is little change in mean and variance from the unlinked STRs' already imprecise estimates, but a large drop in acceptance rate:

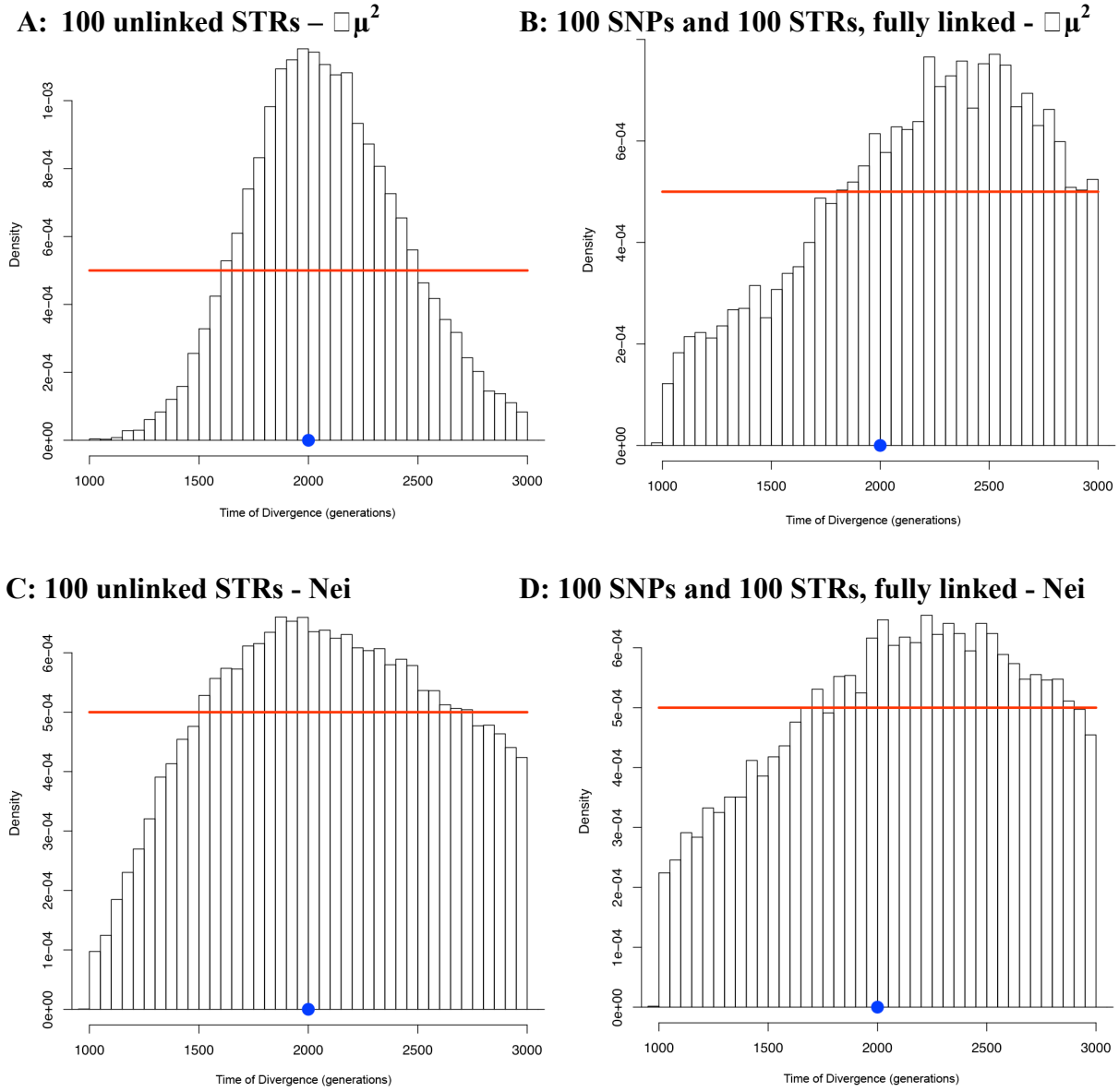


Figure 27. Comparison of unlinked and fully-linked systems. Histograms of divergence time estimates for unlinked STRs and fully linked blocks of 100 SNPs and 100 STRs, accepted within a tolerance value of 0.1. For all the above there were 100,000 total runs of REJECTOR.

Table 8. Rate of acceptance, and mean, variance and 95% credibility interval of time of divergence for a fully-linked block of 100 SNPs and 100 STRs, and for 100 unlinked STRs, using $\Box\mu^2$ and Nei's minimum genetic distance. The change from unlinked STRs to the fully-linked block decreases acceptance rate but increases variance.

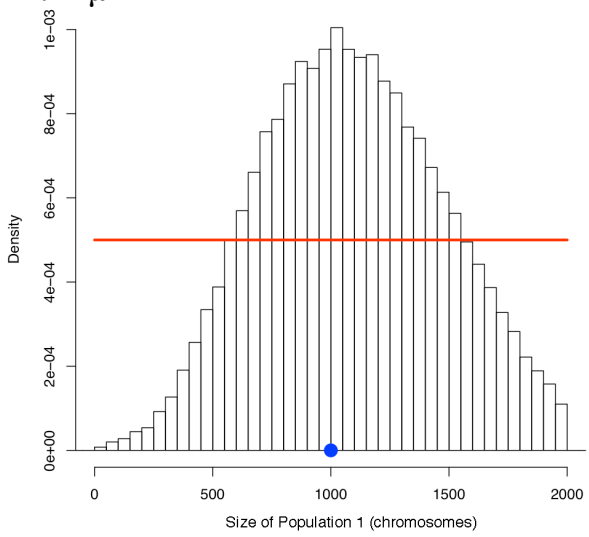
Configuration	Mean time of divergence (gens)	Variance in time of divergence	No. accepted runs (from 100,000)	2.5% Credibility Interval	97.5% Credibility Interval
100 unlinked STRs, $\Box\mu^2$	2082	118551	20407	1451	2792
Block of 100 SNPs and 100 STRs fully linked, $\Box\mu^2$	2186	248011	7551	1147	2952
100 unlinked STRs, Nei's	2081	252520	71880	1172	2941
Block of 100 SNPs and 100 STRs fully linked, Nei's	2103	276367	13112	1104	2944

9.2 Population Size

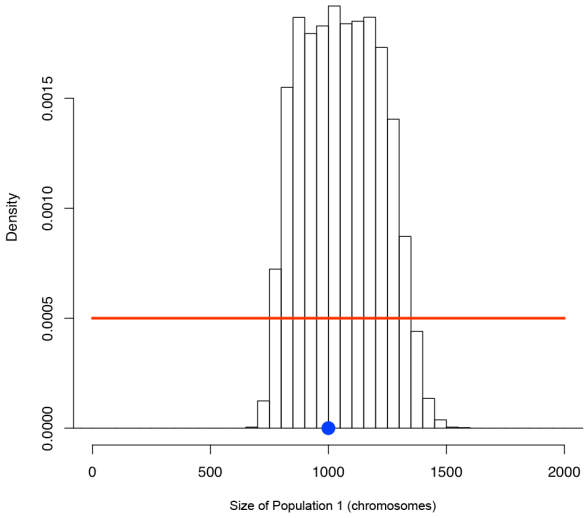
9.2.1 Comparison of Summary Statistics – 1000 STRs

We compared estimates of population size for various summary statistics using 1000 unlinked STRs. Microsatellite variance returned the most precise and accurate estimates of population size.

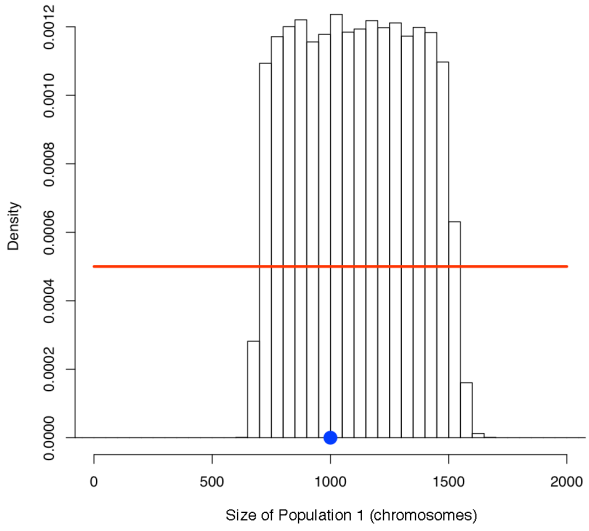
A: □μ²



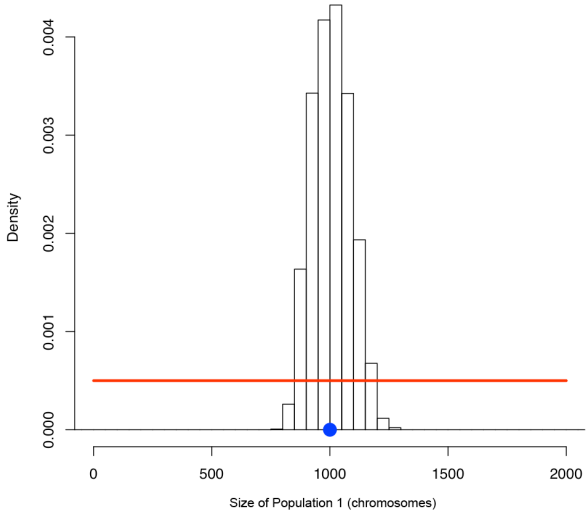
B: F_{ST}



C: Av. Heterozygosity



D: Microsatellite Variance



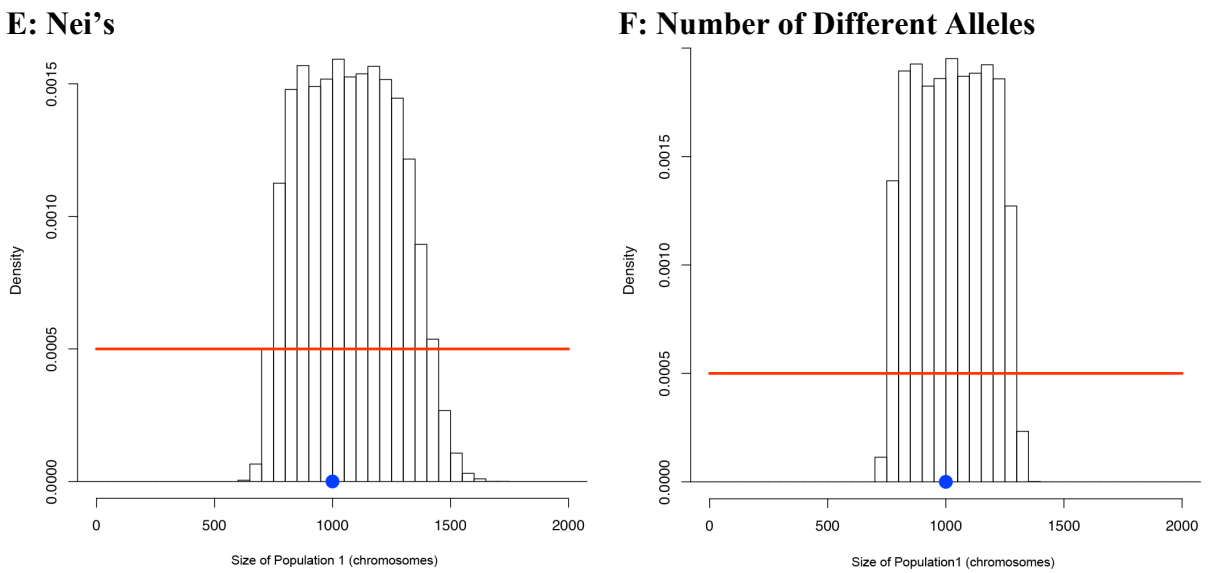


Figure 28. Comparison of summary statistics for estimation of time of divergence. Histograms of estimates of time of divergence for 1000 unlinked STRs, accepted within a tolerance value of 0.1. For all the above there were 100,000 total runs of REJECTOR.

Table 9. Rate of acceptance, and mean, variance, and 95% credibility interval of population size for six summary statistics.

Summary Statistic	No. accepted runs (from 100,000)	Mean time of divergence (gens)	Variance in time of divergence	2.5% Credibility Interval	97.5% Credibility Interval
□μ^2	49100	1090	146002	385	1840
F_{ST}	26959	1059	27109	784	1360
Av. Heterozyg.	41869	1112	59084	713	1519
Microsat. Variance	11529	1008	6513	864	1166
Nei's	32490	1076	39317	747	1439
Number of Different Alleles	26524	1026	23825	770	1285

9.2.2 Gaussian Prior Distribution for Population Size

We compared estimates of population size using 1000 unlinked STRs and the summary statistic microsatellite variance for two prior distributions. The first prior distribution was uniform, with the mean centered on the true parameter value. The second prior distribution was gaussian, with a mean offset from the true parameter value. Comparable posterior distributions resulted in both cases, with means close to the true parameter value, similar variances and similar rates of acceptance:

A: Uniform, centered on true value **B: Gaussian, offset from true value**

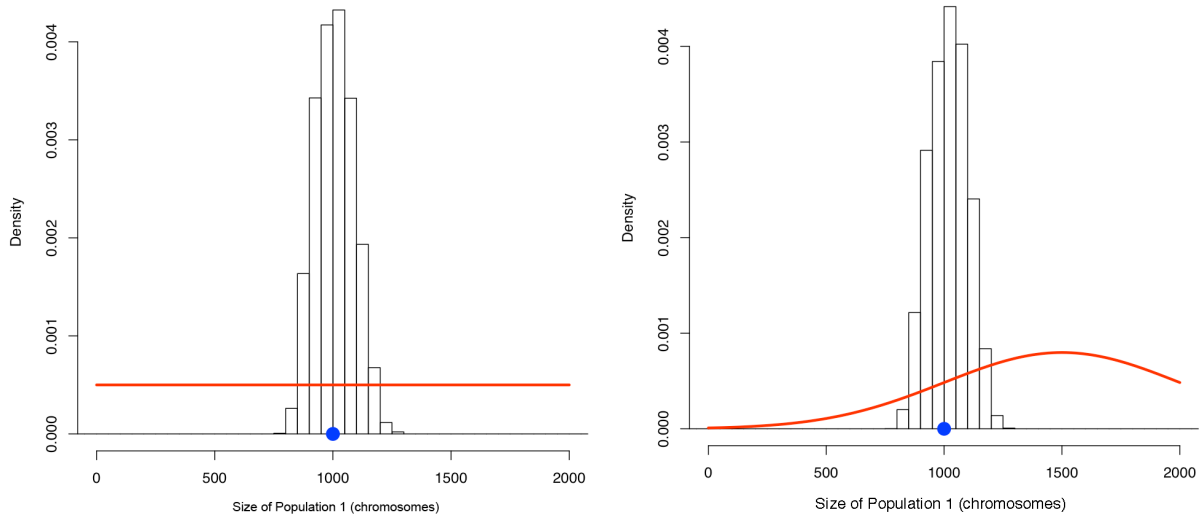


Figure 29. Comparison of uniform and gaussian prior distributions for estimating population size. The mean of the uniform prior distribution is equal to the true parameter value, the mean of the gaussian prior distribution is not. Histograms of population size estimates for 1000 unlinked STRs, using μ^2, accepted within a tolerance value of 0.1. For all the above there were 100,000 total runs of REJECTOR.

Table 10. Rate of acceptance, and mean and variance of population size for 1000 unlinked STRs, using microsatellite variance. Comparison of a uniform distribution with mean centered on the true parameter value and a gaussian distribution with a mean different than the true parameter value.

Prior Type	No. accepted runs (from 100,000)	Mean time of divergence (gens)	Variance in time of divergence	Prior Mean	True Parameter Value
Uniform	11529	1008	6513	864	1166
Gaussian	10689	1020	6424	870	1172

9.2.3 Comparison of SNPs and STRs

Population size can be estimated using both SNPs and STRs, as shown in Figure 30, with the more precise estimate dependent on the summary statistic used. The simulations displayed in that figure which used 100 STRs had a higher rate of acceptance, as shown in Table 11. Also, as shown in Table 11, the simulations using 100 STRs and Nei's minimum genetic distance (Figure 30D) have the lowest variance, despite having a higher acceptance rate than 100 SNPs with Nei's minimum genetic distance, as well as a much lower variance than 100 STRs using average heterozygosity (Figure 30B) despite having a roughly similar acceptance rate.

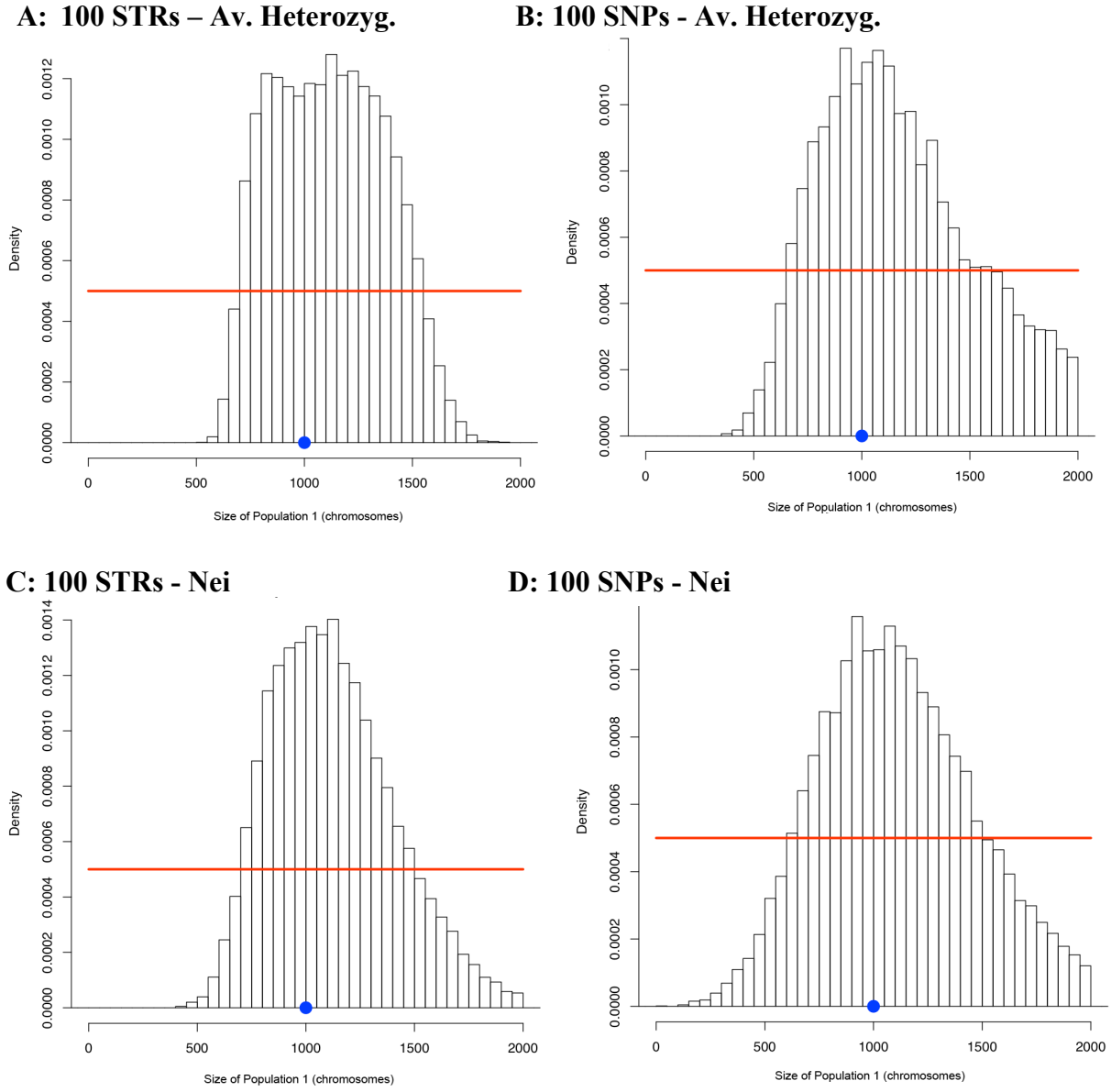


Figure 30. Comparison of population size estimates using SNPs and STRs. Histograms of population size, accepted within a tolerance value of 0.1. For all the above there were 100,000 total runs of REJECTOR.

Table 11. Rate of acceptance, and mean, variance, and 95% credibility interval of population size for 100 SNPs and for 100 STRs for average heterozygosity and Nei's minimum genetic distance. For average heterozygosity, SNPs provide the lower variance and acceptance rate, while for Nei's minimum genetic distance, STRs show a lower variance and a higher acceptance rate.

Configuration	No. accepted runs (from 100,000)	Mean size of Pop. 1	Variance in size of Pop. 1	2.5% Credibility Interval	97.5% Credibility Interval
100 SNPs, Av. Het.	8918	1165	121373	609	1900
100 STRs, Av. Het.	41651	1118	280763	692	1600
100 SNPs, Nei's	18279	1104	123523	478	1839
100 STRs, Nei's	32933	1123	79132.8	663	1742

For all summary statistics increasing the number of loci decreases the variance, as exemplified by Table 12 below:

Table 12. Rate of acceptance, and mean, variance, and 95% credibility interval of population size for unlinked STRs for μ^2. Increasing the number of loci decreases the variance and increases acceptance rate.

No. of STRs	No. accepted runs (from 100,000)	Mean size of Pop. 1	Variance in size of Pop. 1	2.5% Credibility Interval	97.5% Credibility Interval
10	14997	982	324080	50	1944
25	22409	990	318116	55	1945
100	36923	1021	280763	79	1936

9.3 Migration Rate

We evaluated various summary statistics for their ability to recover migration rate by simulating with a uniform prior distribution. The boundaries for the prior distribution were 0 to 0.001 as a proportion of each population permitted to migrate to the other in a 2-population model. A different target population model was used for these simulations (Figure 31). The true migration rate was 0.0001 per generation, demonstrating REJECTOR'S ability to make accurate estimates of parameter values when the mean of the prior distribution is not equal to the true parameter value.

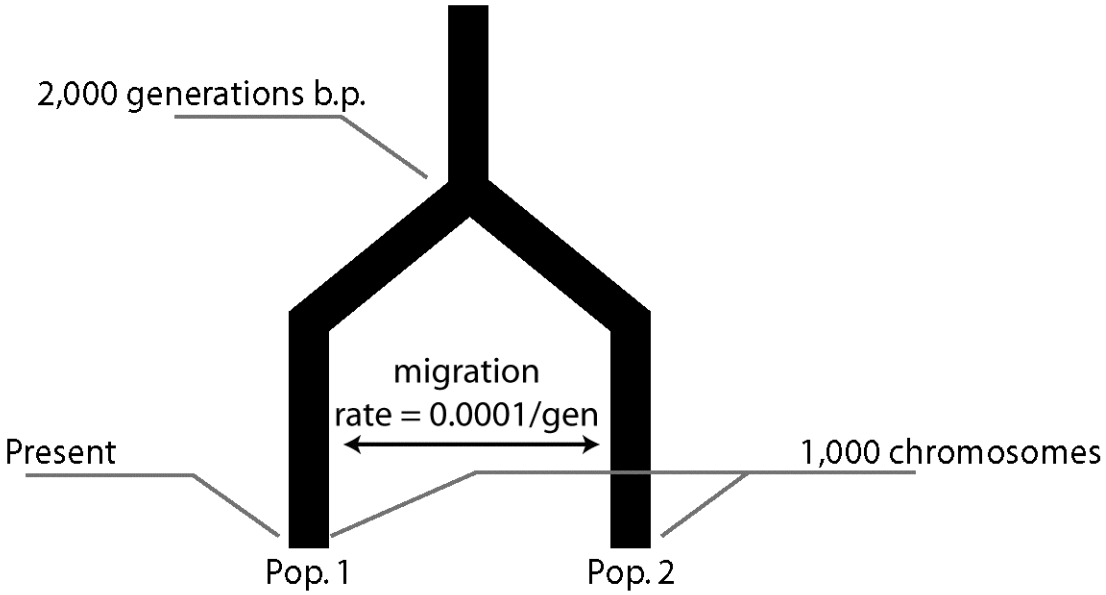
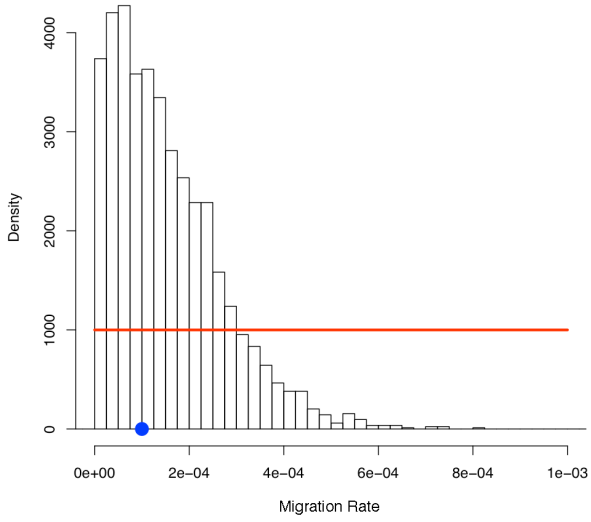


Figure 31. Target population model for simulations with prior distributions for migration rate.

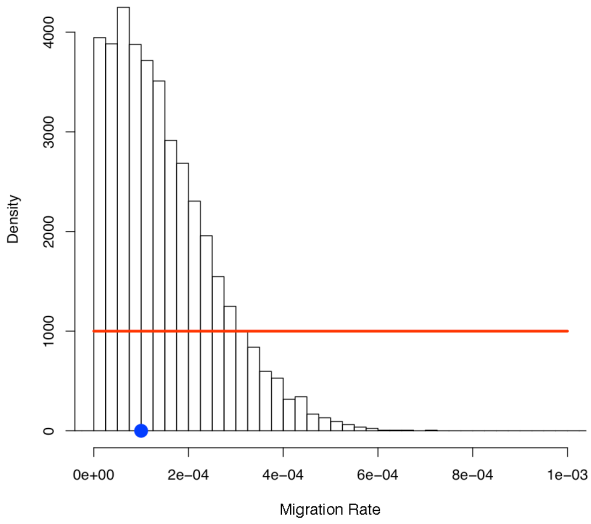
9.3.1 Comparison of Summary Statistics – 100 STRs

We compared estimates of migration rate for various summary statistics using 100 unlinked STRs. The results indicated that the summary statistics $\Delta\mu^2$, $F_{ST,}$ and Nei's minimum genetic distance were most capable of recovering the true value of this parameter. The migration rate estimates were very similar for each direction, so shown here are only the estimates of migration rate from Population 1 to Population 2.

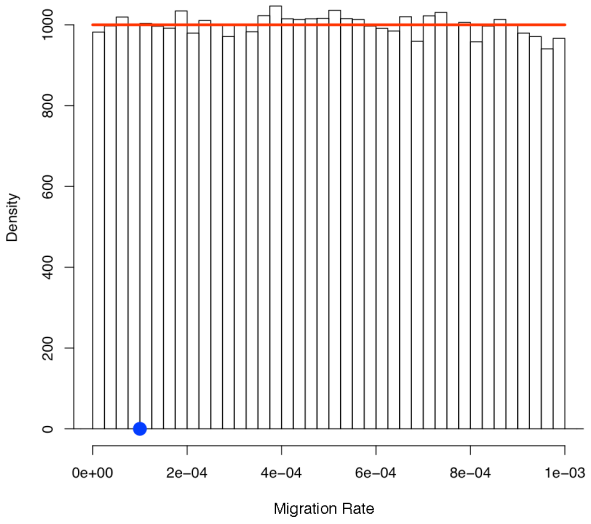
A: $\Delta\mu^2$



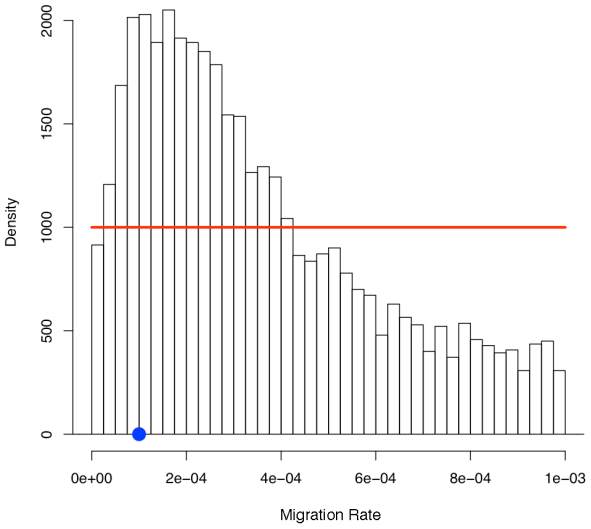
B: F_{ST}



C: Av. Heterozygosity



D: Microsatellite Variance



E: Nei's **F: Number of Different Alleles**

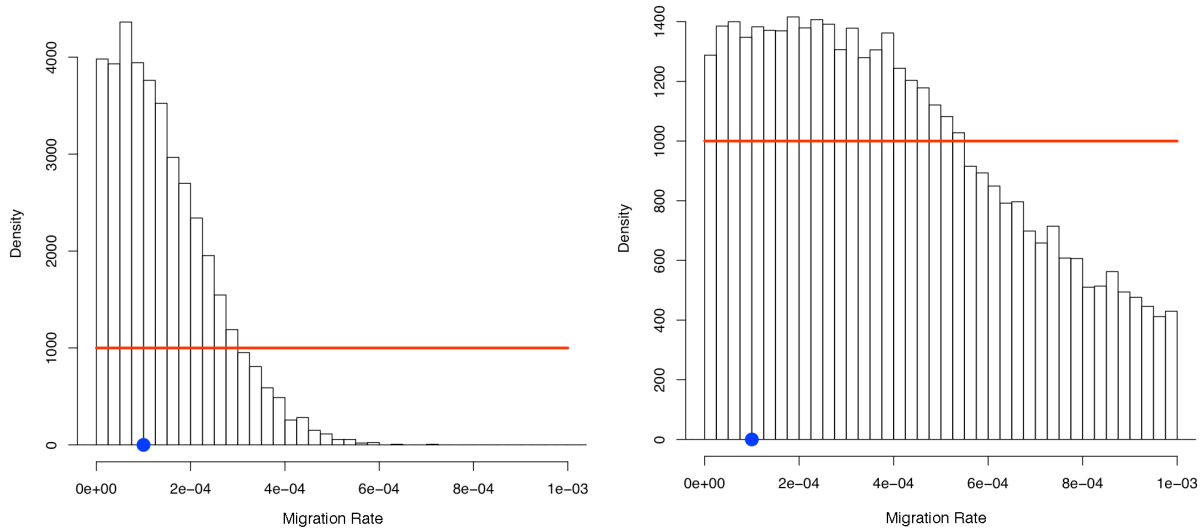


Figure 32. Comparison of summary statistics for estimation of migration rate. Histograms of estimates of time of divergence for 100 unlinked STRs, accepted within a tolerance value of 0.1. For all the above there were 100,000 total runs of REJECTOR.

Table 13. Rate of acceptance, and mean, variance, and 95% credibility interval of migration rate for six summary statistics.

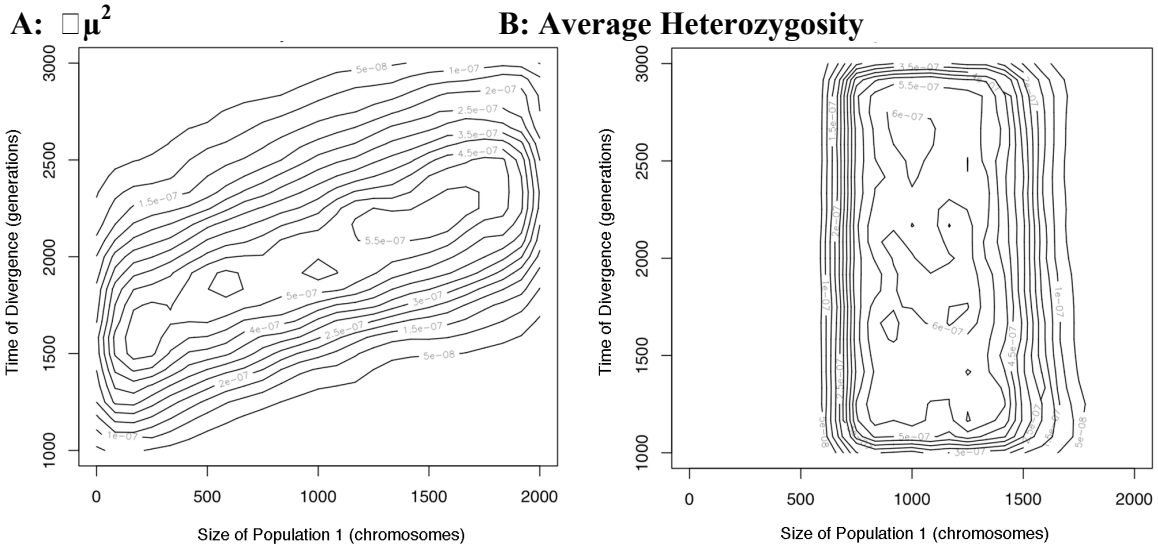
Summary Statistic	No. accepted runs (from 100,000)	Mean time of divergence (gens)	Variance in time of divergence	2.5% Credibility Interval	97.5% Credibility Interval
□μ^2	3360	1.54E-04	1.37E-08	6.72E-06	4.36E-04
F_{ST}	6438	1.49E-04	1.20E-08	6.52E-06	4.13E-04
Av. Heterozyg.	98502	4.98E-04	8.26E-08	2.54E-05	9.74E-04
Microsat. Variance	5599	3.52E-04	6.30E-08	2.69E-05	9.38E-04
Nei's	6391	1.46E-04	1.13E-08	6.57E-06	3.99E-04
Number of Different Alleles	44608	4.01E-04	6.84E-08	1.95E-05	9.42E-04

9.4 Two-parameter Simulations – Population Size and Time of Divergence

The last series of tests allowed both population size for population 1 and the time of divergence to vary simultaneously, both with uniform priors. The results can then be presented as a 2-dimensional density plot to show each summary statistic's response to changes in both parameters.

9.4.1 Comparison of Summary Statistics, Including Multiple Summary Statistics per Run

Figure 33D displays a run of REJECTOR where both $\Box\mu^2$ and average heterozygosity must be within tolerance for an iteration to be accepted. This plot demonstrates that the use of multiple summary statistics in concert can provide more accurate estimates than any summary statistic alone, particularly for models where multiple parameters are allowed to vary.



C: Nei's minimum genetic distance **D: □μ² and Average Heterozygosity**

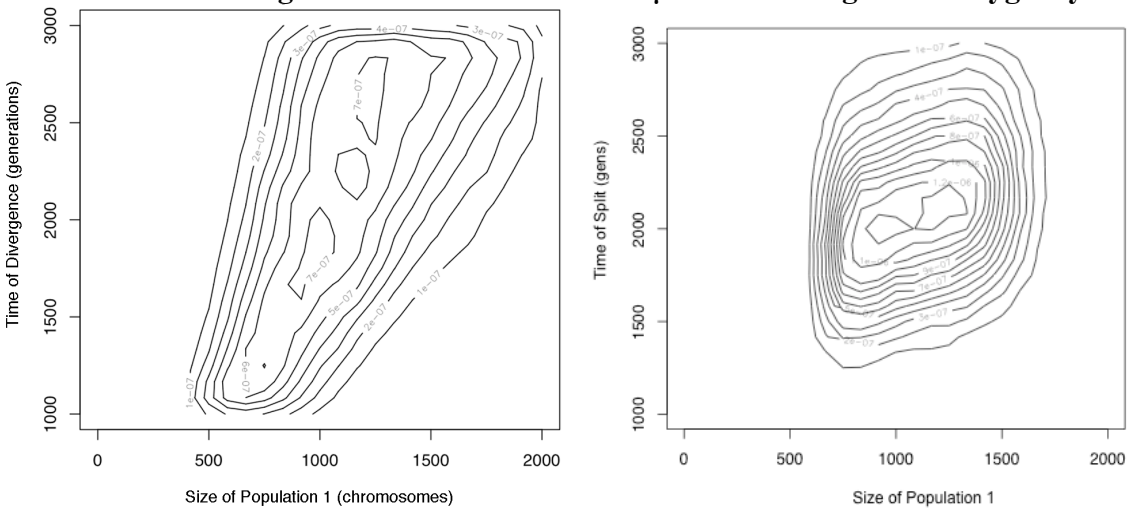


Figure 33. Density plots of time of divergence by population size for 100 unlinked STRs, accepted within a tolerance value of 0.1. For **A**, **B** and **C** there were 100,000 total runs of REJECTOR. **A:** □μ^2, with 20,306 accepted runs. **B:** Average heterozygosity, with 42,332 accepted runs. **C:** Nei's minimum genetic distance, with 31,487 accepted runs. For **D** there were 84,089 accepted runs out of 1,000,000 total runs where both □μ^2 and average heterozygosity needed to be within tolerance to accept.

9.4.2 Effects of Decreased Tolerance

Decreasing the tolerance had the expected effects of decreasing the rate of acceptance while increasing accuracy. Long running simulations using large numbers of unlinked loci and multiple summary statistics at a low tolerance will be the most useful in estimating parameters and comparing models using real data. Care must be taken when setting up such simulations to pick summary statistics that will recover parameters of interest (see Sections 9.1-9.3 above). This paper can serve as a guide for picking statistics for times of divergence, population size, and migration rate, and similar recovery tests to those performed above can be run on other parameters to determine the best summary statistics to use with them.

A: □μ^2 and Av. Het. – Tolerance 0.1 **B: □μ^2 and Av. Het. – Tolerance 0.05**

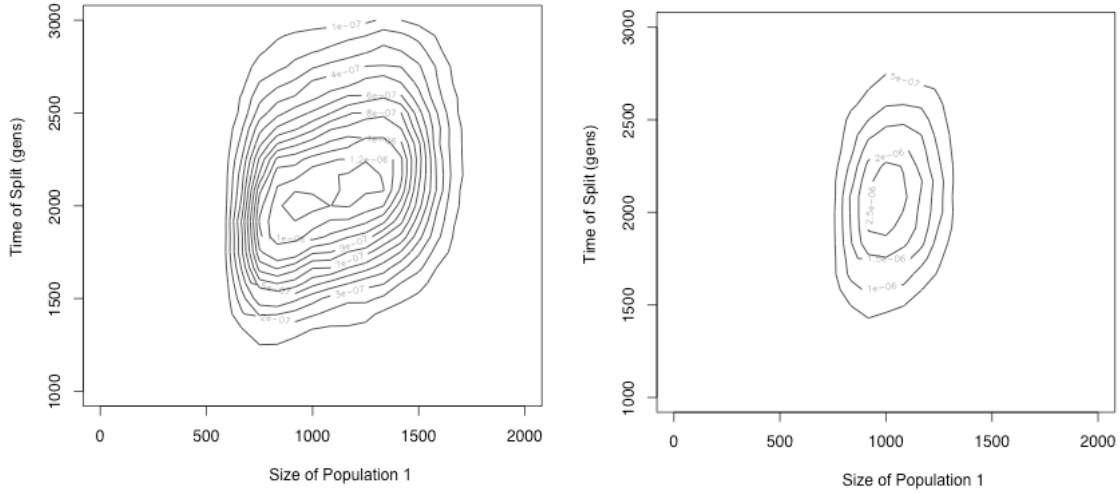


Figure 34. Effects of decreased tolerance. Density plots of time of divergence versus size of population 1 for 100 unlinked STRs. For both the above there were 100,000 total runs of REJECTOR, the target time of divergence is 2,000 generations, and the target population size is 1,000 chromosomes.

9.4.3 Comparison of Uniform and Gaussian Prior Distributions

We tested REJECTOR using μ^2 and microsatellite variance with a tolerance of 0.05 with both uniform and gaussian priors for precision and accuracy in estimating both time of divergence and population size. As shown in Figure 35, the posterior distributions are centered on the true values of both parameters. In the case of Figure 35A, the priors were uniform with means equal to the true parameter values. In the case of Figure 35B, the priors were gaussian distributions with means equal to 1250 chromosomes for population size and 2250 generations for time of divergence, as shown by the light grey contours of the figure. In both cases the posterior distributions centered on the true parameter values.

A: μ^2 and Microsat Variance – Uniform **B: μ^2 and Microsat Variance – Gaussian**

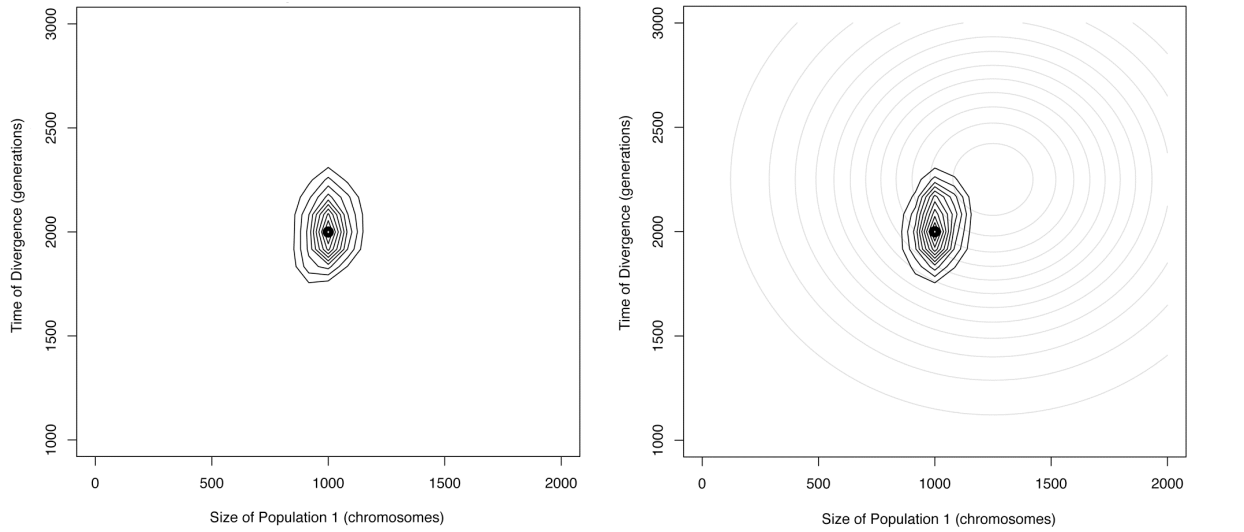


Figure 35. μ^2 and Microsat Variance. Density plot of time of divergence versus size of population 1 for 100 unlinked STRs at a tolerance of 0.05. For both the above there were 100,000 total runs of REJECTOR, the target time of divergence is 2,000 generations, and the target population size is 1,000. For Figure 35A, uniform priors were used equivalent to the whole of the graphed area; for Figure 35B, gaussian priors were used with density as shown by the light grey contour lines. The black dot represents the true values of the parameters.

10. Invoking REJSTATS as a Stand-Alone Program

REJSTATS is a command-line program. To use it, open a terminal window (i.e. the Terminal app on the Macintosh) and navigate to the directory where the program resides. Once there, the program can be most simply invoked by typing:

```
./rejstats
```

which runs the copy of rejstats in that directory. Alternatively, you can place the executable in a directory in your PATH and invoke by typing:

```
rejstats
```

which will invoke REJSTATS from any directory. If you invoke it in this method, rejstats will ask you for the data file name and location. Type it in, hit enter, and REJSTATS will go to work. You can also specify the input file on the command line:

```
./rejstats –f [input file]
```

Note that if the input file is not the same as the current working directory, it must be specified in the command. For example:

```
./rejstats –f /usr/local/rejector/testrejector.txt
```

10.1 Checking for number of alleles per SNP with -sl

The `-sl` switch makes REJSTATS check that there are 2 or less alleles listed for every SNP. Note that when REJSTATS is invoked by REJECTOR this option is used by default.

```
./rejstats –sl –f [input file]
```

10.2 Reading Arlequin-format files

Arlequin-style file formats can be read in if accompanied by a REJSTATS-readable summary file. This is normally done by REJECTOR when invoking REJSTATS to read SIMCOAL2 output, but can be done manually. The Arlequin-style input file goes first, then the summary file:

```
./rejstats –f [Arlequin-style input file] –o
[summary file]
```

10.3 Invoking in batch mode

REJSTATS can be run in batch mode using the `-b` switch. See the examples folder for examples of batch files.

```
./rejstats –b [input batch file]
./rejstats –sl –b –f [Arlequin-style input batch file] –o [summary batch file]
```

10.4 Converting Arlequin-style file to REJECTOR input files

Arlequin-style files plus summary files can be turned back into rejstats input files with the `-i` switch:

```
./rejstats –i [Arlequin-style input file] [summary file]
```

10.5 REJSTATS output

Output files are created as tab-delimited text files with the same name as the input file save that the extension is changed to ".out".

11. Invoking PRIOR as a Stand-alone Program

PRIOR is a simple program that takes the output of REJSTATS and uses it to create input files for SIMCOAL2. Each input to PRIOR consists of a list of priors that contains the distributions for creating a randomized SIMCOAL2 input file. PRIOR uses these distributions to create one input file with random parameters each time it is invoked. PRIOR uses summary files (.sum) from REJSTATS output. It is designed to be invoked by REJECTOR but can be invoked as a stand-alone program.

PRIOR is a command-line program. To use it, open a terminal window (i.e. the Terminal app on the Macintosh) and navigate to the directory where the program resides. Once there, the program can be most simply invoked by typing:

```
./prior
```

which runs the copy of PRIOR in that directory. Alternatively, you can place the executable in a directory in your PATH and invoke by typing:

```
prior
```

which will invoke PRIOR from any directory. If you invoke it in this method, PRIOR will ask you for the summary file name and location. Type it in, hit enter, and PRIOR will go to work. You can also specify the summary file on the command line:

```
./prior –f [summary file]
```

11.1 PRIOR output

PRIOR outputs a file named [file].par, where [file] corresponds to the name of the summary file, without an extension. This .par file can then be read by SIMCOAL2.

12. Scripts

12.1 makerejin.pl – Making dummy input files

You might find it useful to create dummy input files, either for testing purposes or to create a template. The `makerejin.pl` script will do this for the Data section of the file. Create a text file with the priors and statistics you would like to use, then invoke the script:

```
perl makerejin.pl <priors and statistics file> <output file>
```

The script will ask you about the details of the data you would like, then print a file full of random data of the requested type.

12.2 hapmapconvert.pl – Convert files downloaded from HapMap.org

This Perl script takes data downloaded from the HapMap.org browser using the "Download SNP Genotype data" function from the Reports and Analysis section of the browser. Please see http://www.hapmap.org/tutorials.html.en for details on how to use the HapMap browser. Please also see the REJECTOR Tutorial starting on page 13 for a walkthrough of the use of this script.

12.3 runrs – Run REJSTATS on multiple input files

You can use the shell script runrs to run REJSTATS on multiple files in a directory. This very simple script attempts to run REJSTATS on every file in the folder with the extension .txt, so make sure you have only valid input files in the folder before you begin. You can invoke the script using the following command:

```
./runrs
```

12.4 runrej – Run REJECTOR on multiple input files

This shell script works in a similar manner to the runrs script, but it attempts to run REJECTOR on all files with the extension .txt in the directory. It is mainly used to test REJECTOR, and so specifies only 10 iterations. You can change this by editing the runrej file using a text editor. You can invoke the script using the following command:

```
./runrej
```

12.5 rejclean – Remove intermediate files

You can run this script to quickly clean out any intermediate files (.sum files, .out files and so on) from your REJECTOR directory. REJECTOR does clean out most of these intermediate files automatically, but you can run this script to clean things up after an aborted run.

13. References

Anderson, C. N. K., U. Ramakrishnan, et al. (2005). "Serial SimCoal: A population genetics model for data from multiple populations and points in time." Bioinformatics (Oxford) **21**(8): 1733.

Beaumont, M. A. (2004). "Recent developments in genetic data analysis: what can they tell us about human demographic history?" Heredity **92**(5): 365.

Beaumont, M. A., W. Y. Zhang, et al. (2002). "Approximate Bayesian computation in population genetics." Genetics **162**(4): 2025.

Beerli, P. and J. Felsenstein (1999). "Maximum-likelihood estimation of migration rates and effective population numbers in two populations using a coalescent approach." Genetics **152**(2): 763.

Beerli, P. and J. Felsenstein (2001). "Maximum likelihood estimation of a migration matrix and effective population sizes in n subpopulations by using a coalescent approach." Proceedings of the National Academy of Sciences of the United States of America **98**(8): 4563.

Cavalli-Sforza, L. L., A. Piazza, et al. (1994). History and Geography of Human Genes. Princeton, N.J., Princeton University Press.

Excoffier, L. and G. Heckel (2006). "Computer programs for population genetics data analysis: a survival guide." Nat Rev Genet **7**(10): 745.

Excoffier, L., J. Novembre, et al. (2000). "SIMCOAL: A general coalescent program for the simulation of molecular data in interconnected populations with arbitrary demography." Journal Of Heredity **91**: 506-509.

Fu, Y. X. and W. H. Li (1997). "Estimating the age of the common ancestor of a sample of DNA sequences." Molecular Biology and Evolution **14**(2): 195-199.

Goldstein, D. B., A. Ruiz Linares, et al. (1995). "An evaluation of genetic distances for use with microsatellite loci." Genetics **139**: 463-471.

Goldstein, D. B., A. Ruiz Linares, et al. (1995). "Genetic absolute dating based on microsatellites and the origin of modern humans." Proc Natl Acad Sci USA **92**: 6723-6727.

Hill, W. G. and A. Robertson (1968). "Linkage disequilibrium in finite populations." TAG Theoretical and Applied Genetics.

Page 1. Theoretical and Applied Genetics 38, 226--23 t (t968) WG HILL and ALAN ROBERTSON*

Hudson, R. R. (1990). Gene genealogies and the coalescent process. Oxford Sur. Evol. Biol. D. Futuyma and J. Antonovics. Oxford, Oxford Univ. Press. **7:** 1-44.

Kimmel, M., R. Chakraborty, et al. (1998). "Signatures of population expansion in microsatellite repeat data." Genetics **148**: 1921-1930.

Kingman, J. F. C. (1982). "On the genealogy of large populations." **19A**: 27-43.

Laval, G. and L. Excoffier (2004). "SIMCOAL 2.0: a program to simulate genomic diversity over large recombining regions in a subdivided population with a complex history." Bioinformatics (Oxford) **20**(15): 2485.

Macpherson, J. M., S. Ramachandran, et al. (2004). "Demographic estimates from Y chromosome microsatellite polymorphisms: Analysis of a worldwide sample." Human Genomics **1**(5): 345.

Marjoram, P., J. Molitor, et al. (2003). "Markov chain Monte Carlo without likelihoods." Proceedings of the National Academy of Sciences of the United States of America **100**(26): 15324.

Mountain, J. L. and L. L. Cavalli-Sforza (1997). "Multilocus genotypes, a tree of individuals, and human evolutionary history." Am J Hum Genet **61**: 705-718.

Mountain, J. L., A. Knight, et al. (2002). "SNPSTRs: Empirically derived, rapidly typed, autosomal Haplotypes for inference of population history and mutational processes." Genome Research **12**: 1766-1772.

Mountain, J. L., A. A. Lin, et al. (1992). "Evolution of modern humans: evidence from nuclear DNA polymorphisms." Phil. Trans. Royal Soc. B **337**: 159-165.

Nei, M. (1987). Molecular Evolutionary Genetics. New York, Columbia University Press.

Nordborg, M. (2001). Handbook of Statistical Genetics. Chichester, John Wiley and Sons**:** 179-212.

Pamilo, P. and M. Nei (1988). "Relationships between gene trees and species trees." Mol. Biol. Evol. **5**: 568-583.

Pritchard, J. K., M. T. Seielstad, et al. (1999). "Population growth of human Y chromosomes: A study of Y chromosome microsatellites." Molecular Biology and Evolution **16**(12): 1791.

Ramakrishnan, U., E. A. Hadly, et al. (2005). "Detecting past population bottlenecks using temporal genetic data." Molecular Ecology **14**(10): 2915.

Ramakrishnan, U. and J. L. Mountain (2004). "Precision and accuracy of divergence time estimates from STR and SNPSTR variation." Molecular Biology and Evolution **21**(10): 1960.

Ramakrishnan, U. M. A., J. F. Storz, et al. (2004). "Estimation of genetically effective breeding numbers using a rejection algorithm approach." Molecular Ecology **13**(11 %R doi:10.1111/j.1365-294X.2004.02326.x): 3283-3292.

Rosenberg, N. A. and M. Nordborg (2002). "Genealogical trees, coalescent theory and the analysis of genetic polymorphisms." Nature Reviews Genetics **3**(5): 380.

Shoemaker, J. S., I. S. Painter, et al. (1999). "Bayesian statistics in genetics: A guide for the uninitiated." Trends in Genetics **15**(9): 354.

Shriver, M. D., L. Jin, et al. (1995). "A Novel Measure of Genetic-Distance for Highly Polymorphic Tandem Repeat Loci." Molecular Biology And Evolution **12**: 914-920.

Stephens, M. and P. Donnelly (2000). "Inference in molecular population genetics." Journal of the Royal Statistical Society Series B-Statistical Methodology **62**: 605.

Tavare, S., D. J. Balding, et al. (1997). "Inferring coalescence times from DNA sequence data." Genetics **145**(2): 505.

Weber, J. L. and C. Wong (1993). "Mutation of human short tandem repeats." Hum. Molec. Genet. **2**: 1123-1128.

Weir, B. S. (1996). Genetic Data Analysis II: Methods for Discrete Population Genetic Data. Sunderland, Sinauer Associates, Inc.

Weiss, G. and A. von Haeseler (1998). "Inference of population history using a likelihood approach." Genetics **149**(3): 1539.

Wilson, G. A. and B. Rannala (2003). "Bayesian inference of recent migration rates using multilocus genotypes." Genetics **163**(3): 1177.

Zhivotovsky, L. A. (2001). "Estimating divergence time with the use of microsatellite genetic distances: Impacts of population growth and gene flow." Molecular Biology And Evolution **18**: 700-709.