Using Rmetasim

A. Strand J. Niehaus

September 2, 2003

1 Introduction

1.1 How Metasim Works

Metasim itself works by implementing a random Markov model of population dynamics. It is assumed that the reader has a working knowledge of this model before proceeding. More information can be found here: metasim text,web resources, other resources Book: author = "Caswell, Hal" title = "Matrix Population Models. Construction, analysis and interpretation" publisher = "Sinauer" year = "1989"

1.2 Rmetasim Landscape Object

The Rmetasim landscape object is a list of lists at its topmost level. Reading a landscape from a text file is a good way to quickly grab an example landscape. Starting in R: (METASIM_DIR is either the directory that metasim was unzipped to or installed into)

```
> library(rmetasim)
> rland <- read.landscape("<METASIM_DIR>/examples/data/wf/wf1d1e.dat")
> rland
```

The last line will spew out the entire contents of the landscape object in a dizzying blur, however analysis can easily be broken down into pieces:

```
> names(rland)
[1] "intparam"    "switchparam" "floatparam"  "demography"  "loci"
[6] "individuals"
```

This statement lists the six main sections of the landscape: integer parameters, switch parameters, floating point parameters, demography, loci, and individuals. The first three affect how the simulation treats the landscape. The last three are values manipulated in the course of the simulation.

intparam Having a look at intparam:

```
> names(rland$intparam)
[1] "habitats"     "stages"       "locusnum"     "numepochs"    "currentgen"
[6] "currentepoch" "totalgens"    "numdemos"     "maxlandsize"
```

In order these values represent: habitats: the number of different habitats or subpopulations within the landscape stages: the number of stages in the life cycle of the organism; also the size of the S, R, and M matricies for local demography locusnum: the number of different loci currently implemented for the simulation numepochs: the number of different migration scenerios to occur during the simulation currentgen: the current generation the simulation has reached currentepoch: the current epoch the simulation has reached totoalgens: the total number of generations to simulate numdemos: the number of within population demographies maxlandsize: the maxium number of individuals that can exist in the simulation

floatparam The floatparam structure:

```
> names(rland$floatparam)
[1] "selfing"
```

This value being the selfing rate of the species (between 0 and 1 inclusive).

switchparam The switchparam structure:

```
> names(rland$switchparam)
[1] "randepoch" "randdemo"  "multp"
```

These values represent: randepoch: 1=choose the next epoch randomly using their individual probabilities, 0=choose the next epoch according to their ordering randdemo: 1=assign demographies at random, 0=assign demographies in order multp: 1=multiple paternity, 0=entire families from a single mating

demography Next come the demographies. The structure:

```
> names(rland$demography)
[1] "localdem" "epochs"
> names(rland$demography$localdem[[1]])
[1] "LocalS" "LocalR" "LocalM"
> names(rland$demography$epochs[[1]])
[1] "RndChooseProb" "StartGen"      "Extinct"       "Carry"
[5] "Localprob"     "S"             "R"             "M"
```

Here we find both the local S, R, and M matricies for each subpopulation (localdem) and also the S, R, and M matrices for the intersubpopulation movement of individuals (epochs). Notice the subscripts [[1]] after the localdem and epochs. Both localdem and epochs are lists of individual local demographies and epochs respectively. Since both localdem and epochs are lists the subscripts are necessary to view the contents of a single local demography or epoch.

local demographies
LocalS, LocalR, LocalM: the S, R, and M matricies for a particular local demography

epochs
RndChooseProb: the probability of randomly choosing this epoch if

rland$switchparam$randepoch==1
StartGen: the generation this epoch begins if rland$switchparam$randepoch==0
Extinct: a vector of values given the extinction probability for each subpopulation
Carry: a vector of values denoting the maximum individuals for each subpopulation
Localprob: a list of probabilities for choosing local demographies if
rland$switchparam$randedemo==1
S,R,M: The S, R, and M matricies for intersubpopulation movement

loci The next major section specifies genetic parameters. The structure:

```
> names(rland$loci[[1]])
[1] "type"    "ploidy"  "trans"   "rate"    "alleles"
> names(rland$loci[[1]]$alleles[[1]])
[1] "aindex" "birth"  "prop"   "state"
```

Both loci and alleles are lists with elements of the structure shown above.

locus type: The three allele types are 0=Infinite Allele mutation model/Integer state, 1=Strict stepwise mutation model/Integer state, 2=DNA base substitution/variable length sequence state. ploidy: This can be 1 for haploid and 2 for diploid. trans: The mode of inheritance. This can be 0 for bi-parental and 1 for maternal. rate: The per allele (as opposed to per site) mutation rate (range [0 1]) alleles: list of alleles

allele aindex: index number of the allele birth: the generation the allele first appeared prop: frequency of this allele state: an integer for loci type 0 or 1, a string representing sequence of DNA for loci type 2 (only the characters A,T,C,G are allowed)

individuals The last major section is individuals. The structure is merely a large matrix with one individual per row. An example row may look like:

```
0 0 0 1 2 4 5 2   <- stage sex gob loc1a1 loc1a2 loc2a1 loc2a2 loc3a1 (last locus haploid)
```

The first three columns are required and the last five shown here are a product of the loci. The first column contains the individuals subpopulation and lifecycle stage (subpopulation = floor(x/rland$intparam$stages), lifecycle stage = x mod rland$intparam$stages). The second column contains the sex of the individual (currently unimplemented, always 0). The third column contains the generation the individual was born or created. After the first three columns the indivduals genetic code begins. The loci are shown in order with 2 columns for diploid loci and 1 column for haploid loci. The value of these columns represent the allele index of the allele the individual carries.

2 Creating a landscape using R functions.

Landscape creation can best be accomplished using R functions provided by the rmetasim library. It is recommended to create a R source file with all the necessary function calls and simply source that file. This is preferred over calling the functions at the R command prompt because the creation functions

must be called in a specified order to create a correct and usable landscape object. *Remember to use the commmand 'library(rmetasim)' in R before beginning!* All of the creation functions have R help pages that can be accessed by the help() function in R, '?' is the short hand. For example accessing the new.landscape.empty() help page:

```
> help(new.landscape.empty)
```

Or alternately:

```
> ?new.landscape.empty
```

It is recommended that you view these pages during the creation process to understand all the arguments being used.

The creation process has seven steps that correspond to first initialization and then to the six main sections of the Rmetasim landscape. The steps are (1) skeleton creation, (2) integer parameters, (3)switch parameters, (4) floating point parameters, (5) demography, (6) loci, and (7) individuals. It is required to complete these steps in the order above.

(1) skeleton creation Before filling in values the landscape skeleton must be in place. At the top of your source file, blank out your new object and create a new empty landscape:

```
rland <- NULL
rland <- new.landscape.empty()
```

(2) integer parameters Now set the integer parameters for the landscape. This is simple model with 2 habitats and a 2 stage lifecycle:

```
rland <- new.intparam.land(rland, h=2, s=2)
```

There are several things to note. First, rland is a in my partially created landscape object, this important for all the steps except the first. Second, only 2 of the other parameters are used even though many are given in the help file. In this case the defaults are used for all the parameters not specified. For example, the number of total generations for this landscape is now set to the default of 1000.

(3) switch parameters Next set the switch (1 or 0) parameters for this landscape. This call will implement 1 epoch and 1 local demography so random choosing of demographies and epochs is not relevant to this case. However, multiple paternity has been turned off:

```
rland <- new.switchparam.land(rland,mp=0)
```

(4) float parameters Now set the floating point parameters for the landscape. In this case, no selfing rate is specified, therefore the default value of 0 is used:

```
rland <- new.floatparam.land(rland)
```

(5) demography Demography creation is actually broken down into two steps: local demography and epochs.

local demography The S, R, and M matrices for local demographies should be square matricies of order lifecycle stages by lifecycle stages we have previously specified in intparam. For this example that is 2. The matricies are created using the R matrix call and the c() function creates a vector of the values for the first argument:

```
S <- matrix(c(0.1, 0, 0.5, 0.3), nrow = 2)
R <- matrix(c(0, 1.1, 0, 0), nrow = 2)
M <- matrix(c(0, 0, 0, 1), nrow = 2)
```

Now these matrices are assigned to the landscape:

```
rland <- new.local.demo(rland,S,R,M)
```

If I wanted more than one local demography, I could make more calls to new.local.demo(), each time adding a new one.

epochs The larger S, R, and M matricies for epochs describe among population movement. These matricies are square with a dimension (lifecycle stages) * (number of habitats). For this example this is 2 * 2 = 4.

```
S <- matrix(c(rep(0,4),
              rep(0,4),
              rep(0,4),
              rep(0,4)), nrow = 4)
R <- matrix(c(0,0,0,1,
              0,0,0,0,
              0,1,0,0,
              0,0,0,0), byrow=T, nrow = 4)
M <- matrix(c(0,0,0,0,
              0,0,0,.1,
              0,0,0,0,
              0,.1,0,0), nrow = 4)
```

Now insert these into the landscape. Here the carrying capacities are set at 100 and 200 for the 2 habitats and an extinction probability of .01 per generation for the first habitat.

```
rland <- new.epoch(rland,S=S,R=R,M=M,extinct=c(.01,0),carry=c(100,200))
```

If I wanted more than one epoch, I could make more calls to new.epoch(), each time adding a new one.

(6) loci The loci contain genotypic data for each individual. Each new locus requires a call to new.locus() with different parameters. Here, one of each type of locus is specified. Allelesize is needed only for loci of type 2, and a couple of places the defaults are chosen.

```
rland <- new.locus(rland,type=0,ploidy=2,mutationrate=0.001,
                   transmission=0,numalleles=5)
rland <- new.locus(rland,type=1,ploidy=1,mutationrate=0.005,
                   numalleles=3,frequencies=c(.2,.2,.6))
rland <- new.locus(rland,type=2,ploidy=2,mutationrate=0.007,
                   transmission=0,numalleles=6,allelesize=75)
```

(7) individuals The final step is to have metasim generate a list of individuals. The only parameter needed to pass in at this point is the number of individuals in each stage in each subpopulation. This is passed as a vector with the ordering (pop1 stage1, pop1 stage2, ..., pop2 stage1, pop2stage2,....). Here 100 juveniles (stage 1) and 150 adults (stage 2) in the first subpopulation and 200 juveniles and 75 adults in the second subpopulation will be generated. *The frequencies of the alleles at each locus are generated based on the frequencies you assigned earlier with the new.locus call.*

```
rland <- new.individuals(rland,c(100,150,200,75))
```

Having completed the example source file, it now looks like this:

```
# comments in my source file must start with a pound sign!
rland <- NULL
rland <- new.landscape.empty()

rland <- new.intparam.land(rland, h=2, s=2)
rland <- new.switchparam.land(rland,mp=0)
rland <- new.floatparam.land(rland)


S <- matrix(c(0.1, 0, 0.5, 0.3), nrow = 2)
R <- matrix(c(0, 1.1, 0, 0), nrow = 2)
M <- matrix(c(0, 0, 0, 1), nrow = 2)
rland <- new.local.demo(rland,S,R,M)


S <- matrix(c(rep(0,4),
              rep(0,4),
              rep(0,4),
              rep(0,4)), nrow = 4)
R <- matrix(c(0,0,0,1,
              0,0,0,0,
              0,1,0,0,
              0,0,0,0), byrow=T, nrow = 4)
M <- matrix(c(0,0,0,0,
              0,0,0,.1,
              0,0,0,0,
              0,.1,0,0), nrow = 4)
```

```
rland <- new.epoch(rland,S=S,R=R,M=M,extinct=c(.01,0),carry=c(100,200))

rland <- new.locus(rland,type=0,ploidy=2,mutationrate=0.001,transmission=0,numalleles=5)
rland <- new.locus(rland,type=1,ploidy=1,mutationrate=0.005,numalleles=3,frequencies=c(.2,.2,.6))
rland <- new.locus(rland,type=2,ploidy=2,mutationrate=0.007,transmission=0,numalleles=6,allelesize=75)

rland <- new.individuals(rland,c(100,150,200,75))
```

Save this as 'mylandscape.R' in the working directory, and to produce the corresponding Rmetasim landscape start up R and type:

```
> source("mylandscape.R")
```

and now rland is a hot from the oven and fresh-but-not-too-fresh landscape.

Most likely the first time you source a file you will run in to a couple errors. It is a good idea to go back and fix the first error encountered, save the file, and attempt to source it again since the first error could be the cause of later errors. The error messages you recieve should be specific enough to fix the problems quickly. Should you happen to recieve a Segmentation Fault while sourcing your file (this should not happen, but remember to save your work often) an error has occoured in the new.individuals call. Make sure that line is correct in your source file, and be sure to email us with your source file and error conditions.

3 Saving and Reloading a landscape

Saving a landscape to a file is done with the write.landscape() function. The files are written in plain human readible text, that can be parsed by the metasim program from the command prompt. It is a good idea to save your landscape ever so often during simulation runs and if you ever make a few tweaks by hand.

```
> write.landscape(rland,"mylandscape.lnd")
```

Reloading a landscape is just as easy with the read.landscape() function.

```
> newland <- read.landscape("mylandscape.lnd")
```

4 Simulating

Running a landscape through a simulation is accomplished with the simulate.landscape() call. Running a landscape for 10 generations with a random seed:

```
> simulate.landscape(rland,10)
```

The code above returns the new landscape but does not change the landscape passed in. For a simulate and update:

```
> rland <- simulate.landscape(rland,10)
```

Choosing a specific seed for the random number generator produces repeatible results. The examples above produce a slightly differrent landscape each time where as the code below returns the same landscape everytime.

```
> rland <- simulate.landscape(rland,10,seed=5)
```

Landscapes cannot be simulated past their totalgens value (under intparam) despite what may be passed to the simulate.landscape() call. If an attempt is made to simulate them further they will run up to totalgens and return.

Large landscapes running for a large number of generations can take a significant amount of processing time to complete. In these cases we recommend saving your landscape every x generations so that when your processor overheats, OS crashes, lab gets struck by a meteorite, etc. you'll lose no more than x generations worth of simulation. Using a for loop to simulate 100 generations, saving every 10:

```
for(x in 1:10){
      rland <- simulate.landscape(rland,10,seed=5)
      filename <- paste(c("my-simulation-run-", x, ".lnd"),collapse="")
      write.landscape(rland, filename)
}
```

5 Analyzing results

Effective analyzation depends mostly on understanding the landscape object (section 1) and getting comfortable using the R environment. The manuals availible at the R project homepage: www.r-project.org are recommended. Especially 'An Introduction to R'. Nevertheless, Rmetasim comes with a few functions that aid the process of analysis.

populations(rland) Returns a vector of populations assignments for each individual in the landscape

plot.landscape.popsizedist(rland) Plots the frequency distribution of population sizes.

plot.landscape.stgsizedist(rland) Plots the frequency distribution of demographic stage sizes.

locus(lnum=1,Rland) Returns a individual matrix of allele indices for the locus number specified.

ploidy(rland) Returns a vector of integers representing the ploidys for each loci.

locusvec(rland) Returns a vector of locus ID numbers.

states(lnum=1,rland) Takes a locus and returns the different states and their indices

indxfreq(lnum=1,rland) Takes a locus and returns a matrix of frequencies for each allele.

¡contact info¿