A champion's guide to the TOSSM package

Dave Gregovich[1], Karen Martien[1] and Mark Bravington[2]

[1]Southwest Fisheries Science Center, 8604 La Jolla Shores Dr., La Jolla, CA 92037, USA
[2]CSIRO Mathematical and Information Sciences, Marine Laboratory, Castray Esplanade, Hobart 7001, TAS, Australia

Background

The Testing of Spatial Structure Models (TOSSM) package is a set of compiled functions in the 'R' language that were developed by recommendation of the International Whaling Commission (IWC). It relies heavily on the 'rmetasim' package for landscape simulation (Strand, 2002). As currently written, the TOSSM package is closely tied to the IWC's Revised Management Plan (RMP), which attempts to manage and conserve whales that are subject to harvest or other human-induced mortality (IWC, 1994). In the immediate future, it is envisioned that the TOSSM package will be used to assist in the management of whales as directed by the RMP. However, the TOSSM framework is also applicable to the management of virtually any animal population that exhibits spatial structure. The larger scope of TOSSM's potential utility is therefore reflected in some of the more general language included in this document.

The purpose of this document is to serve as a reference document to anyone using the TOSSM package to conduct simulation-based performance testing. We begin by introducing the TOSSM project and defining terms and concepts integral to TOSSM. We then describe the package in detail, with particular attention to the biological interpretation of some of the arguments required by the package. Though our focus here is on the TOSSM package itself, we provide Appendices describing the generation of the TOSSM datasets, which are a required input to the package, and how to develop an interface for using the package to test a particular analytical method.

General Introduction and definition of terms

Because of the difficulty in managing spatially-structured populations, a way of realistically simulating a spatially-structured population, and then testing the performance of analytical methods designed to detect population structure, is desired. In the TOSSM package, an analytical method's performance is tested in terms of 1) accurately inferring the spatial structure of simulated populations statistically and 2) setting appropriate management boundaries accordingly. This is the object of the TOSSM package (Figure 1).

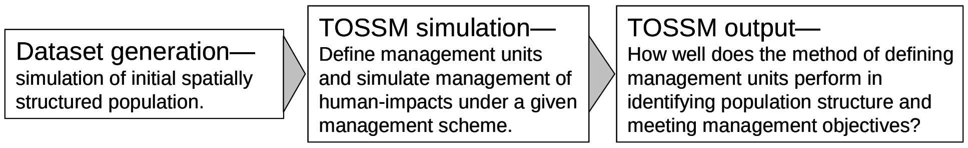

Dataset generation—
simulation of initial spatially structured population.
TOSSM simulation—
Define management units and simulate management of human-impacts under a given management scheme.
TOSSM output—
How well does the method of defining management units perform in identifying population structure and meeting management objectives?

Figure 1. Overview of the TOSSM package.

Archetypes and Breeding populations (BPs)

The TOSSM datasets fall into five broad categories of population structure, which are referred to as 'Archetypes' (Figure 2). The number of simulated breeding populations (BPs) is determined by the number of breeding populations that exist in the initial (ancient), simulated dataset. The datasets exist as 'rmetasim' landscape objects. To generate each initial dataset, the number of BPs, carrying capacity for each BP, and, where applicable, a dispersal rate between BPs, was specified. Further details on generation of the TOSSM datasets can be found in Appendix 1.

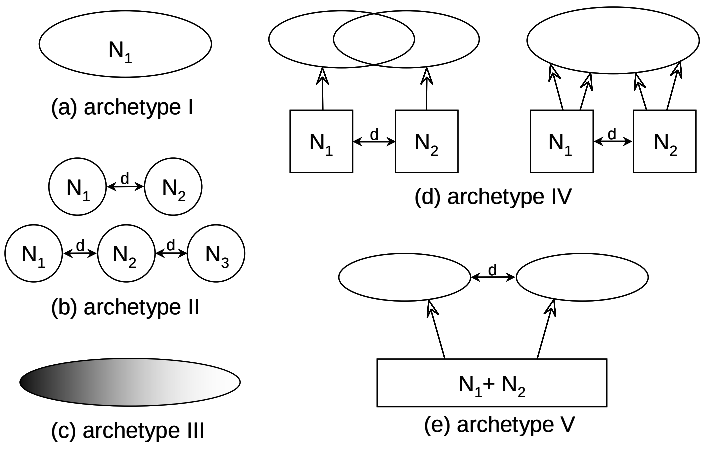


Figure 2. The five archetypes represented in the TOSSM datasets. (a) Archetype I—A single, mixed population that serves as a control. (b) Archetype II—Stepping-stone dispersal pattern between two or three populations, with only adjacent populations mixing. (c) Archetype III—Diffusion-type, where isolation across the population continuously increases with distance. (d) Archetype IV—Two discrete breeding grounds with feeding grounds that overlap partially or completely. (e) Archetype V—A single breeding population with two separate feeding grounds.

Fully-integrated mixed areas (FIMAs)

The TOSSM package assumes that harvest can take place at any time or place in an organism's life history. If harvest takes place in breeding areas during the breeding season, then it might reasonably be assumed that all animals killed in a given location come from a single breeding population. If harvest does not take place on the breeding grounds during breeding, then it is more likely that the animals killed in any given location constitute a mix of individuals from different breeding populations. If harvest takes place in a feeding area known to be shared by multiple breeding populations, then there will almost surely be a mix of individuals from different breeding populations targeted. These harvest scenarios translate similarly to scientific sampling. A sample taken in a given area of space at a point in time will be comprised of a mix of individuals from different breeding populations dictated by where and when the sample was taken.

In order to simulate the mix of individuals from different breeding populations found in a harvest area or sample, it is necessary to define areas in which the proportional composition of separate breeding populations is constant across space. These areas are referred to in the TOSSM package as Fully-integrated Mixed Areas (FIMAs). FIMAs can contain individuals from 1, 2 or several breeding populations. In the TOSSM package, all harvesting, abundance estimation, and collection of genetic samples is assumed to occur in FIMAs. From a genetic standpoint, FIMAs correspond to sampling sites. FIMAs represent the smallest spatial unit at which data are aggregated. In other words, you can combine genetic data and abundance estimates across FIMAs, but this information does not exist at any spatial scale smaller than at the FIMA level.

Another helpful way of stating the function of FIMAs is with the following three statements:

1) If you assume harvest occurs on feeding grounds, the FIMAs are what determine the relationship between BPs and feeding grounds (FGs).
2) If you assume harvest occurs on migration, the FIMAs are what dictate the migratory routes of the different BPs.
3) If you assume harvest occurs on the breeding grounds, the FIMAs will reflect no mixing of BPs.

The mechanism by which BPs are mapped to FIMAs is through a 'mixing' matrix. For example, in a scenario in which there are 2 BPs and 4 FIMAs, the mixing matrix will be a 2 X 4 matrix that might look like the matrix shown in Table 1. This matrix implies that at the time of sampling/harvest, 50% of BP is in FIMA1, 30% is in FIMA 2, 20% is in FIMA3 and 0% is in FIMA4. The spatial distribution of BP2 is the reverse. A visual representation of the matrix in Table 1 is shown in Figure 3. Biologically, this example can be interpreted as two breeding populations that have partially overlapping feeding grounds (Archetype 4; Figure 2d).

Table 1. An example mixing matrix for a simulation in which there are 2 breeding populations and 4 FIMAs.

	FIMA1	FIMA2	FIMA3	FIMA4
BP1	0.5	0.3	0.2	0.0
BP2	0.0	0.2	0.3	0.5

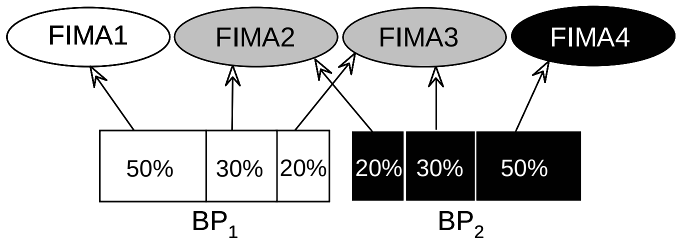


Figure 3. Schematic representation of the mixing matrix given in Table 1. In this case, the geographic location of the BPs is reflected in the composition of FIMAs, with FIMAs 1 and 4 consisting only of individuals from the BPs that are closest in space, and FIMAs 2 and 3 intermediate in location and in composition of the two BPs.

The above example is representative of Archetype 4 in the original datasets. In Archetype 4, BPs mix in the FIMAs. This is not always the case. Archetype 2 consists only of scenarios in which the BPs move to completely separate FIMAs, reflected in the following matrix:

	FIMA1	FIMA2	FIMA3	FIMA4
BP1	0.5	0.5	0.0	0.0
BP2	0.0	0.0	0.5	0.5

Alternatively, in a more extreme case of mixing (representing Archetype 4), BPs could spread across all FIMAs:

	FIMA1	FIMA2	FIMA3	FIMA4
BP1	0.4	0.3	0.2	0.1
BP2	0.1	0.2	0.3	0.4

Note that in all cases illustrated above, the proportions of each BP that move to the various FIMAs—corresponding to each row of the matrix—sum to one. It is possible to input values for the mixing matrix for these proportions of the breeding populations that do not sum to one. However, in that case, the values will simply be normalized as part of the simulation to sum to one.

The number of FIMAs used in the above examples has been limited to four for the sake of simplicity. However, the number of FIMAs is determined by the user, and can be much larger if desired. FIMAs are the smallest unit at which any management decision or scientific sampling occurs. While FIMAs cannot be split, they can be combined when defining management units. Thus, for many analytical methods it will be appropriate to define a large number of FIMAs, as that will provide maximum flexibility when placing management unit boundaries. However, the drawback of defining many FIMAs is that it reduces the number of genetic samples per FIMA, which will result in reduced performance for some methods.

Boundary-setting algorithm's (BSAs)

A Boundary-setting algorithm (BSA) is a function that uses genetic data to define management units. Though there are many analytical methods that accept and analyze genetic data and output information on population structure, most of them don't go so far as to explicitly define management units. Thus, while a BSA will generally have at its core an analytical method for detecting and describing population structure, it must also include a mechanism for using the results of that method to define management units. Specifically, a BSA must make the decision of whether to lump or split FIMAs together in a single or multiple management units. This could be as simple as deciding whether to manage two FIMAs separately or as one management unit. Alternatively, if there are many FIMAs, there could be a number of different decisions the algorithm must make about splitting or grouping the various FIMAs into management units.

A BSA must interface with the TOSSM package by accepting simulated genetic and abundance information from the package and returning a recommended way of combining (or not) the FIMAs into management units. A BSA does not have to use all information about a simulated population that TOSSM makes available; different BSAs may rely on different components of the simulated data provided by a TOSSM simulation. Further details on creating a BSA is given in Appendix 2.

Catch-limit algorithm (CLA)

The algorithm that the IWC uses for calculating the number of whales that can be killed in a management unit is called the catch-limit algorithm, or CLA. It calculates quotas based on the estimated abundance of a management unit and information on historic catches—see Cooke (1994) for further details. The CLA is currently the only algorithm available within the TOSSM package for setting quotas and managing harvest. However, the modular architecture of the package makes it relatively easy for other management schemes to be added. In the immediate future, we plan to expand the TOSSM package so that users have the option of calculating quotas using either the CLA or the Potential Biological Removal (PBR) scheme used in the U.S. Marine Mammal Protection Act (Taylor et al., 2000)

The details of `run.tossm`

(Note: characters printed in `Courier New` font denote 'R' commands and functions)

To install the TOSSM package, simply extract the zip file to your 'R' library, and load TOSSM—`library(tossm).` Make sure that the program MANAGE-D.exe is in the tossm folder. The main function in the TOSSM package is `run.tossm`. Here is the full compliment of arguments to `run.tossm`, including their defaults:

```
function (rland = NULL, choose.f = NULL, choose.f.args = NULL,
    schedule = NULL, n.gs.per.f = NULL, pre.RMP.removals = 1,
    BSA = single.MU.BSA, BSA.args = list(), stop.years = schedule$stop.years,
    gs.years = schedule$gs.years, abund.est.years = schedule$abund.est.years,
    pre.RMP.years = schedule$pre.RMP.years, CLA.years = schedule$CLA.years,
    post.RMP.years = schedule$post.RMP.years, BSA.years = schedule$BSA.years,
    genetic.sampler = def.genetic.sampler, set.coords = def.set.coords,
    abund.for.10pc.CV = 1000, use.rmetasim = FALSE, TAC.args = list(mult.CLA = 5),
    CLA.prog = NULL, CLA.dir = NULL, seed = -1)
```

Here is a rundown of the arguments that `run.tossm` takes:

`rland` – this is an 'rmetasim' landscape that represents a simulated population. A simulated population could be generated by a number of different methods. For the present time, we will assume the user is using simulated initial populations generated specifically for the IWC (aka 'the TOSSM datasets'; Martien 2006). These datasets have been generated via an initial simulation referred to in this document as the 'ancient' phase to reflect that they simulate a long established population in genetic and demographic equilibrium. Further details on the generation of these datasets are contained in Appendix I. These initial datasets exist as '`rmetasim`' landscape objects. To use one, simply load one of the TOSSM datasets into your workspace (e.g., '`load(Arch1_sc1_1.rda)`'). This will create in your 'R' workspace an object called '`rland.end`', which can be passed to `run.tossm`.

`choose.f`—this argument accepts a function describing how individuals choose their FIMAs. It must specify both the mechanism by which a newborn whale chooses a FIMA and the degree of FIMA fidelity individuals display over the course of their lives. Two such functions are provided as part of the TOSSM package. In '`humpbackism`,' individuals learn their migratory route and, therefore, their FIMA fidelity, from their mother. Individuals exhibit high FIMA fidelity their entire lives, with individuals change FIMAs only rarely. In the `'randomism'` function, individuals choose their FIMAs randomly every year. The user is not constrained to these two choices, but functions reflecting different models of movement must be written by the user.

`choose.f.args`—is the argument of `run.tossm` that accepts information on the way in which the breeding populations are distributed across FIMAs. It does this in the form of a matrix with columns corresponding to FIMAs and rows corresponding to breeding populations. If the 'mixing matrix' is input into `choose.f.args` in the following fashion:

```
BP1<-c(0.5,0.3,0.2,0)
BP2<-c(0,0.2,0.3,0.5)
mixmat<-rbind(BP1,BP2)
colnames(mixmat)<-c("FIMA1","FIMA2","FIMA3","FIMA4")
choose.f.args <- list(mixmat)
```

then the resulting proportions in which BP's will be split to each FIMA will be:

```
      FIMA1 FIMA2 FIMA3 FIMA4
BP1   0.5   0.3   0.2   0.0
BP2   0.0   0.2   0.3   0.5
```

This indicates that of the population in BP 1, 50% move to FIMA 1, and 30% to FIMA 2, and 20% to FIMA 3, with a similar, but converse, movement from BP2. Note that the number of FIMAs in a simulation is not explicitly defined. Rather, it is determined by the number of columns in the mixing matrix passed to `choose.f.args`. Also note that as previously stated, the values in the mixing matrix that represent the proportions of each BP that move to each FIMA—corresponding with the rows of the mixing matrix—do not have to sum to one. However, part of the `run.tossm` code normalizes these values to one, so these values are essentially input as weights.

`schedule`—A main prerequisite to running a TOSSM simulation is setting up a schedule of events. This schedule defines a simulation timeline. The simulation schedule includes some terminology specific to International Whaling Commission (IWC) issues. The schedule must include the following components:

`stop.years`–The total number of years in the simulation
`gs.years`—The years in which genetic samples are taken
`abund.est.years`—The years in which abundance estimates are obtained
`pre.RMP.years`—A period that predates RMP management (may include historic harvest levels)
`CLA.years`—Years in which quotas are calculated via the catch limit algorithm
`post.RMP.years` —A recovery period after RMP management (optional)
`BSA.years`—Years in which the boundary-setting algorithm is called to detect spatial structure and define management units accordingly.

These different events are contingent on each other in the following way:

- Genetic samples (`gs.years)` must be obtained before boundaries are set using the BSA (`BSA.years)` (collecting them in the same simulation year is fine).
- Boundaries (`BSA.years)` must be set and abundance estimates (`abund.est.years`) must be obtained before an initial catch limit is set (`CLA.years)`. In other words, the first element in `BSA.years` and `abund.est.years` must be less than or equal to the first element in `CLA.years`. Subsequent calls to the CLA, however, can be made without additional calls to the BSA or abundance estimates.
- All `CLA.years` take place during the RMP phase.

Though it is possible to set up a simulation schedule manually by creating vectors for each of the schedule components and passing them as arguments to `run.tossm`, the TOSSM package includes a function called `def.make.schedule`, which is a convenient way of setting up a schedule of all simulation events. It is recommended to use `def.make.schedule` initially to run simulations; the user can get fancy in customizing this schedule once they are more comfortable with the package.

`def.make.schedule` requires the following arguments:

- `n.pre.RMP` – Number of historic whaling years (before RMP management)
- `n.RMP` – Number of years during which RMP management applies
- `n.post.RMP` – Number of 'recovery' years, during which no harvest occurs
- `abund.gap` – Interval (in years) on which abundance estimates are obtained and CLA is called

`def.make.schedule` generates a schedule in which genetic samples are taken (`gs.years`) and boundaries are set (`BSA.years`) in the last pre-RMP year only. Abundance estimates are obtained (`abund.est.years`) and a quota is calculated (`CLA.years`) in the first RMP year and every `abund.gap` years there after until the end of the RMP phase. Thus, for the following inputs to `def.make.schedule`

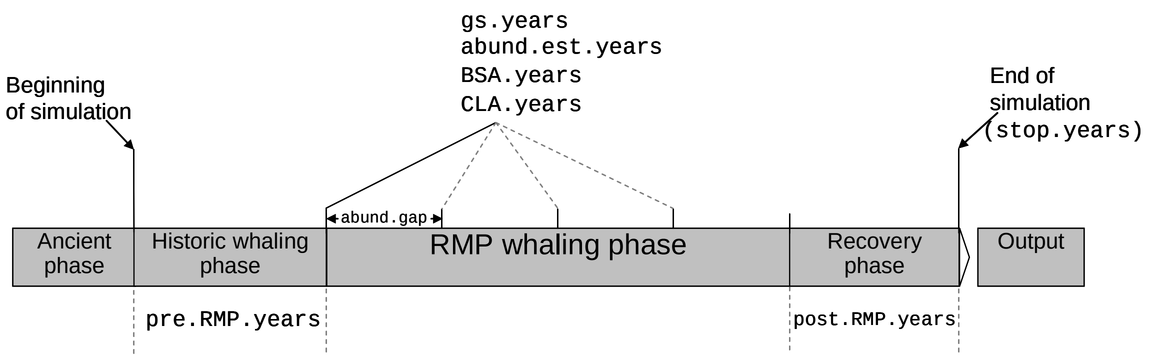
```
my.schedule <- def.make.schedule(n.pre.RMP=10,n.RMP=20,n.post.RMP=10,
abund.gap=5)
```

the function would return a list with the following components:

```
>my.schedule
$stop.years
[1] 40
$gs.years
[1] 10
$abund.est.years
[1] 11 16 21 26
$pre.RMP.years
 [1]  1  2  3  4  5  6  7  8  9 10
$CLA.years
[1] 11 16 21 26
$post.RMP.years
 [1] 31 32 33 34 35 36 37 38 39 40
$BSA.years
[1] 10
```

These events can be envisioned schematically as follows:

Components of `def.make.schedule`



`n.gs.per.f`—the number of genetic samples to collect in each FIMA in each of the `gs.years`. This argument should either be a vector equal in length to the number of FIMAs or a single number, in which the same number of samples is taken from each FIMA

`pre.RMP.removals`—this argument is used to specify any historic harvest before the period in which modern management takes place under the RMP. It consists of either a vector of length equal to the number of FIMAs or a vector of length one, in which case the single harvest value will be applied to all BPs.

`BSA`—The BSA input by the user needs to mesh with `run.tossm`. Leaving the BSA argument of `run.tossm` as the default `(BSA = single.MU.BSA)`, does not set any management boundaries, but is handy just to make sure that run.tossm is running fine before attempting to use a different BSA. See Appendix B for the details of setting up a BSA.

`genetic.sampler`—The genetic sampler dictates the sampling design used for obtaining genetic samples. The default (`def.genetic.sampler`) should be fine for most applications, but the user can supply an alternate sampler if desired.

`set.coords`—defines the locations where genetic samples are obtained. `def.set.coords` assumes that samples are evenly distributed across space, but the user could create alternates to represent various uneven sampling schemes.

`abund.for.10pc.CV` – allows the user to specify the abundance for which the CV of the abundance estimate generated by `run.tossm` will be 10%. This parameter can be used to increase or decrease the variance in the abundance estimates.

`use.rmetasim`—rmetasim is not the only available model for projecting population dynamics, and so it is possible to set this argument to 'FALSE' and use a different modeling method.

`TAC.args`—a list of arguments to be passed to the quota calculating algorithm. The CLA (the only quota calculating algorithm currently implemented in the package) accepts a single optional argument, `mult.CLA`. `mult.CLA` is a multiplier used to scale the quota calculated by the CLA up or down.

`CLA.prog`—The path to the Catch-limit algorithm program (MANAGE-D.exe) to be used. The default should be correct unless MANAGE-D.exe has been moved subsequent to installation of the TOSSM package.

`CLA.dir`—The location of some temporary files created by the CLA program.

`Seed`—integer. If seed>0, it ensures a reproducible sequence of datasets.

The output of `run.tossm`

The output of `run.tossm` includes the following components:

`abund.b`—the true abundance in each breeding ground for each simulation year.
`abund.f`—the true abundance in each FIMA for each simulation year.
`catches`—the true catch per FIMA for each simulation year.
`effort`—a measure of the effort (e.g., distance traveled) required on the part of the simulated whalers in order to fill their quota each year. The value currently output (quota/abundance) is not useful. Suggestions on how best to measure effort are welcome.
`FIMA.mu.map`—Displays the result of the BSA—how FIMAs were mapped to management units—for each year the FIMA was called.
`est.abund.f`— the estimated abundance in each FIMA for each simulation year.
`var.abund.f`—the variance of this estimate.
`gs`—raw allele counts for all loci and individuals sampled in each FIMA during each of the `gs.years.`
`agg.gs`—the results of the genetic sampling aggregated for all years.
`agg.freq`—allelic frequencies across FIMAs and loci.

The first four outputs in the above list should be sufficient for assessing the performance of a particular simulation with respect to the management goals specified by the IWC. The remaining outputs are included to allow for more detailed assessment of the performance of the BSA and as simulation diagnostics.

Acknowledgements
Funding was provided by NMFS and the IWC Secretariat. Thanks to Barbara Taylor for reviewing this document, and to Charles Edwards for help his comments on the TOSSM package and its documentation.

References

Cooke, J.G. 1994. The management of whaling. Aquatic Mammalogy 20:129-135.
Excoffier, L., G. Laval, and S. Schneider. 2005. Arlequin ver. 3.0: An integrated software package for population genetics data analysis. Evolutionary Bioinformatics Online. **1**:47-50.
Glaubitz, J.C. 2004. CONVERT: A user-friendly program to reformat diploid genotypic data for commonly used population genetic software packages. Molecular Ecology Notes. **4**:309-310.
IWC. 1994. The Revised Management Procedure (RMP) for Baleen Whales. Rep. int. Whal. Commn., **44,** 145-167.
IWC. 2004. Report of the Workshop to design simulation-based performance tests for evaluation methods used to infer population structure from genetic data. Journal of Cetacean Research and Management. **6**(Suppl.):469-485.
Laval, G. & Excoffier L. 2004. SIMCOAL 2.0: a program to simulate genomic diversity over large recombining regions in a subdivided population with a complex history. Bioinformatics, **20**:2485-2487.
Martien, K. 2006. Progress on TOSSM dataset generation. Paper SC/58/SD2 submitted to the Annual Meeting of the Scientific Committee of the International Whaling Commission, St. Kitts, June 2006.
Strand A.E. 2002. METASIM 1.0: an individual-based environment for simulating population genetics of complex population dynamics. Molecular Ecology **2**:373-376.
Taylor, B.L., P.R. Wade, D.P. DeMaster, and J. Barlow. 2000. Incorporating uncertainty into management models for marine mammals. Conservation Biology 14:1243-1252.

Appendix A. Dataset Generation

An important step in the TOSSM process is the generation of the initial datasets. As per the recommendations of the TOSSM working group (IWC 2004), these datasets represent five different archetypes (Figure 2). Within each archetype, there are in some cases many population parameterizations that are termed ‘scenarios’. The following excerpt from Martien (2006) summarizes the steps involved in generating these datasets.

Step 1: Estimate N_e. The carrying capacity specified for each TOSSM scenario and used to parameterize the Rmetasim simulations constrains the census population size, not effective population size, N_e. The abundance estimate required to parameterize the coalescent models, however, is N_e. It is therefore necessary to estimate Ne for each scenario by running a simulation that contains a single microsatellite locus and a 500 bp mitochondrial DNA sequence, each with no mutation and initialized with 1,000 alleles. The simulation is run for 2,000 years (100 generations). By calculating heterozygosity at the beginning and end of the simulation, N_e can be estimated using the equation

$$H_t = H_o\left(1 - \frac{1}{2N_e}\right)^t$$

where H_t is heterozygosity at time t and H_o is heterozygosity at time zero. We replicate the simulation 10 times and average across replicates to obtain a mean Ne for both nuclear and mitochondrial markers. It is necessary to estimate N_e separately for the two marker types because the haploid and uni-parentally inherited nature of mitochondrial DNA results in a markedly smaller effective population size for the mitochondrial genome.

Step 2: Generate Coalescent Datasets. The haplotype and allele frequencies used to initialize Rmetasim are generated using the coalescent model SimCoal 2.1.2 (Laval and Excoffier 2004). A 500 bp mtDNA sequence and 18 unlinked microsatellite loci are simulated. 1,000 SimCoal datasets are generated for each scenario.

Step 3: Initialize and Run Rmetasim Simulations. Each SimCoal dataset is used to initialize one Rmetasim landscape using the function ‘coalinput.landscape().’ This function is currently available as an independent script (see *Initializing Rmetasim* above), but will be incorporated into a future release of Rmetasim. Each landscape is simulated in Rmetasim for 1,000 years (50 generations). The final state of the landscape is saved as an R object (*.rda file). In addition, the microsatellite data are saved to a text file formatted for use with the program Convert (Glaubitz 2004) and the mtDNA data are saved in a text file formatted for analysis in the program Arlequin (Excoffier et al. 2005). These formats were chosen because they are used by many different analytical methods and because both Convert and Arlequin include utilities for converting data into multiple other formats (in fact, that is the primary function of Convert). Separate formats are required for mtDNA and microsatellite data because Convert does not support mtDNA data, while Arlequin is not able to convert data into as many formats as Convert.

Appendix B. Writing a BSA

The BSA needs in essence to do two things: 1) analyze the genetic and/or abundance data and 2) set management unit boundaries. The analysis is quite likely to be done exclusively by an ‘outside’ program not written in ‘R’. However, it may be helpful to do some intermediate data processing in R to make the data supplied by the TOSSM package friendly to the analytical method being used. Similarly, it may take some processing to convert the output of an analytical method to a format accepted by `run.tossm`. Regardless, there needs to be at least some ‘R’ code written, even if just a ‘wrapper’ that allows the analytic method and the TOSSM package to interface.

There are a number of examples of mixing and migration models that have been proposed for use as analytic methods. It is foreseen that some of these methods can stand alone or be combined to supply the information desired to analyze TOSSM simulated data and set management boundaries. Whatever the method used, the ‘R’ code written for the BSA must begin as in the following example:

```
My.BSA<- function(genetic samples, abundance estimates, variances, catches by
year/FIMA, optional param1, optional param2){
```

The first four of these arguments are passed automatically to the BSA function by `run.tossm`, so the function must accept these four arguments whether it uses them or not. The last two arguments can optionally be used to accept any additional information that might be required to run the BSA function. Each of the six arguments are specified in the argument `BSA.args`, and can be named as the user wishes.

`run.tossm` can also provide the BSA with genetic data aggregated across years, and can also provide information on allele frequencies across FIMAs. These are not automatically available, but can be made available to the BSA function by including the following lines in the BSA function:

- `agg.gs.tseries()`—creates the objects `agg.gs,n.loci,n.areas,`and `n.alleles`
- `count.alleles.by.f()`—creates the objects `maxal,` and `agg.gfreq`

The only object that a BSA must return to `run.tossm` is a list that groups or doesn’t group the FIMAs into management units (as described in the general introduction above). For instance, `return(list(c(1,2,3),4))`splits four FIMAs into two management units, one comprised of three FIMAs and one of only one FIMA, while `return(list(c(1,2,3,4)),` instructs `run.tossm` to place all four FIMAs in a single management unit.