Episodic Fragmentation Simulation

Peta Hill & Bernd Gruber

2023-09-18

Forward

The code below allows you to rerun the simulations as described in the Manuscript of Hill et al. (2023): Episodic fragmentation and gene flow reveal a trade-off between heterozygosity and allelic richness. As the actual simulation and analysis takes some time to run (about 2 days using 40 cores) we also provide means to run the simulation with fewer parameter combinations.

Please note the simulation is only tested for the stated versions of the R packages and programs and if the reader tries to run it, it may break due to incompatibilities of packages

Required packages and programs

To be able to run the simulation the following programs are needed (links are given which details how to install on your system):

- slim (V 3.7.1) (https://github.com/MesserLab/SLiM)
- plink (V 1.9)(https://www.cog-genomics.org/plink/)

For the R installation you need to have a recent version of R (>4.0) and the following packages. Please note the version might be important as some of the packages are known to not work if incompatible versions are installed.

- seqinr version '4.2-30'
- slimr version '0.2.1'
- dartR version '2.9.7'
- ggplot2 version '3.4.2'
- vcfR version '1.14.0'
- purrr version '1.0.1'
- furrr version '0.3.1'
- parallel version '4.3.1'
- doParallel version 1.0.17

Setting the environment

Loading the packages

```r
library(seqinr)
library(slimr)
library(dartR)
library(ggplot2)
library(vcfR)
library(purrr)
library(furrr)
library(parallel)
library(doParallel)
```

Once loaded package versions can be checked via: `packageVersion("packagename")`

Setting paths

Next it is important to define the paths where you have installed the external programs (plink and slim) and the folder where all output files are stored. Please be aware in case you run the full simulation output, files need a considerable amount of disc space (~2TB)

```r
plinkdir <- "d:/programms/plink"
slimdir <- "d:/programms/slim37"
Sys.setenv(SLIM_HOME=slimdir)

#folder where all simulated files are stored
outdir <- "d:/temp/bb"
```

Setting numbers of cores

You can check the number of processors (cores) your system has to define how many cores you want to use for the simulation. As a default we use the number of cores on your system -1, to allow for smooth input/outputs, but obviously this could be changed to a lower number. It does not make sense to increase the number to more than the number of cores that are available.

```r
cat(paste0("You have ", detectCores()," cores available on your
  system.\nThe maximum recommended number of cores you may use
  is ",detectCores()-1,"."))
```

```
## You have 28 cores available on your
##   system.
## The maximum recommended number of cores you may use
##   is 27.
```

```r
#number of cores to be used
n.cores <- min(detectCores()-1,10)
```

Setting the parameters for the simulation

The parameter space that is simulated is configured by a data.frame. This allows to scale the simulation to a short (toy) run and also for a run of all parameter combinations as in the manuscript.

The parameter are:

- npops: number of subpopulations in the metapopulation. Specified as a vector, e.g. npops <- c(2,4,8) runs simulation with 2,4 and 8 subpopulation. Numbers have to be powers of 2.
- n.loc: number of loci to be simulated.
- mig: migration rates in percentage, e.g. c(25,50,100) runs simulation with 25, 50 and 100% migration rates
- mev: [migration every x generation], specifies the interval of migration events, e.g. c(2,5,10) creates migration events every second, fifth and 10th generation.
- rep: the number of repeats for a parameter combination

Please note the simulation runs all possible parameter combinations above, so the number of simulations increases quite fast if the number of levels of each parameter is high. The default setting below is to run a toy example of only 72 combinations, we also provide the settings for the simulations as run in the manuscript, but be aware that will take quite some time to run. For a more detailed explanation please refer to the manuscript.

```r
#number of subpopulations
pops <- c(2,4,8)

# number of loci (SNPs)
n.loc <- 50
# toy example
df <- expand.grid(rep=1:10,  pNum = pops, mig = c(25,100), mev = c(2,10))

#full run as in the manuscript
#n.loc <- 5000
#df <- expand.grid(rep=1:50,  pNum = c(1,2,4,8,16,32,64),
#mig = c(1,5,10,25,50,100), mev = c(2,5,10,20))

#number of paramter combinations
nrow(df)
```

```
## [1] 120
```

Creating initial genotype frequencies for each subpopulation

We create the metapopulations as specified with initial genotype frequencies of 0.5 for the number of loci specified and using the script below. The initial populations will be saved as vcf files in the folder as specified in outdir above. The naming of the files is:

- gl_n.loc_subpop_i.vcf

```r
ref<-sample(c("A","G","C","T"),n.loc,replace=T, prob=c(0.25,0.25,0.25,0.25))
R<-paste(ref, collapse = "")
#saveRDS(R,file.path(outdir,"TOYreference_allele.rds"))
alt<-comp(ref, forceToLower = FALSE)
locAl<-paste0(ref,"/",alt)
p<-list()
```

```r
gl.set.verbosity(0)
for (ii in 1:length(pops))
{
```

```
n.pop <- pops[ii]
n.ind<-256/n.pop
for (i in 1:n.pop){
  startmatrix0 <- matrix(c(0,1,1,2), nrow=n.ind,ncol=n.loc)
  startmatrix <- apply(startmatrix0,2, function(x) x[order(runif(length(x)))])
  p[[i]] <- new("genlight", startmatrix, ploidy=2)
  pop(p[[i]]) <-  rep(paste0("P",i),n.ind)
  indNames(p[[i]])<- paste0("p",i,"-",1:n.ind)
  locNames(p[[i]]) <- paste0("L",i,"-",1:n.loc)
  alleles(p[[i]])<-locAl
  p[[i]]$chromosome <- as.factor("1")
  position(p[[i]]) <- as.factor(1:nLoc(p[[i]]))
  p[[i]]@other$loc.metrics$pos <- position(p[[i]])
  pvcf<-gl2vcf(p[[i]], plink_path = plinkdir, outpath = outdir,
  outfile = paste0("gl_50_",n.pop,"_",i), snp_pos="pos", verbose=0)
}
}
```

Slim templates

Each parameter file will be used for a simulation executed within slim using the file for the corresponding parameter combination. To prepare all the simulations in slim we use slimr and therein the template function. This function allows to setup templates for the slim runs leaving place holders for the parameter values as set in the data.frame (df).

```
slim_script(
  slim_block(initialize(),
             { initializeSLiMModelType("nonWF");
               #mutation rate
               defineConstant("alpha1", 1e-9);
               # total number of individuals in the metapopulation
               defineConstant("metasz",256);
               #number of subpopulations
               defineConstant("subp_num",slimr_template("pNum"));
               #number of replicate simulations
               defineConstant("rep",slimr_template("rep"));
                #migration rate as percent
               defineConstant("mig", slimr_template("mig"));
               #migration rate as number of individuals
               defineConstant("migRate",asInteger(mig/100*metasz));
               #number of individuals in each subpopulation
               defineConstant("subp_sz", asInteger(metasz/subp_num));
               #output directory
               defineConstant("outpath", slimr_inline(outdir));
               ;# How often to mix
               defineConstant("mev",slimr_template("mev"))
               initializeSLiMOptions(nucleotideBased=T);
               defineConstant("L",slimr_inline(n.loc))
               #reference sequence
               initializeMutationTypeNuc("m1", 0.5, "f", 0.0);
               initializeAncestralNucleotides(slimr_inline(R));
               m1.convertToSubstitution = T;
             m1.mutationStackPolicy = "l";
             mutmat = matrix(c(0.0, alpha1, 0.0,0.0,alpha1,0.0,0.0,0.0,0.0,0.0,0.0,
                               alpha1,0.0,0.0,alpha1,0.0), nrow=4, ncol=4)
             initializeGenomicElementType("g1", m1, 1.0,mutmat);
             initializeGenomicElement(g1, 0, L-1);
             initializeRecombinationRate(0.5);

           }),
  slim_block(reproduction(), {# everybody mates, selfing allowed
    for (j in seqLen(subp_num+1)){
      inds = sim.subpopulations[j];
      if (inds.individualCount > 0)
      {
        parents = inds.sampleIndividuals(inds.individualCount, replace = T);
        for (i in seqLen(inds.individualCount))
        {
          mum = parents[i];
          dad = inds.sampleIndividuals(1);
          child = inds.addCrossed(mum, dad);
        }
      }
      else {j=j+1;
      }
      self.active = 0;}
  }),
  slim_block(1, {
    #set up subpopulations, each of size 'subpop_sz', reading in genotypes
    for (i in seqLen(subp_num)){
      sim.addSubpop(i,subp_sz);#from .vcf files we generated
      subpops = sim.subpopulations;
      f = paste0(outpath+"/gl_50_" + subp_num + "_" + (i+1) + ".vcf");

      subpops[i]%.%genomes.readFromVCF(f, m1);
      subpops[i]%.%individuals.tag = ((subp_sz*i)+1):((i + 1) * subp_sz);
    }
    sim.addSubpop(100,0);# this is for the migrant pool
  }),
  slim_block(2,501, early(), {
    # parents die in line with WF assumptions
    inds = sim.subpopulations.individuals;
    inds[inds.age > 0]%.%fitnessScaling = 0.0;
  }),
  slim_block(2,499, first(), {
    #if the generation is one in which gene flow is to occur;
    #take migrants into the migrant pool
    if(sim.generation %% mev == 0){
      migrants = sample(sim.subpopulations.individuals,migRate);
      p100.takeMigrants(migrants);
    }
  }),
  slim_block(2,500, first(), {
    #if the generation is the one after migration;
    #randomly return migrants to the subpopulations
    if(sim.generation %% mev == 1){
      for (id in seqLen(subp_num)){
        subpops = sim.subpopulations[sim.subpopulations.id == id];
        migrants = sample(p100.individuals, asInteger(subp_sz-subpops.individualCount));
        dest = subpops;
        dest.takeMigrants(migrants);
      }
    }
  }),
  slim_block(470,500, late(), {
    subpops = sim.subpopulations;
    if(500-sim.generation <= (mev+1)){
      subpops.individuals.genomes.outputVCF(paste0(outpath,"/FNAME_",
slimr_template("pNum"),"_mig_",slimr_template("mig"),"_rep_",
slimr_template("rep"),"_mev",slimr_template("mev"),"_gen",
sim.generation,".vcf"),simplifyNucleotides=T, outputNonnucleotides=F);}
  }),
  slim_block(500,late(), {
    sim.simulationFinished();
  })
)->BBSc3

#generates a simulation for all combinations of parameters
script_temp_df <- slim_script_render(BBSc3, template =df)
```

You can check the content of the first template via script_template_df[[1]]

Running simulations in slim

After this setup, the runs in slim are then very simply executed via two commands. The plan() function prepares the parallel execution using the number of cores requested (n.cores), and the slim_run function executes the simulations in parallel with the provided number of cores.

```
plan(future::multisession(workers = n.cores))

resultsNonWF <- slim_run(script_temp_df, parallel = T, throw_error = TRUE)
```

Load and analysis of the simulations

The next step is to load and summarise all simulation output files. Depending on 'mev' the simulations output the mev last generations of each run and then calculate heterozygosity and number of alleles for those last generations and records those in a table (toy.csv). To speed this step we also use a parallel implementation with the same number of cores as specified above (n.cores).

```
#load all files starting with FNAME
filesM <- list.files(path=outdir,pattern="FNAME" )
gls <- list()

ext <- function(i) {
  dumvcf<- read.vcfR(file.path(outdir,filesM[i]),verbose=0)
```

```r
  gls<- vcfR2genlight(dumvcf, n.cores = 1)
  gls<-gl.compliance.check(gls, verbose=0)
  namesgls <- filesM[i]
  pNum<-as.numeric(gsub("FNAME_([0-9]+).*$", "\\1",namesgls))
  rep<-as.numeric(gsub(".*?_rep_([0-9]+).*$","\\1", namesgls))
  mig<-as.numeric(gsub(".*?_mig_([0-9]+).*$","\\1", namesgls))
  mev<-as.numeric(gsub(".*?_mev([0-9]+).*$","\\1", namesgls))
  gen<-as.numeric(gsub(".*?_gen([0-9]+).*$","\\1", namesgls))
  het<-sum(as.matrix(gls)==1)/(nInd(gls)*1*n.loc)
  glout<-gl.filter.monomorphs(gls, verbose = 0)
  Lnum<-(2*nLoc(glout)) + (n.loc-nLoc(glout))
  locd<-rbind(c(pNum, rep, mig, mev,gen, Lnum, het))
  return(locd)
}

#analyse in parallel using the ext(raction) function above.
registerDoParallel(n.cores)
system.time(locd <- foreach(i=1:length(filesM),.combine = rbind,
          .packages = c("vcfR", "dartR")) %dopar% ext(i))
```

```
##    user  system elapsed
##    0.50    0.11   38.36
```

```r
colnames(locd)<- c("pNum", "rep","mig", "mev","gen", "Lnum", "het")

#create the final table (toy)
toy <- data.frame(locd)
write.csv(toy,file.path(outdir,"toy.csv"))
```

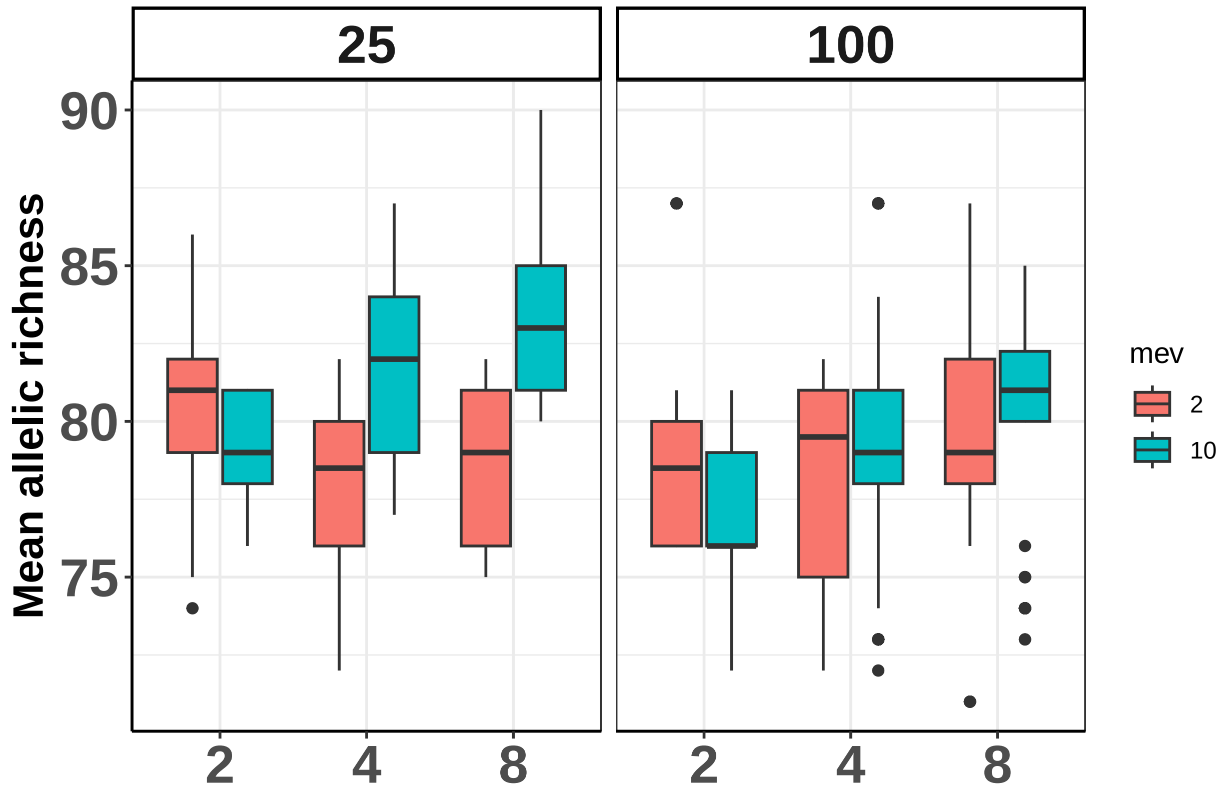
Plotting results

The results of the simulations are recorded in the the toy object. We use ggplot to produce the results and concentrate here on the main to plots, namely the allelic richness and heterozygosity for each of the parameter combinations (e.g., Figure 2 A and B of the manuscript)

```r
#prepare for plotting
toy$pNum <- factor(toy$pNum ,pops)
toy$mig <- factor(toy$mig)
toy$mev<-factor(toy$mev)
```

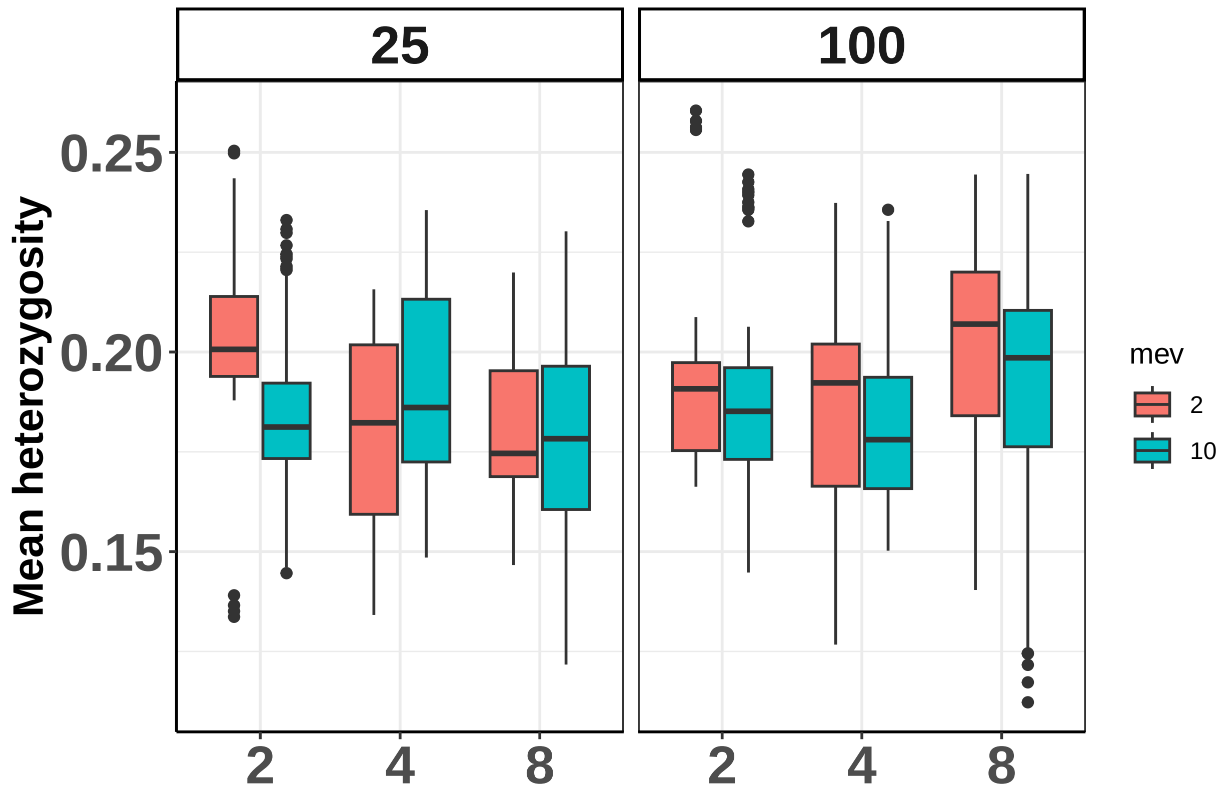
Allelic richness

```r
nma<-ggplot(aes(pNum,Lnum), data=toy)+geom_boxplot(aes(fill=mev))+theme_bw()+
 theme(strip.background = element_rect(color="black",
 fill="white",size=1, linetype="solid"),strip.text.x=element_text
 (size=20,face="bold"), strip.text.y=element_blank())+
  theme(axis.line = element_line(colour ="black"))+xlab("")+
  ylab("Mean allelic richness")+
  theme(axis.title.x=element_text(size=16,
face="bold"),axis.title.y=element_text(size=16,face="bold"),
axis.text.x=element_text(size=20,face="bold"),
axis.text.y=element_text(size=20,face="bold"))+facet_wrap(~mig)

nma
```



Heterozygosity

```
nmh<-ggplot(aes(pNum,het), data=toy)+geom_boxplot(aes(fill=mev))+
 theme_bw()+theme(strip.background =
 element_rect(color="black",
 fill="white", size=1, linetype="solid"),
 strip.text.x =element_text(size=20, face="bold"),
 strip.text.y = element_blank())+theme(axis.line =
 element_line(colour ="black"))+xlab("")+
 ylab("Mean heterozygosity")+ theme(axis.title.x
 =element_text(size=16,face="bold"),axis.title.y
 =element_text(size=16,face="bold"),axis.text.x=
 element_text(size=20,face="bold"),axis.text.y=
 element_text(size=20,face="bold"))+facet_wrap(~mig)

nmh
```


25
100
Mean heterozygosity
0.25
0.20
0.15
2
4
8
2
4
8
mev
2
10