Appendix S3. Modified functions from the PopGenReport R package for simulation from a resistance surface accommodating unequal population size. All text below this caption can be inserted into an R script and used as currently formatted as a companion to the PopGenReport package.

###
### 3 functions from PopGenReport R package modified to accommodate variable population size
###

init.popgensim_mod<-function(n.pops, n.ind, sex.ratio, n.loci, n.allels, locs = NULL, n.cov = 3)
{
  # Arguments (as described in PopGenReport documentation)
  # n.pops: number of subpopulations
  # n.ind: number of individuals per subpopulations --> modified to a vector, length of n.pops
  # sex.ratio: sex ratio of males to females
  # n.loci: number of loci
  # n.allels: number of alleles per locus
  # locs: xy coordinates of subpopulations
  # n.cov: columns of information for each simulated population preceeding genotypes (fixed at 3; population ID, sex, age)

  noffset <- n.cov # index for where genotype columns begin
  pops <- list() # create storage for population genotypes
  for (i in 1:n.pops) {
    # for each population i
    # create storage for genotypes
    empty.loci <- matrix(NA, ncol = n.loci * 2, nrow = n.ind[i])
    colnames(empty.loci) <- paste("locus", rep(seq(1, n.loci, 1), each = 2), rep(c("A", "B"), n.loci), sep = "")
    # bind genotypes to population ID, sex, and age columns and store in population storage list
    pops[[i]] <- data.frame(cbind(pop = NA, sex = NA, age = NA, empty.loci))
    # populate the population ID column
    pops[[i]][, "pop"] <- i
    # assign male sex to individuals
    pops[[i]][, "sex"] <- "male"
    # calculate the number of females in population based on sex ratio and assign female sex
    pops[[i]][1:(round(n.ind[i] * sex.ratio, 0)), "sex"] <- "female"
    # make the sex column type=factor
    pops[[i]][, "sex"] <- factor(pops[[i]][, "sex"])
    # sample alleles to generate the initial genotytes
    pops[[i]][, (n.offset + 1):(n.loci * 2 + noffset)] <- sample(1:n.allels, n.loci * 2 * n.ind[i], replace = T)
  }

  if (is.null(locs))
    names(pops) <- 1:n.pops
  else names(pops) <- row.names(locs)
return(pops)
}
run.popgensim_mod<-function (simpops, steps, cost.mat, n.offspring, n.ind, mig.rate,
disp.max, disp.rate, n.allels, mut.rate, n.cov = 3, rec = "none",
emi.table = NULL)
{
  # Arguments (as described in PopGenReport documentation)
  # simpops: list of simulated population genotypes from init.popgensim_mod function
  # steps: years of reproduction
  # cost.matrix: resistance distance matrix between population pairs
  # n.offspring: number of offspring per female
  # n.ind: number of individuals per subpopulations --> modified to a vector, length of n.pops
  # mig.rate: migration rate
  # disp.max: dispersal distance of disp.rate individuals
  # disp.rate: percentage of individuals achieving disp.max
  # n.allels: number of alleles per locus
  # mut.rate: mutation rate
  # n.cov: columns of information for each simulated population preceeding genotypes (fixed at 3;
  # population ID, sex, age)
  # rec: indicates if emigration matrix should be recorded
  # emi.table: option to provide an emigration matrix

  pops <- simpops # create new variable for simulated population list
distances <- cost.mat # create new variable for resistance distance matrix
disdis <- p2p(distances, d0 = disp.max, p = disp.rate)# convert resistance distance to probability of
dispersal
stdemi.table = disdis/rep(colSums(disdis), each = ncol(disdis)) # standardize the dispersal probability
emi.tableint = NULL # create empty variable
for (year in 1:steps) {
  # if emigration matrix is provided
  if (!is.null(emi.table)) {
    # calculate the number of individuals migrating among
    # population pairs
    emi.table <- stdemi.table * mig.rate * n.ind + emi.table
    emi.tableint <- floor(emi.table)
    emi.table <- emi.table - emi.tableint
  }
  # if emigration matrix is not provided
  # use included emigration_mod to select and move individuals
  # among populations
  emi <- emigration_mod(pops, mig.rate, emi.m = disdis, emi.table = emi.tableint)
  pops <- emi[[1]] # select the list of populations post-emigration step
  # for each population simulate reproduction
  pops<-lapply(1:length(pops), function(y)reproduction(x=pops[[y]],K=n.ind[y],type = "K.limit",n.off = n.offspring,n.cov = n.cov))
  # for each population allow for mutation of genetic alleles
  pops <- lapply(pops, mutation, n.allels, mutrate = mut.rate,n.cov = n.cov)
}
out <- pops # assign list of simulated populations to out variable
# if emigration matrix should be recorded
if (rec == "emi") {
    # add the emigration matrix as a second element in the output list
    out <- list()
    out[[1]] <- pops
    out[[2]] <- emi[[2]]
}
return(out)
}
emigration_mod<-function (xp, perc.mig, emi.m, emi.table = NULL) {
    # xp: simulated population list
    # perc.mig: percentage of migrating individuals
    # emi.m: emigration probability (based on cost dispersal distance)
    # emi.table: option to provide an emigration matrix

    n.pops = length(xp) # calculate number of simulated populations
    migs <- matrix(0, nrow = n.pops, ncol = n.pops) # create empty matrix
    migrants <- NA
    # if no emigration table is provided
    if (is.null(emi.table)) {
        # for each population
        for (i in 1:n.pops) {
            # check that the population is not NULL
            if (is.null(pop.size)) {
                # draw a random uniform number for each simulated individual
                # and assign a 1 to become a migrant if value is less than perc.mig
                migrants[i] <- sum(ifelse(runif(nrow(xp[[i]])) < perc.mig, 1, 0))
            # allow migration of the above defined number of migrants among
            # population as a function of the probability of migration for
            # each population pair
                fromto <- table(sample(1:n.pops, migrants[i], replace = T, prob = emi.m[, i]))
                # convert names of the vector moving migrants to numeric
                to <- as.numeric(names(fromto))
                # populate the pair-wise migrant matrix with the number of
                # migrants moving among population pairs
                for (ii in 1:length(to)) migs[i, to[ii]] <- migs[i, to[ii]] + fromto[ii]
            }
        }
    }
    # if emigration table is provided
    # over-write the empty migs matrix with the provied table
    else migs <- emi.table
    # for each population pair
    for (from in 1:n.pops) {
for (to in 1:n.pops) {
  psize <- nrow(xp[[from]]) # calculate the size of each source simulated population
  m <- migs[from, to] # find the number of migrants to move to the recipient population
  if (m > 0) { # if the number of migrants is greater than zero
    ind.from <- sample(1:psize, m, replace = F) # randomly select m migrants
    xp[[to]] <- rbind(xp[[to]], xp[[from]][ind.from, ]) # add randomly selected migrants from source population to recipient population
    xp[[from]] <- xp[[from]][-ind.from, ] # remove individual from source population that migrated to recipient population
    xp[[to]][, "pop"] <- to # re-define population ID column of recipient population
  }
}

results <- list(xp, migs)
return(results)