Modelling Biological Systems Exam 2016

Sergii Gladchuk

December 28, 2016

1. Describe shortly step-by-step how a genetic algorithm works.

I will describe genetic algorithm, using example to solve travel problem - find shortest way to visit all the cities from the distance matrix:

```
# citi matrix
  cities = c("Lund", "Kalmar", "Stockholm", "Karlstad", "Göteborg", "Jönkoping");

  dist = c(0,   267,    602,    506,    262,    282,
         267,  0,  412,    467,    341,    241,
         602,  412,    0,  305,    467,    321,
         506,  467,    305,    0,  247,    243,
         262,  341,    467,    247,    0,  149,
         282,  241,    321,    243,    149,    0);
  dist = matrix(dist,nrow=6,ncol=6)
  rownames(dist) = cities;
  colnames(dist) = cities;
  dist
```

```
##           Lund Kalmar Stockholm Karlstad Göteborg Jönkoping
## Lund         0    267       602      506      262       282
## Kalmar     267      0       412      467      341       241
## Stockholm  602    412         0      305      467       321
## Karlstad   506    467       305        0      247       243
## Göteborg   262    341       467      247        0       149
## Jönkoping  282    241       321      243      149         0
```

The main idea of GM algorithm is to use natural selection process, which produced all the life on Earth including us, to solve computational problems. So genetic algorithm imitate natural process of chromosome: replication, recombination, mutation to find best possible solution to the defined problem.

There are 7 steps in genetic algorithm:

1. **Represent problems as genes in chromosome**

In travel problem it is no so hard since the aim is to find shortest way to visit all the cities given distances for all the combination of cities. Unique combination of cities indexes from 1 to 6 should be is specific order - if driver goes in that order the distance is the lowest possible. Each gene represent one index - one chromosome represent specific order or indexes, and fitness is total distance.

2. **Random generation of starting chromosom population**

We have to start with some random set of chromosome, which do not represent the best solution yet. In my case I randomly create 50 combination of 6 indexes and record them to chromosomes with -1 fitness.

```
Create_pop = function(POPSIZE,GENE_NUMBER) {     # This function creates
                                                 # a random population of 50 chromosomes

   IndL = list()           # create an empty list to put on Inds.
                       # A list of inidividuals = a population :)

   for(Count1 in 1:POPSIZE) {     # For counter for the chromosomes

     Gene = c()
     pos = 1:6
     for(Count2 in 1:GENE_NUMBER) {  # For counter for genes inside chromosomes

       g = runif(1,1, 7 - Count2)    # g is a random number between 0 and 1
       Gene[Count2] = pos[g]

       #remove position from list
       pos = pos[-g]

     }

     IndL[[Count1]] = list(Gene = Gene, Fitness = -1)

   }

   return(IndL)
 }
```

3. **Evaluate fitness of individual chromosomes**

The evaluation function calculates the distances between the cities for particular chromosome, based on the order of genes, and record the value into the structure, for further steps:

```
Evaluate_fitness = function(IndL,POPSIZE,GENE_NUMBER) {

    for(Count1 in 1:POPSIZE) {
      Added_Value = 0;
      for(Count2 in 2:GENE_NUMBER) {
        row = IndL[[Count1]]$Gene[Count2 - 1]
        col = IndL[[Count1]]$Gene[Count2]
        Added_Value = Added_Value + dist[row, col]
      }
      Added_Value = Added_Value +
        dist[IndL[[Count1]]$Gene[GENE_NUMBER],IndL[[Count1]]$Gene[1]]

      IndL[[Count1]]$Fitness = Added_Value
    }

    return(IndL)
  }
```

4. **Sort after avaluation**

Sorts all chromosomes in population from best (lowest distance) to worst one.

```
Sort = function(IndL) {

    IndL = IndL[order(sapply(IndL,function(x) x$Fitness))] # sort the list by fitness

    return(IndL)
  }
```

5. **Recombination (pairing)**

Most important step in whole algorithm, which takes top rated chromosomes and pairs them, produces new combination (offsprings) based on very best solutions so far. This step can lead to even better solutions. There are several ways to produce offsptings:

- In binary GA - it is simple recombination
- In continuous GA - proportional blending, linear blending, combinations of previous.

In my case it is not purely continuous GA - I split randomly every pair of chromosomes from 1 to 20 of top rated in random point, swap and marge two halves of chromosomes into two new chromosomes.

6. **Inserting of offspring**

Newly created chromosomes substitute worst rated chromosomes in our population. So our best solutions are not touched.

```
Reproduce = function(IndL,POPSIZE,NUM_BREED) {

    countback = POPSIZE
    for(Count1 in seq(1,POPSIZE/2,2)) {
      if(Count1 <= NUM_BREED) {
        Gene_No = round(runif(1,1,GENE_NUMBER)) #to find split point

        offspring1 = c(IndL[[Count1]]$Gene)

        offspring2 = c(IndL[[Count1 + 1]]$Gene)

        for(Count2 in 1:GENE_NUMBER) {
          if(Count2 <= Gene_No) {
            oldPos = match(offspring2[Count2],offspring1)
            rGene = offspring1[Count2]
            offspring1[Count2] = offspring2[Count2]
            offspring1[oldPos] = rGene

          } else {
            oldPos = match(offspring1[Count2],offspring2)
            rGene = offspring2[Count2]
            offspring2[Count2] = offspring1[Count2]
            offspring2[oldPos] = rGene

          }
        }
        #record new offsprings
        IndL[[countback]]$Gene = offspring1
        IndL[[countback - 1]]$Gene = offspring2
        countback = countback - 2
      }
    }

    return(IndL)
  }
```

7. **Mutation**

Also crucial step in genetic algorithm. It just introduces some randomness during the evolution of population and can help to solve the problem in case there is no best possible solutions in population, and recombination cannot produce one either. The important thing is that mutation is introduced only to low-fitness chromosomes, so we do not mutate our best solutions.

In my case I just randomly do swap for set of worst chromosomes based on rate of mutation constants.

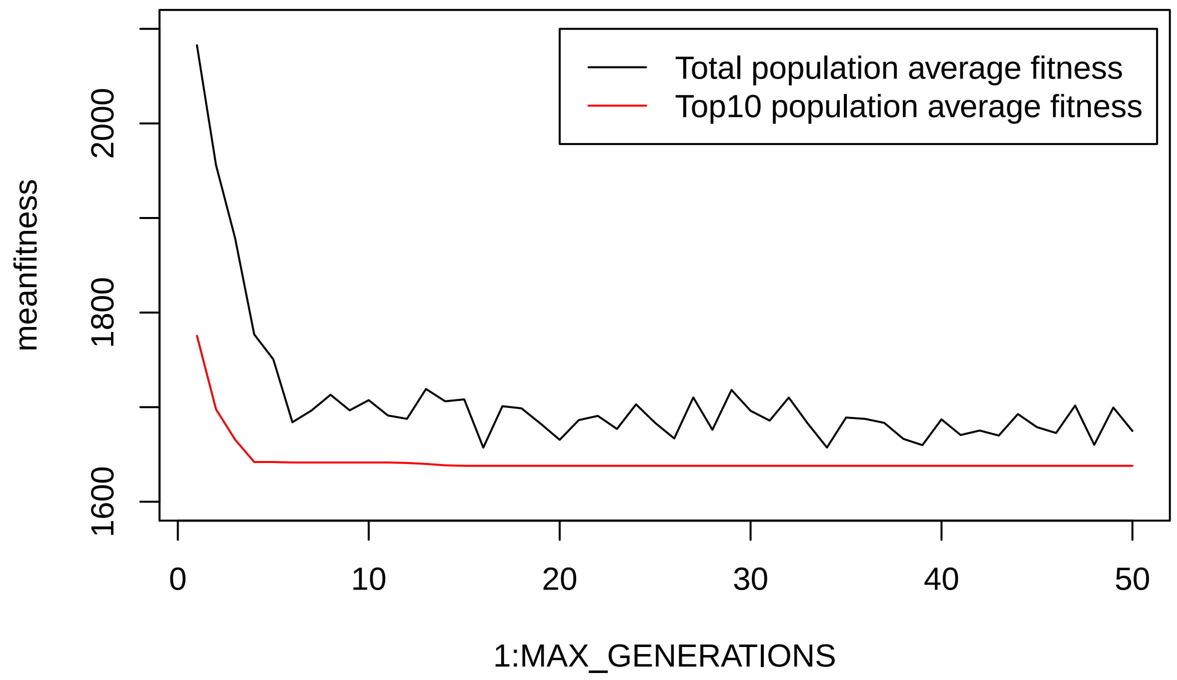
```
Mutate = function(IndL,POPSIZE,NUM_BREED,mutation_free,mutation_rate) {

    for(Count1 in mutation_free:POPSIZE) {
      for(Count2 in 1 : GENE_NUMBER) {
        Chance = runif(1,0,1)
        if(Chance <= mutation_rate) {
          removedGene = IndL[[Count1]]$Gene[Count2]
          newGene = round(runif(1,1,GENE_NUMBER))
          oldGenePos = match(newGene,IndL[[Count1]]$Gene)
          IndL[[Count1]]$Gene[Count2] = newGene
          IndL[[Count1]]$Gene[oldGenePos] = removedGene

        }

      }
    }

    return(IndL)
  }
```

Steps 3 - 7 should be put into the loop to produce several generations till some cut-off time. Each next generation potentially improve solution of previous one, so very last generation will have preserved very best solution.

The loop routine and output of final solution in my case:

```
# variables:
 POPSIZE = 50         # popsize is number of chromosomes in the population
 GENE_NUMBER = 6      # gene number is number of genes in each chromosome,


 MAX_GENERATIONS = 50  # number of iterations
 NUM_BREED = 20        # out of the 50 chromsomes (= possible solutions)
                       # the 20 best are allowed to reproduce

 # genetic operators
```

```r
  mutation_rate = 0.04  # how large percentage of the genes experience mutations
  mutation_free = 10    # number of topranked Individuals
                        # that should be saved from mutation



IndL = Create_pop(POPSIZE,GENE_NUMBER) # calls function that generates random chromosomes

  # main loop over all of our functions

  meantopfitness = c() # empty vectors to store results
  meanfitness = c()

  for(generation in 1:MAX_GENERATIONS) {
    IndL = Evaluate_fitness(IndL,POPSIZE,GENE_NUMBER) # give back altered population with fitness

    IndL = Sort(IndL)
    IndL = Reproduce(IndL,POPSIZE,NUM_BREED)
    IndL = Mutate(IndL,POPSIZE,NUM_BREED,mutation_free,mutation_rate)

    # output
    addfitness = 0
    add_top = 0
    for(Count in 1:POPSIZE) {
      addfitness = addfitness + IndL[[Count]]$Fitness
      if(Count <= 10) {
        add_top = add_top + IndL[[Count]]$Fitness
      }
    }
    meantopfitness[generation] = add_top/10
    meanfitness[generation] = addfitness/POPSIZE

  }

  ## plot results

  plot(1:MAX_GENERATIONS,meanfitness,ylim=c(1600,2100),pch=19,type="l")
  points(1:MAX_GENERATIONS,meantopfitness,pch=19,col="red",type="l")
  legend(20,2100, legend=c('Total population average fitness',
                          'Top10 population average fitness'),
         col=c('black','red'),lty=c(1,1))
```



```
route = ''
for (Count in 1 : GENE_NUMBER) {
  if (Count != GENE_NUMBER){
    route = c(route, cities[IndL[[1]]$Gene[Count]])
  } else {
    route = c(route, cities[IndL[[1]]$Gene[Count]])
  }
}
cat("The optimal route is:\n", route, "\nTotal distance: ", IndL[[1]]$Fitness)
```

```
## The optimal route is:
##   Göteborg Lund Kalmar Stockholm Karlstad Jönkoping
## Total distance:  1638
```