Discrete Optimization

Evan Carey

March 7, 2019

Contents

Introduction

In this assignment, you will solve a discrete optimization problem in the context of allocating healthcare resources in the state of MO. Let's assume there are a set of clinics in the state of MO that currently do not have specialty mental health care. Out of all these clinics, you must pick 5 clinics to implement specialty mental health care. How can you go about picking which 5 clinics (out of the larger set of clinics) is the 'best' 5 ?

If you know the home locations of a patient population, and you also know the locations of a bunch of clinics, can you choose which set of clinics to expand resources to in order to maximally expand access to care?

Data Overview

I have simulated fake home locations for ~6 million Missouri residents and provided their latitude and longitude. I have also included an Urban / Rural designation for each person. This file is called `Mo_pop_Sim.csv`.

Additionally, I have provided the location of all federally qualified health centers in MO (FQHC) as a shape file: MO_2018_Federally_Qualified_Health_Center_Locations

Code Demo and Questions:

Overview

I will start by demonstrating different coding approaches. I have embedded questions below, you will need to slightly modify my coding approaches to answer the questions that are presented inline.

Data import:

First, we must import the datasets of interest. For the MO residents, I will take a small random sample to make the math easier on our computers. Also, I will convert the shape file to a data.table object.

```
library(data.table)
library(rgdal)
```

```
## Loading required package: sp

## rgdal: version: 1.2-20, (SVN revision 725)
##  Geospatial Data Abstraction Library extensions to R successfully loaded
##  Loaded GDAL runtime: GDAL 2.2.3, released 2017/11/20
##  Path to GDAL shared files: C:/Users/evancarey/Documents/R/win-library/3.5/rgdal/gdal
##  GDAL binary built with GEOS: TRUE
##  Loaded PROJ.4 runtime: Rel. 4.9.3, 15 August 2016, [PJ_VERSION: 493]
##  Path to PROJ.4 shared files: C:/Users/evancarey/Documents/R/win-library/3.5/rgdal/proj
##  Linking to sp version: 1.2-7
```

```
#### Import Simulated MO residents ####

## includes urban/rural designation
MO_residents <-
  fread('C:/Users/evancarey/Dropbox/Work/SLU/health_data/missouri_spatial/MO_people_sim/Mo_pop_Sim.csv')

## take small random sample (1%)
set.seed(32)
MO_residents_sample <-
  MO_residents[sample(.N,size = .N*.01)]
MO_residents_sample
```

```
##        UR      long      lat
##     1:  R -91.64121 40.03330
##     2:  R -91.69872 37.89038
##     3:  U -90.21123 38.77534
##     4:  U -90.31673 38.52737
##     5:  U -94.48445 39.23081
##    ---
## 63320:  U -93.35018 37.24793
## 63321:  R -93.57726 36.89733
## 63322:  R -93.58427 38.50715
## 63323:  U -93.31100 37.14882
## 63324:  U -90.39841 38.30173
```

```
#### Import Missouri FQHC locations ####
data_path <- 'C:/Users/evancarey/Dropbox/Work/SLU/health_data/missouri_spatial/MO_2018_Federally_Qualifi
MO_FQHC <-
  readOGR(data_path,
          'MO_2018_Federally_Qualified_Health_Center_Locations')
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "C:\Users\evancarey\Dropbox\Work\SLU\health_data\missouri_spatial\MO_2018_Federally_Qualifie
## with 197 features
## It has 12 fields
## Integer64 fields read as strings:  OBJECTID
```

```r
## create data.table for later use
MO_FQHC_df <-
  data.table(as.data.frame(MO_FQHC))[,list(OBJECTID, Latitude,Longitude)]
MO_FQHC_df
```

```
##      OBJECTID Latitude Longitude
##   1:        1 38.43595 -90.55468
##   2:        2 37.71462 -91.13398
##   3:        3 38.16026 -92.60146
##   4:        4 36.77257 -90.45721
##   5:        5 38.96288 -94.49885
##  ---
## 193:      193 37.94593 -91.77395
## 194:      194 37.22432 -93.29159
## 195:      195 36.56534 -91.56262
## 196:      196 38.67776 -90.23025
## 197:      197 38.56633 -92.20211
```

Simple case: 8 total clinics, pick 4

Let's first simplify the problem by considering only 8 total clinics, then picking 4 of the 8 clinics to expand services to. We will use the small sample of MO residents as well.

If we have 8 total sites, and we want to pick 4 out of the 8 to expand services, how many total options are there? This is a permutation where order does not matter. This is often called the 'binomial coefficient', with 'n choose k'. Check the wiki page for more info if you are unfamiliar with this: https://en.wikipedia.org/wiki/Binomial_coefficient

R has a function called `choose()` that will calculate this for us:

```r
choose(8,4) # 70 possible solutions. We can calculate all of these
```

```
## [1] 70
```

There are 70 possible solutions to our problem. With so few possible solutions, we can actually just do a brute force calculation here (calculate every possible solution). We will do that in a moment.

Let's grab 8 random sites from our full list of 194 sites for now. We will pretend we only have 8 sites for the first part of this assignment:

```r
## Grab 8 random sites
set.seed(32)
MO_FQHC_df_8 <- MO_FQHC_df[sample(8)]
```

If we have 8 total sites, and we want to pick 4 out of the 8 sites, what does one 'solution' to the problem look like? It would simpy be picking 4 out of the 8 sites! So how can we generate a matrix of all possible solutions, where each row in the matrix is a single solution? in this case, we can use the `combn()` function. First I will test the code using a small test set:

```r
## A solution is picking any 4 of these 8 sites (order doesn't matter)
## We can generate all possible solutions using the combn function
t(combn(x=1:4,m = 2)) # each row is a permuation
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    1    3
## [3,]    1    4
```

```
## [4,]    2    3
## [5,]    2    4
## [6,]    3    4
```

```r
choose(4,2) # as expected, 6
```

```
## [1] 6
```

Now let's implement it on the our 8 choose 4 sites:

```r
## generate all possible solutions
possible_solutions <-
  MO_FQHC_df_8[,t(combn(x = as.character(OBJECTID),
                        m = 4))]
head(possible_solutions)
```

```
##      [,1] [,2] [,3] [,4]
## [1,] "5"  "8"  "7"  "4"
## [2,] "5"  "8"  "7"  "1"
## [3,] "5"  "8"  "7"  "3"
## [4,] "5"  "8"  "7"  "2"
## [5,] "5"  "8"  "7"  "6"
## [6,] "5"  "8"  "4"  "1"
```

Now that we have all 70 solutions, how do we evaluate how good each solution is? If we consider access to care to be a function of how far someone lives from a clinic, we might decide to minimize the mean distance to the nearest clinic. We could also decide to maximize the number of people who live within 30 miles of a clinic, or something else. For either of those options, our first step is to calculate how far each person lives from each clinic. I have chosen to create a cartesian product of every patient / clinic combination, then store it in a long format:

```r
## how good is each solution? Depends on our definition of good.
## need to define and calculate an objective function (cost function)
## First we calculate the geodesic distance between each Patient and each clinic
## create a data.table of every clinic/patient pair:
## generate ID's for the residents
MO_residents_sample[,pat_ID:=1:.N]
## create a cartesian product with all combinations
pat_clinic_combinations <-
  CJ(OBJECTID=as.character(MO_FQHC_df_8$OBJECTID),
     pat_ID=MO_residents_sample$pat_ID)
## merge in the lat and long for clinics
setkey(pat_clinic_combinations,OBJECTID)
setkey(MO_FQHC_df_8,OBJECTID)
pat_clinic_combinations[MO_FQHC_df_8,':='(Latitude_clinic=Latitude,
                                          Longitude_clinic=Longitude)]
## merge in the lat and long for patients
setkey(pat_clinic_combinations,pat_ID)
setkey(MO_residents_sample,pat_ID)
pat_clinic_combinations[MO_residents_sample,':='(Latitude_patient=lat,
                                                 Longitude_patient=long)]
```

Now that I have this pair-wise list, I can calculate the distance between every patient / clinic pair:

```r
## now use geosphere::distGeo to calculate the distance
library(geosphere)
?distGeo()
```

```
## starting httpd help server ... done
```

```
pat_clinic_combinations[,distancemiles := distGeo(matrix(c(Longitude_clinic,
                                                           Latitude_clinic),
                                                         ncol=2),
                                                  matrix(c(Longitude_patient,
                                                           Latitude_patient),
                                                         ncol=2))/1609.344]
## now we have all the distances from each patient home to each clinic.
```

Now that we have the distances, we are ready to write our objective function. The objective function is simply a function that calculate how good a given solution to our problem is. Recall that a single solution is simply a list of 4 sites (out of the 8 sites) that we will choose for care expansion. As a first step, we will consider the objective function to be the average distance from every patient's home to the closest clinic.

```
## Let's write an objective function that will evaluate the cost for
## a potential solution
## it should take in as arguments: (1) the solution and (2) the distance matrix
## first we will hard code it, then turn into function.
## here is a sample solution
possible_solutions[1,]
```

```
## [1] "5" "8" "7" "4"
```

```
## calculate the minimum distance by patientID (closest clinic)
## then take the average
setkey(pat_clinic_combinations,OBJECTID)
pat_clinic_combinations[J(possible_solutions[1,]),
                        list(min_dist=min(distancemiles)),
                        by=pat_ID][,mean(min_dist)]
```

```
## [1] 53.31197
```

Now that I have the code working for a single solution, let's turn it into a function:

```
## Turn the code from above into a function
obj_func_min_mean <-
  function(solution,distMat) {
    setkey(distMat,OBJECTID)
    distMat[J(solution),
            list(min_dist=min(distancemiles)),
            by=pat_ID][,mean(min_dist)]
  }
```

And now we can test that function:

```
## function for first solution
obj_func_min_mean(solution = possible_solutions[1,],
                  distMat = pat_clinic_combinations)
```

```
## [1] 53.31197
```

```
## function for second solution
obj_func_min_mean(solution = possible_solutions[2,],
                  distMat = pat_clinic_combinations)
```

```
## [1] 44.59366
```

Now that it works, we can apply it to every single possible solution then find the smallest distance. This is considered the 'brute force' approach - calculate every possibility, then take the 'best' possibility.

```r
## apply to all solutions
dist_temp_8 <-
  apply(possible_solutions,
        MARGIN = 1,
        obj_func_min_mean,
        distMat = pat_clinic_combinations)
## create dataframe of results
temp_results_8 <-
  data.table(dist_temp_8,possible_solutions)
temp_results_8
```

```
##     dist_temp_8 V1 V2 V3 V4
##  1:    53.31197  5  8  7  4
##  2:    44.59366  5  8  7  1
##  3:    53.60984  5  8  7  3
##  4:    55.33742  5  8  7  2
##  5:    54.38686  5  8  7  6
##  6:    52.83664  5  8  4  1
##  7:    52.73023  5  8  4  3
##  8:    62.28142  5  8  4  2
##  9:    60.56054  5  8  4  6
## 10:    44.17391  5  8  1  3
## 11:    53.00498  5  8  1  2
## 12:    50.63307  5  8  1  6
## 13:    55.02716  5  8  3  2
## 14:    54.05351  5  8  3  6
## 15:    62.00088  5  8  2  6
## 16:    42.63234  5  7  4  1
## 17:    74.14007  5  7  4  3
## 18:    60.20792  5  7  4  2
## 19:    74.56267  5  7  4  6
## 20:    44.50680  5  7  1  3
## 21:    44.31540  5  7  1  2
## 22:    43.73270  5  7  1  6
## 23:    60.60856  5  7  3  2
## 24:    73.50600  5  7  3  6
## 25:    61.80998  5  7  2  6
## 26:    42.00713  5  4  1  3
## 27:    50.80101  5  4  1  2
## 28:    49.33872  5  4  1  6
## 29:    59.80304  5  4  3  2
## 30:    73.05816  5  4  3  6
## 31:    67.47657  5  4  2  6
## 32:    44.18858  5  1  3  2
## 33:    43.53176  5  1  3  6
## 34:    50.73409  5  1  2  6
## 35:    61.49534  5  3  2  6
## 36:    56.91308  8  7  4  1
## 37:    65.94932  8  7  4  3
## 38:    67.99970  8  7  4  2
## 39:    67.61609  8  7  4  6
## 40:    57.39300  8  7  1  3
## 41:    58.93987  8  7  1  2
## 42:    57.98798  8  7  1  6
```

```
## 43:    68.24626  8  7  3  2
## 44:    67.27265  8  7  3  6
## 45:    69.38392  8  7  2  6
## 46:    65.45594  8  4  1  3
## 47:    99.58582  8  4  1  2
## 48:    99.16437  8  4  1  6
## 49:    76.62889  8  4  3  2
## 50:    76.22217  8  4  3  6
## 51:   108.81303  8  4  2  6
## 52:    67.75288  8  1  3  2
## 53:    66.77923  8  1  3  6
## 54:    98.45003  8  1  2  6
## 55:    78.10335  8  3  2  6
## 56:    55.22622  7  4  1  3
## 57:    56.73589  7  4  1  2
## 58:    56.66729  7  4  1  6
## 59:    73.02213  7  4  3  2
## 60:    86.27730  7  4  3  6
## 61:    74.82157  7  4  2  6
## 62:    57.40767  7  1  3  2
## 63:    56.75089  7  1  3  6
## 64:    58.33374  7  1  2  6
## 65:    74.71447  7  3  2  6
## 66:    65.54891  4  1  3  2
## 67:    65.40618  4  1  3  6
## 68:    96.81320  4  1  2  6
## 69:    83.44637  4  3  2  6
## 70:    67.23700  1  3  2  6
##     dist_temp_8 V1 V2 V3 V4
```

```
## find minimum (brute force complete)
temp_results_8[min(dist_temp_8) == dist_temp_8]
```

```
##    dist_temp_8 V1 V2 V3 V4
## 1:    42.00713  5  4  1  3
```

Implement a genetic algorithm

Instead of a brute force approach, now we can implement a genetic algorithm to optimize the placement of 4 new programs out of the 8 programs (our example from above). Since we know the right answer from our brute force approach above, we can verify the genetic algorithm works as expected.

I will use the GA package to implement the genetic algorithm. This package has the ability to maximize instead of minimize, so if I want a minimal solution, I must define a new function that is the inverse of the objective function.

Another wrinkle to this approach is the 'solution' the GA is providing is simply a reordered vector with a lower and upper bound defined (when we say `type = "permutation"`). That worked well in our 'traveling salesman' problem, where we were interested in the order of all sites. In this case, we are not interested in reording all the sites - instead, we are interested in picking 4 of the sites (out of the 8). One way to accomplish this is to make the objective function take in a 'solution' which is all 8 sites in a random order, then only use the first 4 sites in our calculation of distances. I have hard coded that into the function below using the `k` argument:

```r
## Implement genetic algorithm
library(GA)
```

```
## Loading required package: foreach

## Loading required package: iterators

## Package 'GA' version 3.1.1
## Type 'citation("GA")' for citing this R package in publications.
```

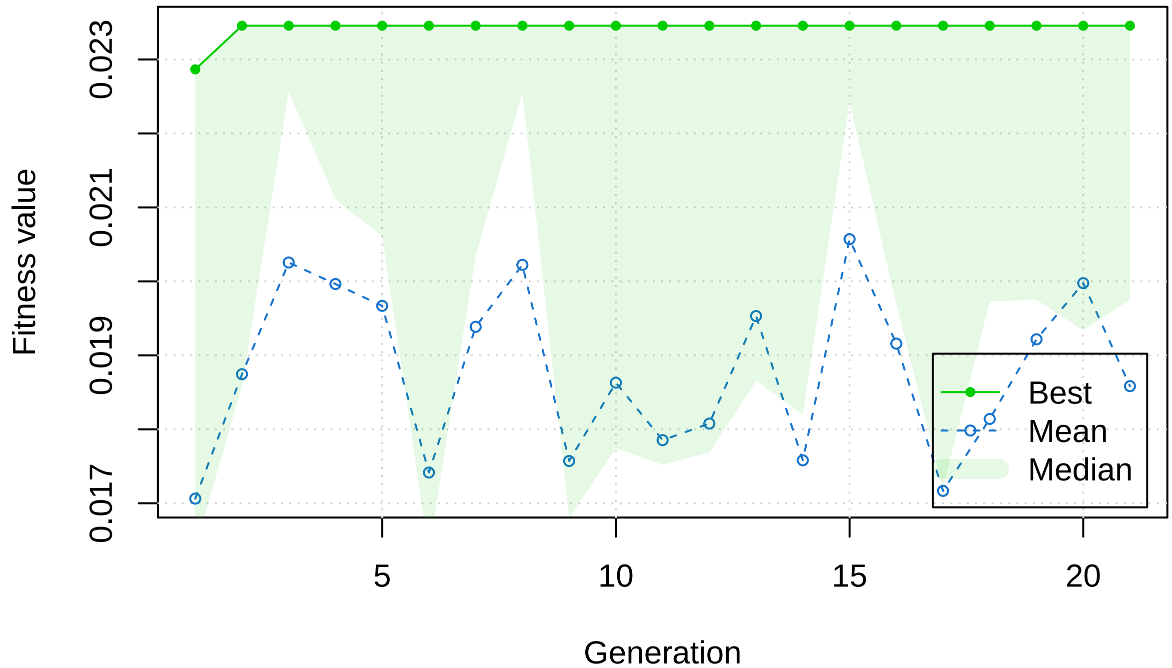
```r
## make inverse to maximize
## Alter the objective function a bit so it works with ga()
## I rewrote this function so it only uses the first 4 sites of the 'solution'
## the 'solution' is all every site in a random order.
obj_func_min_mean2 <-
  function(solution,distMat,k) {
    setkey(distMat,OBJECTID)
    distMat[J(distMat[,unique(OBJECTID)[solution[1:k]]]), ## here is where I subset to only 4 sites
            list(min_dist=min(distancemiles)),
            by=pat_ID][,mean(min_dist)]
  }
## make the inverse function so the maximum is the minimum...
Fitness_f <-
  function(solution,...) 1/obj_func_min_mean2(solution, ...)
```

Now we can run the genetic algorithm:

```r
## Run the genetic algorithm
## it is picking a 'solution' which is a random combination of all sites
## I coded is to only use the first 'k' of those sites in the function above
GA <- ga(type = "permutation", fitness = Fitness_f,
         distMat = pat_clinic_combinations, k=4,
         lower = 1, upper = 8, # these are the boundaries of the solution space (number of sites)
         popSize = 10, # total number of solutions per generation
         maxiter = 1000,  # max number of generations
         run = 20, # if we get the same best answer 20 times in a row, stop
         pmutation = 0.2) # this is the mutation rate per generation
## did it find the optimal solution?
summary(GA)
```

```
## -- Genetic Algorithm -------------------
##
## GA settings:
## Type                  =  permutation
## Population size       =  10
## Number of generations =  1000
## Elitism               =  1
## Crossover probability =  0.8
## Mutation probability  =  0.2
##
## GA results:
## Iterations             = 21
## Fitness function value = 0.02345637
## Solution =
##      x1 x2 x3 x4 x5 x6 x7 x8
## [1,]  7  4  5  1  8  6  3  2
```

```r
plot(GA)
```




```r
GA@solution # first 4 are the ones we choose
```

```
##      x1 x2 x3 x4 x5 x6 x7 x8
## [1,]  7  4  5  1  8  6  3  2
```

```r
1/GA@fitnessValue
```

```
## [1] 42.63234
```

```
## It found the 'best' answer, which is an average distance of 42.00713
```

The genetic algorithm found the max solution!

Extend this code (questions for you to complete):

Using the above code as a template, implement the following objective functions. Use both the brute force approach and the genetic algorithm to 'solve' the problem.

1. Change the objective function to minimize the median distance instead of the mean distance. Do you get a different optimal solution than when I used the mean?
2. Change the objective function to maximize the number of patients that live within 40 miles of a clinic. Do you get a different optimal solution than the median or mean?
3. In the prior 2 questions, we have not worried about the differences between urban and rural patients (they essentially had equal weights in our objective function). Now, I want you to make the rural patients 'worth more' in the objective function. You could simply exclude all the urban patients from

the data and only consider the rural patients; But I still want you to consider the urban patients in the objective equation, I just want them to be worth 'less' than the rural patients. Use your answer from number two to construct a reasonable objective function for the following: 'Maximize the number of patients that live within 40 miles of a clinic. Rural patients should be worth 5 times as much as urban patients.' Does this give you a different answer than you got in number 2?

Extending this to all 197 sites, choosing 10 for care expansion

Now we will expand this code to look at all 197 sites (instead of just 8), and we will pick 10 of these sites (instead of 4).

Note - if you get memory errors when you try to run this, you can use a random sample of 100 sites instead of all 197 sites. Email me if you have issues

Question to answer: How many potential solutions are there to this problem?

```
## What about using the full list of 197 clinics and choosing 10?
total_sites <-
  197
chosen_sites <-
  10
## use the choose function to calculate total number of solutions:
```

Since we are considering all 197 sites, we need to calculate a new distance matrix from every patient home to every clinic site. I will follow the same approach I used above, creating a 'long' form of this table:

```
## We need to construct the full distance matrix of each patient to all 197 hospitals:
## create a data.table of every clinic/patient pair using the sample of patients
## (not the full 6 million!):
MO_residents_sample[,pat_ID:=1:.N]
pat_clinic_combinations2 <-
  CJ(OBJECTID=as.character(MO_FQHC_df$OBJECTID),
     pat_ID=MO_residents_sample$pat_ID) ## use the sample, not the full list!
```

What if we used the full `MO_Residents` file instead of just our sample? How large would the distance table be?

```
## Calculate the size of the distance table for all MO patients instead of just the sample
## note you can't create the table, its too big.
## Just use the number of rows in each table to calculate the size of the final table.
```

That is a big table, too big to calculate on one PC. So we will just use the sample of patients. I will merge in the latitudes and longitudes, then calculate the distance:

```
## look up lat and long for each one
setkey(pat_clinic_combinations2,OBJECTID)
setkey(MO_FQHC_df,OBJECTID)
pat_clinic_combinations2[MO_FQHC_df,':='(Latitude_clinic=Latitude,
                                        Longitude_clinic=Longitude)]
setkey(pat_clinic_combinations2,pat_ID)
setkey(MO_residents_sample,pat_ID)
pat_clinic_combinations2[MO_residents_sample,':='(Latitude_patient=lat,
                                                 Longitude_patient=long)]
## calculate distances
pat_clinic_combinations2[,distancemiles := distGeo(matrix(c(Longitude_clinic,
                                                            Latitude_clinic),
                                                          ncol=2),
```

```r
                                                            matrix(c(Longitude_patient,
                                                                     Latitude_patient),
                                                                   ncol=2))/1609.344]
```

Now we will use the same objective function as we used above to minimize the mean distance to the closest clinic across all patients.

```r
## use this same objective function
obj_func_min_mean2 <-
  function(solution,distMat,k) {
    setkey(distMat,OBJECTID)
    distMat[J(distMat[,unique(OBJECTID)[solution[1:k]]]),
            list(min_dist=min(distancemiles)),
            by=pat_ID][,mean(min_dist)]
  }
Fitness_f <-
  function(solution,...) 1/obj_func_min_mean2(solution,...)
```

We can test that this works on a single solution by simply generating a vector from 1 to 197, which will pick only the first 10 sites:

```r
## test it works on one possible solution:
obj_func_min_mean2(1:197,
                   distMat=pat_clinic_combinations2,
                   k=10)
```

```
## [1] 66.15685
```

```r
Fitness_f(1:197,
          distMat=pat_clinic_combinations2,
          k=10)
```

```
## [1] 0.01511559
```

Can we find a better solution? We can't do brute force, there are too many possible solutions. But perhaps we can just randomly generate solutions and test them. . . here I randomly find 15 permutations of the numbers 1 to 197, then I test those solutions:

```r
set.seed(23)
## test it on 15 random solutions:
mat_1 <-
  t(sapply(1:15, # number of replicates
           function(x) sample(197,197)))
str(mat_1) # this is a matrix of solutions, each row is a solution
```

```
##  int [1:15, 1:197] 114 193 164 98 45 67 179 13 189 148 ...
```

```r
## score them all by applying the objective function by row
res1 <- apply(mat_1,
              1,
              obj_func_min_mean2,
              distMat=pat_clinic_combinations2,
              k=10)
## find the best one
min(res1)
```

```
## [1] 27.286
```

```r
mat_1[which(res1 == min(res1)),][1:10]
```

```
##  [1]  67 115 146  61  98 129  95  46  53 177
```

This gives me a solution with an average distance of 27.286

Rerun this 5 more times (with 10 random solutions per time) without a seed. What is the best solution you can come up with? What is the average distance for your best solution?

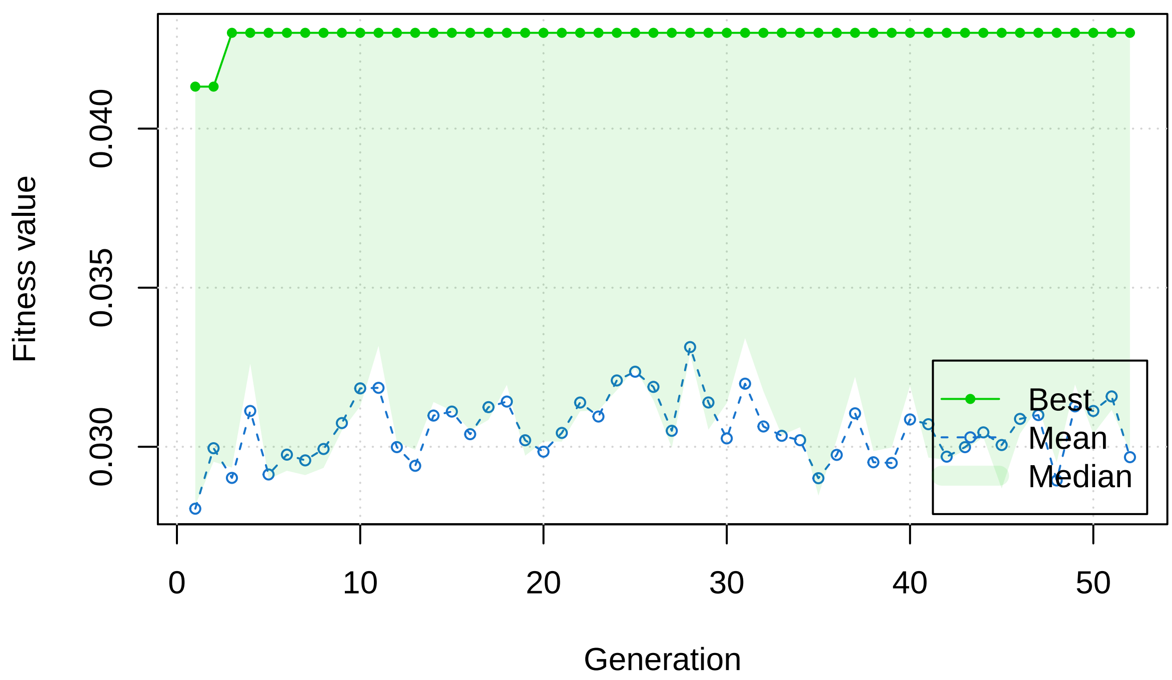
Run the Genetic Algorithm on this:

Now we will run a genetic algorithm on this full set of 197 sites and the random sample of patients. We can use the same approach we used above:

```r
## Run the genetic algorithm
## it is picking a 'solution' which is a random combination of all sites
## I coded is to only use the first 'k' of those sites in the function above
GA2 <- ga(type = "permutation", fitness = Fitness_f,
         distMat = pat_clinic_combinations2, k=chosen_sites,
         lower = 1, upper = total_sites,
         elitism = 5, # this is how many top solution to keep every generation
         popSize = 50, # this is the number of solutions per generation
         maxiter = 1000,
         run = 50,
         pmutation = 0.3)
## Check your final solution, did it work well?
summary(GA2)
```

```
## -- Genetic Algorithm -------------------
##
## GA settings:
## Type                  =  permutation
## Population size       =  50
## Number of generations =  1000
## Elitism               =  5
## Crossover probability =  0.8
## Mutation probability  =  0.3
##
## GA results:
## Iterations             = 52
## Fitness function value = 0.04301099
## Solutions =
##       x1 x2 x3  x4 x5  x6  x7 x8  x9 x10  ...  x196 x197
## [1,] 72 34 61 122 47 111 162 10 143  88        179   44
## [2,] 72 34 61 122 47 111 162 10 143  88        179   44
## [3,] 72 34 61 122 47 111 162 10 143  88        179   44
## [4,] 72 34 61 122 47 111 162 10 143  88        179   44
## [5,] 72 34 61 122 47 111 162 10 143  88        179   44
```

```r
plot(GA2)
```




```
GA2@solution # first 10 are the ones we choose
```

```
##       x1 x2 x3  x4 x5  x6  x7 x8  x9 x10 x11 x12 x13 x14 x15 x16 x17 x18
## [1,] 72 34 61 122 47 111 162 10 143  88  84 168 161   1  38 130  67 157
## [2,] 72 34 61 122 47 111 162 10 143  88  84 168 161   1  38 130  67 157
## [3,] 72 34 61 122 47 111 162 10 143  88  84 168 161   1  38 130  67 157
## [4,] 72 34 61 122 47 111 162 10 143  88  84 168 161   1  38 130  67 157
## [5,] 72 34 61 122 47 111 162 10 143  88  84 168 161   1  38 130  67 157
##       x19 x20 x21 x22 x23 x24 x25 x26 x27 x28 x29 x30 x31 x32 x33 x34 x35
## [1,] 125  58 144  19 185  39 110  32   8 182 191  11  33  26 138 187  46
## [2,] 125  58 144  19 185  39 110  32   8 182 191  11  33  26 138 187  46
## [3,] 125  58 144  19 185  39 110  32   8 182 191  11  33  26 138 187  46
## [4,] 125  58 144  19 185  39 110  32   8 182 191  11  33  26 138 187  46
## [5,] 125  58 144  19 185  39 110  32   8 182 191  11  33  26 138 187  46
##       x36 x37 x38 x39 x40 x41 x42 x43 x44 x45 x46 x47 x48 x49 x50 x51 x52
## [1,] 159  62 127 189 172  75 107 177  20 164   6   7 169 145   5 123  37
## [2,] 159  62 172  75 107 177 147  20 164 153 109  82   6   7 123   5 145
## [3,] 159  62 172  75 107 177 147  20 164 153 109  82   6   7 123   5 145
## [4,] 159  62 172  75 107 177 147  20 164 153 109  82   6   7 123   5 145
## [5,] 159  62 172  75 107 177  20 164   6   7 169 145   5 123  37  15 186
##       x53 x54 x55 x56 x57 x58 x59 x60 x61 x62 x63 x64 x65 x66 x67 x68 x69
## [1,]  15 186  87  36 115 146   9   3  95 178  71  78  98  68 119  56 152
## [2,] 169  53  21  64  37   4  15 121 128 186 100  87  36 115 146   9   3
## [3,] 169  53  37   4  15 121 128 186 100  87  36 115 146   9   3  95 158
## [4,] 169  53  21  64  37   4  15 121 128 186 100  87  36 115 146   9   3
## [5,]  87  36 115 146   9   3  95 178  71  78  98  68 119  56 152  73   2
```

```
##      x70 x71 x72 x73 x74 x75 x76 x77 x78 x79 x80 x81 x82 x83 x84 x85 x86
## [1,]  73   2 190  83  12 116 124 173  45  54  18 137 104 194 174  70 136
## [2,]  95 158  91 178 183  71  78 152  56 119  68  98  73   2 190  83  12
## [3,]  91 178 183  71  78 152  56  64  21 119  68  98  73   2 190  83  12
## [4,]  95 158  91 178 183  71  78 152  56 119  68  98  73   2 190  83  12
## [5,] 190  83  12 116 124 173  45  54  18 137 104 194 174  70 136  48 195
##      x87 x88 x89 x90 x91 x92 x93 x94 x95 x96 x97 x98 x99 x100 x101 x102
## [1,]  48 195 151  92  42  22 193 135  17 160  63 165 139   50  101  120
## [2,] 116 124 173  45  54  18 137 104 194 174  70 136  48  195  151   92
## [3,] 116 124 173  45  54  18 137 104 194 174  70 136  48  195  151   92
## [4,] 116 124 173  45  54  18 137 104 194 174  70 136  48  195  151   92
## [5,] 151  92  42  22 193 135  17 160  63 165 139  50 101  120  197  105
##      x103 x104 x105 x106 x107 x108 x109 x110 x111 x112 x113 x114 x115 x116
## [1,]  197  105   49   24  163   81   30   97   35   60   86   29  184   25
## [2,]   42   22  193  135   17  160  149   93   55   80   41  192  188   25
## [3,]   42   22  193  135   17  160  149   93   55   80   41  192  188   25
## [4,]   42   22  193  135   17  160  149   93   55   80   41  192  188   25
## [5,]   49   24  163   81   30   97   35   60   86   29  184   25  188   80
##      x117 x118 x119 x120 x121 x122 x123 x124 x125 x126 x127 x128 x129 x130
## [1,]  188  142   51  133   79   65  112   89  118  106  155  103  176  102
## [2,]  184   29   86   60   35   97   30   81  163   24   49  105  197  120
## [3,]  184   29   86   60   35   97   30   81  163   24   49  105  197  120
## [4,]  184   29   86   60   35   97   30   81  163   24   49  105  197  120
## [5,]   55   93  171   96  134  114   40   23  132   59  167  150  192   41
##      x131 x132 x133 x134 x135 x136 x137 x138 x139 x140 x141 x142 x143 x144
## [1,]   69   76   80   55   93  171   96  134  114   40   23  132   59  167
## [2,]  101   50  139  165   63  171   96  134  114   40   23  132  150  141
## [3,]  101   50  139  165   63  171   96  134  114   40   23  132   59  167
## [4,]  101   50  139  165   63  171   96  134  114   40   23  132   59  167
## [5,]  166   43  170  181   94  117   57   27  180   16  147  153  109   82
##      x145 x146 x147 x148 x149 x150 x151 x152 x153 x154 x155 x156 x157 x158
## [1,]  150   66  141  156  108  154   52  192   41   31   77   13   28   85
## [2,]  156  108  154   52  142   51  133   79   65  112   89  118  106  155
## [3,]  150   66  141  156  108  154   52  142   51  133   79   65  112   89
## [4,]  150   66  141  156  108  154   52  142   51  133   79   65  112   89
## [5,]   53   66  141  156  108  154   52  142   51  133   79   65  112   89
##      x159 x160 x161 x162 x163 x164 x165 x166 x167 x168 x169 x170 x171 x172
## [1,]  148  113  131  166   43  170  181   94  117   57   27  180   16  147
## [2,]  103  176  102  189  127   69   76   31   77   13   66  167   59   28
## [3,]  118  106  155  103  176  102  189  127   69   76   31   77   13   28
## [4,]  118  106  155  103  176  102  189  127   69   76   31   77   13   28
## [5,]  118  106  155  103  176  102  189  127   69   76   31   77   13   28
##      x173 x174 x175 x176 x177 x178 x179 x180 x181 x182 x183 x184 x185 x186
## [1,]  153  109   82   53   21   64    4  149  121  128  100  158   91  183
## [2,]   85  148  113  131  166   43  170  181   94  117   57   27  180   16
## [3,]   85  148  113  131  166   43  170  181   94  117   57   27  180   16
## [4,]   85  148  113  131  166   43  170  181   94  117   57   27  180   16
## [5,]   85  148  113  131   21   64    4  149  121  128  100  158   91  183
##      x187 x188 x189 x190 x191 x192 x193 x194 x195 x196 x197
## [1,]  196  140  129   74   14   99  175  126   90  179   44
## [2,]  196  140  129   74   14   99  175  126   90  179   44
## [3,]  196  140  129   74   14   99  175  126   90  179   44
## [4,]  196  140  129   74   14   99  175  126   90  179   44
## [5,]  196  140  129   74   14   99  175  126   90  179   44
```

```r
1/GA2@fitnessValue
```

```
## [1] 23.24987
```