Week 9 Homework - Solutions

Jeff Baggett

11/07/2016

## Problem 1 - Redistricting with a Genetic Algorithm

Solve problem 13.10-6. Install the 'gramEvol' package to get access to a genetic algorithm that uses integer encoding called GeneticAlg.int(). Use that algorithm to solve this problem. You'll have to read the documentation to figure out how to use the algorithm.

In the block below add your code to use the genetic algorithm. Either experiment with different random number seeds or use a for loop to conduct the optimization many times to find the best solution you can.

# Solution  
set.seed(108) # this seed is for a good solution   
library('gramEvol')  
GA <- GeneticAlg.int(18,1,10,allowrepeat = TRUE,popSize=50,iterations=200,evalFunc=demrepFit) # find minimum  
df <- distAssign(GA$best$genome);  
df

## Dem Rep Tot Win  
## 1 100 124 224 TRUE  
## 2 75 83 158 TRUE  
## 3 158 165 323 TRUE  
## 4 226 111 337 FALSE  
## 5 138 150 288 TRUE  
## 6 109 112 221 TRUE  
## 7 160 167 327 TRUE  
## 8 72 98 170 TRUE  
## 9 145 162 307 TRUE  
## 10 136 149 285 TRUE

numRepDist = sum(df$Win);  
numRepDist

## [1] 9

GA$best$genome

## [1] 4 6 2 1 9 10 7 4 10 1 9 3 3 6 7 5 5 8

Make sure you print out your best solution. How many districts do Republicans win with your solution? (I don't think Republicans can win all 10 in this example, but they can get close.)

Solution: This solution shows it is possible for Republicans to win 9 out of 10 districts. I haven't been able to find a better solution. Notice that the district numbers are arbitrary so there are many equivalent solutions.

## Problem 2 - TSP with a Genetic Algorithm

Use the ga() function with permutation encoding from the 'GA' package to approximate a solution to this 48 city TSP problem. Try different random number seeds and report the best result you can find. Copy the code from saTSP.R to make a graph of the tour. Your fitness function is tspFitness().

## [1] 2.980537e-05

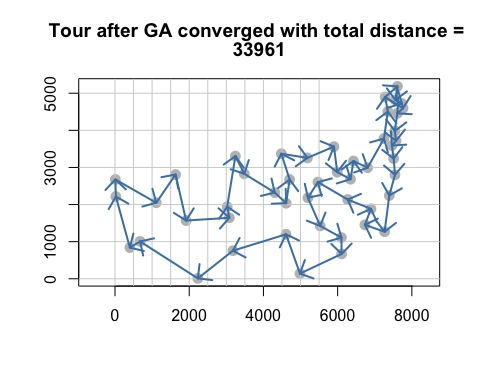
## Loading required package: GA

## Loading required package: foreach

## Loading required package: iterators

## Package 'GA' version 3.0.2

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



You may have to play with the optimization parameters a bit, the best solution I was able to find had total "distance" of 33,961. The genetic algorithm does a really nice job with this problem.

## Problem 3 - TSP with Simulated Annealing

Modify the code in saTSP.R and include it below to approximate an optimal tour for the 48 city TSP problem in Problem 2. Include a graph of the best tour you are able to find.

distmat <- as.matrix(read.table("att48\_d.txt"))  
loc <- as.matrix(read.table("att48\_xy.txt"))  
  
distance <- function(sq) { # Target function  
 sq2 <- embed(sq, 2)  
 sum(distmat[cbind(sq2[,2], sq2[,1])])  
}  
  
genseq <- function(sq) { # Generate new candidate sequence  
 idx <- seq(2, NROW(distmat)-1)  
 changepoints <- sample(idx, size = 2, replace = FALSE)  
 tmp <- sq[changepoints[1]]  
 sq[changepoints[1]] <- sq[changepoints[2]]  
 sq[changepoints[2]] <- tmp  
 sq  
}  
  
sq <- c(1:nrow(distmat), 1) # Initial sequence: alphabetic  
distance(sq)

## [1] 157553

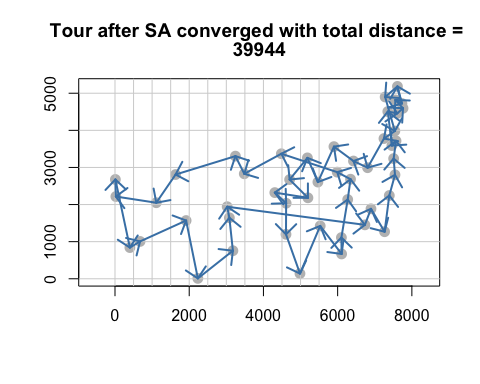
x <- loc[,1]; y <- loc[,2]  
s <- seq\_len(nrow(distmat))  
tspinit <- loc[sq,]  
  
set.seed(123) # first one I found that make total dist < 40000  
res <- optim(sq, distance, genseq, method = "SANN",  
 control = list(maxit = 500000, temp = 5000,   
 tmax = 10, trace = TRUE,  
 REPORT = 10000))

## sann objective function values  
## initial value 157553.000000  
## iter 100000 value 43589.000000  
## iter 200000 value 43589.000000  
## iter 300000 value 43278.000000  
## iter 400000 value 40811.000000  
## iter 499999 value 39944.000000  
## final value 39944.000000  
## sann stopped after 499999 iterations

totaldist <- res$value  
totaldist

## [1] 39944

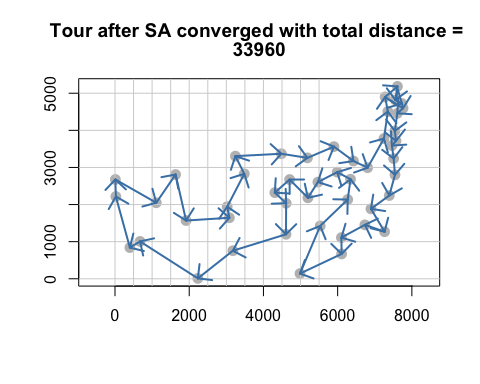
str = c("Tour after SA converged with total distance = ",as.character(totaldist));  
plot(x, y, type = "n", asp = 1, xlab = "", ylab = "", main = str);  
points(x, y, pch = 16, cex = 1.5, col = "grey")  
abline(h = pretty(range(x), 10), v = pretty(range(y), 10), col = "lightgrey")  
  
tour <- res$par  
n <- length(tour)  
arrows(x[tour[-n]], y[tour[-n]],   
 x[tour[-1]], y[tour[-1]],   
 length = 0.15, angle = 45,   
 col = "steelblue", lwd = 2)



Solution: SA does not work nearly as well as the genetic algorithm on this problem. GA usually finds a solution with total distance less than 36000, but SA has a hard time getting below 40000. Looking at the picture of the SA solution, there are clearly moves that should be made, but SA doesn't see them or doesn't make them. The performance of SA depends critically on the temperature schedule and I was never able to get the optim() version of SA to work well.

Here is a version of SA for the TSP problem where I was able to get more control over the temperature schedule. It has performance similar to GA now.

distmat <- as.matrix(read.table("att48\_d.txt"))  
loc <- as.matrix(read.table("att48\_xy.txt"))  
  
calculate\_tour\_distance = function(tour, distance\_matrix) {  
 sum(distance\_matrix[embed(c(tour, tour[1]), 2)])  
}  
  
current\_temperature = function(iter, s\_curve\_amplitude, s\_curve\_center, s\_curve\_width) {  
 s\_curve\_amplitude \* s\_curve(iter, s\_curve\_center, s\_curve\_width)  
}  
  
s\_curve = function(x, center, width) {  
 1 / (1 + exp((x - center) / width))  
}  
  
run\_annealing\_process = function(distance\_matrix, tour, number\_of\_iterations,  
 s\_curve\_amplitude, s\_curve\_center, s\_curve\_width) {  
 n\_cities = nrow(distance\_matrix);  
 best\_distance = calculate\_tour\_distance(tour, distance\_matrix)  
 tour\_distance = best\_distance  
   
 for(iter in 1:number\_of\_iterations) {  
 temp = current\_temperature(iter, s\_curve\_amplitude, s\_curve\_center, s\_curve\_width)  
   
 candidate\_tour = tour  
 swap = sample(n\_cities, 2)  
 candidate\_tour[swap[1]:swap[2]] = rev(candidate\_tour[swap[1]:swap[2]])  
 candidate\_dist = calculate\_tour\_distance(candidate\_tour, distance\_matrix)  
   
 if (temp > 0) {  
 ratio = exp((tour\_distance - candidate\_dist) / temp)  
 } else {  
 ratio = as.numeric(candidate\_dist < tour\_distance)  
 }  
   
 if (runif(1) < ratio) {  
 tour = candidate\_tour  
 tour\_distance = candidate\_dist  
   
 if (tour\_distance < best\_distance) {  
 best\_tour = tour  
 best\_distance = tour\_distance  
 }  
 }  
 }  
   
 return(list(tour=tour, tour\_distance=tour\_distance, best\_tour=best\_tour, best\_distance=best\_distance))  
}  
  
s\_curve\_amplitude = 4000;  
s\_curve\_center = 0;  
s\_curve\_width = 3000;  
number\_of\_iterations = 100000;  
set.seed(111); # 111 produces the same tour as GA did  
tour = sample(nrow(distmat)); # generate initial tour  
  
result <- run\_annealing\_process(distmat, tour, number\_of\_iterations,  
 s\_curve\_amplitude, s\_curve\_center, s\_curve\_width)   
  
str = c("Tour after SA converged with total distance = ",as.character(result$best\_distance));  
plot(x, y, type = "n", asp = 1, xlab = "", ylab = "", main = str);  
points(x, y, pch = 16, cex = 1.5, col = "grey")  
abline(h = pretty(range(x), 10), v = pretty(range(y), 10), col = "lightgrey")  
  
tour <- result$best\_tour  
tour <- c(tour,tour[1])  
n <- length(tour)  
arrows(x[tour[-n]], y[tour[-n]],   
 x[tour[-1]], y[tour[-1]],   
 length = 0.15, angle = 45,   
 col = "steelblue", lwd = 2)



## Problem 4 - Comparing Algorithms for a 30 dimensional Rastrigin function

The 30 dimensional Rastrigin function is considered very difficult to optimize and is a test case for many optimization algorithms. We know that the global minimum value of 0 occurs at the origin. For this problem you should compare the performance of Naive Multistart, the Genetic Algorithm plus local search, and the Simulated Annealing algorithm GenSA() from the 'GenSA' package. If you can get it to work, then also try the mlsl() function in the 'nloptr' package as it should work considerably better than Naive Multistart.

This is a somewhat open problem, but at the very least you should try each algorithm multiple times (possibly in for loop) and report on which algorithms are most efficient (fewest function calls) and which are most reliable (able to consistently identify the global minimum). Experiment with the algorithm parameters (population size, number of iterations of local search, etc.) You'll likely have to increase population sizes and the maximum number of iterations to successfully solve the 30 dimensional problem. Look at the source code in the presentation .Rmd file included in the download packet for guidance in setting up your algorithms.

# your code goes in this block  
dimension = 30;  
lower = rep(-5.12,dimension); upper = rep(5.12,dimension);   
  
## Naive Multistart  
## won't work well unless you increase the number of searches  
bestmin <- 100000; numlocalSearch = 5000;   
numevals = 0;  
set.seed(126)  
for (j in 1:numlocalSearch){  
 x0 <- as.vector(runif(dimension,min=-5.12,max=5.12));  
 result <- optim(x0,Rastrigin,method="L-BFGS-B",lower=lower,upper=upper);  
 numevals = numevals + result$counts[1]  
 if (result$value<bestmin){ bestmin = result$value; bestx = result$par}  
}  
  
  
bestmin

## [1] 78.60158

# even with 10000 local searches the method doesn't work all that well  
# and required a ton of function evaluations  
numevals

## function   
## 94975

## MLSL - you didn't have to include this  
# we will use the same number of function evaluations is in Naive multistart (94975)  
require(nloptr);

## Loading required package: nloptr

set.seed(126)  
x0 <- as.vector(runif(dimension,min=-5.12,max=5.12));  
result <- mlsl(x0,Rastrigin,lower=lower,upper=upper,nl.info=TRUE,  
 control=list(maxeval = 94975))

##   
## Call:  
##   
## nloptr(x0 = x0, eval\_f = fn, eval\_grad\_f = gr, lb = lower, ub = upper,   
## opts = opts)  
##   
##   
## Minimization using NLopt version 2.4.2   
##   
## NLopt solver status: 5 ( NLOPT\_MAXEVAL\_REACHED: Optimization stopped   
## because maxeval (above) was reached. )  
##   
## Number of Iterations....: 94975   
## Termination conditions: stopval: -Inf xtol\_rel: 1e-06 maxeval: 94975 ftol\_rel: 0 ftol\_abs: 0   
## Number of inequality constraints: 0   
## Number of equality constraints: 0   
## Current value of objective function: 24.8737229345422   
## Current value of controls: -6.342811e-12 1.15723e-12 1.377861e-13 6.604215e-12 -1.169869e-12   
## -2.756872e-12 7.995475e-13 -6.832385e-12 -4.974691 -1.297653e-12   
## 5.670719e-12 -7.79424e-12 -8.312829e-13 -8.342736e-13 -8.326644e-12   
## 7.871368e-12 -8.894805e-13 4.960131e-12 -4.74476e-12 6.234635e-12   
## -4.758566e-12 -1.070167e-12 -2.789505e-12 -5.027405e-12 3.501727e-13   
## -3.531167e-12 3.214682e-12 -5.126481e-12 9.292991e-13 -1.467116e-12

## finds a minimum value of 24.9, not great better than naive approach  
  
  
## Genetic Algorithm with local search  
# note ga() maximizes so we'll have to redefine Rastrigin with a negative  
Rastrigin2 <- function(x) {  
 -(sum(x^2 - 10 \* cos(2 \* pi \* x)) + 10 \* length(x))  
}  
set.seed(123)  
result = ga(type="real-valued",fitness=Rastrigin2,  
 popSize = 50,  
 min=lower,max=upper,maxiter=1300,optim=TRUE);  
round(result@solution,5)

## x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17 x18 x19  
## [1,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## x20 x21 x22 x23 x24 x25 x26 x27 x28 x29 x30  
## [1,] 0 0 0 0 0 0 0 0 0 0 0

result@fitnessValue

## [1] 0

# 1300 or so iterations and ga with local search has identified the global min  
# if we estimate 30 or so function evaluations for each local search then  
# this is about 40000 function calls to find the global min. ... better than mlsl  
  
# Simulated Annealing  
  
library(GenSA)  
set.seed(1234) # The user can use any seed.  
out <- GenSA(lower = lower, upper = upper, fn = Rastrigin,  
 control=list(verbose=TRUE))

## Initializing par with random data inside bounds  
## It: 1, obj value: 198.9905811  
## It: 10, obj value: 100.4904555  
## It: 13, obj value: 55.71761643  
## It: 16, obj value: 45.76805613  
## It: 17, obj value: 43.77812285  
## It: 18, obj value: 38.8033528  
## It: 19, obj value: 30.84371062  
## It: 21, obj value: 29.84875156  
## It: 22, obj value: 26.86387943  
## It: 24, obj value: 23.87900729  
## It: 31, obj value: 21.88909422  
## It: 33, obj value: 17.90926303  
## It: 35, obj value: 16.91430397  
## It: 37, obj value: 15.91934491  
## It: 38, obj value: 14.92438586  
## It: 44, obj value: 12.93446774  
## It: 53, obj value: 11.93950869  
## It: 66, obj value: 9.949590571  
## It: 72, obj value: 8.954631514  
## It: 95, obj value: 6.9647134  
## It: 96, obj value: 5.969754343  
## It: 102, obj value: 4.974795285  
## It: 157, obj value: 3.979836228  
## It: 165, obj value: 1.989918114  
## It: 168, obj value: 0.9949590571  
## It: 171, obj value: 0  
## ................................................................................................................................

out[c("value","par","counts")]

## $value  
## [1] 0  
##   
## $par  
## [1] 2.712917e-11 -2.960680e-12 1.582701e-13 5.669022e-12 -1.235440e-11  
## [6] -2.861435e-12 -3.094681e-11 -1.252898e-11 6.343722e-12 -3.320250e-11  
## [11] -4.005316e-12 8.829808e-12 1.656851e-13 1.360880e-12 -1.455698e-11  
## [16] -3.650985e-12 -1.451763e-11 1.682576e-09 3.106473e-12 1.036181e-13  
## [21] -9.015259e-12 -3.211081e-11 3.785124e-12 -2.053973e-12 4.886316e-10  
## [26] 3.208469e-12 -8.708271e-12 3.157677e-11 -7.472686e-12 2.107131e-11  
##   
## $counts  
## [1] 370460

Discuss your algorithm comparison here:

We should probably compare the number of function evaluations for each of the methods, but we will just give some observations. Naive multistart doesn't work that well for this problem. Using about 100,000 function evaluations we did not get very close to the globabl minimum value. MLSL works better, but fails to find the global minimum as well.

The GA algorithm with local search found the global min with about 40000 evaluations. SA found it with about 370000 evaluations, but that number could probably be decreased by experimenting with parameters.