Week08_Assignment

John Reddy Peasari

4/1/2021

Brute Force and GA for the fraction 0.00001

```
## Load the population data
pop.data <- data.table::fread("Mo_pop_Sim.csv")
#str(pop.data)
frac <- 0.00001
small.data <- pop.data[sample(1:nrow(pop.data),
                              size = round(nrow(pop.data) * frac),
                              replace = F), ## extract a sample of randomlychosen 1% rows
                       ]  ## and choose all columns
#head(small.data)
```

```
## Load the FQHC data
data_path <- 'Uploads-week-8'
fqhc.data <- data.table(as.data.frame(readOGR(data_path,
                    'MO_2018_Federally_Qualified_Health_Center_Locations')))
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "D:\Spring 2021\HPC\Modules\week08\Uploads-week-8", layer: "MO_2018_Federally_Qualified_Heal
## with 197 features
## It has 12 fields
## Integer64 fields read as strings:  OBJECTID
```

```
#str(fqhc.data)
## Select a subset of 4 rows, drawn at random, and cols: OBJECTID, Long, Lat
set.seed(8888)
no.ctrs = 8
sites.dt <- fqhc.data[sample(1:nrow(fqhc.data), no.ctrs, replace = F),
                      list(OBJECTID = as.character(OBJECTID),
                           Longitude,
                           Latitude)]
#head(sites.dt)
```

```
## Create combinations of residences and centers
small.data <- cbind(small.data, resid = c(1:nrow(small.data)))
setkey(small.data, resid)
#head(small.data)
```

```
dist.dt <- data.table(CJ(as.character(small.data$resid),
                         as.character(sites.dt$OBJECTID)))
names(dist.dt)
```

```
## [1] "V1" "V2"
```

```
setnames(dist.dt, c("V1", "V2"), c("resid", "OBJECTID"))
#str(dist.dt)
#head(dist.dt)
```

```
## Compute distances on a small subset of values - this takes a long time
# if run on all rows in the dataset
v <- map_dbl(1:nrow(dist.dt),
        function(rid){
          r.rsd <- small.data[resid == dist.dt[rid,resid]]
          r.fqhc <- fqhc.data[OBJECTID == as.character(dist.dt[rid,OBJECTID])]

          distm(c(r.rsd$long, r.rsd$lat),
                c(r.fqhc$Longitude, r.fqhc$Latitude),
                fun = distHaversine)

          ## note that the above distance is in meters
          ## convert it to miles
        })

dist.dt[, distance := v]
head(dist.dt)
```

```
##    resid OBJECTID  distance
## 1:     1      119 518571.07
## 2:     1       12 268383.72
## 3:     1      132  69366.34
## 4:     1      193 182951.43
## 5:     1       60  43348.54
## 6:     1       64 192500.59
```

```
## Next, use the distance information to make decisions on, let's say top 2 centers
## The function combn() produces all possible combinations of elements in x, taking m at a time
## If we want to take all possible combinations of 2 items, out of a set of 4...
#combn(4, 2)
## the combinations are in columns; a simpler-to-read format is to show them in rows
#t(combn(x=1:4,m = 2)) ## transpose the results using t()

## If we want to identify the two best centers, out of four, for providing the extended services
## we need identify the two best ones based on, say, the total average distance between each of them
## and each residence in our sample.
#unique(sites.dt$OBJECTID)
## Build combinations
combinations <- data.table(as.data.frame(t(combn(unique(sites.dt$OBJECTID), 2))))
names(combinations) <- c("loc1", "loc2")
##
meandists <- dist.dt[, mean(distance), by=OBJECTID]
```

```
ind = 1

get_combined_distance <- function(loc1, loc2, ctr.id){

}
map_dbl(1:nrow(combinations),
        function(ind){
          mean(c(meandists[meandists$OBJECTID == combinations[ind,loc1]]$V1,
                 meandists[meandists$OBJECTID == combinations[ind,loc2]]$V1))
        })
```

```
##  [1] 258696.1 286639.8 237045.1 297692.8 260008.2 240472.8 239035.4 249515.2
##  [9] 199920.6 260568.3 222883.7 203348.3 201910.8 227864.3 288512.0 250827.3
## [17] 231291.9 229854.5 238917.3 201232.7 181697.3 180259.9 261880.4 242345.0
## [25] 240907.6 204660.4 203223.0 183687.5
```

```
combinations[, meandist :=
               map_dbl(1:nrow(combinations),
                       function(ind){
                         mean(c(meandists[meandists$OBJECTID == combinations[ind,loc1]]$V1,
                                meandists[meandists$OBJECTID == combinations[ind,loc2]]$V1))
                       })]
## Find the best one - this is the brute-force approach
combinations[meandist == min(combinations$meandist)]
```

```
##    loc1 loc2 meandist
## 1:  193   12 180259.9
```

```
p <- rep(1, length(unique(dist.dt$OBJECTID)))
w <- dist.dt[,mean(distance), by = OBJECTID]$V1
W <- max(w) * 1.14
knapsack <- function(x){
  f <- sum(x*p)
  penalty <- sum(w) * abs(sum(x*w) - max(W))
  (f - penalty)
}
gamodel <- ga(type = "binary",
              fitness = knapsack,
              nBits = length(w))


## Brute-force approach
combinations[meandist == min(combinations$meandist)]
```

```
##    loc1 loc2 meandist
## 1:  193   12 180259.9
```

```
## Genetic algorithm based approach
summary(gamodel)
```

```
## -- Genetic Algorithm -------------------
```

```
##
## GA settings:
## Type                  =  binary
## Population size       =  50
## Number of generations =  100
## Elitism               =  2
## Crossover probability =  0.8
## Mutation probability  =  0.1
##
## GA results:
## Iterations             = 100
## Fitness function value = -3.545e+10
## Solution =
##      x1 x2 x3 x4 x5 x6 x7 x8
## [1,]  0  1  0  1  0  0  0  0
```

```
unique(dist.dt$OBJECTID)
```

```
## [1] "119" "12"  "132" "193" "60"  "64"  "67"  "95"
```

Brute Force and GA for the fraction 0.0001

```
## Load the population data
pop.data <- data.table::fread("Mo_pop_Sim.csv")
#str(pop.data)
frac <- 0.0001
small.data <- pop.data[sample(1:nrow(pop.data),
                              size = round(nrow(pop.data) * frac),
                              replace = F), ## extract a sample of randomlychosen 1% rows
                       ]  ## and choose all columns
#head(small.data)
```

```
## Load the FQHC data
data_path <- 'Uploads-week-8'
fqhc.data <- data.table(as.data.frame(readOGR(data_path,
                     'MO_2018_Federally_Qualified_Health_Center_Locations')))
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "D:\Spring 2021\HPC\Modules\week08\Uploads-week-8", layer: "MO_2018_Federally_Qualified_Heal
## with 197 features
## It has 12 fields
## Integer64 fields read as strings:  OBJECTID
```

```
#str(fqhc.data)
## Select a subset of 4 rows, drawn at random, and cols: OBJECTID, Long, Lat
set.seed(8888)
no.ctrs = 8
sites.dt <- fqhc.data[sample(1:nrow(fqhc.data), no.ctrs, replace = F),
                      list(OBJECTID = as.character(OBJECTID),
                           Longitude,
```

```
                            Latitude)]
#head(sites.dt)
```

```
## Create combinations of residences and centers
small.data <- cbind(small.data, resid = c(1:nrow(small.data)))
setkey(small.data, resid)
#head(small.data)

dist.dt <- data.table(CJ(as.character(small.data$resid),
                         as.character(sites.dt$OBJECTID)))
names(dist.dt)
```

```
## [1] "V1" "V2"
```

```
setnames(dist.dt, c("V1", "V2"), c("resid", "OBJECTID"))
#str(dist.dt)
#head(dist.dt)
```

```
## Compute distances on a small subset of values - this takes a long time
# if run on all rows in the dataset
v <- map_dbl(1:nrow(dist.dt),
        function(rid){
          r.rsd <- small.data[resid == dist.dt[rid,resid]]
          r.fqhc <- fqhc.data[OBJECTID == as.character(dist.dt[rid,OBJECTID])]

          distm(c(r.rsd$long, r.rsd$lat),
                c(r.fqhc$Longitude, r.fqhc$Latitude),
                fun = distHaversine)

          ## note that the above distance is in meters
          ## convert it to miles
        })

dist.dt[, distance := v]
head(dist.dt)
```

```
##    resid OBJECTID distance
## 1:     1      119 328265.6
## 2:     1       12 258872.5
## 3:     1      132 370846.6
## 4:     1      193 203777.8
## 5:     1       60 338876.5
## 6:     1       64 353118.7
```

```
## Next, use the distance information to make decisions on, let's say top 2 centers
## The function combn() produces all possible combinations of elements in x, taking m at a time
## If we want to take all possible combinations of 2 items, out of a set of 4...
#combn(4, 2)
## the combinations are in columns; a simpler-to-read format is to show them in rows
#t(combn(x=1:4,m = 2)) ## transpose the results using t()

## If we want to identify the two best centers, out of four, for providing the extended services
```

```
## we need identify the two best ones based on, say, the total average distance between each of them
## and each residence in our sample.
#unique(sites.dt$OBJECTID)
## Build combinations
combinations <- data.table(as.data.frame(t(combn(unique(sites.dt$OBJECTID), 2))))
names(combinations) <- c("loc1", "loc2")
##
meandists <- dist.dt[, mean(distance), by=OBJECTID]
ind = 1

get_combined_distance <- function(loc1, loc2, ctr.id){

}
map_dbl(1:nrow(combinations),
        function(ind){
          mean(c(meandists[meandists$OBJECTID == combinations[ind,loc1]]$V1,
                 meandists[meandists$OBJECTID == combinations[ind,loc2]]$V1))
        })
```

```
##  [1] 268964.0 289110.3 241011.9 299422.3 249300.1 243256.8 239701.6 251845.8
##  [9] 203747.4 262157.8 212035.6 205992.3 202437.1 223893.7 282304.1 232181.9
## [17] 226138.6 222583.4 234205.7 184083.5 178040.2 174485.0 242494.0 236450.6
## [25] 232895.4 186328.4 182773.2 176729.9
```

```
combinations[, meandist :=
               map_dbl(1:nrow(combinations),
                       function(ind){
                         mean(c(meandists[meandists$OBJECTID == combinations[ind,loc1]]$V1,
                                meandists[meandists$OBJECTID == combinations[ind,loc2]]$V1))
                       })]
## Find the best one - this is the brute-force approach
combinations[meandist == min(combinations$meandist)]
```

```
##    loc1 loc2 meandist
## 1:  193   12   174485
```

```
p <- rep(1, length(unique(dist.dt$OBJECTID)))
w <- dist.dt[,mean(distance), by = OBJECTID]$V1
W <- max(w) * 1.14
knapsack <- function(x){
  f <- sum(x*p)
  penalty <- sum(w) * abs(sum(x*w) - max(W))
  (f - penalty)
}
gamodel <- ga(type = "binary",
              fitness = knapsack,
              nBits = length(w))


## Brute-force approach
combinations[meandist == min(combinations$meandist)]
```

```
##    loc1 loc2 meandist
```

```
## 1:  193   12   174485
```

```
## Genetic algorithm based approach
summary(gamodel)
```

```
## -- Genetic Algorithm -------------------
##
## GA settings:
## Type                  =  binary
## Population size       =  50
## Number of generations =  100
## Elitism               =  2
## Crossover probability =  0.8
## Mutation probability  =  0.1
##
## GA results:
## Iterations             = 100
## Fitness function value = -237983775
## Solution =
##      x1 x2 x3 x4 x5 x6 x7 x8
## [1,]  0  1  0  1  0  0  0  0
```

```
unique(dist.dt$OBJECTID)
```

```
## [1] "119" "12"  "132" "193" "60"  "64"  "67"  "95"
```

Brute Force and GA for the fraction 0.0001

```
## Load the population data
pop.data <- data.table::fread("Mo_pop_Sim.csv")
#str(pop.data)
frac <- 0.001
small.data <- pop.data[sample(1:nrow(pop.data),
                              size = round(nrow(pop.data) * frac),
                              replace = F), ## extract a sample of randomlychosen 1% rows
                       ]  ## and choose all columns
#head(small.data)
```

```
## Load the FQHC data
data_path <- 'Uploads-week-8'
fqhc.data <- data.table(as.data.frame(readOGR(data_path,
                    'MO_2018_Federally_Qualified_Health_Center_Locations')))
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "D:\Spring 2021\HPC\Modules\week08\Uploads-week-8", layer: "MO_2018_Federally_Qualified_Heal
## with 197 features
## It has 12 fields
## Integer64 fields read as strings:  OBJECTID
```

```
#str(fqhc.data)
## Select a subset of 4 rows, drawn at random, and cols: OBJECTID, Long, Lat
set.seed(8888)
no.ctrs = 8
sites.dt <- fqhc.data[sample(1:nrow(fqhc.data), no.ctrs, replace = F),
                      list(OBJECTID = as.character(OBJECTID),
                           Longitude,
                           Latitude)]
#head(sites.dt)
```

```
## Create combinations of residences and centers
small.data <- cbind(small.data, resid = c(1:nrow(small.data)))
setkey(small.data, resid)
#head(small.data)

dist.dt <- data.table(CJ(as.character(small.data$resid),
                         as.character(sites.dt$OBJECTID)))
names(dist.dt)
```

```
## [1] "V1" "V2"
```

```
setnames(dist.dt, c("V1", "V2"), c("resid", "OBJECTID"))
#str(dist.dt)
#head(dist.dt)
```

```
## Compute distances on a small subset of values - this takes a long time
# if run on all rows in the dataset
v <- map_dbl(1:nrow(dist.dt),
        function(rid){
          r.rsd <- small.data[resid == dist.dt[rid,resid]]
          r.fqhc <- fqhc.data[OBJECTID == as.character(dist.dt[rid,OBJECTID])]

          distm(c(r.rsd$long, r.rsd$lat),
                c(r.fqhc$Longitude, r.fqhc$Latitude),
                fun = distHaversine)

          ## note that the above distance is in meters
          ## convert it to miles
        })

dist.dt[, distance := v]
head(dist.dt)
```

```
##    resid OBJECTID distance
## 1:     1      119 328265.6
## 2:     1       12 258872.5
## 3:     1      132 370846.6
## 4:     1      193 203777.8
## 5:     1       60 338876.5
## 6:     1       64 353118.7
```

```
## Next, use the distance information to make decisions on, let's say top 2 centers
## The function combn() produces all possible combinations of elements in x, taking m at a time
## If we want to take all possible combinations of 2 items, out of a set of 4...
#combn(4, 2)
## the combinations are in columns; a simpler-to-read format is to show them in rows
#t(combn(x=1:4,m = 2)) ## transpose the results using t()

## If we want to identify the two best centers, out of four, for providing the extended services
## we need identify the two best ones based on, say, the total average distance between each of them
## and each residence in our sample.
#unique(sites.dt$OBJECTID)
## Build combinations
combinations <- data.table(as.data.frame(t(combn(unique(sites.dt$OBJECTID), 2))))
names(combinations) <- c("loc1", "loc2")
##
meandists <- dist.dt[, mean(distance), by=OBJECTID]
ind = 1

get_combined_distance <- function(loc1, loc2, ctr.id){

}
map_dbl(1:nrow(combinations),
        function(ind){
          mean(c(meandists[meandists$OBJECTID == combinations[ind,loc1]]$V1,
                 meandists[meandists$OBJECTID == combinations[ind,loc2]]$V1))
        })
```

```
##  [1] 265069.7 287726.9 237998.6 298294.7 248448.2 238614.9 235919.1 254567.0
##  [9] 204838.7 265134.8 215288.3 205455.0 202759.2 227495.9 287792.0 237945.5
## [17] 228112.2 225416.4 238063.6 188217.2 178383.9 175688.1 248513.2 238680.0
## [25] 235984.1 188833.5 186137.7 176304.4
```

```
combinations[, meandist :=
               map_dbl(1:nrow(combinations),
                       function(ind){
                         mean(c(meandists[meandists$OBJECTID == combinations[ind,loc1]]$V1,
                                meandists[meandists$OBJECTID == combinations[ind,loc2]]$V1))
                       })]
## Find the best one - this is the brute-force approach
combinations[meandist == min(combinations$meandist)]
```

```
##    loc1 loc2 meandist
## 1:  193   12 175688.1
```

```
p <- rep(1, length(unique(dist.dt$OBJECTID)))
w <- dist.dt[,mean(distance), by = OBJECTID]$V1
W <- max(w) * 1.14
knapsack <- function(x){
  f <- sum(x*p)
  penalty <- sum(w) * abs(sum(x*w) - max(W))
  (f - penalty)
}
```

```
gamodel <- ga(type = "binary",
              fitness = knapsack,
              nBits = length(w))


## Brute-force approach
combinations[meandist == min(combinations$meandist)]
```

```
##    loc1 loc2 meandist
## 1:  193   12 175688.1
```

```
## Genetic algorithm based approach
summary(gamodel)
```

```
## -- Genetic Algorithm -------------------
##
## GA settings:
## Type                  =  binary
## Population size       =  50
## Number of generations =  100
## Elitism               =  2
## Crossover probability =  0.8
## Mutation probability  =  0.1
##
## GA results:
## Iterations             = 100
## Fitness function value = -20633823071
## Solution =
##      x1 x2 x3 x4 x5 x6 x7 x8
## [1,]  0  1  0  1  0  0  0  0
```

```
unique(dist.dt$OBJECTID)
```

```
## [1] "119" "12"  "132" "193" "60"  "64"  "67"  "95"
```

Used avarage closeness from the residences to the health care services. Locations that are closest to all the residences are assumed to be selected

Fitness Function

Initially, profit vector (p) was initialized with all ones

Mean distance was used as the weights

Fitness function was defined by penalizing the objective function f with the penalty value

Here, binary GA metric was used

W <- max(w) * 1.14 was found to be the optimal value where the GA approach was same

as the Brute FOrce approach

For the fraction frac=0.00001 and n.ctrs=8

GA based

x1 x2 x3 x4 x5 x6 x7 x8

0 1 0 1 0 0 0 0

“119” “12” “132” “193” “60” “64” “67” “95”

Brute FOrce approach

loc1 loc2 meandist

193 12 196469.3

For the fraction frac=0.0001 and n.ctrs=8 (increased dataset values)

GA based

x1 x2 x3 x4 x5 x6 x7 x8

0 1 0 1 0 0 0 0

“119” “12” “132” “193” “60” “64” “67” “95”

Brute FOrce approach

loc1 loc2 meandist

193 12 174485

For the fraction frac=0.001 and n.ctrs=8 (increased dataset values)

GA based

```
##
```