ACS Homework 2 于丁一 2021310532

于丁一

2022/3/25

3.1

a.

Move strategy: Immediate adoption of the first randomly selected downhill neighbor.

For each starting number and each step, it will randomly try at most 100 times.

```
## INITIAL VALUES
setwd('C:/Users/yudyd/Desktop/ACS')
baseball.dat = read.table('baseball.dat',header=T)
#baseball.dat = read.table(file.choose(),header=TRUE)
baseball.dat$freeagent = factor(baseball.dat$freeagent)
baseball.dat$arbitration = factor(baseball.dat$arbitration)
baseball.sub = baseball.dat[,-1]
salary.log = log(baseball.dat$salary)
n = length(salary.log)
m = length(baseball.sub[1,])
```

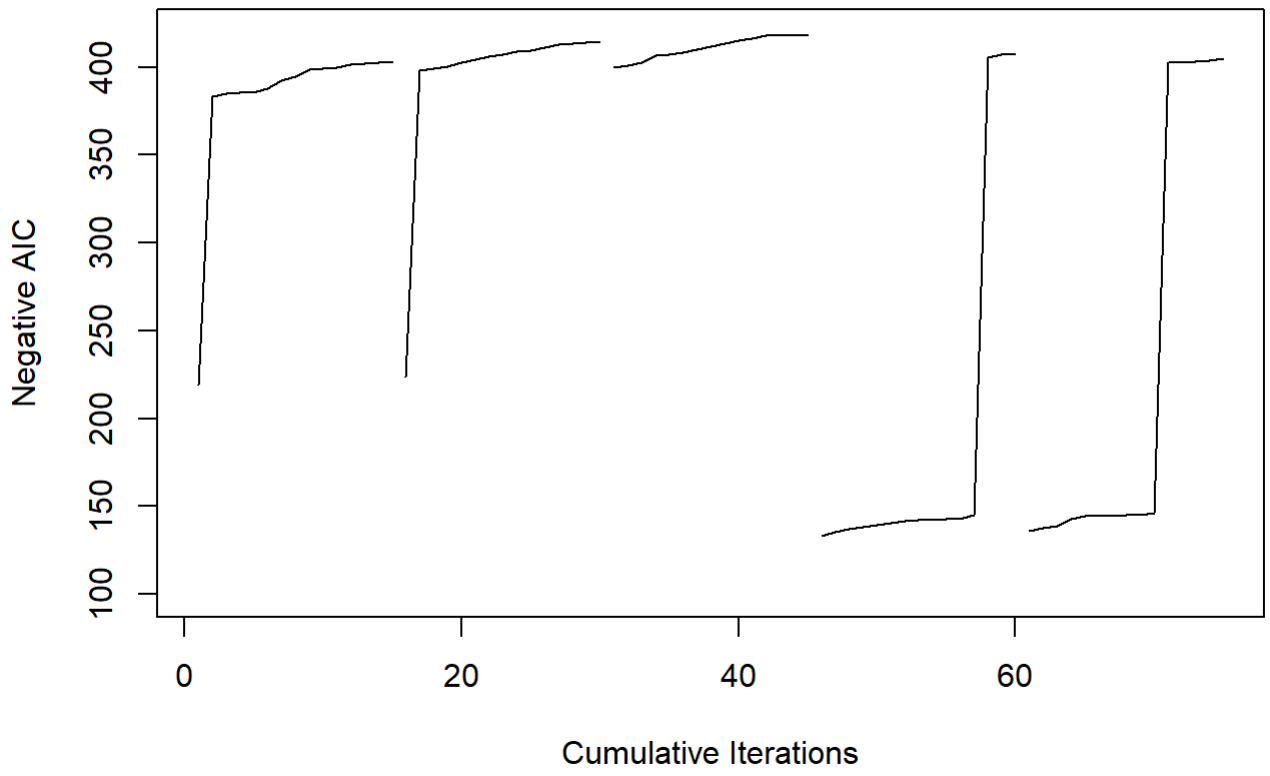
```
num.starts = 5
runs = matrix(0,num.starts,m)
itr = 15
runs.aic = matrix(0,num.starts,itr)
maxitr.step = 100

# INITIALIZES STARTING RUNS
set.seed(19676)
for(i in 1:num.starts){runs[i,] = rbinom(m,1,.5)}

## MAIN
for(k in 1:num.starts){
    run.current = runs[k,]

    # ITERATES EACH RANDOM START
    for(j in 1:itr){
        run.vars = baseball.sub[,run.current==1]
        g = lm(salary.log~.,run.vars)
        run.aic = extractAIC(g)[2]
        run.next = run.current

        # TESTS ALL MODELS IN THE 1-NEIGHBORHOOD AND SELECTS THE
        # MODEL WITH THE LOWEST AIC
        set.seed(2022)
        ii = 0
        while(1){
          i <- sample(1:m,1)
            run.step = run.current
            run.step[i] = !run.current[i]
            run.vars = baseball.sub[,run.step==1]
            g = lm(salary.log~.,run.vars)
            run.step.aic = extractAIC(g)[2]
            if(run.step.aic < run.aic){
                run.next = run.step
                run.aic = run.step.aic
                break
            }
            ii = ii + 1
            if(ii == maxitr.step){
              break
            }
        }
        run.current = run.next
        runs.aic[k,j]=run.aic
    }
    runs[k,] = run.current
}

## OUTPUT
runs        # LISTS OF PREDICTORS
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
## [1,]    1    0    0    1    0    0    1    1    0     0     0     0     1     1
## [2,]    0    0    0    0    0    0    0    1    1     1     0     0     1     1
## [3,]    0    1    1    0    0    0    0    1    0     1     0     0     1     1
## [4,]    1    0    0    1    0    0    1    0    1     1     1     0     1     1
## [5,]    0    0    1    1    0    0    1    0    1     1     0     1     1     1
##      [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26]
## [1,]     0     0     0     1     0     0     1     1     0     1     1     0
## [2,]     0     1     1     1     0     1     1     1     0     0     1     0
## [3,]     1     1     0     0     0     0     0     0     0     1     1     0
## [4,]     0     0     0     0     0     1     1     1     0     0     0     0
## [5,]     0     0     0     0     0     1     0     1     0     0     1     0
##      [,27]
## [1,]     1
## [2,]     0
## [3,]     1
## [4,]     0
## [5,]     0
```

```
runs.aic    # AIC VALUES
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] -219.2491 -383.1598 -384.8845 -385.2670 -385.5486 -387.7295 -392.4281
## [2,] -223.9421 -397.8533 -399.0655 -400.4496 -402.3143 -404.0909 -406.0190
## [3,] -399.9767 -400.9211 -402.7400 -406.5377 -407.2893 -408.0385 -409.8765
## [4,] -133.1792 -135.1758 -136.9254 -138.0628 -139.6416 -140.6238 -141.9388
## [5,] -135.9155 -137.6556 -138.6794 -142.8906 -144.4592 -144.6582 -144.7527
##           [,8]      [,9]     [,10]     [,11]     [,12]     [,13]     [,14]
## [1,] -394.3687 -398.8490 -399.0062 -399.7377 -401.3536 -401.8140 -402.4844
## [2,] -407.2632 -408.6217 -409.6959 -411.3662 -412.6075 -413.2450 -413.7213
## [3,] -411.6624 -413.5920 -415.3194 -415.9826 -417.7432 -417.7432 -417.7432
## [4,] -142.3828 -142.4026 -142.9623 -143.0717 -145.0678 -405.6801 -406.9117
## [5,] -144.8277 -144.8767 -146.4994 -402.3639 -402.4116 -403.2609 -403.5843
##          [,15]
## [1,] -402.5519
## [2,] -413.8955
## [3,] -417.7432
## [4,] -407.3752
## [5,] -405.0748
```

```
##PLOT
plot(1:(itr*num.starts),-c(t(runs.aic)),xlab="Cumulative Iterations",
  ylab="Negative AIC",ylim=c(100,420),type="n")
for(i in 1:num.starts) {
  lines((i-1)*itr+(1:itr),-runs.aic[i,]) }
```



b.

Moving strategy: 2-neighborhoods.

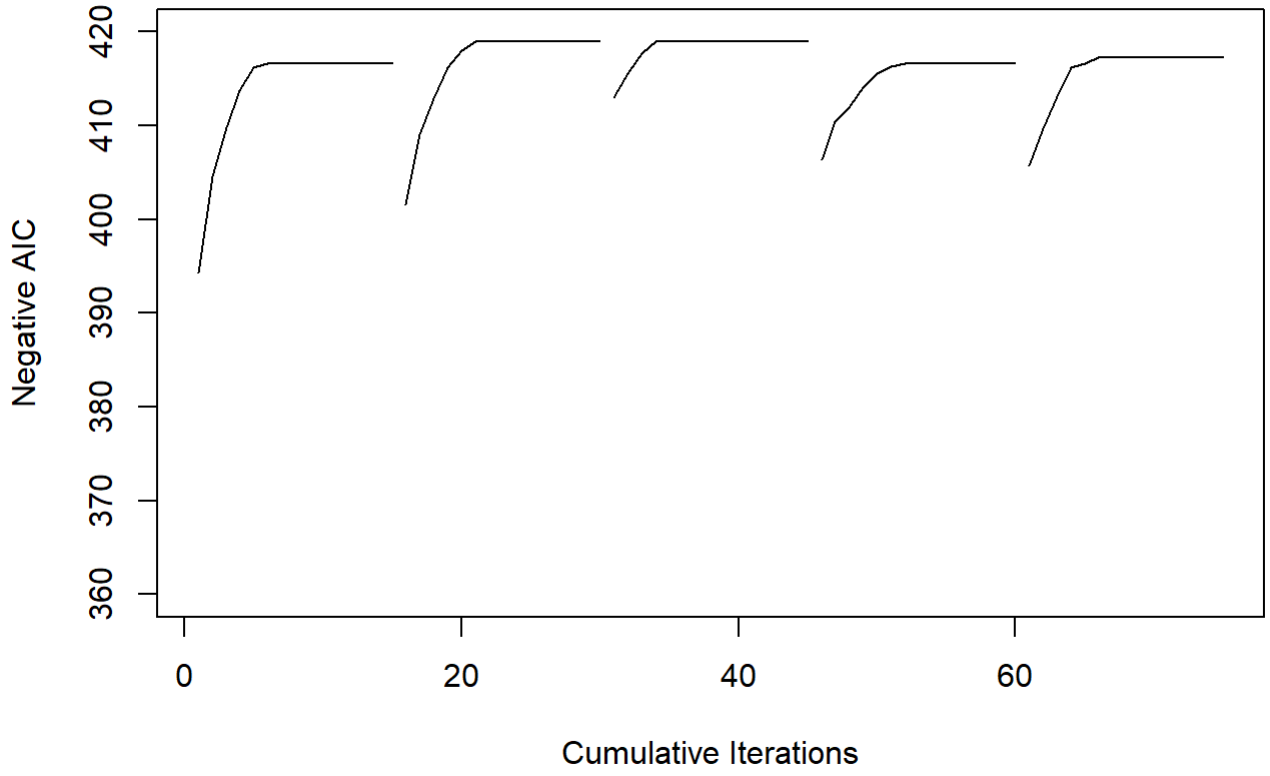
```
set.seed(19676)
for(i in 1:num.starts){runs[i,] = rbinom(m,1,.5)}

## MAIN
for(k in 1:num.starts){
    run.current = runs[k,]

    # ITERATES EACH RANDOM START
    for(j in 1:itr){
        run.vars = baseball.sub[,run.current==1]
        g = lm(salary.log~.,run.vars)
        run.aic = extractAIC(g)[2]
        run.next = run.current

        # TESTS ALL MODELS IN THE 1-NEIGHBORHOOD AND SELECTS THE
        # MODEL WITH THE LOWEST AIC
        for (i in 0:(m-1)) {
        for(ii in (i+1):m){
            run.step = run.current
            run.step[i] = !run.current[i]
            run.step[ii] = !run.current[ii]
            run.vars = baseball.sub[,run.step==1]
            g = lm(salary.log~.,run.vars)
            run.step.aic = extractAIC(g)[2]
            if(run.step.aic < run.aic){
                run.next = run.step
                run.aic = run.step.aic
            }
        }
        }
        run.current = run.next
        runs.aic[k,j]=run.aic
    }
    runs[k,] = run.current
}

## OUTPUT
runs        # LISTS OF PREDICTORS
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
## [1,]    0    0    1    0    0    1    0    1    0     1     0     0     1     1
## [2,]    0    1    1    0    0    1    0    1    0     1     0     0     1     1
## [3,]    0    1    1    0    0    1    0    1    0     1     0     0     1     1
## [4,]    0    0    1    0    0    1    0    1    0     1     0     1     1     1
## [5,]    0    0    1    0    0    1    0    1    0     1     0     1     1     1
##      [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26]
## [1,]     0     0     0     0     0     1     1     1     0     1     1     0
## [2,]     1     1     0     0     0     0     0     0     0     1     1     1
## [3,]     1     1     0     0     0     0     0     0     0     1     1     1
## [4,]     0     0     0     0     0     1     1     1     0     0     0     0
## [5,]     1     1     0     0     0     1     0     1     0     0     1     1
##      [,27]
## [1,]     0
## [2,]     0
## [3,]     0
## [4,]     0
## [5,]     0
```

```
runs.aic    # AIC VALUES
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] -394.2750 -404.5787 -409.7884 -413.6918 -416.2450 -416.6119 -416.6119
## [2,] -401.5732 -409.0429 -412.8614 -416.2275 -418.0125 -418.9472 -418.9472
## [3,] -413.0112 -415.6115 -417.7187 -418.9472 -418.9472 -418.9472 -418.9472
## [4,] -406.4152 -410.3917 -411.9585 -414.0506 -415.5199 -416.3035 -416.6030
## [5,] -405.7847 -409.7152 -413.0607 -416.1504 -416.5907 -417.2714 -417.2714
##           [,8]      [,9]     [,10]     [,11]     [,12]     [,13]     [,14]
## [1,] -416.6119 -416.6119 -416.6119 -416.6119 -416.6119 -416.6119 -416.6119
## [2,] -418.9472 -418.9472 -418.9472 -418.9472 -418.9472 -418.9472 -418.9472
## [3,] -418.9472 -418.9472 -418.9472 -418.9472 -418.9472 -418.9472 -418.9472
## [4,] -416.6030 -416.6030 -416.6030 -416.6030 -416.6030 -416.6030 -416.6030
## [5,] -417.2714 -417.2714 -417.2714 -417.2714 -417.2714 -417.2714 -417.2714
##          [,15]
## [1,] -416.6119
## [2,] -418.9472
## [3,] -418.9472
## [4,] -416.6030
## [5,] -417.2714
```

```
##PLOT
plot(1:(itr*num.starts),-c(t(runs.aic)),xlab="Cumulative Iterations",
  ylab="Negative AIC",ylim=c(360,420),type="n")
for(i in 1:num.starts) {
  lines((i-1)*itr+(1:itr),-runs.aic[i,]) }
```



The algorithm based on 2-neighborhoods run slowly than algorithm based on 1-neighborhoods and immediate adoption because it search more candidate solution. As the result, the algorithm based on 2-neighborhoods outperform the algorithm based on 1-neighborhoods and immediate adoption.

3.2

a.

Try different tabu tenures.

For tabu tenures equals to 10,

```
tabu = rep(0,m)
tabu.term = 10
itr = 75
aics = rep(0,itr+1)

# INITIALIZES STARTING RUN
set.seed(139992)
run = rbinom(m,1,.5)
run.current = run
run.vars = baseball.sub[,run.current==1]
g = lm(salary.log~.,run.vars)
run.aic = extractAIC(g)[2]
best.aic = run.aic
aics[1] = run.aic

## MAIN
for(j in 1:itr){
    run.aic = 0

    # TESTS ALL MODELS IN THE 1-NEIGHBORHOOD AND CHOOSES THE BEST
    # MODEL IF THE MODEL IS NOT TABU, OTHERWISE IT SELECTS THE
    # LEAST UNFAVORABLE UNLESS THE MODEL IS THE BEST SEEN OVERALL
    for(i in 1:m){
        run.step = run.current
        run.step[i] = !run.current[i]
        run.vars = baseball.sub[,run.step==1]
        g = lm(salary.log~.,run.vars)
        run.step.aic = extractAIC(g)[2]
        if(run.step.aic < run.aic && tabu[i]==0){
          run.next = run.step
          run.aic = run.step.aic
          pos = i
      }
      if(run.step.aic < run.aic && tabu[i]!=0 &&
        run.step.aic < best.aic){
        run.next = run.step
        run.aic = run.step.aic
        pos = i
      }

    # DECREMENT TABU TERMS
      if(tabu[i]!=0){
        tabu[i]=tabu[i]-1
        }
    }

    tabu[pos] = tabu.term
    run.current = run.next
    if(run.aic < best.aic){
        best.aic = run.aic
        run = run.current
    }
```

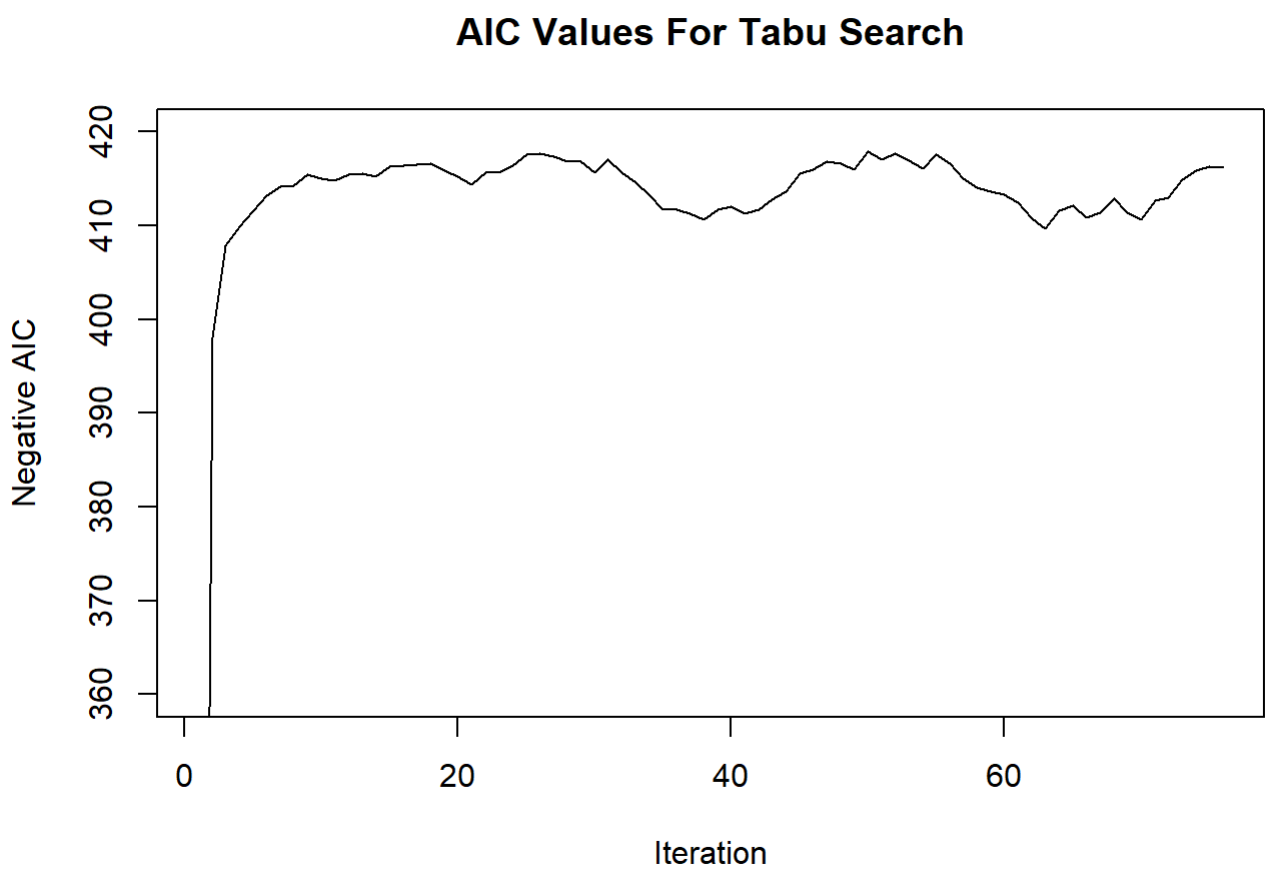
```
    aics[j+1] = run.aic
}

## OUTPUT
run         # BEST LIST OF PREDICTORS FOUND
```

```
##  [1] 1 0 1 0 0 0 0 1 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 1 1 0 1
```

```
best.aic    # AIC VALUE
```

```
## [1] -417.8858
```

```
#aics       # VECTOR OF AIC VALUES

## PLOT OF AIC VALUES

plot(-aics,type="n",ylab="Negative AIC",xlab="Iteration",
    main="AIC Values For Tabu Search",ylim=c(360,420))
lines(-aics)
```




```
sort(aics)
```

```
##  [1] -417.8858 -417.7432 -417.7432 -417.5813 -417.5622 -417.4223 -417.0554
##  [8] -417.0463 -416.9614 -416.8776 -416.8081 -416.8033 -416.6447 -416.6119
## [15] -416.5833 -416.5411 -416.4249 -416.3614 -416.3035 -416.2576 -416.1533
## [22] -416.0728 -415.9821 -415.9666 -415.9155 -415.9064 -415.6902 -415.6902
## [29] -415.6631 -415.6516 -415.5280 -415.5199 -415.4259 -415.4118 -415.2833
## [36] -415.2214 -415.0604 -414.9912 -414.9363 -414.8595 -414.6162 -414.3713
## [43] -414.2939 -414.1899 -414.0777 -413.6519 -413.5906 -413.2981 -413.2981
## [50] -413.2326 -412.9754 -412.8515 -412.7407 -412.6596 -412.5132 -412.0902
## [57] -412.0696 -411.7305 -411.7103 -411.7007 -411.6773 -411.6593 -411.5917
## [64] -411.3499 -411.3417 -411.3392 -411.2925 -410.8289 -410.7821 -410.6878
## [71] -410.6033 -409.8627 -409.6594 -407.8643 -398.0523 -214.3571
```

```
(1:76)[aics==min(aics)]
```

```
## [1] 50
```

For tabu tenures equals to 5,

```
tabu = rep(0,m)
tabu.term = 5
itr = 75
aics = rep(0,itr+1)

# INITIALIZES STARTING RUN
set.seed(139992)
run = rbinom(m,1,.5)
run.current = run
run.vars = baseball.sub[,run.current==1]
g = lm(salary.log~.,run.vars)
run.aic = extractAIC(g)[2]
best.aic = run.aic
aics[1] = run.aic

## MAIN
for(j in 1:itr){
    run.aic = 0

    # TESTS ALL MODELS IN THE 1-NEIGHBORHOOD AND CHOOSES THE BEST
    # MODEL IF THE MODEL IS NOT TABU, OTHERWISE IT SELECTS THE
    # LEAST UNFAVORABLE UNLESS THE MODEL IS THE BEST SEEN OVERALL
    for(i in 1:m){
        run.step = run.current
        run.step[i] = !run.current[i]
        run.vars = baseball.sub[,run.step==1]
        g = lm(salary.log~.,run.vars)
        run.step.aic = extractAIC(g)[2]
        if(run.step.aic < run.aic && tabu[i]==0){
          run.next = run.step
          run.aic = run.step.aic
          pos = i
      }
      if(run.step.aic < run.aic && tabu[i]!=0 &&
        run.step.aic < best.aic){
        run.next = run.step
        run.aic = run.step.aic
        pos = i
      }

    # DECREMENT TABU TERMS
      if(tabu[i]!=0){
        tabu[i]=tabu[i]-1
        }
    }

    tabu[pos] = tabu.term
    run.current = run.next
    if(run.aic < best.aic){
        best.aic = run.aic
        run = run.current
    }
```

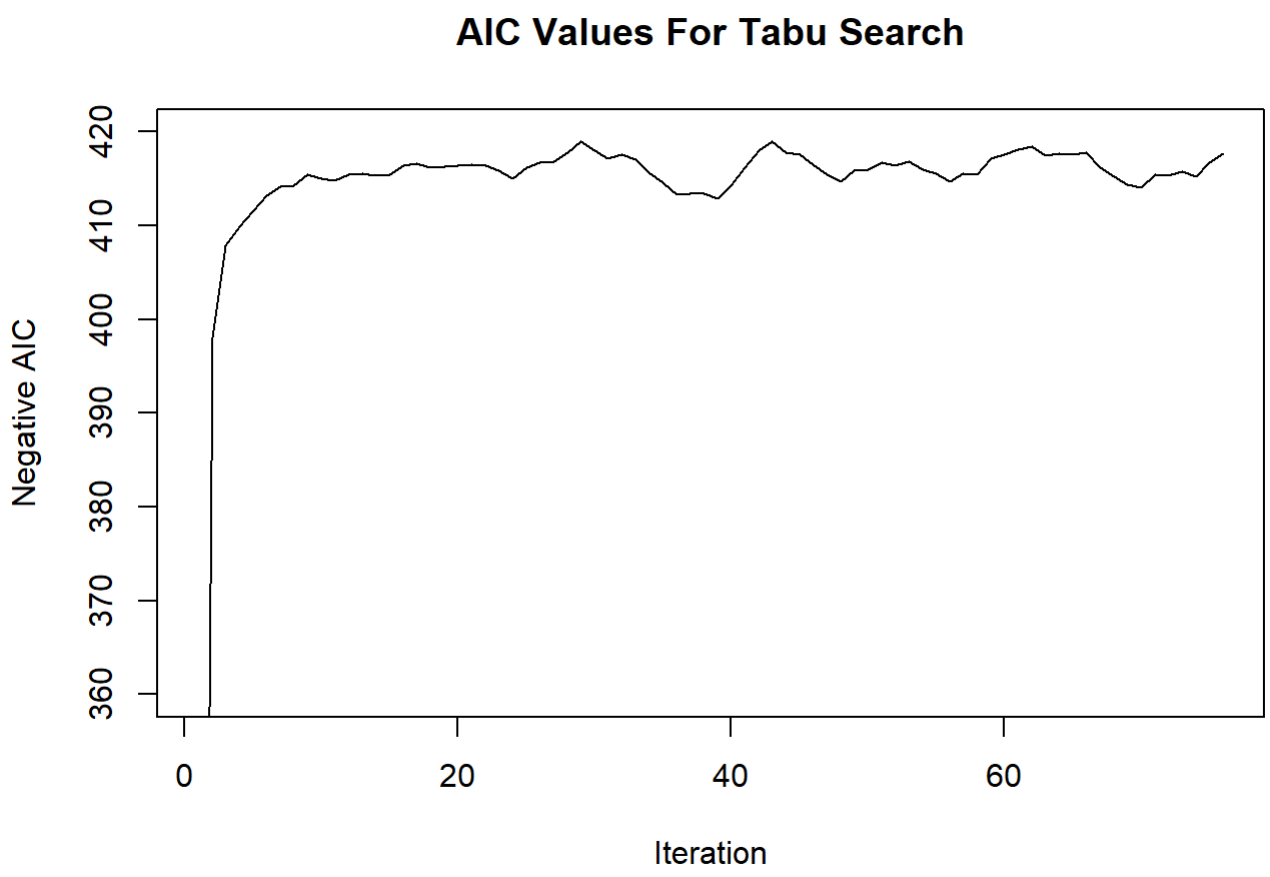
```
    aics[j+1] = run.aic
}

## OUTPUT
run         # BEST LIST OF PREDICTORS FOUND
```

```
##  [1] 0 1 1 0 0 1 0 1 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 1 1 1 0
```

```
best.aic    # AIC VALUE
```

```
## [1] -418.9472
```

```
#aics       # VECTOR OF AIC VALUES

## PLOT OF AIC VALUES

plot(-aics,type="n",ylab="Negative AIC",xlab="Iteration",
    main="AIC Values For Tabu Search",ylim=c(360,420))
lines(-aics)
```




```
sort(aics)
```

```
##  [1] -418.9472 -418.9472 -418.4511 -418.1540 -418.0125 -418.0125 -417.8494
##  [8] -417.8494 -417.7992 -417.7187 -417.7187 -417.5816 -417.5813 -417.5813
## [15] -417.5423 -417.4739 -417.2010 -417.1452 -417.0463 -416.8813 -416.8033
## [22] -416.7620 -416.7536 -416.7464 -416.6030 -416.5484 -416.5069 -416.4283
## [29] -416.4097 -416.3790 -416.3702 -416.3233 -416.2450 -416.2392 -416.2010
## [36] -416.1878 -415.9890 -415.9666 -415.9046 -415.8448 -415.7739 -415.6902
## [43] -415.5311 -415.5280 -415.5199 -415.4646 -415.4532 -415.4301 -415.4259
## [50] -415.4157 -415.4118 -415.3496 -415.3377 -415.2439 -415.2242 -415.0604
## [57] -415.0239 -414.8595 -414.7020 -414.6872 -414.6162 -414.4247 -414.2939
## [64] -414.2886 -414.1899 -414.0117 -413.4579 -413.4147 -413.2981 -413.2326
## [71] -412.8449 -411.5917 -409.8627 -407.8643 -398.0523 -214.3571
```

```
(1:76)[aics==min(aics)]
```

```
## [1] 29 43
```

For tabu tenures equals to 3,

```
tabu = rep(0,m)
tabu.term = 3
itr = 75
aics = rep(0,itr+1)

# INITIALIZES STARTING RUN
set.seed(139992)
run = rbinom(m,1,.5)
run.current = run
run.vars = baseball.sub[,run.current==1]
g = lm(salary.log~.,run.vars)
run.aic = extractAIC(g)[2]
best.aic = run.aic
aics[1] = run.aic

## MAIN
for(j in 1:itr){
    run.aic = 0

    # TESTS ALL MODELS IN THE 1-NEIGHBORHOOD AND CHOOSES THE BEST
    # MODEL IF THE MODEL IS NOT TABU, OTHERWISE IT SELECTS THE
    # LEAST UNFAVORABLE UNLESS THE MODEL IS THE BEST SEEN OVERALL
    for(i in 1:m){
        run.step = run.current
        run.step[i] = !run.current[i]
        run.vars = baseball.sub[,run.step==1]
        g = lm(salary.log~.,run.vars)
        run.step.aic = extractAIC(g)[2]
        if(run.step.aic < run.aic && tabu[i]==0){
          run.next = run.step
          run.aic = run.step.aic
          pos = i
      }
      if(run.step.aic < run.aic && tabu[i]!=0 &&
        run.step.aic < best.aic){
        run.next = run.step
        run.aic = run.step.aic
        pos = i
      }

    # DECREMENT TABU TERMS
      if(tabu[i]!=0){
        tabu[i]=tabu[i]-1
        }
    }

    tabu[pos] = tabu.term
    run.current = run.next
    if(run.aic < best.aic){
        best.aic = run.aic
        run = run.current
    }
```

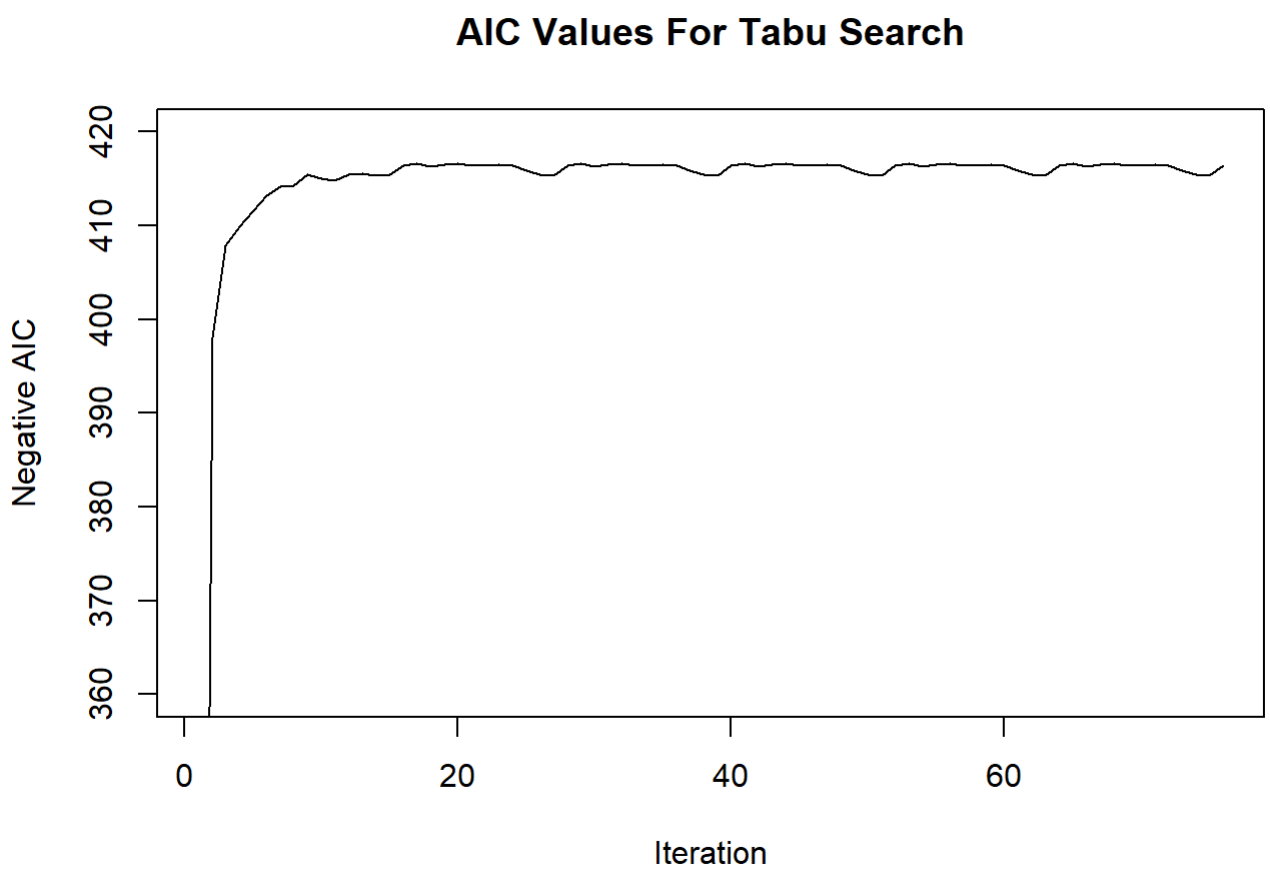
```
    aics[j+1] = run.aic
}

## OUTPUT
run         # BEST LIST OF PREDICTORS FOUND
```

```
##  [1] 0 0 1 0 0 1 0 1 0 1 0 0 1 1 0 0 0 0 0 1 1 1 0 1 1 0 0
```

```
best.aic    # AIC VALUE
```

```
## [1] -416.6119
```

```
#aics       # VECTOR OF AIC VALUES

## PLOT OF AIC VALUES

plot(-aics,type="n",ylab="Negative AIC",xlab="Iteration",
    main="AIC Values For Tabu Search",ylim=c(360,420))
lines(-aics)
```




```
sort(aics)
```

```
##  [1] -416.6119 -416.6119 -416.6119 -416.6119 -416.6119 -416.6030 -416.6030
##  [8] -416.6030 -416.6030 -416.6030 -416.5411 -416.5411 -416.5411 -416.5411
## [15] -416.5411 -416.5069 -416.5069 -416.5069 -416.5069 -416.5069 -416.4101
## [22] -416.4101 -416.4101 -416.4101 -416.4101 -416.4097 -416.4097 -416.4097
## [29] -416.4097 -416.4097 -416.3790 -416.3790 -416.3790 -416.3790 -416.3790
## [36] -416.3702 -416.3702 -416.3702 -416.3702 -416.3702 -416.3702 -416.2547
## [43] -416.2547 -416.2547 -416.2547 -416.2547 -415.9080 -415.9080 -415.9080
## [50] -415.9080 -415.9080 -415.5199 -415.4259 -415.4157 -415.4118 -415.4026
## [57] -415.4026 -415.4026 -415.4026 -415.4026 -415.3920 -415.3920 -415.3920
## [64] -415.3920 -415.3920 -415.3496 -415.0604 -414.8595 -414.2939 -414.1899
## [71] -413.2326 -411.5917 -409.8627 -407.8643 -398.0523 -214.3571
```

```
(1:76)[aics==min(aics)]
```

```
## [1] 20 32 44 56 68
```

When tabu tenure equal to 10, the algorithm find the solution AIC = 417.8858, which is not the global minimum. It is because that the tabu tenure is too large that it the algorithm has too little choices in one local search.

When tabu tenure equal to 5, the algorithm find the solution AIC = -418.9472, which is the global minimum.It can show that 5 is an appropriate tabu tenure.

When tabu tenure equal to 3, the algorithm find the solution AIC = -416.6119, which is not the global minimum. It is because the tabu tenure is too small and the algorithm converge into local minimum.

b.

Choosing tabu tenures as 5, and AIC upper bound as 5.

```
tabu = rep(0,m)
tabu.bound = 5
tabu.state = 0
tabu.term = 5
itr = 75
aics = rep(0,itr+1)
aic.change = rep(0,itr)

# INITIALIZES STARTING RUN
set.seed(139992)
run = rbinom(m,1,.5)
run.current = run
run.vars = baseball.sub[,run.current==1]
g = lm(salary.log~.,run.vars)
run.aic = extractAIC(g)[2]
best.aic = run.aic
aics[1] = run.aic

## MAIN
for(j in 1:itr){
    run.aic = 0

    # TESTS ALL MODELS IN THE 1-NEIGHBORHOOD AND CHOOSES THE BEST
    # MODEL IF THE MODEL IS NOT TABU, OTHERWISE IT SELECTS THE
    # LEAST UNFAVORABLE UNLESS THE MODEL IS THE BEST SEEN OVERALL
    for(i in 1:m){
        run.step = run.current
        run.step[i] = !run.current[i]
        run.vars = baseball.sub[,run.step==1]
        g = lm(salary.log~.,run.vars)
        run.step.aic = extractAIC(g)[2]
        if(run.step.aic < run.aic && tabu[i]==0 && tabu.state == 0){
          run.next = run.step
          run.aic = run.step.aic
          pos = i
        }
        if(run.step.aic < run.aic && tabu[i]==0 && tabu.state != 0 &&
           (aics[j] - run.step.aic) < 1){
          run.next = run.step
          run.aic = run.step.aic
          pos = i
        }
      if(run.step.aic < run.aic && tabu[i]!=0 &&
        run.step.aic < best.aic){
        run.next = run.step
        run.aic = run.step.aic
        pos = i
      }

    # DECREMENT TABU TERMS
      if(tabu[i]!=0){
        tabu[i]=tabu[i]-1
```

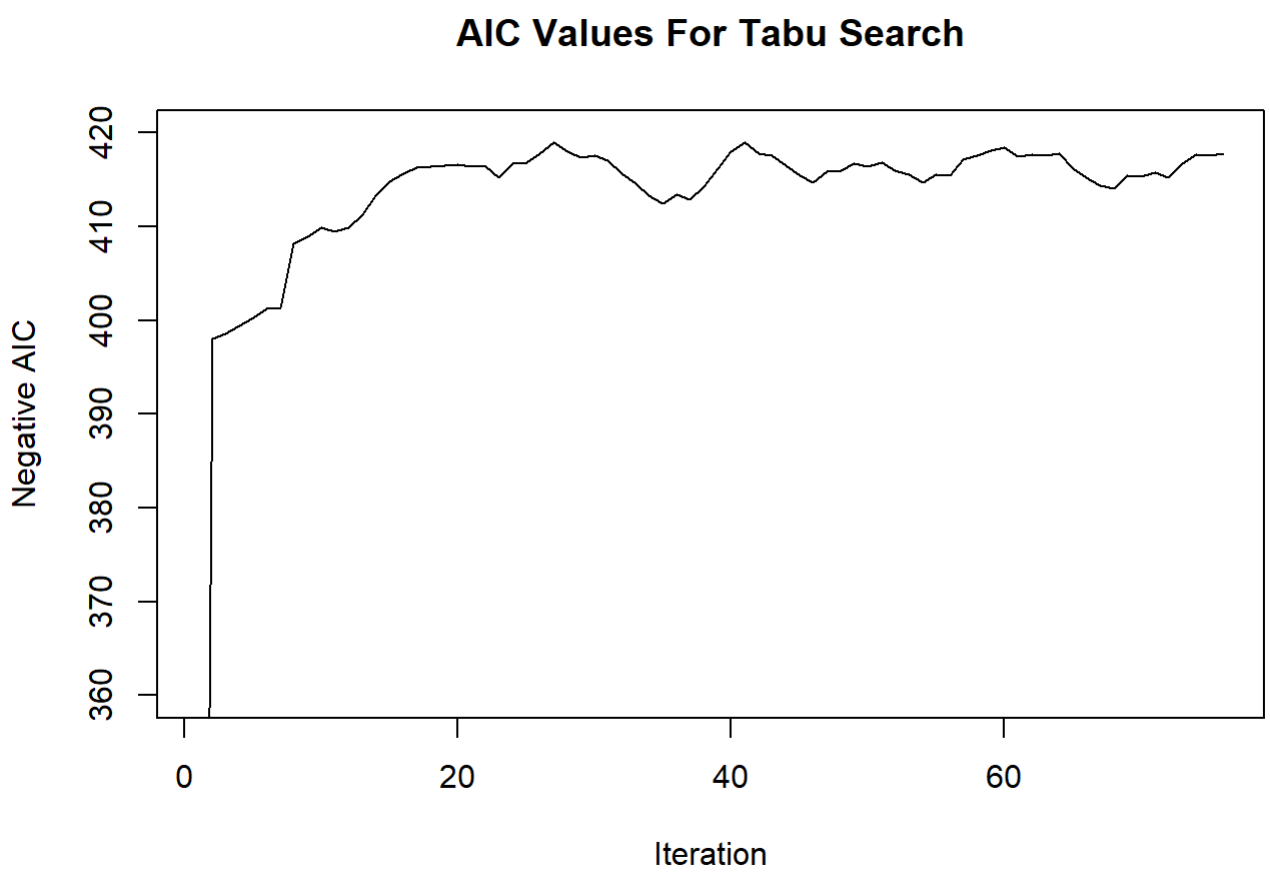
```
        }
    }
    if(tabu.state != 0){
      tabu.state = tabu.state - 1
    }

    aic.change[j] = aics[j] - run.aic

    if(aic.change[j] > 5 && tabu.state == 0){
      tabu.state = tabu.term
    }
    tabu[pos] = tabu.term
    run.current = run.next
    if(run.aic < best.aic){
        best.aic = run.aic
        run = run.current
    }
    aics[j+1] = run.aic
}

## OUTPUT
run         # BEST LIST OF PREDICTORS FOUND
```

```
##  [1] 0 1 1 0 0 1 0 1 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 1 1 1 0
```

```
best.aic    # AIC VALUE
```

```
## [1] -418.9472
```

```
#aics       # VECTOR OF AIC VALUES

## PLOT OF AIC VALUES

plot(-aics,type="n",ylab="Negative AIC",xlab="Iteration",
    main="AIC Values For Tabu Search",ylim=c(360,420))
lines(-aics)
```




```
sort(aics)
```

```
##  [1] -418.9472 -418.9472 -418.4511 -418.1540 -418.0125 -418.0125 -417.8494
##  [8] -417.8494 -417.8494 -417.7992 -417.7187 -417.7187 -417.5816 -417.5813
## [15] -417.5813 -417.5813 -417.5423 -417.4739 -417.3942 -417.1452 -417.0463
## [22] -416.8813 -416.8033 -416.7620 -416.7536 -416.7464 -416.6119 -416.5484
## [29] -416.5411 -416.4283 -416.4249 -416.4101 -416.3790 -416.3035 -416.2392
## [36] -416.1878 -415.9890 -415.9666 -415.8448 -415.7739 -415.6902 -415.6375
## [43] -415.5311 -415.5280 -415.4646 -415.4532 -415.4301 -415.3377 -415.2439
## [50] -415.2242 -415.2148 -414.8266 -414.7020 -414.6872 -414.6162 -414.4247
## [57] -414.2886 -414.0117 -413.4579 -413.2981 -413.2665 -412.8449 -412.5132
## [64] -411.2706 -409.9396 -409.8926 -409.4222 -408.9593 -408.2226 -401.4035
## [71] -401.2140 -400.3319 -399.3782 -398.5703 -398.0523 -214.3571
```

```
(1:76)[aics==min(aics)]
```

```
## [1] 27 41
```

Adding attribute that if the AIC change for one iteration is larger than 5, then the AIC change of next five iterations can not exceed 1.

With this attribute on the tabu list, the algorithm find the minimum AIC value -418.9472 in the 27th and 41st iteration.

C.

```
tabu = rep(0,m)
tabu.term = 5
itr = 75
aics = rep(0,itr+1)
rsq = rep(0,itr+1)

# INITIALIZES STARTING RUN
set.seed(139992)
run = rbinom(m,1,.5)
run.current = run
run.vars = baseball.sub[,run.current==1]
g = lm(salary.log~.,run.vars)
run.aic = extractAIC(g)[2]
run.rsq = summary(g)$r.squared
best.aic = run.aic
aics[1] = run.aic
rsq[1] = run.rsq
## MAIN
for(j in 1:itr){
    run.aic = 0
  run.rsq = 0
    # TESTS ALL MODELS IN THE 1-NEIGHBORHOOD AND CHOOSES THE BEST
    # MODEL IF THE MODEL IS NOT TABU, OTHERWISE IT SELECTS THE
    # LEAST UNFAVORABLE UNLESS THE MODEL IS THE BEST SEEN OVERALL
    for(i in 1:m){
        run.step = run.current
        run.step[i] = !run.current[i]
        run.vars = baseball.sub[,run.step==1]
        g = lm(salary.log~.,run.vars)
        run.step.aic = extractAIC(g)[2]
        run.step.rsq = summary(g)$r.squared
        if(run.step.aic < run.aic && tabu[i]==0){
          run.next = run.step
          run.aic = run.step.aic
          run.rsq = run.step.rsq
          pos = i
        }
        if(run.step.aic < run.aic && tabu[i]!=0 &&
           (rsq[j]-rsq[j-1])>0.1 && (run.step.rsq-rsq[j])<0.05){
        run.next = run.step
        run.aic = run.step.aic
        run.rsq = run.step.rsq
        pos = i
      }
      if(run.step.aic < run.aic && tabu[i]!=0 &&
        run.step.aic < best.aic){
        run.next = run.step
        run.aic = run.step.aic
        run.rsq = run.step.rsq
        pos = i
```

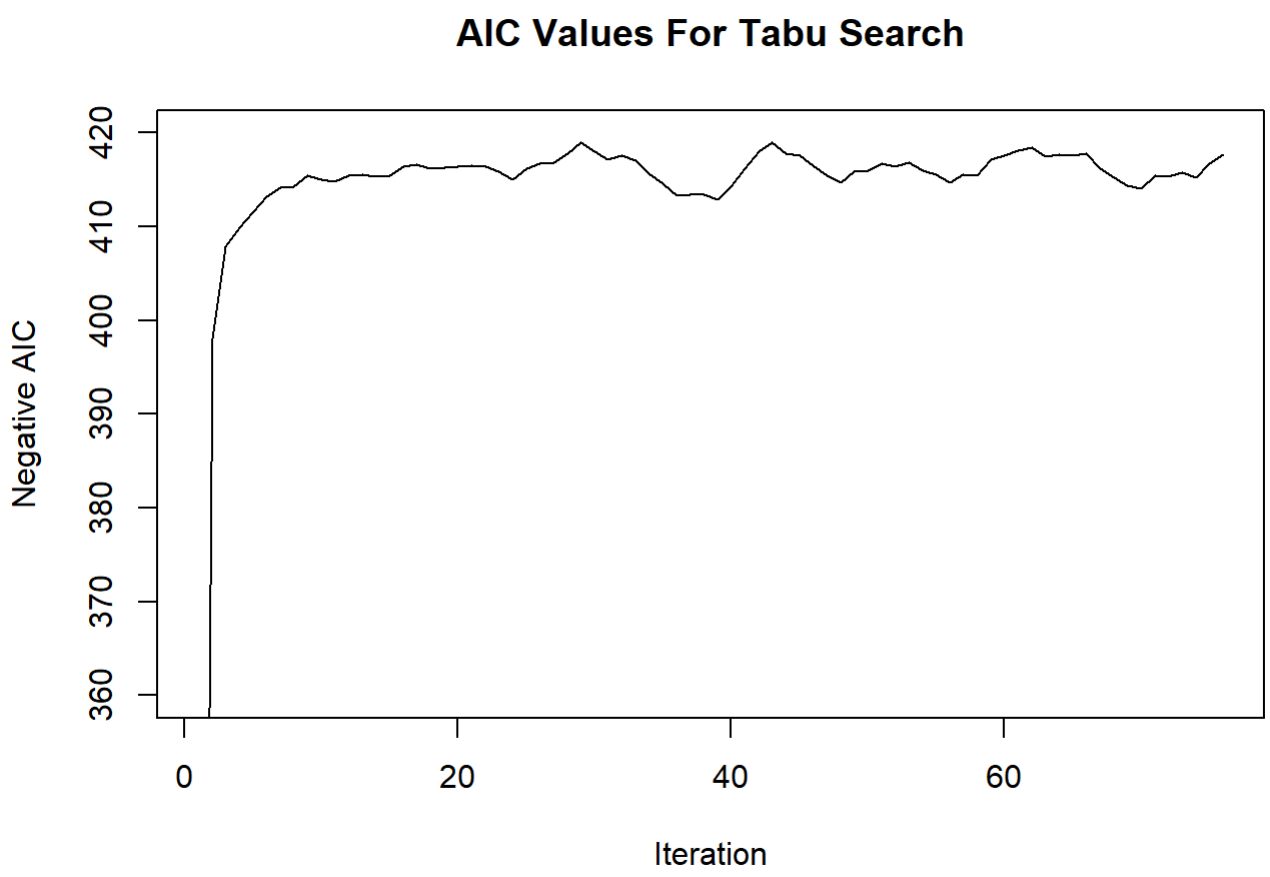
```
      }

    # DECREMENT TABU TERMS
      if(tabu[i]!=0){
        tabu[i]=tabu[i]-1
        }
    }

    tabu[pos] = tabu.term
    run.current = run.next
    if(run.aic < best.aic){
        best.aic = run.aic
        run = run.current
    }
    aics[j+1] = run.aic
    rsq[j+1] = run.rsq
}

## OUTPUT
run         # BEST LIST OF PREDICTORS FOUND
```

```
##  [1] 0 1 1 0 0 1 0 1 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 1 1 1 0
```

```
best.aic    # AIC VALUE
```

```
## [1] -418.9472
```

```
#aics       # VECTOR OF AIC VALUES

## PLOT OF AIC VALUES

plot(-aics,type="n",ylab="Negative AIC",xlab="Iteration",
    main="AIC Values For Tabu Search",ylim=c(360,420))
lines(-aics)
```




```
sort(aics)
```

```
##  [1] -418.9472 -418.9472 -418.4511 -418.1540 -418.0125 -418.0125 -417.8494
##  [8] -417.8494 -417.7992 -417.7187 -417.7187 -417.5816 -417.5813 -417.5813
## [15] -417.5423 -417.4739 -417.2010 -417.1452 -417.0463 -416.8813 -416.8033
## [22] -416.7620 -416.7536 -416.7464 -416.6030 -416.5484 -416.5069 -416.4283
## [29] -416.4097 -416.3790 -416.3702 -416.3233 -416.2450 -416.2392 -416.2010
## [36] -416.1878 -415.9890 -415.9666 -415.9046 -415.8448 -415.7739 -415.6902
## [43] -415.5311 -415.5280 -415.5199 -415.4646 -415.4532 -415.4301 -415.4259
## [50] -415.4157 -415.4118 -415.3496 -415.3377 -415.2439 -415.2242 -415.0604
## [57] -415.0239 -414.8595 -414.7020 -414.6872 -414.6162 -414.4247 -414.2939
## [64] -414.2886 -414.1899 -414.0117 -413.4579 -413.4147 -413.2981 -413.2326
## [71] -412.8449 -411.5917 -409.8627 -407.8643 -398.0523 -214.3571
```

```
(1:76)[aics==min(aics)]
```

```
## [1] 29 43
```

Adding aspiration that if the r-square change large in the previous move(high-influence), then for neighborhood that change r-square little(low-influence), it could be the next solution even if it is on the tabu list.

Under this aspiration, the algorithm find the minimum AIC -418.9472.

3.3

a.

cooling schedules 1.

For initial temperature 10, and $\alpha = 0.9$, and at the first 5 temperatures, the duration is 60, the next five duration is 120, the last five the duration is 220.

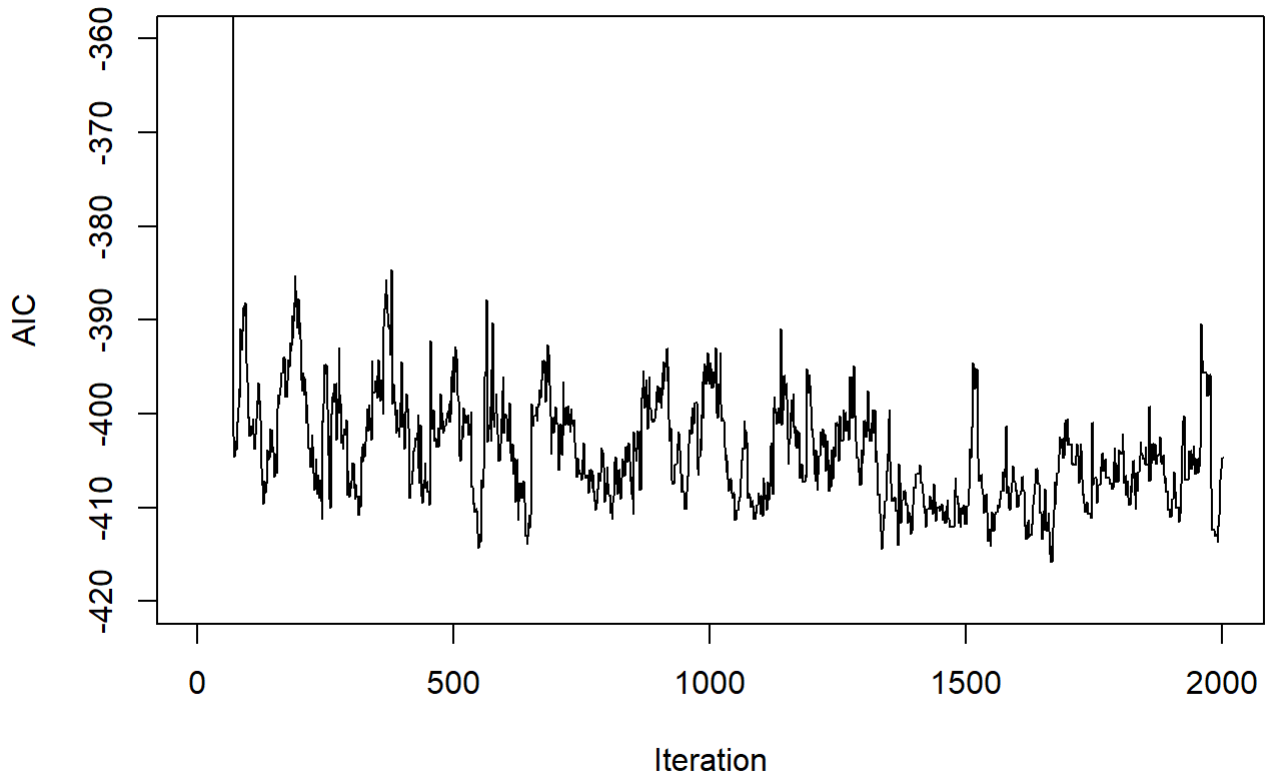
```
cooling = c(rep(60,5),rep(120,5),rep(220,5))
tau.start = 10
tau = rep(tau.start,15)
aics = NULL
# INITIALIZES STARTING RUN, TEMPERATURE SCHEDULE
set.seed(1999)
run = rbinom(m,1,.5)
run.current = run
run.vars = baseball.sub[,run.current==1]
g = lm(salary.log~.,run.vars)
run.aic = extractAIC(g)[2]
best.aic = run.aic
aics = run.aic
for(j in 2:15){tau[j] = 0.9*tau[j-1]}

## MAIN
for(j in 1:15){

    # RANDOMLY SELECTS A PREDICTOR TO ADD/REMOVE FROM THE MODEL
    # AND ACCEPTS THE NEW MODEL IF IT IS BETTER OR WITH PROBABILITY p
    for(i in 1:cooling[j]){
        pos = sample(1:m,1)
        run.step = run.current
        run.step[pos] = !run.current[pos]
        run.vars = baseball.sub[,run.step==1]
        g = lm(salary.log~.,run.vars)
        run.step.aic = extractAIC(g)[2]
        p = min(1,exp((run.aic-extractAIC(g)[2])/tau[j]))
        if(run.step.aic < run.aic){
            run.current = run.step
            run.aic = run.step.aic}
        if(rbinom(1,1,p)){
            run.current = run.step
            run.aic = run.step.aic}
        if(run.step.aic < best.aic){
            run = run.step
            best.aic = run.step.aic}
        aics = c(aics,run.aic)
    }
}

## OUTPUT
run         # BEST LIST OF PREDICTORS FOUND
```

```
##  [1] 0 1 1 0 0 1 0 1 0 1 1 0 1 1 1 1 1 0 0 0 0 0 0 1 0 0 1
```

```
best.aic    # AIC VALUE
```

```
## [1] -415.8437
```

```
#aics       # VECTOR OF AIC VALUES

## PLOT OF AIC VALUES
plot(aics,ylim=c(-420,-360),type="n",ylab="AIC", xlab="Iteration")
lines(aics)
```




```
(1:2001)[aics==min(aics)]
```

```
## [1] 1666 1667 1668
```

cooling schedules 2.

For initial temperature 15, and $\alpha = 0.95$, and at the first 5 temperatures, the duration is 30, the next five duration is 120, the last five the duration is 320.

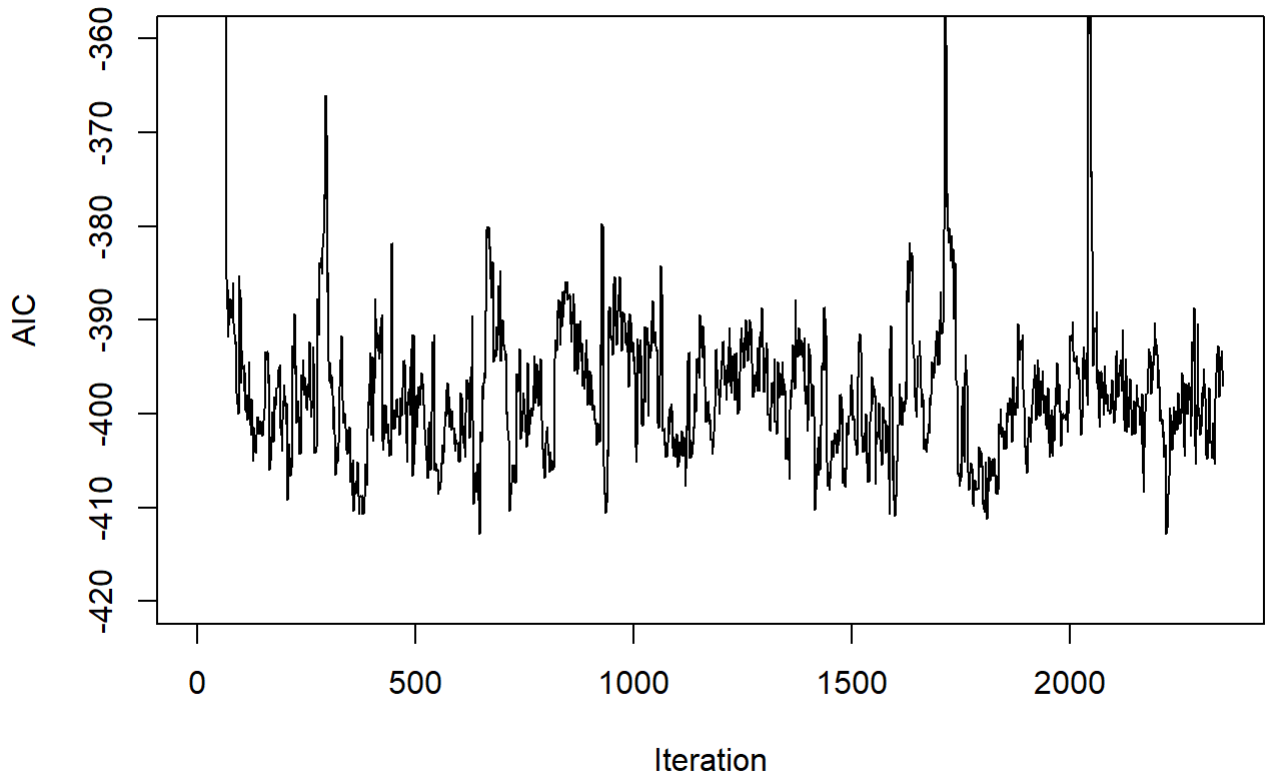
```
cooling = c(rep(30,5),rep(120,5),rep(320,5))
tau.start = 15
tau = rep(tau.start,15)
aics = NULL
# INITIALIZES STARTING RUN, TEMPERATURE SCHEDULE
set.seed(1999)
run = rbinom(m,1,.5)
run.current = run
run.vars = baseball.sub[,run.current==1]
g = lm(salary.log~.,run.vars)
run.aic = extractAIC(g)[2]
best.aic = run.aic
aics = run.aic
for(j in 2:15){tau[j] = 0.95*tau[j-1]}

## MAIN
for(j in 1:15){

    # RANDOMLY SELECTS A PREDICTOR TO ADD/REMOVE FROM THE MODEL
    # AND ACCEPTS THE NEW MODEL IF IT IS BETTER OR WITH PROBABILITY p
    for(i in 1:cooling[j]){
        pos = sample(1:m,1)
        run.step = run.current
        run.step[pos] = !run.current[pos]
        run.vars = baseball.sub[,run.step==1]
        g = lm(salary.log~.,run.vars)
        run.step.aic = extractAIC(g)[2]
        p = min(1,exp((run.aic-extractAIC(g)[2])/tau[j]))
        if(run.step.aic < run.aic){
            run.current = run.step
            run.aic = run.step.aic}
        if(rbinom(1,1,p)){
            run.current = run.step
            run.aic = run.step.aic}
        if(run.step.aic < best.aic){
            run = run.step
            best.aic = run.step.aic}
        aics = c(aics,run.aic)
    }
}

## OUTPUT
run         # BEST LIST OF PREDICTORS FOUND
```

```
##  [1] 0 0 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 0
```

```
best.aic    # AIC VALUE
```

```
## [1] -412.8295
```

```
#aics       # VECTOR OF AIC VALUES

## PLOT OF AIC VALUES
plot(aics,ylim=c(-420,-360),type="n",ylab="AIC", xlab="Iteration")
lines(aics)
```




```
(1:2001)[aics==min(aics)]
```

```
## [1] 646 647
```

cooling schedules 3.

For initial temperature 10, and $\alpha = 0.8$, and at the first 5 temperatures, the duration is 40, the next five duration is 120, the last five the duration is 320.

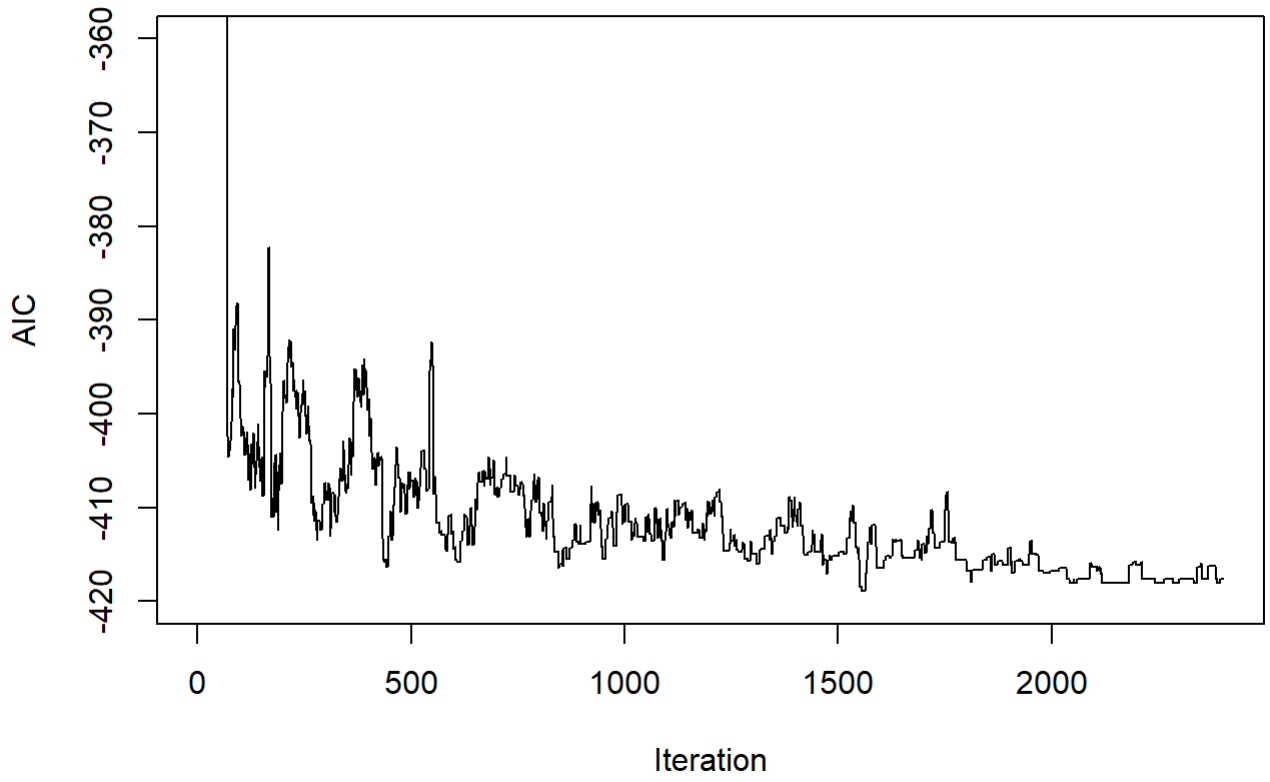
```
cooling = c(rep(40,5),rep(120,5),rep(320,5))
tau.start = 10
tau = rep(tau.start,15)
aics = NULL
# INITIALIZES STARTING RUN, TEMPERATURE SCHEDULE
set.seed(1999)
run = rbinom(m,1,.5)
run.current = run
run.vars = baseball.sub[,run.current==1]
g = lm(salary.log~.,run.vars)
run.aic = extractAIC(g)[2]
best.aic = run.aic
aics = run.aic
for(j in 2:15){tau[j] = 0.8*tau[j-1]}

## MAIN
for(j in 1:15){

    # RANDOMLY SELECTS A PREDICTOR TO ADD/REMOVE FROM THE MODEL
    # AND ACCEPTS THE NEW MODEL IF IT IS BETTER OR WITH PROBABILITY p
    for(i in 1:cooling[j]){
        pos = sample(1:m,1)
        run.step = run.current
        run.step[pos] = !run.current[pos]
        run.vars = baseball.sub[,run.step==1]
        g = lm(salary.log~.,run.vars)
        run.step.aic = extractAIC(g)[2]
        p = min(1,exp((run.aic-extractAIC(g)[2])/tau[j]))
        if(run.step.aic < run.aic){
            run.current = run.step
            run.aic = run.step.aic}
        if(rbinom(1,1,p)){
            run.current = run.step
            run.aic = run.step.aic}
        if(run.step.aic < best.aic){
            run = run.step
            best.aic = run.step.aic}
        aics = c(aics,run.aic)
    }
}

## OUTPUT
run         # BEST LIST OF PREDICTORS FOUND
```

```
##  [1] 1 0 1 0 0 1 0 1 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 1 1 1 0
```

```
best.aic    # AIC VALUE
```

```
## [1] -418.9421
```

```
#aics       # VECTOR OF AIC VALUES

## PLOT OF AIC VALUES
plot(aics,ylim=c(-420,-360),type="n",ylab="AIC", xlab="Iteration")
lines(aics)
```




```
(1:2001)[aics==min(aics)]
```

```
## [1] 1555 1556 1557 1558 1559 1560 1561 1562 1563
```

For cooling schedule 1 and 3, they have the same initial temperature, but schedule 3 cools faster than 1 and fewer steps when temperature is high, more steps when temperature is low. Hence, schedule 3 get better result than schedule, which is -415.8437 versus -418.9421. For cooling schedule 2, it has a high initial temperature, hence the AIC is the highest among three schedules, which is -412.8295.

b.

2-neighborhoods

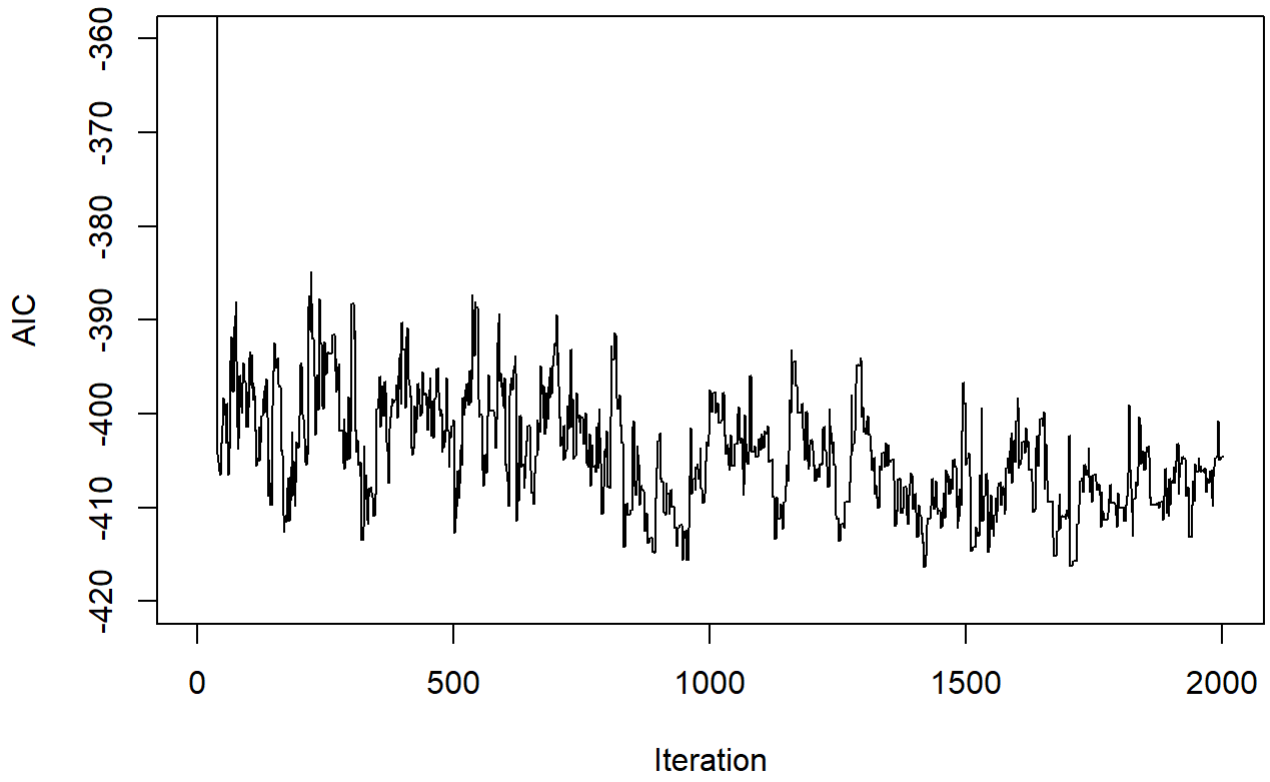
```
cooling = c(rep(60,5),rep(120,5),rep(220,5))
tau.start = 10
tau = rep(tau.start,15)
aics = NULL
# INITIALIZES STARTING RUN, TEMPERATURE SCHEDULE
set.seed(1999)
run = rbinom(m,1,.5)
run.current = run
run.vars = baseball.sub[,run.current==1]
g = lm(salary.log~.,run.vars)
run.aic = extractAIC(g)[2]
best.aic = run.aic
aics = run.aic
for(j in 2:15){tau[j] = 0.9*tau[j-1]}

## MAIN
for(j in 1:15){

    # RANDOMLY SELECTS A PREDICTOR TO ADD/REMOVE FROM THE MODEL
    # AND ACCEPTS THE NEW MODEL IF IT IS BETTER OR WITH PROBABILITY p
    for(i in 1:cooling[j]){
        pos = sample(1:m,2)
        run.step = run.current
        run.step[pos] = !run.current[pos]
        run.vars = baseball.sub[,run.step==1]
        g = lm(salary.log~.,run.vars)
        run.step.aic = extractAIC(g)[2]
        p = min(1,exp((run.aic-extractAIC(g)[2])/tau[j]))
        if(run.step.aic < run.aic){
            run.current = run.step
            run.aic = run.step.aic}
        if(rbinom(1,1,p)){
            run.current = run.step
            run.aic = run.step.aic}
        if(run.step.aic < best.aic){
            run = run.step
            best.aic = run.step.aic}
        aics = c(aics,run.aic)
    }
}

## OUTPUT
run         # BEST LIST OF PREDICTORS FOUND
```

```
##  [1] 1 0 1 0 0 0 0 1 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 1 0 0 0
```

```
best.aic    # AIC VALUE
```

```
## [1] -416.3802
```

```
#aics       # VECTOR OF AIC VALUES

## PLOT OF AIC VALUES
plot(aics,ylim=c(-420,-360),type="n",ylab="AIC", xlab="Iteration")
lines(aics)
```




```
(1:2001)[aics==min(aics)]
```

```
## [1] 1418 1419 1420
```

3-neighborhoods

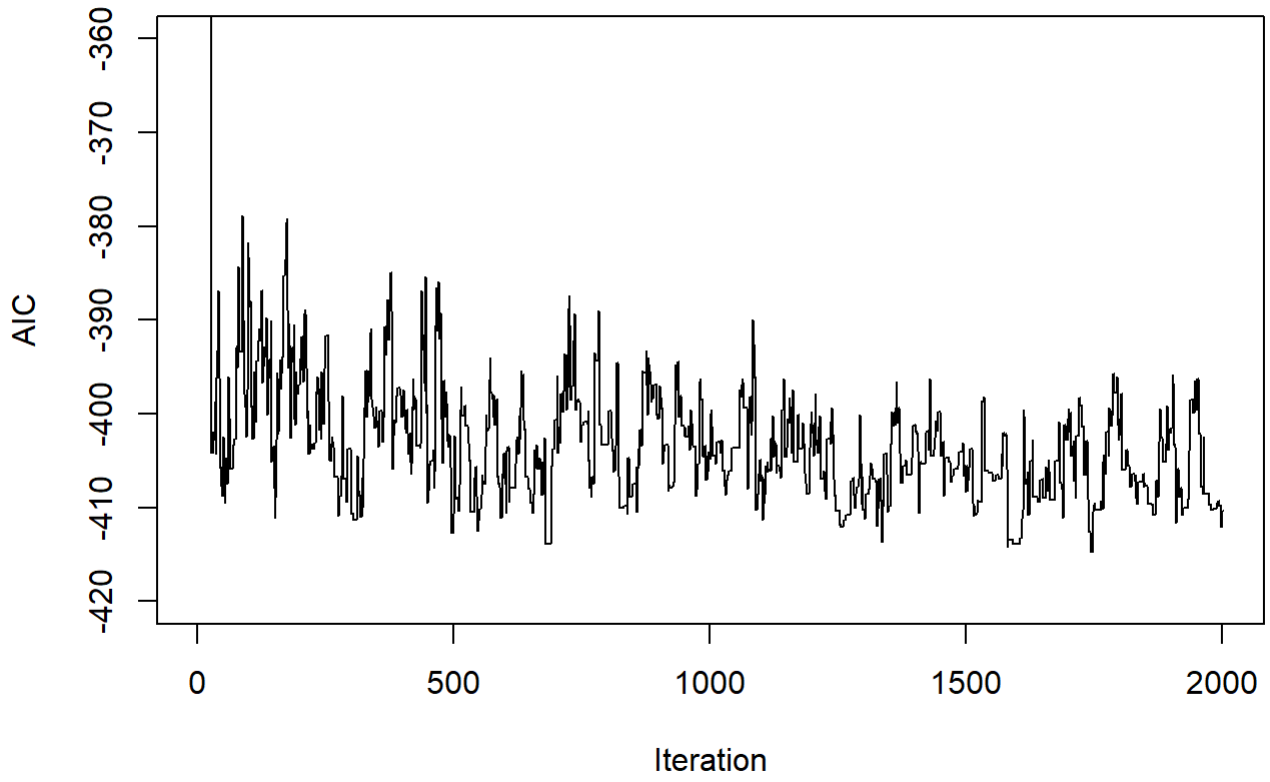
```
cooling = c(rep(60,5),rep(120,5),rep(220,5))
tau.start = 10
tau = rep(tau.start,15)
aics = NULL
# INITIALIZES STARTING RUN, TEMPERATURE SCHEDULE
set.seed(1999)
run = rbinom(m,1,.5)
run.current = run
run.vars = baseball.sub[,run.current==1]
g = lm(salary.log~.,run.vars)
run.aic = extractAIC(g)[2]
best.aic = run.aic
aics = run.aic
for(j in 2:15){tau[j] = 0.9*tau[j-1]}

## MAIN
for(j in 1:15){

    # RANDOMLY SELECTS A PREDICTOR TO ADD/REMOVE FROM THE MODEL
    # AND ACCEPTS THE NEW MODEL IF IT IS BETTER OR WITH PROBABILITY p
    for(i in 1:cooling[j]){
        pos = sample(1:m,3)
        run.step = run.current
        run.step[pos] = !run.current[pos]
        run.vars = baseball.sub[,run.step==1]
        g = lm(salary.log~.,run.vars)
        run.step.aic = extractAIC(g)[2]
        p = min(1,exp((run.aic-extractAIC(g)[2])/tau[j]))
        if(run.step.aic < run.aic){
            run.current = run.step
            run.aic = run.step.aic}
        if(rbinom(1,1,p)){
            run.current = run.step
            run.aic = run.step.aic}
        if(run.step.aic < best.aic){
            run = run.step
            best.aic = run.step.aic}
        aics = c(aics,run.aic)
    }
}

## OUTPUT
run         # BEST LIST OF PREDICTORS FOUND
```

```
##  [1] 0 0 1 0 0 0 0 1 0 1 0 0 1 1 1 1 1 0 0 0 0 0 0 0 1 1 0
```

```
best.aic    # AIC VALUE
```

```
## [1] -414.7412
```

```
#aics       # VECTOR OF AIC VALUES

## PLOT OF AIC VALUES
plot(aics,ylim=c(-420,-360),type="n",ylab="AIC", xlab="Iteration")
lines(aics)
```




```
(1:2001)[aics==min(aics)]
```

```
## [1] 1744 1745 1746 1747
```

The results show that 2-neighborhoods algorithm outperform the 3-neighborhoods algorithm. Their AICs are -416.3802 and -414.7412, respectively.

3.4

a.

For mutation rates $\mu = 0.01$,

```
P = 20
itr = 100
m.rate = .01
r = matrix(0,P,1)
phi = matrix(0,P,1)
runs = matrix(0,P,m)
runs.next = matrix(0,P,m)
runs.aic = matrix(0,P,1)
aics = matrix(0,P,itr)
run = NULL
best.aic = 0
best.aic.gen = rep(0,itr)

# INITIALIZES STARTING GENERATION, FITNESS VALUES
set.seed(3219553)
for(i in 1:P){
    runs[i,] = rbinom(m,1,.5)
    run.vars = baseball.sub[,runs[i,]==1]
    g = lm(salary.log~.,run.vars)
    runs.aic[i] = extractAIC(g)[2]
    aics[i,1] = runs.aic[i]
    if(runs.aic[i] < best.aic){
        run = runs[i,]
        best.aic = runs.aic[i]
    }
}
r = rank(-runs.aic)
phi = 2*r/(P*(P+1))
best.aic.gen[1]=best.aic

## MAIN
for(j in 1:itr-1){

    # BUILDS THE NEW GENERATION, SELECTING FIRST PARENT BASED ON
    # FITNESS AND THE SECOND PARENT AT RANDOM
    for(i in 1:10){
        parent.1 = runs[sample(1:P,1,prob=phi),]
        parent.2 = runs[sample(1:P,1),]
        pos = sample(1:(m-1),1)
        mutate = rbinom(m,1,m.rate)
        runs.next[i,] = c(parent.1[1:pos],parent.2[(pos+1):m])
        runs.next[i,] = (runs.next[i,]+mutate)%%2
        mutate = rbinom(m,1,m.rate)
        runs.next[P+1-i,] = c(parent.2[1:pos],parent.1[(pos+1):m])
        runs.next[P+1-i,] = (runs.next[P+1-i,]+mutate)%%2
    }
    runs = runs.next

    # UPDATES AIC VALUES, FITNESS VALUES FOR NEW GENERATION
    for(i in 1:P){
        run.vars = baseball.sub[,runs[i,]==1]
        g = lm(salary.log~.,run.vars)
```

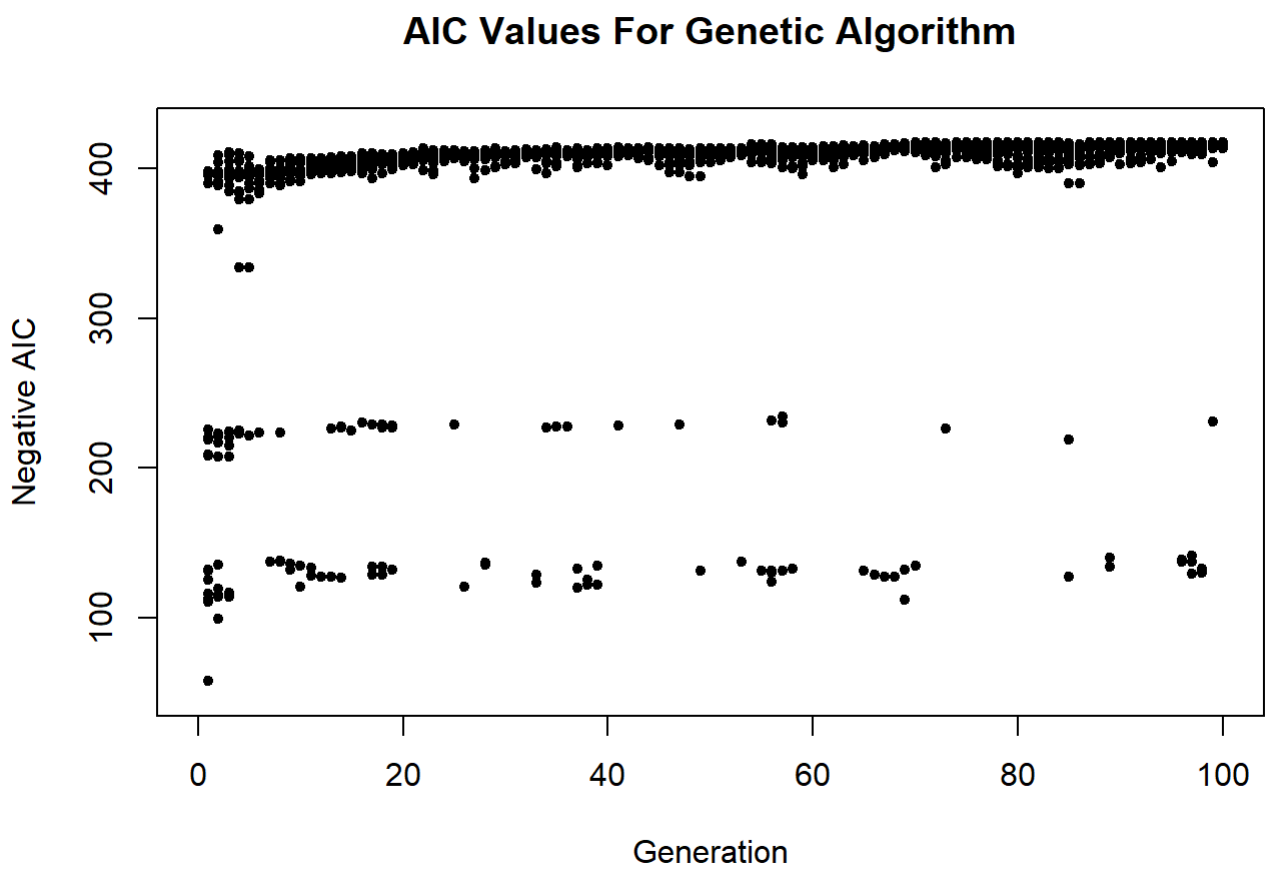
```
        runs.aic[i] = extractAIC(g)[2]
        aics[i,j+1] = runs.aic[i]
        if(runs.aic[i] < best.aic){
            run = runs[i,]
            best.aic = runs.aic[i]
        }
    }
    best.aic.gen[j+1]=best.aic
    r = rank(-runs.aic)
    phi = 2*r/(P*(P+1))
}

## OUTPUT
run         # BEST LIST OF PREDICTORS FOUND
```

```
##  [1] 0 1 1 0 0 0 0 1 1 1 0 0 1 1 1 1 1 0 0 0 0 0 0 1 1 1 0
```

```
best.aic    # AIC VALUE
```

```
## [1] -417.5816
```

```
## PLOT OF AIC VALUES
plot(-aics,xlim=c(0,itr),ylim=c(50,425),type="n",ylab="Negative AIC",
    xlab="Generation",main="AIC Values For Genetic Algorithm")
for(i in 1:itr){points(rep(i,P),-aics[,i],pch=20)}
```



For mutation rates $\mu = 0.001$,

```
P = 20
itr = 100
m.rate = .001
r = matrix(0,P,1)
phi = matrix(0,P,1)
runs = matrix(0,P,m)
runs.next = matrix(0,P,m)
runs.aic = matrix(0,P,1)
aics = matrix(0,P,itr)
run = NULL
best.aic = 0
best.aic.gen = rep(0,itr)

# INITIALIZES STARTING GENERATION, FITNESS VALUES
set.seed(3219553)
for(i in 1:P){
    runs[i,] = rbinom(m,1,.5)
    run.vars = baseball.sub[,runs[i,]==1]
    g = lm(salary.log~.,run.vars)
    runs.aic[i] = extractAIC(g)[2]
    aics[i,1] = runs.aic[i]
    if(runs.aic[i] < best.aic){
        run = runs[i,]
        best.aic = runs.aic[i]
    }
}
r = rank(-runs.aic)
phi = 2*r/(P*(P+1))
best.aic.gen[1]=best.aic

## MAIN
for(j in 1:itr-1){

    # BUILDS THE NEW GENERATION, SELECTING FIRST PARENT BASED ON
    # FITNESS AND THE SECOND PARENT AT RANDOM
    for(i in 1:10){
        parent.1 = runs[sample(1:P,1,prob=phi),]
        parent.2 = runs[sample(1:P,1),]
        pos = sample(1:(m-1),1)
        mutate = rbinom(m,1,m.rate)
        runs.next[i,] = c(parent.1[1:pos],parent.2[(pos+1):m])
        runs.next[i,] = (runs.next[i,]+mutate)%%2
        mutate = rbinom(m,1,m.rate)
        runs.next[P+1-i,] = c(parent.2[1:pos],parent.1[(pos+1):m])
        runs.next[P+1-i,] = (runs.next[P+1-i,]+mutate)%%2
    }
    runs = runs.next

    # UPDATES AIC VALUES, FITNESS VALUES FOR NEW GENERATION
    for(i in 1:P){
        run.vars = baseball.sub[,runs[i,]==1]
        g = lm(salary.log~.,run.vars)
```

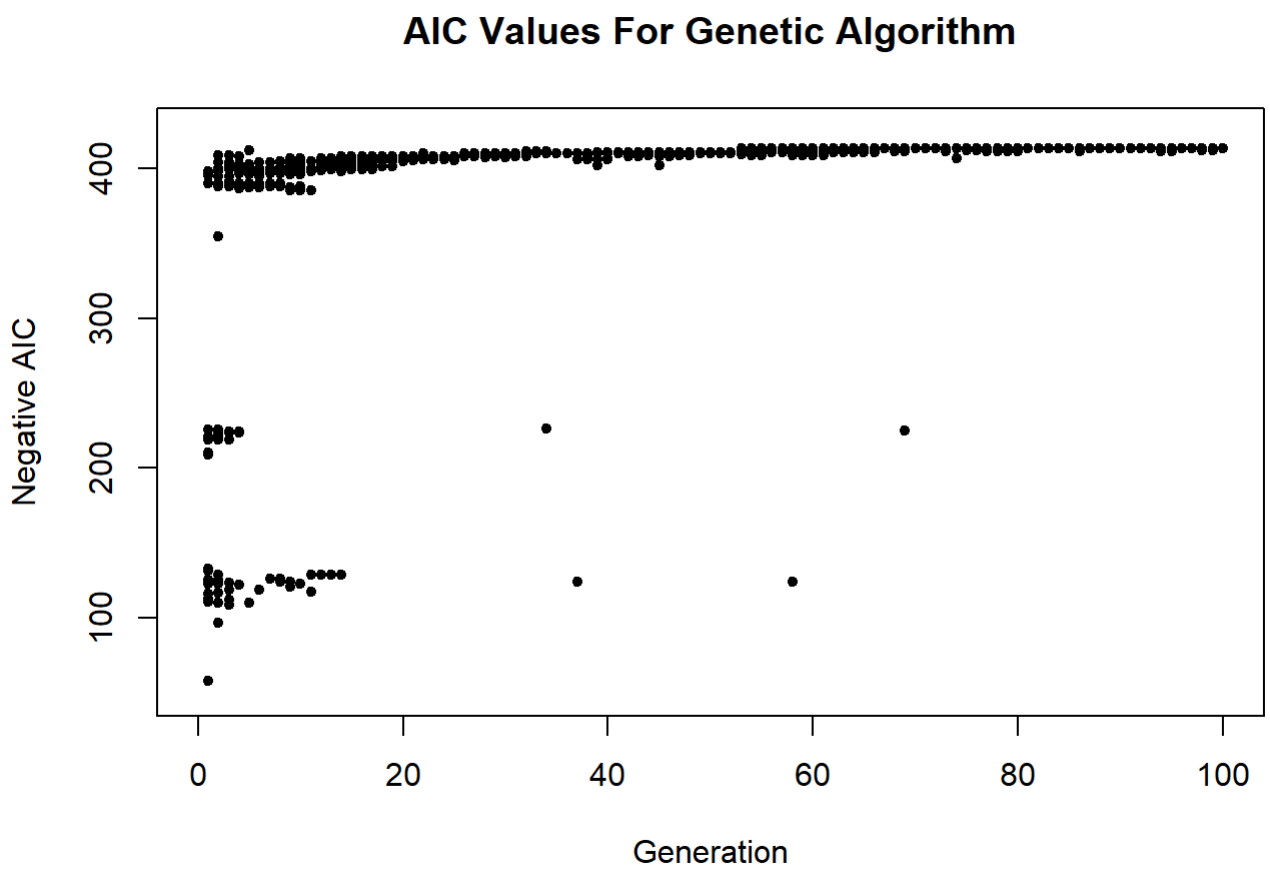
```
        runs.aic[i] = extractAIC(g)[2]
        aics[i,j+1] = runs.aic[i]
        if(runs.aic[i] < best.aic){
            run = runs[i,]
            best.aic = runs.aic[i]
        }
    }
    best.aic.gen[j+1]=best.aic
    r = rank(-runs.aic)
    phi = 2*r/(P*(P+1))
}

## OUTPUT
run         # BEST LIST OF PREDICTORS FOUND
```

```
##  [1] 0 0 1 1 0 0 1 1 0 1 1 0 1 1 1 1 1 1 0 0 0 0 0 1 0 0 1
```

```
best.aic    # AIC VALUE
```

```
## [1] -413.4002
```

```
## PLOT OF AIC VALUES
plot(-aics,xlim=c(0,itr),ylim=c(50,425),type="n",ylab="Negative AIC",
    xlab="Generation",main="AIC Values For Genetic Algorithm")
for(i in 1:itr){points(rep(i,P),-aics[,i],pch=20)}
```



For mutation rates $\mu = 0.1$,

```
P = 20
itr = 100
m.rate = .1
r = matrix(0,P,1)
phi = matrix(0,P,1)
runs = matrix(0,P,m)
runs.next = matrix(0,P,m)
runs.aic = matrix(0,P,1)
aics = matrix(0,P,itr)
run = NULL
best.aic = 0
best.aic.gen = rep(0,itr)

# INITIALIZES STARTING GENERATION, FITNESS VALUES
set.seed(3219553)
for(i in 1:P){
    runs[i,] = rbinom(m,1,.5)
    run.vars = baseball.sub[,runs[i,]==1]
    g = lm(salary.log~.,run.vars)
    runs.aic[i] = extractAIC(g)[2]
    aics[i,1] = runs.aic[i]
    if(runs.aic[i] < best.aic){
        run = runs[i,]
        best.aic = runs.aic[i]
    }
}
r = rank(-runs.aic)
phi = 2*r/(P*(P+1))
best.aic.gen[1]=best.aic

## MAIN
for(j in 1:itr-1){

    # BUILDS THE NEW GENERATION, SELECTING FIRST PARENT BASED ON
    # FITNESS AND THE SECOND PARENT AT RANDOM
    for(i in 1:10){
        parent.1 = runs[sample(1:P,1,prob=phi),]
        parent.2 = runs[sample(1:P,1),]
        pos = sample(1:(m-1),1)
        mutate = rbinom(m,1,m.rate)
        runs.next[i,] = c(parent.1[1:pos],parent.2[(pos+1):m])
        runs.next[i,] = (runs.next[i,]+mutate)%%2
        mutate = rbinom(m,1,m.rate)
        runs.next[P+1-i,] = c(parent.2[1:pos],parent.1[(pos+1):m])
        runs.next[P+1-i,] = (runs.next[P+1-i,]+mutate)%%2
    }
    runs = runs.next

    # UPDATES AIC VALUES, FITNESS VALUES FOR NEW GENERATION
    for(i in 1:P){
        run.vars = baseball.sub[,runs[i,]==1]
        g = lm(salary.log~.,run.vars)
```

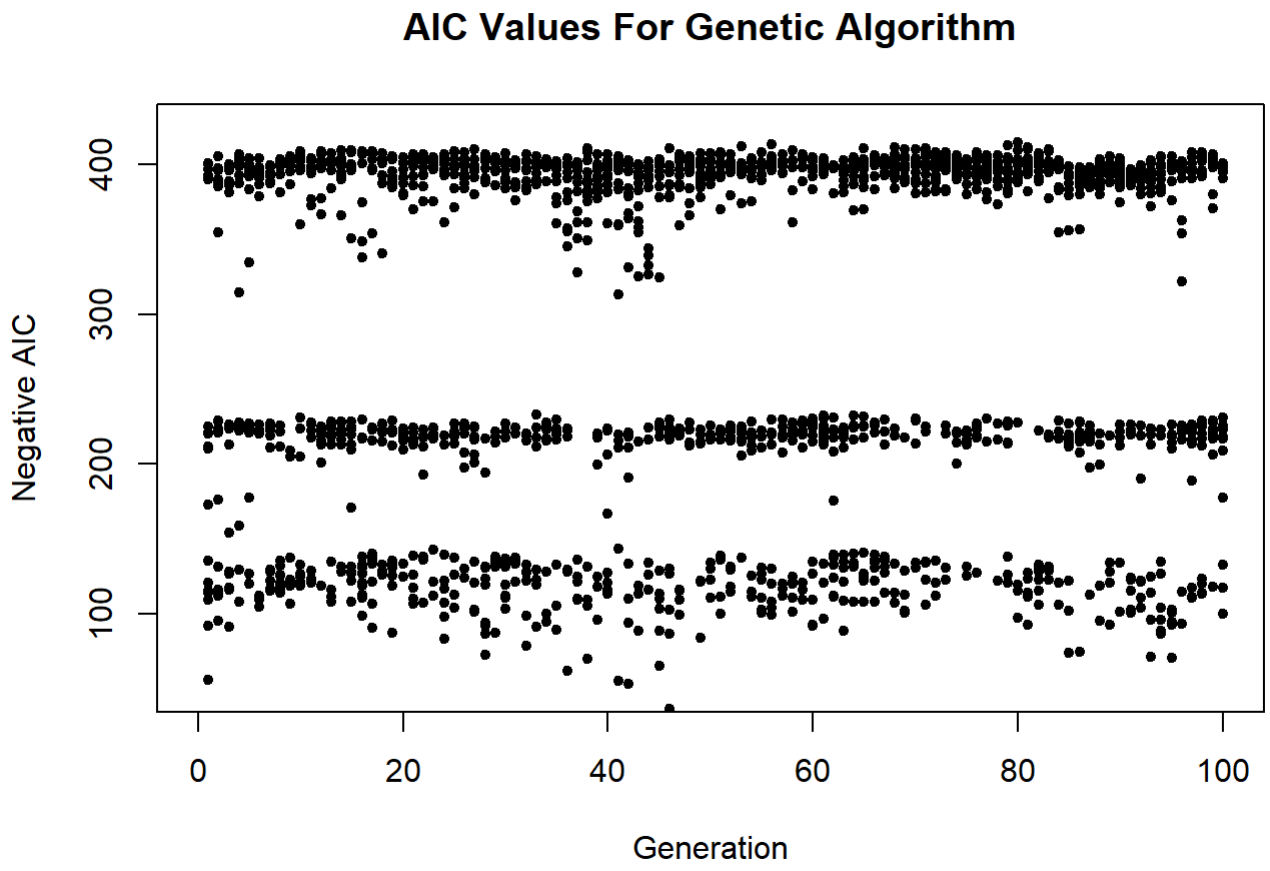
```
        runs.aic[i] = extractAIC(g)[2]
        aics[i,j+1] = runs.aic[i]
        if(runs.aic[i] < best.aic){
            run = runs[i,]
            best.aic = runs.aic[i]
        }
    }
    best.aic.gen[j+1]=best.aic
    r = rank(-runs.aic)
    phi = 2*r/(P*(P+1))
}

## OUTPUT
run         # BEST LIST OF PREDICTORS FOUND
```

```
##  [1] 0 1 1 0 0 1 0 1 0 1 0 1 1 1 1 1 0 0 0 0 0 1 0 0 1 1 1
```

```
best.aic    # AIC VALUE
```

```
## [1] -414.3174
```

```
## PLOT OF AIC VALUES
plot(-aics,xlim=c(0,itr),ylim=c(50,425),type="n",ylab="Negative AIC",
    xlab="Generation",main="AIC Values For Genetic Algorithm")
for(i in 1:itr){points(rep(i,P),-aics[,i],pch=20)}
```



For mutation rate $\mu = 0.01$, the algorithm find the solution AIC = -417.5816, which indicates it is an appropriate mutation rate.

For mutation rate $\mu = 0.001$, the algorithm find the solution AIC = -413.4002. The graph shows that all the chromosome in the later generation are all the same, which is because the mutation rate is too low that only a few candidate solution has been searched, and the iterations soon converge to a local minimum.

For mutation rate $\mu = 0.1$, the algorithm find the solution AIC = -414.3174. The graph shows that even in the 100th generation, there are many chromosome that has low negative AIC, which is because the mutation rate is too high and causes the unstable of solution.

b.

For generation sizes $P = 20$,

```
P = 20
itr = 100
m.rate = .01
r = matrix(0,P,1)
phi = matrix(0,P,1)
runs = matrix(0,P,m)
runs.next = matrix(0,P,m)
runs.aic = matrix(0,P,1)
aics = matrix(0,P,itr)
run = NULL
best.aic = 0
best.aic.gen = rep(0,itr)

# INITIALIZES STARTING GENERATION, FITNESS VALUES
set.seed(3219553)
for(i in 1:P){
    runs[i,] = rbinom(m,1,.5)
    run.vars = baseball.sub[,runs[i,]==1]
    g = lm(salary.log~.,run.vars)
    runs.aic[i] = extractAIC(g)[2]
    aics[i,1] = runs.aic[i]
    if(runs.aic[i] < best.aic){
        run = runs[i,]
        best.aic = runs.aic[i]
    }
}
r = rank(-runs.aic)
phi = 2*r/(P*(P+1))
best.aic.gen[1]=best.aic

## MAIN
for(j in 1:itr-1){

    # BUILDS THE NEW GENERATION, SELECTING FIRST PARENT BASED ON
    # FITNESS AND THE SECOND PARENT AT RANDOM
    for(i in 1:10){
        parent.1 = runs[sample(1:P,1,prob=phi),]
        parent.2 = runs[sample(1:P,1),]
        pos = sample(1:(m-1),1)
        mutate = rbinom(m,1,m.rate)
        runs.next[i,] = c(parent.1[1:pos],parent.2[(pos+1):m])
        runs.next[i,] = (runs.next[i,]+mutate)%%2
        mutate = rbinom(m,1,m.rate)
        runs.next[P+1-i,] = c(parent.2[1:pos],parent.1[(pos+1):m])
        runs.next[P+1-i,] = (runs.next[P+1-i,]+mutate)%%2
    }
    runs = runs.next

    # UPDATES AIC VALUES, FITNESS VALUES FOR NEW GENERATION
    for(i in 1:P){
        run.vars = baseball.sub[,runs[i,]==1]
        g = lm(salary.log~.,run.vars)
```

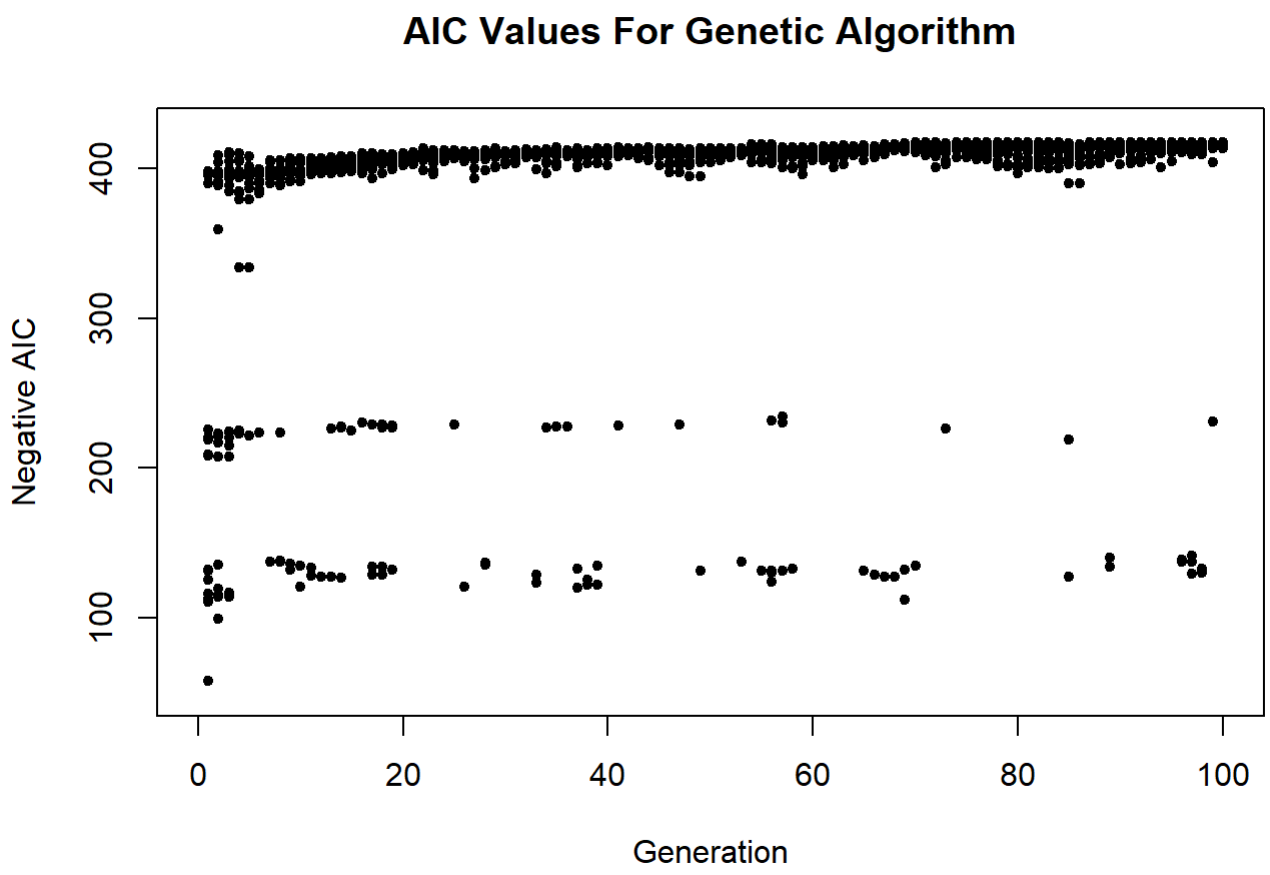
```
        runs.aic[i] = extractAIC(g)[2]
        aics[i,j+1] = runs.aic[i]
        if(runs.aic[i] < best.aic){
            run = runs[i,]
            best.aic = runs.aic[i]
        }
    }
    best.aic.gen[j+1]=best.aic
    r = rank(-runs.aic)
    phi = 2*r/(P*(P+1))
}

## OUTPUT
run         # BEST LIST OF PREDICTORS FOUND
```

```
##  [1] 0 1 1 0 0 0 0 1 1 1 0 0 1 1 1 1 1 0 0 0 0 0 0 1 1 1 0
```

```
best.aic    # AIC VALUE
```

```
## [1] -417.5816
```

```
## PLOT OF AIC VALUES
plot(-aics,xlim=c(0,itr),ylim=c(50,425),type="n",ylab="Negative AIC",
    xlab="Generation",main="AIC Values For Genetic Algorithm")
for(i in 1:itr){points(rep(i,P),-aics[,i],pch=20)}
```



For generation sizes $P = 10$,

```
P = 10
itr = 100
m.rate = .01
r = matrix(0,P,1)
phi = matrix(0,P,1)
runs = matrix(0,P,m)
runs.next = matrix(0,P,m)
runs.aic = matrix(0,P,1)
aics = matrix(0,P,itr)
run = NULL
best.aic = 0
best.aic.gen = rep(0,itr)

# INITIALIZES STARTING GENERATION, FITNESS VALUES
set.seed(3219553)
for(i in 1:P){
    runs[i,] = rbinom(m,1,.5)
    run.vars = baseball.sub[,runs[i,]==1]
    g = lm(salary.log~.,run.vars)
    runs.aic[i] = extractAIC(g)[2]
    aics[i,1] = runs.aic[i]
    if(runs.aic[i] < best.aic){
        run = runs[i,]
        best.aic = runs.aic[i]
    }
}
r = rank(-runs.aic)
phi = 2*r/(P*(P+1))
best.aic.gen[1]=best.aic

## MAIN
for(j in 1:itr-1){

    # BUILDS THE NEW GENERATION, SELECTING FIRST PARENT BASED ON
    # FITNESS AND THE SECOND PARENT AT RANDOM
    for(i in 1:5){
        parent.1 = runs[sample(1:P,1,prob=phi),]
        parent.2 = runs[sample(1:P,1),]
        pos = sample(1:(m-1),1)
        mutate = rbinom(m,1,m.rate)
        runs.next[i,] = c(parent.1[1:pos],parent.2[(pos+1):m])
        runs.next[i,] = (runs.next[i,]+mutate)%%2
        mutate = rbinom(m,1,m.rate)
        runs.next[P+1-i,] = c(parent.2[1:pos],parent.1[(pos+1):m])
        runs.next[P+1-i,] = (runs.next[P+1-i,]+mutate)%%2
    }
    runs = runs.next

    # UPDATES AIC VALUES, FITNESS VALUES FOR NEW GENERATION
    for(i in 1:P){
        run.vars = baseball.sub[,runs[i,]==1]
        g = lm(salary.log~.,run.vars)
```

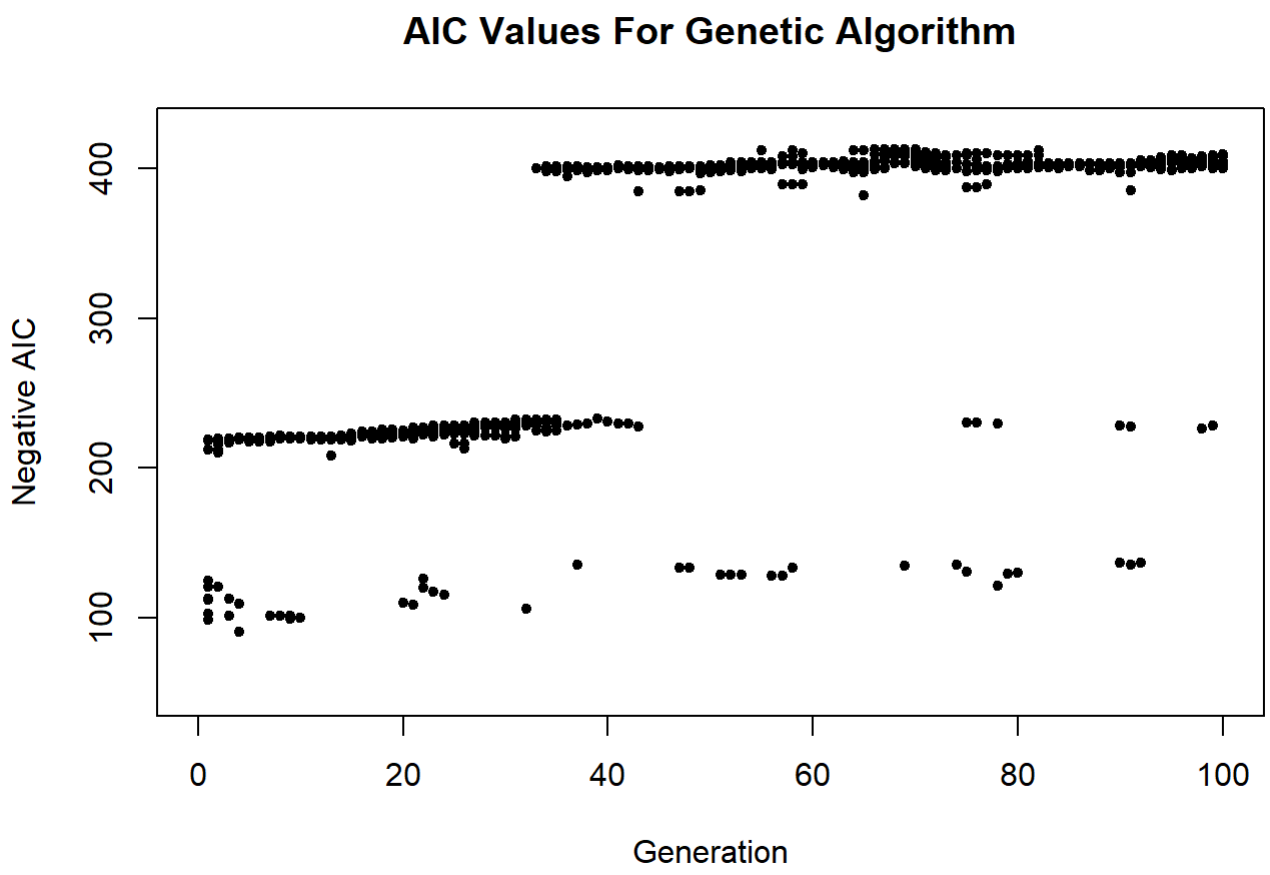
```
        runs.aic[i] = extractAIC(g)[2]
        aics[i,j+1] = runs.aic[i]
        if(runs.aic[i] < best.aic){
            run = runs[i,]
            best.aic = runs.aic[i]
        }
    }
    best.aic.gen[j+1]=best.aic
    r = rank(-runs.aic)
    phi = 2*r/(P*(P+1))
}

## OUTPUT
run         # BEST LIST OF PREDICTORS FOUND
```

```
##  [1] 1 0 1 1 0 0 0 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 1 1 1 0
```

```
best.aic    # AIC VALUE
```

```
## [1] -412.7442
```

```
## PLOT OF AIC VALUES
plot(-aics,xlim=c(0,itr),ylim=c(50,425),type="n",ylab="Negative AIC",
    xlab="Generation",main="AIC Values For Genetic Algorithm")
for(i in 1:itr){points(rep(i,P),-aics[,i],pch=20)}
```



For generation sizes $P = 40$,

```
P = 40
itr = 100
m.rate = .01
r = matrix(0,P,1)
phi = matrix(0,P,1)
runs = matrix(0,P,m)
runs.next = matrix(0,P,m)
runs.aic = matrix(0,P,1)
aics = matrix(0,P,itr)
run = NULL
best.aic = 0
best.aic.gen = rep(0,itr)

# INITIALIZES STARTING GENERATION, FITNESS VALUES
set.seed(3219553)
for(i in 1:P){
    runs[i,] = rbinom(m,1,.5)
    run.vars = baseball.sub[,runs[i,]==1]
    g = lm(salary.log~.,run.vars)
    runs.aic[i] = extractAIC(g)[2]
    aics[i,1] = runs.aic[i]
    if(runs.aic[i] < best.aic){
        run = runs[i,]
        best.aic = runs.aic[i]
    }
}
r = rank(-runs.aic)
phi = 2*r/(P*(P+1))
best.aic.gen[1]=best.aic

## MAIN
for(j in 1:itr-1){

    # BUILDS THE NEW GENERATION, SELECTING FIRST PARENT BASED ON
    # FITNESS AND THE SECOND PARENT AT RANDOM
    for(i in 1:20){
        parent.1 = runs[sample(1:P,1,prob=phi),]
        parent.2 = runs[sample(1:P,1),]
        pos = sample(1:(m-1),1)
        mutate = rbinom(m,1,m.rate)
        runs.next[i,] = c(parent.1[1:pos],parent.2[(pos+1):m])
        runs.next[i,] = (runs.next[i,]+mutate)%%2
        mutate = rbinom(m,1,m.rate)
        runs.next[P+1-i,] = c(parent.2[1:pos],parent.1[(pos+1):m])
        runs.next[P+1-i,] = (runs.next[P+1-i,]+mutate)%%2
    }
    runs = runs.next

    # UPDATES AIC VALUES, FITNESS VALUES FOR NEW GENERATION
    for(i in 1:P){
        run.vars = baseball.sub[,runs[i,]==1]
        g = lm(salary.log~.,run.vars)
```

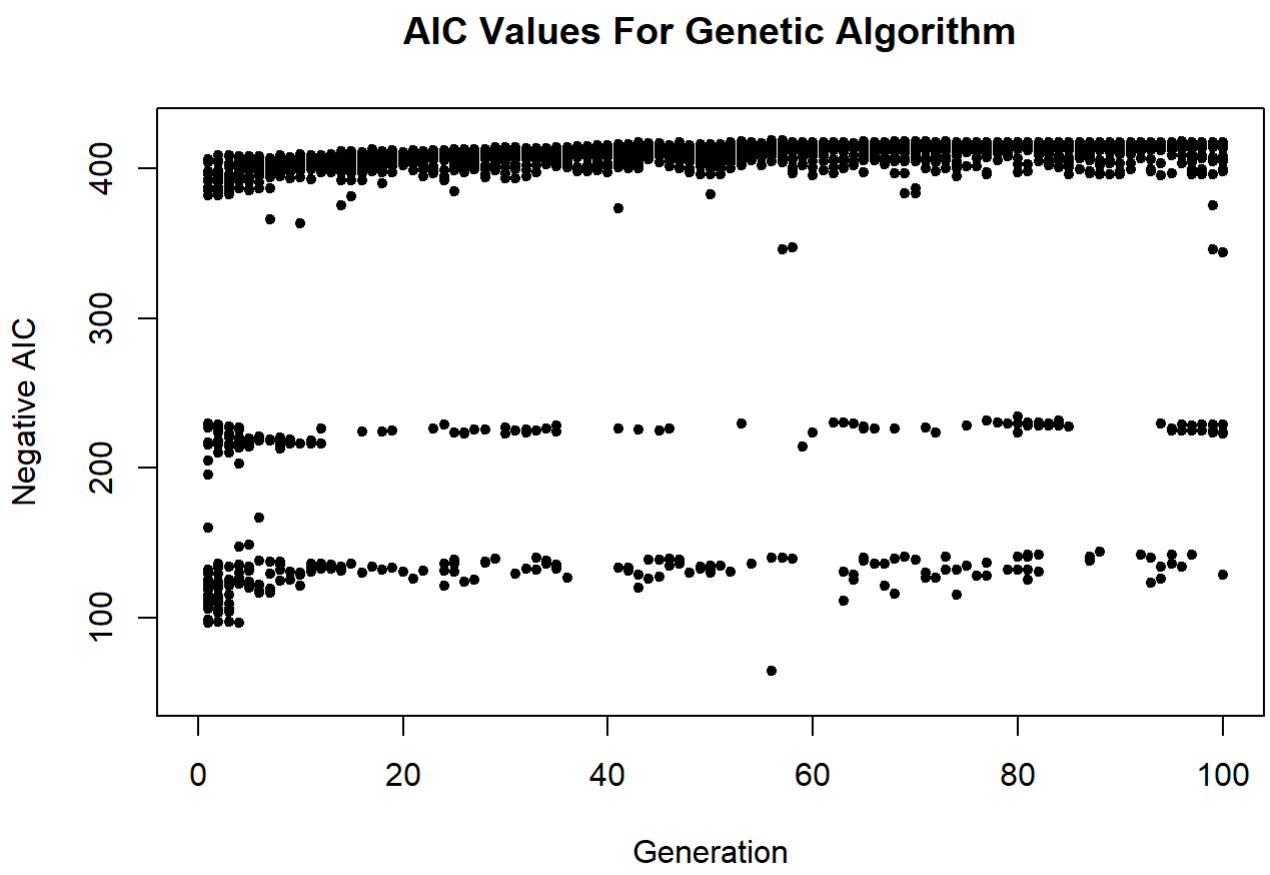
```
        runs.aic[i] = extractAIC(g)[2]
        aics[i,j+1] = runs.aic[i]
        if(runs.aic[i] < best.aic){
            run = runs[i,]
            best.aic = runs.aic[i]
        }
    }
    best.aic.gen[j+1]=best.aic
    r = rank(-runs.aic)
    phi = 2*r/(P*(P+1))
}

## OUTPUT
run         # BEST LIST OF PREDICTORS FOUND
```

```
##  [1] 1 0 1 0 0 0 0 1 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 1 1 1 0
```

```
best.aic    # AIC VALUE
```

```
## [1] -418.4511
```

```
## PLOT OF AIC VALUES
plot(-aics,xlim=c(0,itr),ylim=c(50,425),type="n",ylab="Negative AIC",
    xlab="Generation",main="AIC Values For Genetic Algorithm")
for(i in 1:itr){points(rep(i,P),-aics[,i],pch=20)}
```



For generation size $P = 10, 20, 40$, the algorithm find the solution AIC = -412.7442,-417.5816,-418.4511,respectively. This results show that the more chromosome in one generation, the more candidate solution the algorithm has been searched, hence will get better solution.

C.

i.

To scale the fitness function, consider

$$\phi\left(\boldsymbol{\vartheta}_i^{(t)}\right) = af\left(\boldsymbol{\theta}_i^{(t)}\right) + b$$

and let a is the reciprocal of negative standard deviation of all AICs in t-th generation, and b is the scaled negative smallest AIC in t-th generation.

```
P = 20
itr = 100
m.rate = .01
r = matrix(0,P,1)
phi = matrix(0,P,1)
runs = matrix(0,P,m)
runs.next = matrix(0,P,m)
runs.aic = matrix(0,P,1)
aics = matrix(0,P,itr)
run = NULL
best.aic = 0
best.aic.gen = rep(0,itr)

# INITIALIZES STARTING GENERATION, FITNESS VALUES
set.seed(3219553)
for(i in 1:P){
    runs[i,] = rbinom(m,1,.5)
    run.vars = baseball.sub[,runs[i,]==1]
    g = lm(salary.log~.,run.vars)
    runs.aic[i] = extractAIC(g)[2]
    aics[i,1] = runs.aic[i]
    if(runs.aic[i] < best.aic){
        run = runs[i,]
        best.aic = runs.aic[i]
    }
}
a <- -1/sd(runs.aic)
b <- -max(runs.aic)*a
scale.fit <- a*runs.aic + b
phi <- scale.fit/sum(scale.fit)
#r = rank(-runs.aic)
#phi = 2*r/(P*(P+1))
best.aic.gen[1]=best.aic

## MAIN
for(j in 1:itr-1){

    # BUILDS THE NEW GENERATION, SELECTING FIRST PARENT BASED ON
    # FITNESS AND THE SECOND PARENT AT RANDOM
    for(i in 1:10){
        parent.1 = runs[sample(1:P,1,prob=phi),]
        parent.2 = runs[sample(1:P,1),]
        pos = sample(1:(m-1),1)
        mutate = rbinom(m,1,m.rate)
        runs.next[i,] = c(parent.1[1:pos],parent.2[(pos+1):m])
        runs.next[i,] = (runs.next[i,]+mutate)%%2
        mutate = rbinom(m,1,m.rate)
        runs.next[P+1-i,] = c(parent.2[1:pos],parent.1[(pos+1):m])
        runs.next[P+1-i,] = (runs.next[P+1-i,]+mutate)%%2
    }
    runs = runs.next
```

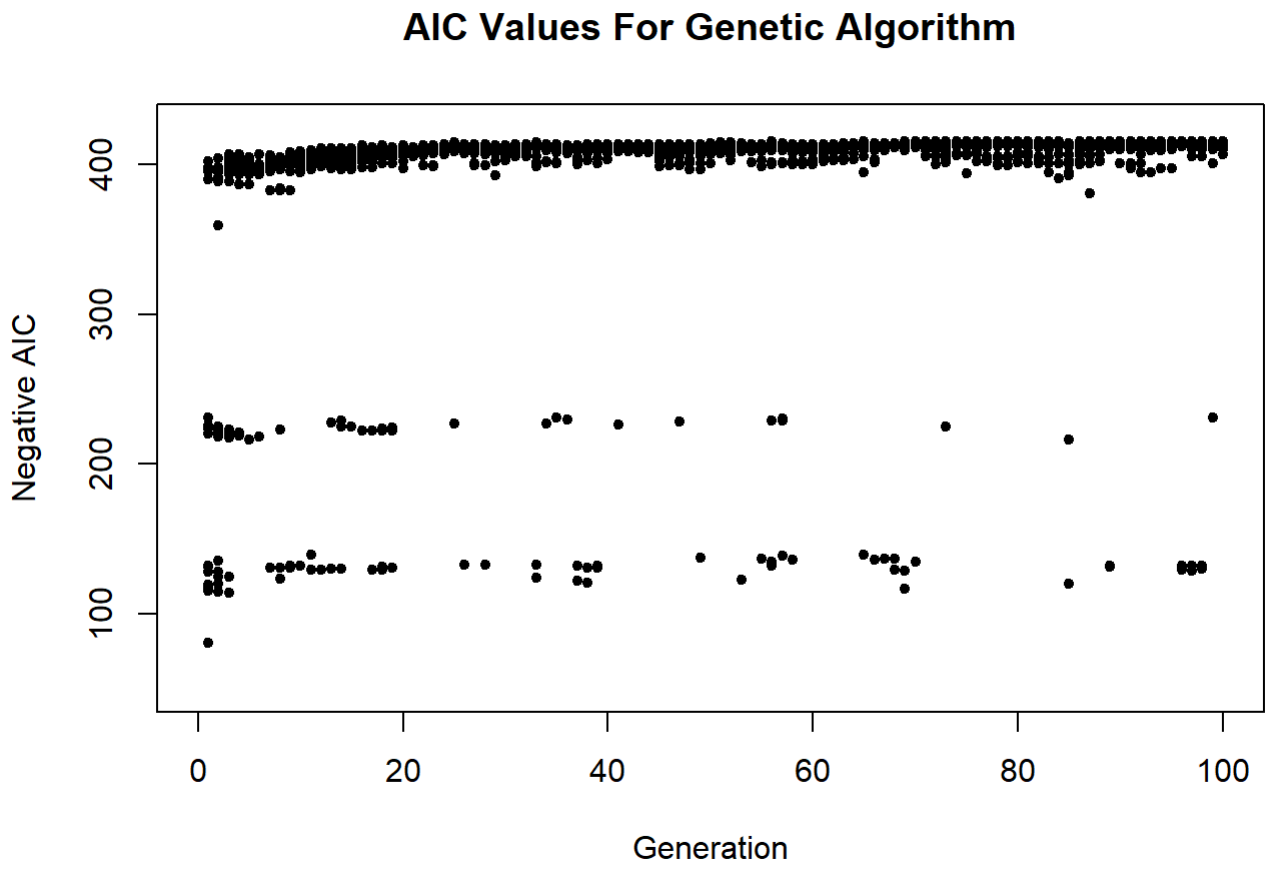
```
    # UPDATES AIC VALUES, FITNESS VALUES FOR NEW GENERATION
    for(i in 1:P){
        run.vars = baseball.sub[,runs[i,]==1]
        g = lm(salary.log~.,run.vars)
        runs.aic[i] = extractAIC(g)[2]
        aics[i,j+1] = runs.aic[i]
        if(runs.aic[i] < best.aic){
            run = runs[i,]
            best.aic = runs.aic[i]
        }
    }
    best.aic.gen[j+1]=best.aic
    r = rank(-runs.aic)
    phi = 2*r/(P*(P+1))
}

## OUTPUT
run         # BEST LIST OF PREDICTORS FOUND
```

```
##  [1] 1 0 1 0 0 1 0 1 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1
```

```
best.aic    # AIC VALUE
```

```
## [1] -415.48
```

```
## PLOT OF AIC VALUES
plot(-aics,xlim=c(0,itr),ylim=c(50,425),type="n",ylab="Negative AIC",
    xlab="Generation",main="AIC Values For Genetic Algorithm")
for(i in 1:itr){points(rep(i,P),-aics[,i],pch=20)}
```


AIC Values For Genetic Algorithm
Negative AIC
100
200
300
400
0
20
40
60
80
100
Generation

####

ii.

```
P = 20
itr = 100
m.rate = .01
r = matrix(0,P,1)
phi = matrix(0,P,1)
runs = matrix(0,P,m)
runs.next = matrix(0,P,m)
runs.aic = matrix(0,P,1)
aics = matrix(0,P,itr)
run = NULL
best.aic = 0
best.aic.gen = rep(0,itr)

# INITIALIZES STARTING GENERATION, FITNESS VALUES
set.seed(3219553)
for(i in 1:P){
    runs[i,] = rbinom(m,1,.5)
    run.vars = baseball.sub[,runs[i,]==1]
    g = lm(salary.log~.,run.vars)
    runs.aic[i] = extractAIC(g)[2]
    aics[i,1] = runs.aic[i]
    if(runs.aic[i] < best.aic){
        run = runs[i,]
        best.aic = runs.aic[i]
    }
}
a <- -1/sd(runs.aic)
b <- -max(runs.aic)*a
scale.fit <- a*runs.aic + b
phi <- scale.fit/sum(scale.fit)
#r = rank(-runs.aic)
#phi = 2*r/(P*(P+1))
best.aic.gen[1]=best.aic

## MAIN
for(j in 1:itr-1){

    # BUILDS THE NEW GENERATION, SELECTING FIRST PARENT BASED ON
    # FITNESS AND THE SECOND PARENT AT RANDOM
    for(i in 1:10){
        parent.1 = runs[sample(1:P,1,prob=phi),]
        parent.2 = runs[sample(1:P,1,prob=phi),]
        pos = sample(1:(m-1),1)
        mutate = rbinom(m,1,m.rate)
        runs.next[i,] = c(parent.1[1:pos],parent.2[(pos+1):m])
        runs.next[i,] = (runs.next[i,]+mutate)%%2
        mutate = rbinom(m,1,m.rate)
        runs.next[P+1-i,] = c(parent.2[1:pos],parent.1[(pos+1):m])
        runs.next[P+1-i,] = (runs.next[P+1-i,]+mutate)%%2
    }
    runs = runs.next
```

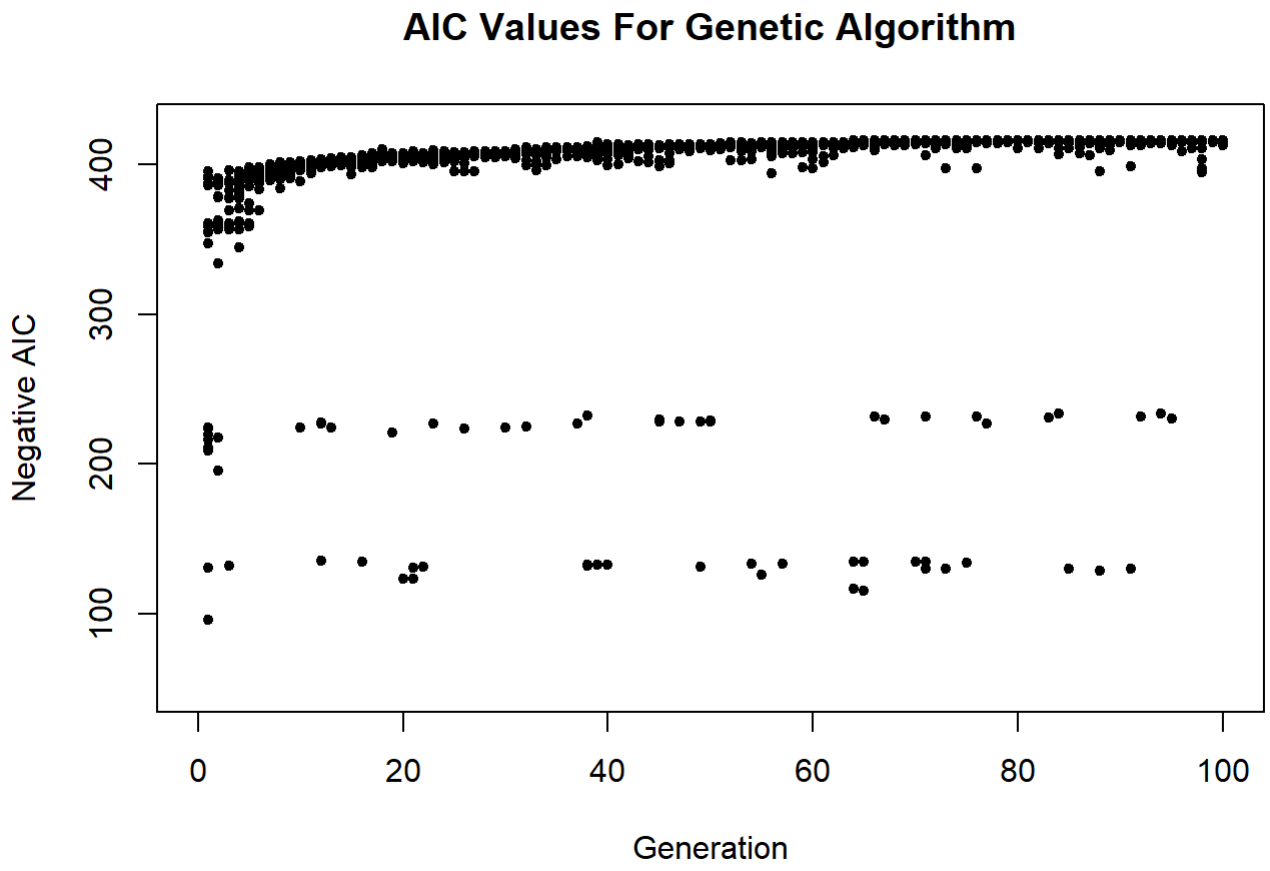
```
    # UPDATES AIC VALUES, FITNESS VALUES FOR NEW GENERATION
    for(i in 1:P){
        run.vars = baseball.sub[,runs[i,]==1]
        g = lm(salary.log~.,run.vars)
        runs.aic[i] = extractAIC(g)[2]
        aics[i,j+1] = runs.aic[i]
        if(runs.aic[i] < best.aic){
            run = runs[i,]
            best.aic = runs.aic[i]
        }
    }
    best.aic.gen[j+1]=best.aic
    r = rank(-runs.aic)
    phi = 2*r/(P*(P+1))
}

## OUTPUT
run         # BEST LIST OF PREDICTORS FOUND
```

```
##  [1] 0 0 1 0 0 0 0 1 0 1 0 1 1 1 1 1 0 0 0 1 0 1 0 0 0 0 0
```

```
best.aic    # AIC VALUE
```

```
## [1] -416.1567
```

```
## PLOT OF AIC VALUES
plot(-aics,xlim=c(0,itr),ylim=c(50,425),type="n",ylab="Negative AIC",
    xlab="Generation",main="AIC Values For Genetic Algorithm")
for(i in 1:itr){points(rep(i,P),-aics[,i],pch=20)}
```



iii.

Tournament selection with P/5 strata,

```
P = 20
itr = 100
m.rate = .01
r = matrix(0,P,1)
phi = matrix(0,P,1)
runs = matrix(0,P,m)
runs.next = matrix(0,P,m)
runs.aic = matrix(0,P,1)
aics = matrix(0,P,itr)
run = NULL
best.aic = 0
best.aic.gen = rep(0,itr)

# INITIALIZES STARTING GENERATION, FITNESS VALUES
set.seed(3219553)
for(i in 1:P){
    runs[i,] = rbinom(m,1,.5)
    run.vars = baseball.sub[,runs[i,]==1]
    g = lm(salary.log~.,run.vars)
    runs.aic[i] = extractAIC(g)[2]
    aics[i,1] = runs.aic[i]
    if(runs.aic[i] < best.aic){
        run = runs[i,]
        best.aic = runs.aic[i]
    }
}
best.aic.gen[1]=best.aic

## MAIN
for(j in 1:(itr-1)){
  set.seed(2022 + j)
    # BUILDS THE NEW GENERATION, SELECTING FIRST PARENT BASED ON
    # FITNESS AND THE SECOND PARENT AT RANDOM
    for(i in 1:5){
      cpt1 <- sample(1:P,5)
      cpt2 <- sample((1:P)[-cpt1],5)
      cpt3 <- sample((1:P)[-c(cpt1,cpt2)],5)
      cpt4 <- (1:P)[-c(cpt1,cpt2,cpt3)]
      aicmax1 <- min(aics[cpt1,j])
      aicmax2 <- min(aics[cpt2,j])
      aicmax3 <- min(aics[cpt3,j])
      aicmax4 <- min(aics[cpt4,j])

      parent.1 <- runs[cpt1,][which(aics[cpt1,j]==aicmax1),]

      location <- aics[cpt2,j]
      parent2 <- runs[cpt2,]
      parent.2 <- parent2[which(location==aicmax2),]

      location <- aics[cpt3,j]
      parent3 <- runs[cpt3,]
      parent.3 <- parent3[which(location==aicmax3),]
```

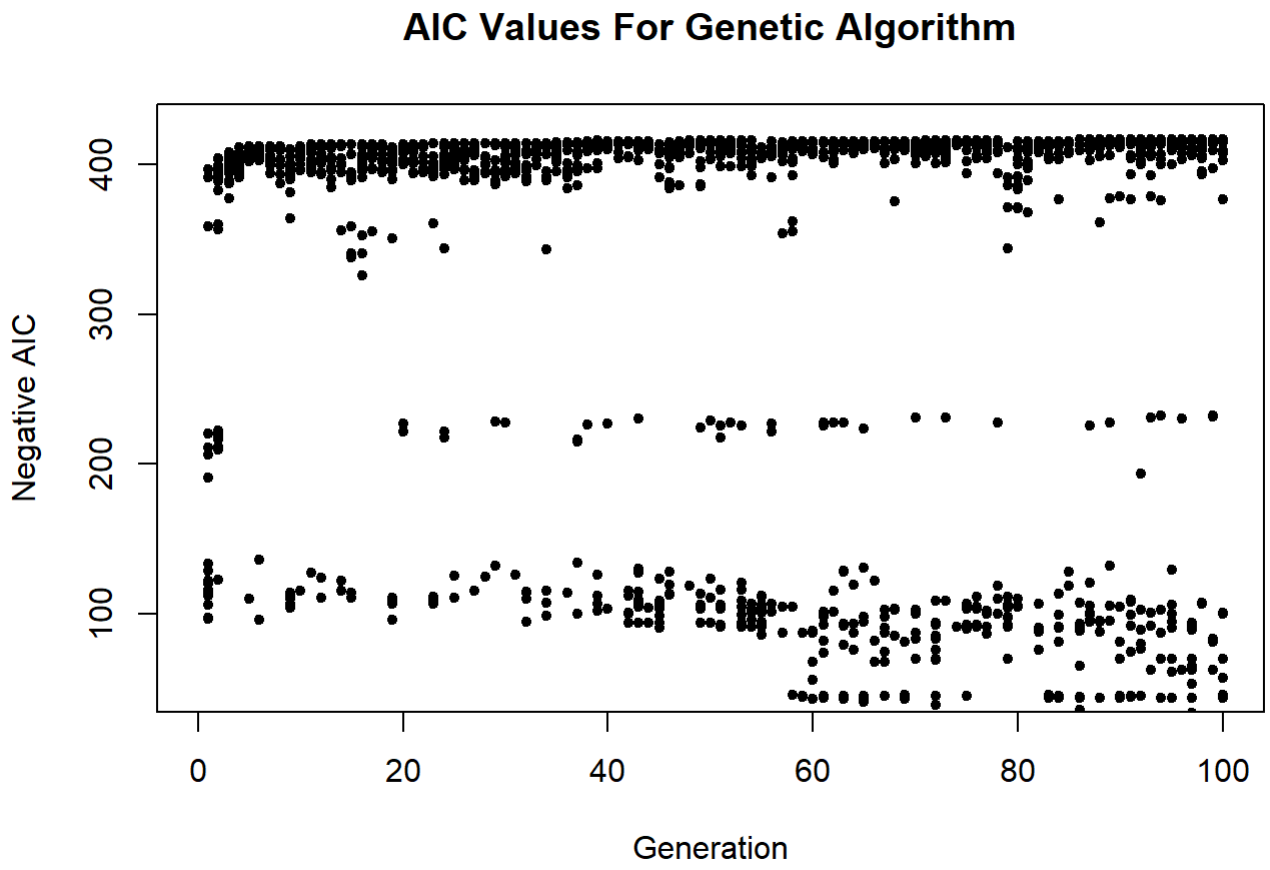
```
      location <- aics[cpt4,j]
      parent4 <- runs[cpt4,]
      parent.4 <- parent4[which(location==aicmax4),]

        pos = sample(1:(m-1),1)
        mutate = rbinom(m,1,m.rate)
        runs.next[i,] = c(parent.1[1:pos],parent.2[(pos+1):m])
        runs.next[i,] = (runs.next[i,]+mutate)%%2
        mutate = rbinom(m,1,m.rate)
        runs.next[P+1-i,] = c(parent.2[1:pos],parent.1[(pos+1):m])
        runs.next[P+1-i,] = (runs.next[P+1-i,]+mutate)%%2

        pos = sample(1:(m-1),1)
        mutate = rbinom(m,1,m.rate)
        runs.next[i+5,] = c(parent.3[1:pos],parent.4[(pos+1):m])
        runs.next[i+5,] = (runs.next[i+5,]+mutate)%%2
        mutate = rbinom(m,1,m.rate)
        runs.next[P-4-i,] = c(parent.4[1:pos],parent.3[(pos+1):m])
        runs.next[P-4-i,] = (runs.next[P-4-i,]+mutate)%%2
    }
    runs = runs.next

    # UPDATES AIC VALUES, FITNESS VALUES FOR NEW GENERATION
    for(i in 1:P){
        run.vars = baseball.sub[,runs[i,]==1]
        g = lm(salary.log~.,run.vars)
        runs.aic[i] = extractAIC(g)[2]
        aics[i,j+1] = runs.aic[i]
        if(runs.aic[i] < best.aic){
            run = runs[i,]
            best.aic = runs.aic[i]
        }
    }
    best.aic.gen[j+1]=best.aic
    #r = rank(-runs.aic)
    #phi = 2*r/(P*(P+1))
}

## OUTPUT
run         # BEST LIST OF PREDICTORS FOUND
```

```
##  [1] 0 0 1 0 0 0 0 1 0 1 0 0 1 1 1 1 1 0 0 1 0 1 0 0 1 1 0
```

```
best.aic    # AIC VALUE
```

```
## [1] -416.3201
```

```
## PLOT OF AIC VALUES
plot(-aics,xlim=c(0,itr),ylim=c(50,425),type="n",ylab="Negative AIC",
    xlab="Generation",main="AIC Values For Genetic Algorithm")
for(i in 1:itr){points(rep(i,P),-aics[,i],pch=20)}
```



mechanism i,ii,iii give the solution AIC = -415.48, -416.1567, -416.3201, respectively. This shows that randomly choose one parent will get the worst solution among three, and both parent chosen by proportional to fitness is better than random choose. The tournament selection get the best solution, but in the last generation it will remain many offspring that are not very competitive.

d.

```
P = 20
itr = 100
m.rate = .01
r = matrix(0,P,1)
phi = matrix(0,P,1)
runs = matrix(0,P,m)
runs.next = matrix(0,1,m)
runs.aic = matrix(0,P,1)
aics = matrix(0,P,itr)
run = NULL
best.aic = 0
best.aic.gen = rep(0,itr)

# INITIALIZES STARTING GENERATION, FITNESS VALUES
set.seed(3219553)
for(i in 1:P){
    runs[i,] = rbinom(m,1,.5)
    run.vars = baseball.sub[,runs[i,]==1]
    g = lm(salary.log~.,run.vars)
    runs.aic[i] = extractAIC(g)[2]
    aics[i,1] = runs.aic[i]
    if(runs.aic[i] < best.aic){
        run = runs[i,]
        best.aic = runs.aic[i]
    }
}
r = rank(-runs.aic)
phi = 2*r/(P*(P+1))
best.aic.gen[1]=best.aic

## MAIN
for(j in 1:(itr-1)){

    # BUILDS THE NEW GENERATION, SELECTING FIRST PARENT BASED ON
    # FITNESS AND THE SECOND PARENT AT RANDOM
    for(i in 1:1){
        parent.1 = runs[sample(1:P,1,prob=phi),]
        parent.2 = runs[sample(1:P,1),]
        pos = sample(1:(m-1),1)
        mutate = rbinom(m,1,m.rate)
        runs.next[1,] = c(parent.1[1:pos],parent.2[(pos+1):m])
        runs.next[1,] = (runs.next[i,]+mutate)%%2
    }
  location = which(aics[,j]==max(aics[,j]))[1]
    runs[location,] = runs.next[1,]

    # UPDATES AIC VALUES, FITNESS VALUES FOR NEW GENERATION
    for(i in 1:P){
        run.vars = baseball.sub[,runs[i,]==1]
        g = lm(salary.log~.,run.vars)
```

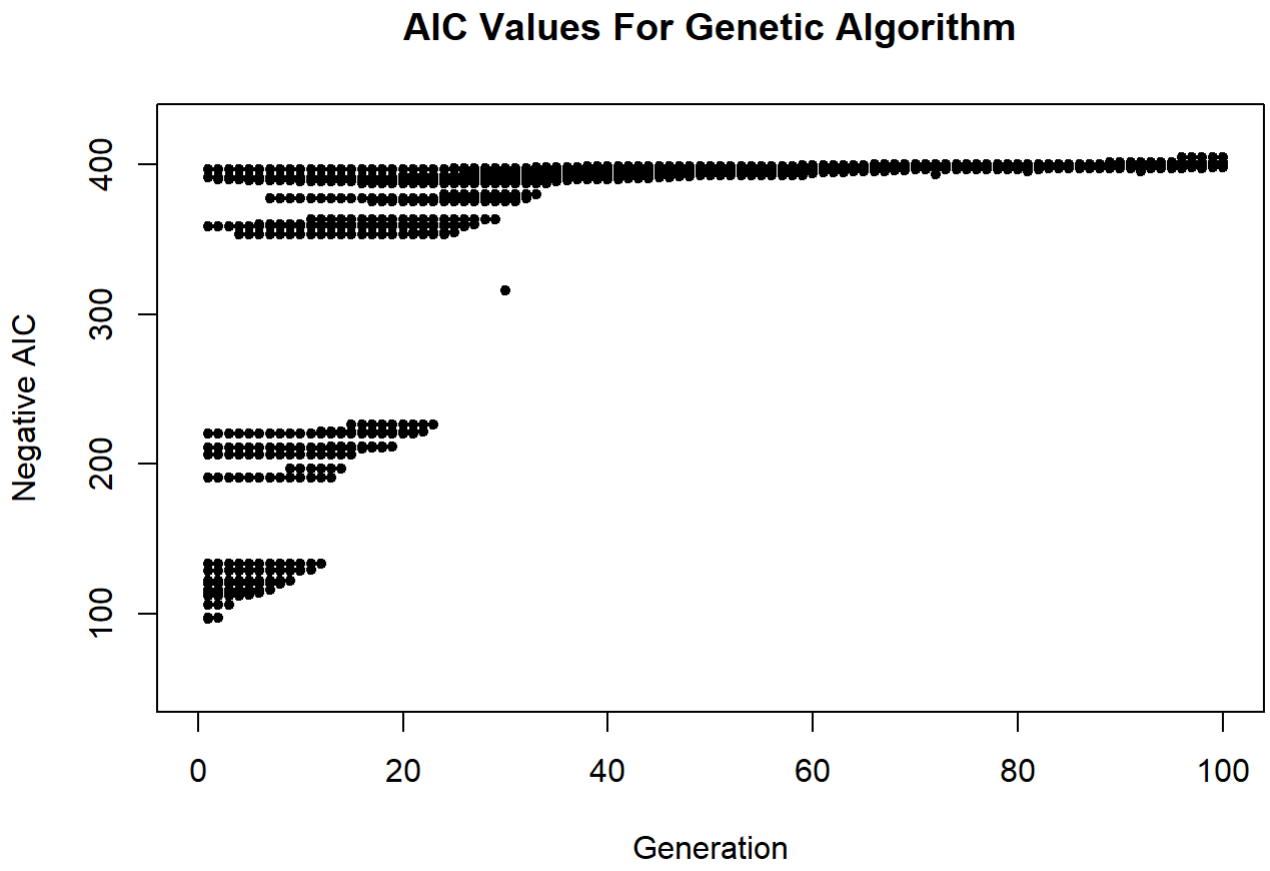
```
        runs.aic[i] = extractAIC(g)[2]
        aics[i,j+1] = runs.aic[i]
        if(runs.aic[i] < best.aic){
            run = runs[i,]
            best.aic = runs.aic[i]
        }
    }
    best.aic.gen[j+1]=best.aic
    r = rank(-runs.aic)
    phi = 2*r/(P*(P+1))
}

## OUTPUT
run         # BEST LIST OF PREDICTORS FOUND
```

```
##  [1] 0 0 1 1 1 0 1 1 0 1 1 1 1 1 1 1 0 1 0 1 0 0 1 1 0 0 1
```

```
best.aic    # AIC VALUE
```

```
## [1] -404.8097
```

```
## PLOT OF AIC VALUES
plot(-aics,xlim=c(0,itr),ylim=c(50,425),type="n",ylab="Negative AIC",
    xlab="Generation",main="AIC Values For Genetic Algorithm")
for(i in 1:itr){points(rep(i,P),-aics[,i],pch=20)}
```



a steady-state genetic algorithm find the solution AIC = -404.8097, which is not good because updating only one offspring at each generation causes the limitation of searching area.

e.

```
P = 20
itr = 100
m.rate = .01
r = matrix(0,P,1)
phi = matrix(0,P,1)
runs = matrix(0,P,m)
runs.next = matrix(0,P,m)
runs.aic = matrix(0,P,1)
aics = matrix(0,P,itr)
run = NULL
best.aic = 0
best.aic.gen = rep(0,itr)

# INITIALIZES STARTING GENERATION, FITNESS VALUES
set.seed(3219553)
for(i in 1:P){
    runs[i,] = rbinom(m,1,.5)
    run.vars = baseball.sub[,runs[i,]==1]
    g = lm(salary.log~.,run.vars)
    runs.aic[i] = extractAIC(g)[2]
    aics[i,1] = runs.aic[i]
    if(runs.aic[i] < best.aic){
        run = runs[i,]
        best.aic = runs.aic[i]
    }
}
r = rank(-runs.aic)
phi = 2*r/(P*(P+1))
best.aic.gen[1]=best.aic

## MAIN
for(j in 1:(itr-1)){

    # BUILDS THE NEW GENERATION, SELECTING FIRST PARENT BASED ON
    # FITNESS AND THE SECOND PARENT AT RANDOM
    for(i in 1:10){
        parent.1 = runs[sample(1:P,1,prob=phi),]
        parent.2 = runs[sample(1:P,1),]
        parent = rbind(parent.1,parent.2)
        mutate = rbinom(m,1,m.rate)
        for (k in 1:m) {
          runs.next[i,k] = parent[sample(1:2,1),k]
        }
        runs.next[i,] = (runs.next[i,]+mutate)%%2
        mutate = rbinom(m,1,m.rate)
        for (k in 1:m) {
          runs.next[P+1-i,] = parent[sample(1:2,1),k]
        }
        runs.next[P+1-i,] = (runs.next[P+1-i,]+mutate)%%2
    }
```

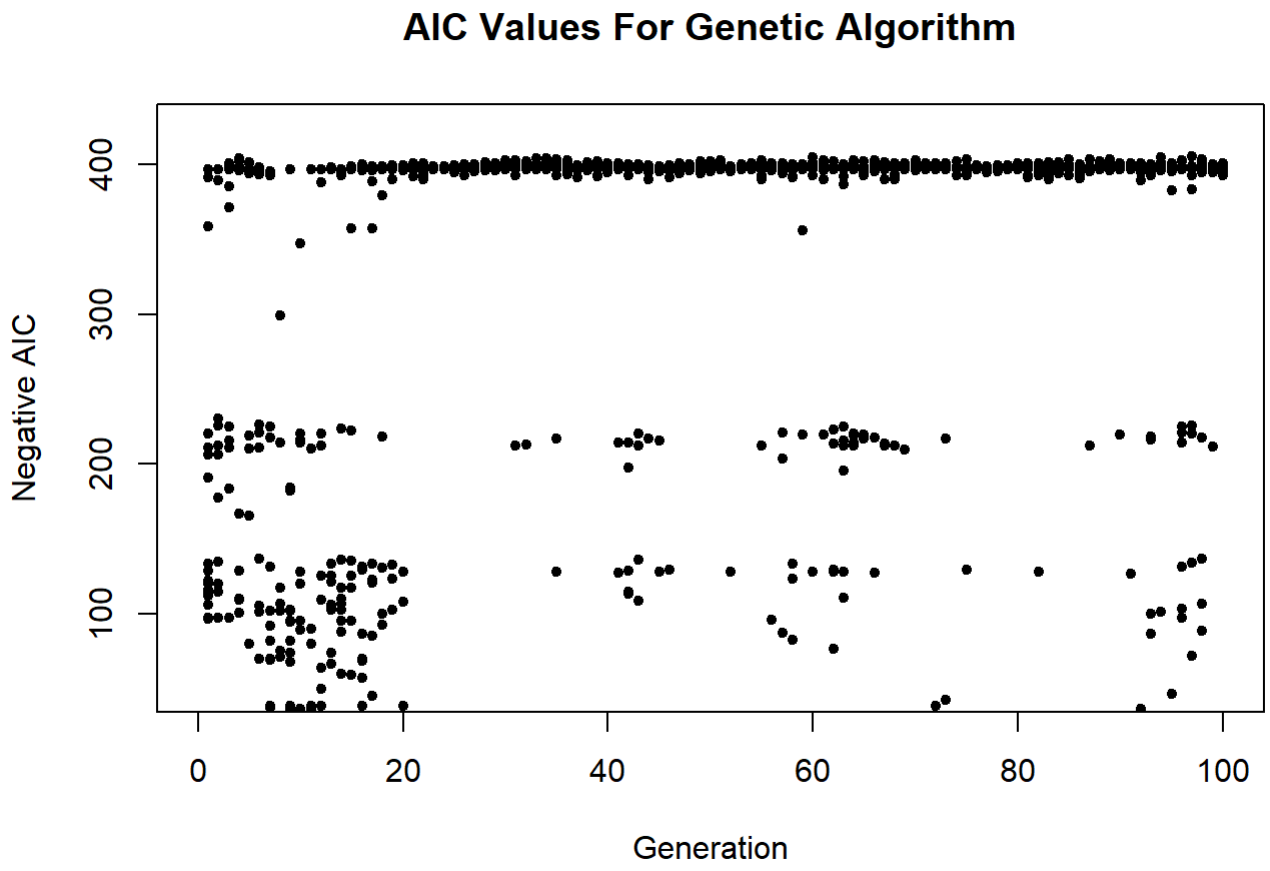
```
    runs = runs.next
    # UPDATES AIC VALUES, FITNESS VALUES FOR NEW GENERATION
    for(i in 1:P){
      if(sum(runs[i,])!=0){
        run.vars = data.frame(baseball.sub[,runs[i,]==1])
          g = lm(salary.log~.,run.vars)
          runs.aic[i] = extractAIC(g)[2]
      }
      else{
        runs.aic[i] = 0
      }
        aics[i,j+1] = runs.aic[i]
        if(runs.aic[i] < best.aic){
            run = runs[i,]
            best.aic = runs.aic[i]
        }
    }
    best.aic.gen[j+1]=best.aic
    r = rank(-runs.aic)
    phi = 2*r/(P*(P+1))
}

## OUTPUT
run         # BEST LIST OF PREDICTORS FOUND
```

```
##  [1] 1 0 1 1 1 1 0 1 0 1 0 1 1 1 1 1 1 0 0 0 0 0 0 1 0 1 1
```

```
best.aic    # AIC VALUE
```

```
## [1] -405.4286
```

```
## PLOT OF AIC VALUES
plot(-aics,xlim=c(0,itr),ylim=c(50,425),type="n",ylab="Negative AIC",
    xlab="Generation",main="AIC Values For Genetic Algorithm")
for(i in 1:itr){points(rep(i,P),-aics[,i],pch=20)}
```



by *uniform crossover*, the algorithm find AIC = -405.4286.

3.5

a.

```
setwd('C:/Users/yudyd/Desktop/ACS')
gene.dat = read.table('geneticmapping.dat',header=TRUE)
n = length(gene.dat[,1])
p = length(gene.dat[1,])

d <- function(theta,i,j){
  mean(abs(gene.dat[,theta[i]]-gene.dat[,theta[j]]))
}

TT <- function(theta,j){
  if(d(theta,j,j+1) == 0 || d(theta,j,j+1) == 1){
    return(0)
  }
  else{
    n*(d(theta,j,j+1)*log(d(theta,j,j+1))+
         (1-d(theta,j,j+1))*log(1-d(theta,j,j+1)))
  }
}

ll <- function(theta){
  tmp <- 0
  for (j in 1:(p-1)) {
    tmp <- tmp + TT(theta,j)
  }
  return(tmp)
}
```

```
num.starts = 15
runs = matrix(0,num.starts,p)
itr = 50
runs.ll = matrix(0,num.starts,itr)

# INITIALIZES STARTING RUNS
set.seed(19676)
for(i in 1:num.starts){runs[i,] = sample(1:p)}

## MAIN
for(k in 1:num.starts){
    run.current = runs[k,]

    # ITERATES EACH RANDOM START
    for(j in 1:itr){

        run.ll = ll(run.current)
        run.next = run.current

    #set.seed(2022+j)
        for(i in 1:20){
            run.step = run.current
            run.step.tmp = run.step
            local = sample(1:p,2)
            tmp = run.step.tmp[local[1]]
            run.step.tmp[local[1]] = run.step.tmp[local[2]]
            run.step.tmp[local[2]] = tmp
            run.step.ll = ll(run.step.tmp)
            if(run.step.ll > run.ll){
                run.next = run.step.tmp
                run.ll = run.step.ll
                #break
            }
        }
        run.current = run.next
        runs.ll[k,j]=ll(run.current)
    }
    runs[k,] = run.current
}

## OUTPUT
runs        # LISTS OF PREDICTORS
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
##  [1,]    2    4   12    6    3    7    1   11    9     8     5    10
##  [2,]    7    3    6   12    4    2   10    5    8     9    11     1
##  [3,]    7    1   11    9    8    5   10    2    4    12     6     3
##  [4,]   11    1    2    4   12    6    3    7   10     5     8     9
##  [5,]   12    6    3    7    1   11    9    8    5    10     4     2
##  [6,]    2    4   12    6    3    7    1   11    9     8     5    10
##  [7,]    9   11    1    7    3    8    5   10    2     4    12     6
##  [8,]    9    8    5   10   11    1    2    4   12     6     3     7
##  [9,]    2    4   12    6    3    7    1   11    9     8     5    10
## [10,]    9    8    5   10    2    4   12    6    3     7     1    11
## [11,]   10    5    2    4   12    6    3    7    1    11     9     8
## [12,]   10    5    8    9   11    1    2    4   12     6     3     7
## [13,]    9   11    1    7    3    6   12    4    2     8     5    10
## [14,]    2    4   12    6    9    8    5   10   11     1     7     3
## [15,]   12    6    3    7    1   10    5    8    9    11     4     2
```

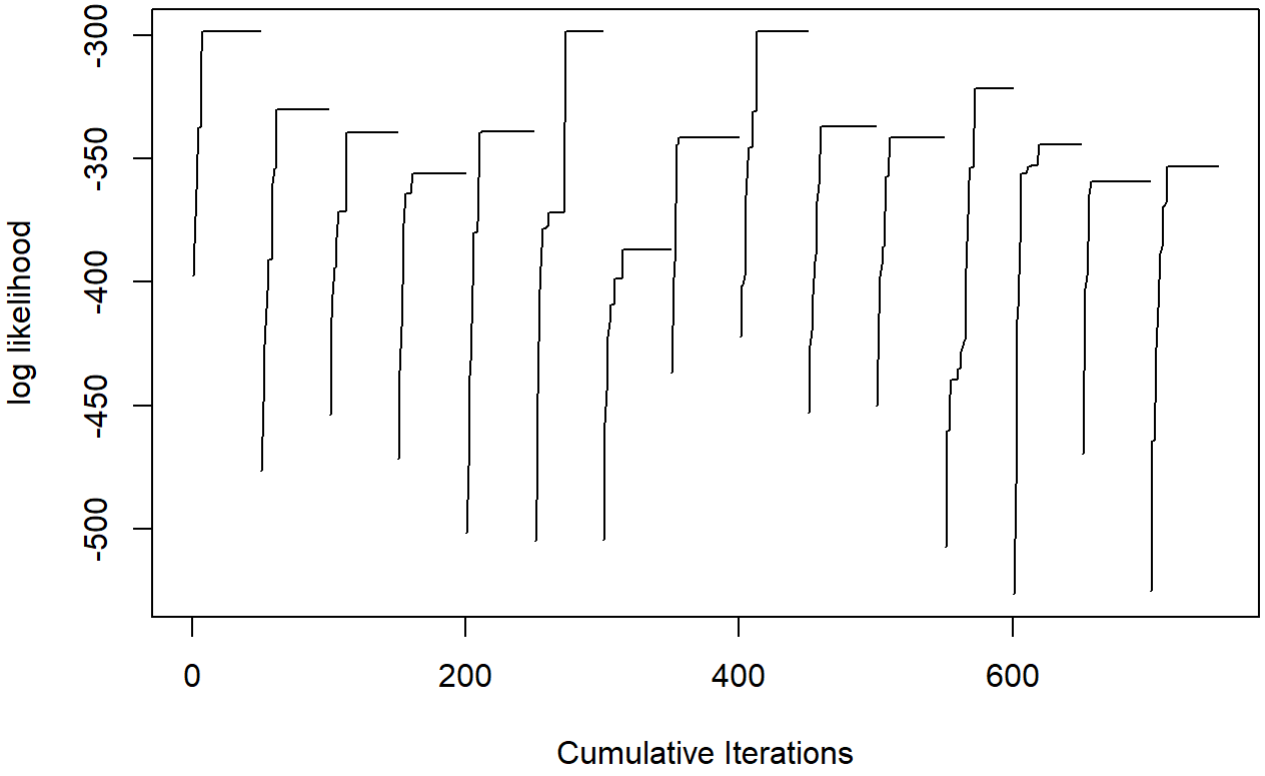
```
runs.ll     # AIC VALUES
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
##  [1,] -397.5413 -384.5394 -362.3979 -348.0822 -337.3432 -337.3432 -306.3045
##  [2,] -476.7062 -448.4826 -425.3157 -417.7567 -403.5225 -390.9984 -390.9984
##  [3,] -453.6484 -414.4557 -405.1156 -394.2990 -394.2990 -381.7909 -371.2578
##  [4,] -471.8667 -435.7771 -418.2625 -400.9723 -376.2965 -364.0284 -364.0284
##  [5,] -501.7394 -474.5022 -437.8197 -431.8742 -401.5793 -380.0848 -380.0848
##  [6,] -505.1627 -488.5286 -445.4802 -403.0127 -393.6932 -378.3923 -378.3923
##  [7,] -504.7989 -456.9070 -443.9451 -422.2802 -416.3579 -409.2823 -409.2823
##  [8,] -436.9288 -398.0570 -391.6850 -344.2820 -344.2820 -341.4800 -341.4800
##  [9,] -422.1859 -401.4928 -400.9788 -397.5070 -364.8725 -350.8024 -345.3473
## [10,] -453.0096 -425.9585 -419.6950 -405.2252 -391.8388 -388.8265 -367.1787
## [11,] -450.2460 -412.6413 -397.8610 -392.9526 -385.6593 -385.6593 -357.1409
## [12,] -507.6283 -460.2564 -460.2564 -445.7232 -439.5729 -439.5729 -439.5729
## [13,] -526.6342 -481.6808 -415.3012 -409.9527 -384.1507 -355.9271 -355.9271
## [14,] -469.7405 -435.7660 -402.7248 -397.2781 -383.5437 -364.3186 -359.3654
## [15,] -525.1376 -464.3456 -464.3456 -449.0789 -426.4761 -410.3696 -399.1015
##           [,8]      [,9]     [,10]     [,11]     [,12]     [,13]     [,14]
##  [1,] -298.3949 -298.3949 -298.3949 -298.3949 -298.3949 -298.3949 -298.3949
##  [2,] -390.9984 -359.8340 -354.0911 -354.0911 -330.0099 -330.0099 -330.0099
##  [3,] -371.2578 -371.2578 -371.2578 -371.2578 -371.2578 -339.1177 -339.1177
##  [4,] -364.0284 -364.0284 -364.0284 -356.1188 -356.1188 -356.1188 -356.1188
##  [5,] -380.0848 -375.5802 -339.1177 -339.1177 -338.8451 -338.8451 -338.8451
##  [6,] -378.3923 -377.2811 -377.2811 -371.7105 -371.7105 -371.7105 -371.7105
##  [7,] -409.2823 -398.4702 -398.4702 -398.4702 -398.4702 -398.4702 -398.4702
##  [8,] -341.4800 -341.4800 -341.4800 -341.4800 -341.4800 -341.4800 -341.4800
##  [9,] -345.3473 -345.3473 -330.5351 -330.5351 -330.5351 -298.3949 -298.3949
## [10,] -360.2607 -348.3800 -336.6488 -336.6488 -336.6488 -336.6488 -336.6488
## [11,] -357.1409 -357.1409 -341.1535 -341.1535 -341.1535 -341.1535 -341.1535
## [12,] -439.5729 -439.5729 -435.0682 -435.0682 -428.1502 -426.8296 -424.0276
## [13,] -355.9271 -355.9271 -355.9271 -353.1767 -353.1767 -352.5891 -352.5891
## [14,] -359.3654 -359.3654 -359.3654 -359.3654 -359.3654 -359.3654 -359.3654
## [15,] -388.2894 -385.6660 -369.2651 -368.6870 -367.6954 -352.8832 -352.8832
##          [,15]     [,16]     [,17]     [,18]     [,19]     [,20]     [,21]
##  [1,] -298.3949 -298.3949 -298.3949 -298.3949 -298.3949 -298.3949 -298.3949
##  [2,] -330.0099 -330.0099 -330.0099 -330.0099 -330.0099 -330.0099 -330.0099
##  [3,] -339.1177 -339.1177 -339.1177 -339.1177 -339.1177 -339.1177 -339.1177
##  [4,] -356.1188 -356.1188 -356.1188 -356.1188 -356.1188 -356.1188 -356.1188
##  [5,] -338.8451 -338.8451 -338.8451 -338.8451 -338.8451 -338.8451 -338.8451
##  [6,] -371.7105 -371.7105 -371.7105 -371.7105 -371.7105 -371.7105 -371.7105
##  [7,] -386.6762 -386.6762 -386.6762 -386.6762 -386.6762 -386.6762 -386.6762
##  [8,] -341.4800 -341.4800 -341.4800 -341.4800 -341.4800 -341.4800 -341.4800
##  [9,] -298.3949 -298.3949 -298.3949 -298.3949 -298.3949 -298.3949 -298.3949
## [10,] -336.6488 -336.6488 -336.6488 -336.6488 -336.6488 -336.6488 -336.6488
## [11,] -341.1535 -341.1535 -341.1535 -341.1535 -341.1535 -341.1535 -341.1535
## [12,] -423.4446 -394.5451 -372.5309 -358.9225 -353.3963 -353.3963 -353.3963
## [13,] -352.5891 -352.5891 -352.5891 -352.5891 -343.9554 -343.9554 -343.9554
## [14,] -359.3654 -359.3654 -359.3654 -359.3654 -359.3654 -359.3654 -359.3654
## [15,] -352.8832 -352.8832 -352.8832 -352.8832 -352.8832 -352.8832 -352.8832
##          [,22]     [,23]     [,24]     [,25]     [,26]     [,27]     [,28]
##  [1,] -298.3949 -298.3949 -298.3949 -298.3949 -298.3949 -298.3949 -298.3949
##  [2,] -330.0099 -330.0099 -330.0099 -330.0099 -330.0099 -330.0099 -330.0099
##  [3,] -339.1177 -339.1177 -339.1177 -339.1177 -339.1177 -339.1177 -339.1177
```

```
##  [4,] -356.1188 -356.1188 -356.1188 -356.1188 -356.1188 -356.1188 -356.1188
##  [5,] -338.8451 -338.8451 -338.8451 -338.8451 -338.8451 -338.8451 -338.8451
##  [6,] -371.7105 -298.3949 -298.3949 -298.3949 -298.3949 -298.3949 -298.3949
##  [7,] -386.6762 -386.6762 -386.6762 -386.6762 -386.6762 -386.6762 -386.6762
##  [8,] -341.4800 -341.4800 -341.4800 -341.4800 -341.4800 -341.4800 -341.4800
##  [9,] -298.3949 -298.3949 -298.3949 -298.3949 -298.3949 -298.3949 -298.3949
## [10,] -336.6488 -336.6488 -336.6488 -336.6488 -336.6488 -336.6488 -336.6488
## [11,] -341.1535 -341.1535 -341.1535 -341.1535 -341.1535 -341.1535 -341.1535
## [12,] -321.2562 -321.2562 -321.2562 -321.2562 -321.2562 -321.2562 -321.2562
## [13,] -343.9554 -343.9554 -343.9554 -343.9554 -343.9554 -343.9554 -343.9554
## [14,] -359.3654 -359.3654 -359.3654 -359.3654 -359.3654 -359.3654 -359.3654
## [15,] -352.8832 -352.8832 -352.8832 -352.8832 -352.8832 -352.8832 -352.8832
##          [,29]     [,30]     [,31]     [,32]     [,33]     [,34]     [,35]
##  [1,] -298.3949 -298.3949 -298.3949 -298.3949 -298.3949 -298.3949 -298.3949
##  [2,] -330.0099 -330.0099 -330.0099 -330.0099 -330.0099 -330.0099 -330.0099
##  [3,] -339.1177 -339.1177 -339.1177 -339.1177 -339.1177 -339.1177 -339.1177
##  [4,] -356.1188 -356.1188 -356.1188 -356.1188 -356.1188 -356.1188 -356.1188
##  [5,] -338.8451 -338.8451 -338.8451 -338.8451 -338.8451 -338.8451 -338.8451
##  [6,] -298.3949 -298.3949 -298.3949 -298.3949 -298.3949 -298.3949 -298.3949
##  [7,] -386.6762 -386.6762 -386.6762 -386.6762 -386.6762 -386.6762 -386.6762
##  [8,] -341.4800 -341.4800 -341.4800 -341.4800 -341.4800 -341.4800 -341.4800
##  [9,] -298.3949 -298.3949 -298.3949 -298.3949 -298.3949 -298.3949 -298.3949
## [10,] -336.6488 -336.6488 -336.6488 -336.6488 -336.6488 -336.6488 -336.6488
## [11,] -341.1535 -341.1535 -341.1535 -341.1535 -341.1535 -341.1535 -341.1535
## [12,] -321.2562 -321.2562 -321.2562 -321.2562 -321.2562 -321.2562 -321.2562
## [13,] -343.9554 -343.9554 -343.9554 -343.9554 -343.9554 -343.9554 -343.9554
## [14,] -359.3654 -359.3654 -359.3654 -359.3654 -359.3654 -359.3654 -359.3654
## [15,] -352.8832 -352.8832 -352.8832 -352.8832 -352.8832 -352.8832 -352.8832
##          [,36]     [,37]     [,38]     [,39]     [,40]     [,41]     [,42]
##  [1,] -298.3949 -298.3949 -298.3949 -298.3949 -298.3949 -298.3949 -298.3949
##  [2,] -330.0099 -330.0099 -330.0099 -330.0099 -330.0099 -330.0099 -330.0099
##  [3,] -339.1177 -339.1177 -339.1177 -339.1177 -339.1177 -339.1177 -339.1177
##  [4,] -356.1188 -356.1188 -356.1188 -356.1188 -356.1188 -356.1188 -356.1188
##  [5,] -338.8451 -338.8451 -338.8451 -338.8451 -338.8451 -338.8451 -338.8451
##  [6,] -298.3949 -298.3949 -298.3949 -298.3949 -298.3949 -298.3949 -298.3949
##  [7,] -386.6762 -386.6762 -386.6762 -386.6762 -386.6762 -386.6762 -386.6762
##  [8,] -341.4800 -341.4800 -341.4800 -341.4800 -341.4800 -341.4800 -341.4800
##  [9,] -298.3949 -298.3949 -298.3949 -298.3949 -298.3949 -298.3949 -298.3949
## [10,] -336.6488 -336.6488 -336.6488 -336.6488 -336.6488 -336.6488 -336.6488
## [11,] -341.1535 -341.1535 -341.1535 -341.1535 -341.1535 -341.1535 -341.1535
## [12,] -321.2562 -321.2562 -321.2562 -321.2562 -321.2562 -321.2562 -321.2562
## [13,] -343.9554 -343.9554 -343.9554 -343.9554 -343.9554 -343.9554 -343.9554
## [14,] -359.3654 -359.3654 -359.3654 -359.3654 -359.3654 -359.3654 -359.3654
## [15,] -352.8832 -352.8832 -352.8832 -352.8832 -352.8832 -352.8832 -352.8832
##          [,43]     [,44]     [,45]     [,46]     [,47]     [,48]     [,49]
##  [1,] -298.3949 -298.3949 -298.3949 -298.3949 -298.3949 -298.3949 -298.3949
##  [2,] -330.0099 -330.0099 -330.0099 -330.0099 -330.0099 -330.0099 -330.0099
##  [3,] -339.1177 -339.1177 -339.1177 -339.1177 -339.1177 -339.1177 -339.1177
##  [4,] -356.1188 -356.1188 -356.1188 -356.1188 -356.1188 -356.1188 -356.1188
##  [5,] -338.8451 -338.8451 -338.8451 -338.8451 -338.8451 -338.8451 -338.8451
##  [6,] -298.3949 -298.3949 -298.3949 -298.3949 -298.3949 -298.3949 -298.3949
##  [7,] -386.6762 -386.6762 -386.6762 -386.6762 -386.6762 -386.6762 -386.6762
```

```
##  [8,] -341.4800 -341.4800 -341.4800 -341.4800 -341.4800 -341.4800 -341.4800
##  [9,] -298.3949 -298.3949 -298.3949 -298.3949 -298.3949 -298.3949 -298.3949
## [10,] -336.6488 -336.6488 -336.6488 -336.6488 -336.6488 -336.6488 -336.6488
## [11,] -341.1535 -341.1535 -341.1535 -341.1535 -341.1535 -341.1535 -341.1535
## [12,] -321.2562 -321.2562 -321.2562 -321.2562 -321.2562 -321.2562 -321.2562
## [13,] -343.9554 -343.9554 -343.9554 -343.9554 -343.9554 -343.9554 -343.9554
## [14,] -359.3654 -359.3654 -359.3654 -359.3654 -359.3654 -359.3654 -359.3654
## [15,] -352.8832 -352.8832 -352.8832 -352.8832 -352.8832 -352.8832 -352.8832
##           [,50]
##  [1,] -298.3949
##  [2,] -330.0099
##  [3,] -339.1177
##  [4,] -356.1188
##  [5,] -338.8451
##  [6,] -298.3949
##  [7,] -386.6762
##  [8,] -341.4800
##  [9,] -298.3949
## [10,] -336.6488
## [11,] -341.1535
## [12,] -321.2562
## [13,] -343.9554
## [14,] -359.3654
## [15,] -352.8832
```

```
##PLOT
plot(1:(itr*num.starts),c(t(runs.ll)),xlab="Cumulative Iterations",
  ylab="log likelihood",type="n")
for(i in 1:num.starts) {
  lines((i-1)*itr+(1:itr),runs.ll[i,]) }
```



By random local search, the algorithm perform well and find the true genetic map:10,5,8,9,11,1,7,3,6,12,4,2, with the max likelihood -298.3949.

```
num.starts = 15
runs = matrix(0,num.starts,p)
itr = 50
runs.ll = matrix(0,num.starts,itr)

# INITIALIZES STARTING RUNS
set.seed(19676)
for(i in 1:num.starts){runs[i,] = sample(1:p)}

## MAIN
for(k in 1:num.starts){
    run.current = runs[k,]

    # ITERATES EACH RANDOM START
    for(j in 1:itr){

      run.ll = ll(run.current)
        run.next = run.current

    #set.seed(2022+j)
        for(kk in 1:(p-1)){
          for (mm in (kk+1):p) {
            run.step.tmp = run.current
            tmp = run.step.tmp[kk]
            run.step.tmp[kk] = run.step.tmp[mm]
            run.step.tmp[mm] = tmp
            run.step.ll = ll(run.step.tmp)
            if(run.step.ll > run.ll){
                run.next = run.step.tmp
                run.ll = run.step.ll
            }
          }
        }
        run.current = run.next
        runs.ll[k,j]=run.ll
    }
    runs[k,] = run.current
}

## OUTPUT
runs        # LISTS OF PREDICTORS
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
##  [1,]    7    3    6   12    4    2    1   11    9     8     5    10
##  [2,]    9   11    1   12    4    2   10    5    8     7     6     3
##  [3,]    3    6   12    4    2   10    5    8    9    11     1     7
##  [4,]    1    7    3    6   12    4    2   10    5     8     9    11
##  [5,]    1    7    3    6   12    4    2   11    9     8     5    10
##  [6,]    9   11    1    7    6    3   10    5    8    12     4     2
##  [7,]    1    7    3    6   12    4    2   10    5     8     9    11
##  [8,]    9    8    5   10   11    1    7    3    6    12     4     2
##  [9,]    2    4   12    6    3    7    1   11   10     5     8     9
## [10,]   11    9    8    5   10    1    7    3    6    12     4     2
## [11,]   10    5    8    2    4   12    6    9   11     1     7     3
## [12,]    8    9   11    2    4   12    6    3    7     1     5    10
## [13,]    9   11    1    7    3    6   12    4    2     8     5    10
## [14,]    7    3    6   12    4   10    5    8    9    11     1     2
## [15,]    2    4   12    6    3   10    5    8    9    11     1     7
```

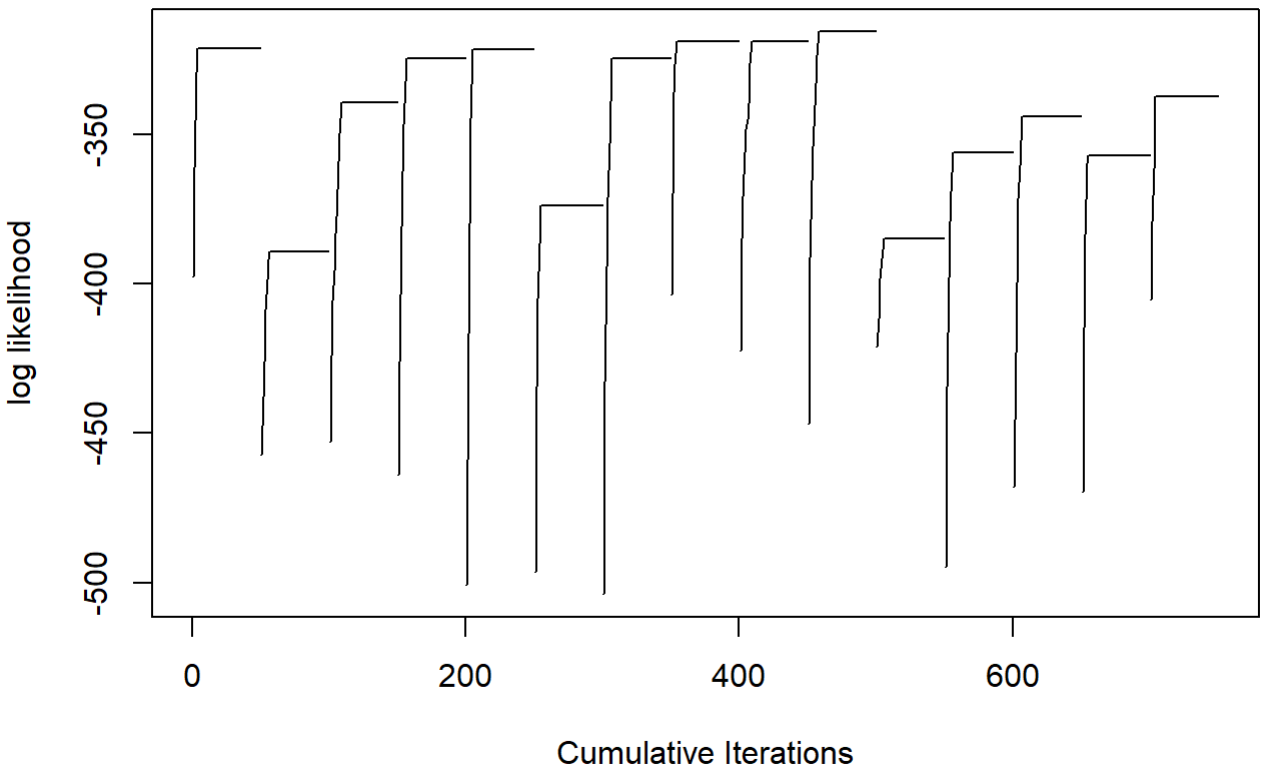
```
runs.ll     # AIC VALUES
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
##  [1,] -397.5413 -361.1197 -329.1658 -321.2562 -321.2562 -321.2562 -321.2562
##  [2,] -457.1805 -437.8280 -426.1922 -408.6205 -398.6933 -391.6970 -389.0736
##  [3,] -453.0046 -423.9210 -405.5976 -395.5340 -383.0861 -377.4110 -358.9702
##  [4,] -464.0315 -425.9248 -380.7503 -362.3801 -346.7548 -332.2151 -324.3054
##  [5,] -500.7137 -416.5045 -379.6252 -344.6591 -321.4922 -321.4922 -321.4922
##  [6,] -496.5598 -434.3145 -404.9645 -386.4932 -373.8785 -373.6973 -373.6973
##  [7,] -503.8677 -432.7455 -394.1887 -371.0218 -352.8353 -338.8451 -324.3054
##  [8,] -403.5652 -335.8479 -326.5284 -318.6187 -318.6187 -318.6187 -318.6187
##  [9,] -422.1859 -384.8053 -370.7352 -355.9230 -348.3508 -344.8322 -339.4908
## [10,] -446.8947 -391.2782 -365.4305 -353.8403 -340.1144 -332.7809 -323.4614
## [11,] -421.1098 -411.6891 -398.0807 -391.6201 -388.5962 -384.7021 -384.7021
## [12,] -494.6680 -434.5267 -398.0144 -379.0265 -368.9354 -356.0847 -356.0847
## [13,] -467.8267 -422.8733 -399.6325 -371.4089 -360.4988 -351.8651 -343.9554
## [14,] -469.7405 -414.5998 -383.3729 -359.7074 -356.9054 -356.9054 -356.9054
## [15,] -405.2817 -378.9545 -361.4075 -337.3262 -337.3262 -337.3262 -337.3262
##           [,8]      [,9]     [,10]     [,11]     [,12]     [,13]     [,14]
##  [1,] -321.2562 -321.2562 -321.2562 -321.2562 -321.2562 -321.2562 -321.2562
##  [2,] -389.0736 -389.0736 -389.0736 -389.0736 -389.0736 -389.0736 -389.0736
##  [3,] -349.6507 -339.1177 -339.1177 -339.1177 -339.1177 -339.1177 -339.1177
##  [4,] -324.3054 -324.3054 -324.3054 -324.3054 -324.3054 -324.3054 -324.3054
##  [5,] -321.4922 -321.4922 -321.4922 -321.4922 -321.4922 -321.4922 -321.4922
##  [6,] -373.6973 -373.6973 -373.6973 -373.6973 -373.6973 -373.6973 -373.6973
##  [7,] -324.3054 -324.3054 -324.3054 -324.3054 -324.3054 -324.3054 -324.3054
##  [8,] -318.6187 -318.6187 -318.6187 -318.6187 -318.6187 -318.6187 -318.6187
##  [9,] -326.1909 -318.6187 -318.6187 -318.6187 -318.6187 -318.6187 -318.6187
## [10,] -315.5517 -315.5517 -315.5517 -315.5517 -315.5517 -315.5517 -315.5517
## [11,] -384.7021 -384.7021 -384.7021 -384.7021 -384.7021 -384.7021 -384.7021
## [12,] -356.0847 -356.0847 -356.0847 -356.0847 -356.0847 -356.0847 -356.0847
## [13,] -343.9554 -343.9554 -343.9554 -343.9554 -343.9554 -343.9554 -343.9554
## [14,] -356.9054 -356.9054 -356.9054 -356.9054 -356.9054 -356.9054 -356.9054
## [15,] -337.3262 -337.3262 -337.3262 -337.3262 -337.3262 -337.3262 -337.3262
##          [,15]     [,16]     [,17]     [,18]     [,19]     [,20]     [,21]
##  [1,] -321.2562 -321.2562 -321.2562 -321.2562 -321.2562 -321.2562 -321.2562
##  [2,] -389.0736 -389.0736 -389.0736 -389.0736 -389.0736 -389.0736 -389.0736
##  [3,] -339.1177 -339.1177 -339.1177 -339.1177 -339.1177 -339.1177 -339.1177
##  [4,] -324.3054 -324.3054 -324.3054 -324.3054 -324.3054 -324.3054 -324.3054
##  [5,] -321.4922 -321.4922 -321.4922 -321.4922 -321.4922 -321.4922 -321.4922
##  [6,] -373.6973 -373.6973 -373.6973 -373.6973 -373.6973 -373.6973 -373.6973
##  [7,] -324.3054 -324.3054 -324.3054 -324.3054 -324.3054 -324.3054 -324.3054
##  [8,] -318.6187 -318.6187 -318.6187 -318.6187 -318.6187 -318.6187 -318.6187
##  [9,] -318.6187 -318.6187 -318.6187 -318.6187 -318.6187 -318.6187 -318.6187
## [10,] -315.5517 -315.5517 -315.5517 -315.5517 -315.5517 -315.5517 -315.5517
## [11,] -384.7021 -384.7021 -384.7021 -384.7021 -384.7021 -384.7021 -384.7021
## [12,] -356.0847 -356.0847 -356.0847 -356.0847 -356.0847 -356.0847 -356.0847
## [13,] -343.9554 -343.9554 -343.9554 -343.9554 -343.9554 -343.9554 -343.9554
## [14,] -356.9054 -356.9054 -356.9054 -356.9054 -356.9054 -356.9054 -356.9054
## [15,] -337.3262 -337.3262 -337.3262 -337.3262 -337.3262 -337.3262 -337.3262
##          [,22]     [,23]     [,24]     [,25]     [,26]     [,27]     [,28]
##  [1,] -321.2562 -321.2562 -321.2562 -321.2562 -321.2562 -321.2562 -321.2562
##  [2,] -389.0736 -389.0736 -389.0736 -389.0736 -389.0736 -389.0736 -389.0736
##  [3,] -339.1177 -339.1177 -339.1177 -339.1177 -339.1177 -339.1177 -339.1177
```

```
##  [4,] -324.3054 -324.3054 -324.3054 -324.3054 -324.3054 -324.3054 -324.3054
##  [5,] -321.4922 -321.4922 -321.4922 -321.4922 -321.4922 -321.4922 -321.4922
##  [6,] -373.6973 -373.6973 -373.6973 -373.6973 -373.6973 -373.6973 -373.6973
##  [7,] -324.3054 -324.3054 -324.3054 -324.3054 -324.3054 -324.3054 -324.3054
##  [8,] -318.6187 -318.6187 -318.6187 -318.6187 -318.6187 -318.6187 -318.6187
##  [9,] -318.6187 -318.6187 -318.6187 -318.6187 -318.6187 -318.6187 -318.6187
## [10,] -315.5517 -315.5517 -315.5517 -315.5517 -315.5517 -315.5517 -315.5517
## [11,] -384.7021 -384.7021 -384.7021 -384.7021 -384.7021 -384.7021 -384.7021
## [12,] -356.0847 -356.0847 -356.0847 -356.0847 -356.0847 -356.0847 -356.0847
## [13,] -343.9554 -343.9554 -343.9554 -343.9554 -343.9554 -343.9554 -343.9554
## [14,] -356.9054 -356.9054 -356.9054 -356.9054 -356.9054 -356.9054 -356.9054
## [15,] -337.3262 -337.3262 -337.3262 -337.3262 -337.3262 -337.3262 -337.3262
##           [,29]     [,30]     [,31]     [,32]     [,33]     [,34]     [,35]
##  [1,] -321.2562 -321.2562 -321.2562 -321.2562 -321.2562 -321.2562 -321.2562
##  [2,] -389.0736 -389.0736 -389.0736 -389.0736 -389.0736 -389.0736 -389.0736
##  [3,] -339.1177 -339.1177 -339.1177 -339.1177 -339.1177 -339.1177 -339.1177
##  [4,] -324.3054 -324.3054 -324.3054 -324.3054 -324.3054 -324.3054 -324.3054
##  [5,] -321.4922 -321.4922 -321.4922 -321.4922 -321.4922 -321.4922 -321.4922
##  [6,] -373.6973 -373.6973 -373.6973 -373.6973 -373.6973 -373.6973 -373.6973
##  [7,] -324.3054 -324.3054 -324.3054 -324.3054 -324.3054 -324.3054 -324.3054
##  [8,] -318.6187 -318.6187 -318.6187 -318.6187 -318.6187 -318.6187 -318.6187
##  [9,] -318.6187 -318.6187 -318.6187 -318.6187 -318.6187 -318.6187 -318.6187
## [10,] -315.5517 -315.5517 -315.5517 -315.5517 -315.5517 -315.5517 -315.5517
## [11,] -384.7021 -384.7021 -384.7021 -384.7021 -384.7021 -384.7021 -384.7021
## [12,] -356.0847 -356.0847 -356.0847 -356.0847 -356.0847 -356.0847 -356.0847
## [13,] -343.9554 -343.9554 -343.9554 -343.9554 -343.9554 -343.9554 -343.9554
## [14,] -356.9054 -356.9054 -356.9054 -356.9054 -356.9054 -356.9054 -356.9054
## [15,] -337.3262 -337.3262 -337.3262 -337.3262 -337.3262 -337.3262 -337.3262
##           [,36]     [,37]     [,38]     [,39]     [,40]     [,41]     [,42]
##  [1,] -321.2562 -321.2562 -321.2562 -321.2562 -321.2562 -321.2562 -321.2562
##  [2,] -389.0736 -389.0736 -389.0736 -389.0736 -389.0736 -389.0736 -389.0736
##  [3,] -339.1177 -339.1177 -339.1177 -339.1177 -339.1177 -339.1177 -339.1177
##  [4,] -324.3054 -324.3054 -324.3054 -324.3054 -324.3054 -324.3054 -324.3054
##  [5,] -321.4922 -321.4922 -321.4922 -321.4922 -321.4922 -321.4922 -321.4922
##  [6,] -373.6973 -373.6973 -373.6973 -373.6973 -373.6973 -373.6973 -373.6973
##  [7,] -324.3054 -324.3054 -324.3054 -324.3054 -324.3054 -324.3054 -324.3054
##  [8,] -318.6187 -318.6187 -318.6187 -318.6187 -318.6187 -318.6187 -318.6187
##  [9,] -318.6187 -318.6187 -318.6187 -318.6187 -318.6187 -318.6187 -318.6187
## [10,] -315.5517 -315.5517 -315.5517 -315.5517 -315.5517 -315.5517 -315.5517
## [11,] -384.7021 -384.7021 -384.7021 -384.7021 -384.7021 -384.7021 -384.7021
## [12,] -356.0847 -356.0847 -356.0847 -356.0847 -356.0847 -356.0847 -356.0847
## [13,] -343.9554 -343.9554 -343.9554 -343.9554 -343.9554 -343.9554 -343.9554
## [14,] -356.9054 -356.9054 -356.9054 -356.9054 -356.9054 -356.9054 -356.9054
## [15,] -337.3262 -337.3262 -337.3262 -337.3262 -337.3262 -337.3262 -337.3262
##           [,43]     [,44]     [,45]     [,46]     [,47]     [,48]     [,49]
##  [1,] -321.2562 -321.2562 -321.2562 -321.2562 -321.2562 -321.2562 -321.2562
##  [2,] -389.0736 -389.0736 -389.0736 -389.0736 -389.0736 -389.0736 -389.0736
##  [3,] -339.1177 -339.1177 -339.1177 -339.1177 -339.1177 -339.1177 -339.1177
##  [4,] -324.3054 -324.3054 -324.3054 -324.3054 -324.3054 -324.3054 -324.3054
##  [5,] -321.4922 -321.4922 -321.4922 -321.4922 -321.4922 -321.4922 -321.4922
##  [6,] -373.6973 -373.6973 -373.6973 -373.6973 -373.6973 -373.6973 -373.6973
##  [7,] -324.3054 -324.3054 -324.3054 -324.3054 -324.3054 -324.3054 -324.3054
```

```
##  [8,] -318.6187 -318.6187 -318.6187 -318.6187 -318.6187 -318.6187 -318.6187
##  [9,] -318.6187 -318.6187 -318.6187 -318.6187 -318.6187 -318.6187 -318.6187
## [10,] -315.5517 -315.5517 -315.5517 -315.5517 -315.5517 -315.5517 -315.5517
## [11,] -384.7021 -384.7021 -384.7021 -384.7021 -384.7021 -384.7021 -384.7021
## [12,] -356.0847 -356.0847 -356.0847 -356.0847 -356.0847 -356.0847 -356.0847
## [13,] -343.9554 -343.9554 -343.9554 -343.9554 -343.9554 -343.9554 -343.9554
## [14,] -356.9054 -356.9054 -356.9054 -356.9054 -356.9054 -356.9054 -356.9054
## [15,] -337.3262 -337.3262 -337.3262 -337.3262 -337.3262 -337.3262 -337.3262
##           [,50]
##  [1,] -321.2562
##  [2,] -389.0736
##  [3,] -339.1177
##  [4,] -324.3054
##  [5,] -321.4922
##  [6,] -373.6973
##  [7,] -324.3054
##  [8,] -318.6187
##  [9,] -318.6187
## [10,] -315.5517
## [11,] -384.7021
## [12,] -356.0847
## [13,] -343.9554
## [14,] -356.9054
## [15,] -337.3262
```

```
##PLOT
plot(1:(itr*num.starts),c(t(runs.ll)),xlab="Cumulative Iterations",
  ylab="log likelihood",type="n")
for(i in 1:num.starts) {
  lines((i-1)*itr+(1:itr),runs.ll[i,]) }
```



The algorithm find the solution 11,9,8,5,10,1,7,3,6,12,4,2, with likelihood -315.5517.

Play with GA

```
library(GA)
```

```
## Warning: 程辑包'GA'是用R版本4.1.3 来建造的
```

```
## 载入需要的程辑包：foreach
```

```
## 载入需要的程辑包：iterators
```

```
## Warning: 程辑包'iterators'是用R版本4.1.2 来建造的
```

```
## Package 'GA' version 3.2.2
## Type 'citation("GA")' for citing this R package in publications.
```

```
##
## 载入程辑包：'GA'
```

```
## The following object is masked from 'package:utils':
##
##     de
```

```
fitness <- function(theta){
  run.vars = baseball.sub[,theta==1]
    g = lm(salary.log~.,run.vars)
    extractAIC(g)[2]
}

runs = matrix(0,P,m)
# INITIALIZES STARTING GENERATION, FITNESS VALUES
set.seed(3219553)
for(i in 1:P){
    runs[i,] = rbinom(m,1,.5)
}
```

Using GA in Problem 3.4

Compared with method in Problem 3.4,

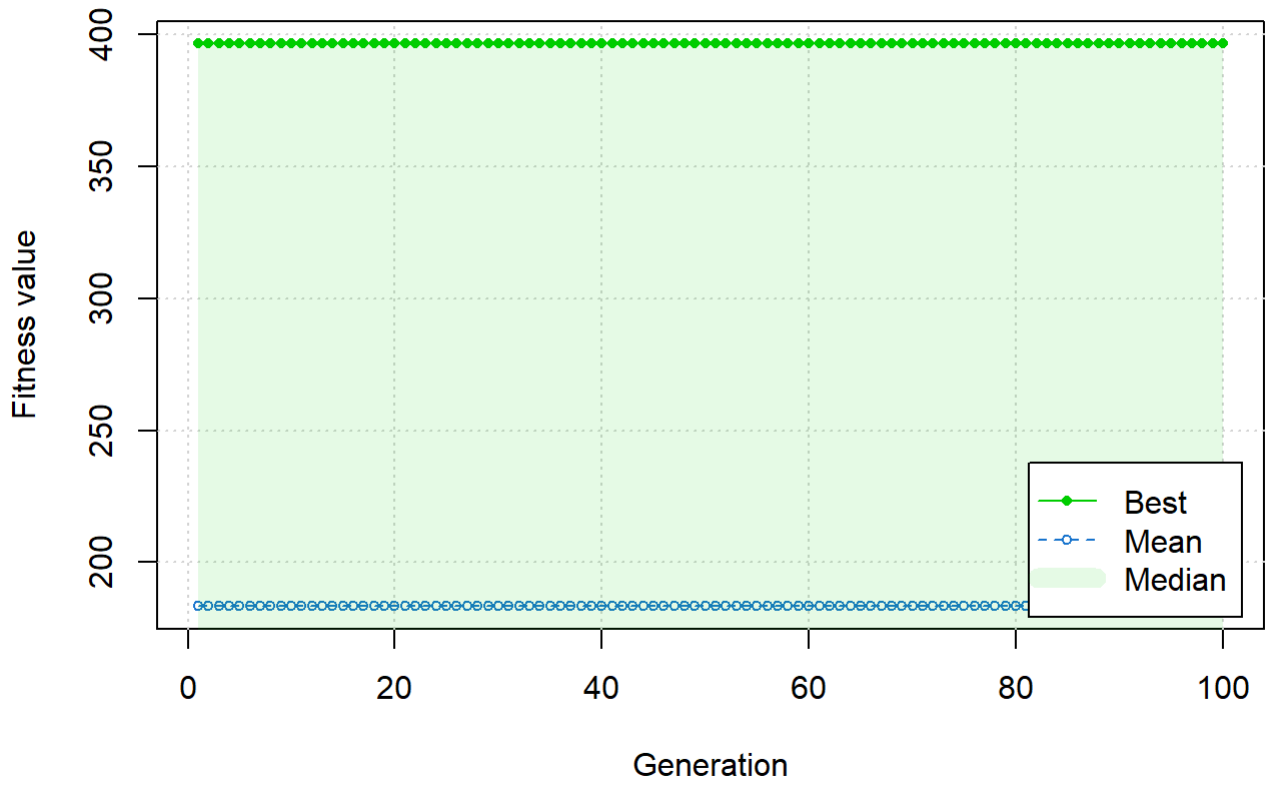
a.

$\mu = 0.01$

```
GA <- ga(type = "binary",
         fitness =  function(theta) -fitness(theta),suggestions = runs,
         popSize = 20, maxiter = 100,elitism = 20,pmutation = 0.01,
         nBits=m,seed=2022,selection = ga_lrSelection,pcrossover = 1)
summary(GA)
```

```
## -- Genetic Algorithm -------------------
##
## GA settings:
## Type                  =  binary
## Population size       =  20
## Number of generations =  100
## Elitism               =  20
## Crossover probability =  1
## Mutation probability  =  0.01
## Suggestions =
##       x1 x2 x3 x4 x5 x6 x7 x8 x9 x10  ...  x26 x27
## 1      0  0  1  0  0  1  1  1  0   1         1   0
## 2      0  0  1  0  1  1  1  0  0   0         0   0
## 3      1  1  1  0  0  0  0  1  0   1         0   0
## 4      0  0  1  0  1  0  1  1  1   0         0   1
## 5      1  0  1  0  0  0  0  0  0   0         0   0
## 6      1  0  1  1  1  0  0  0  0   0         1   1
## 7      0  1  0  1  1  0  1  1  0   1         0   1
## 8      1  0  0  0  0  0  0  1  1   0         1   0
## 9      1  0  1  1  0  0  1  1  1   1         1   1
## 10     1  0  0  1  1  0  1  0  1   0         1   0
##  ...
## 19     0  1  0  1  1  1  0  1  0   0         0   0
## 20     1  1  0  1  1  0  1  1  1   1         0   0
##
## GA results:
## Iterations             = 100
## Fitness function value = 396.6147
## Solution =
##      x1 x2 x3 x4 x5 x6 x7 x8 x9 x10  ...  x26 x27
## [1,]  0  1  0  1  1  0  1  1  0   1         0   1
```

```
plot(GA)
```

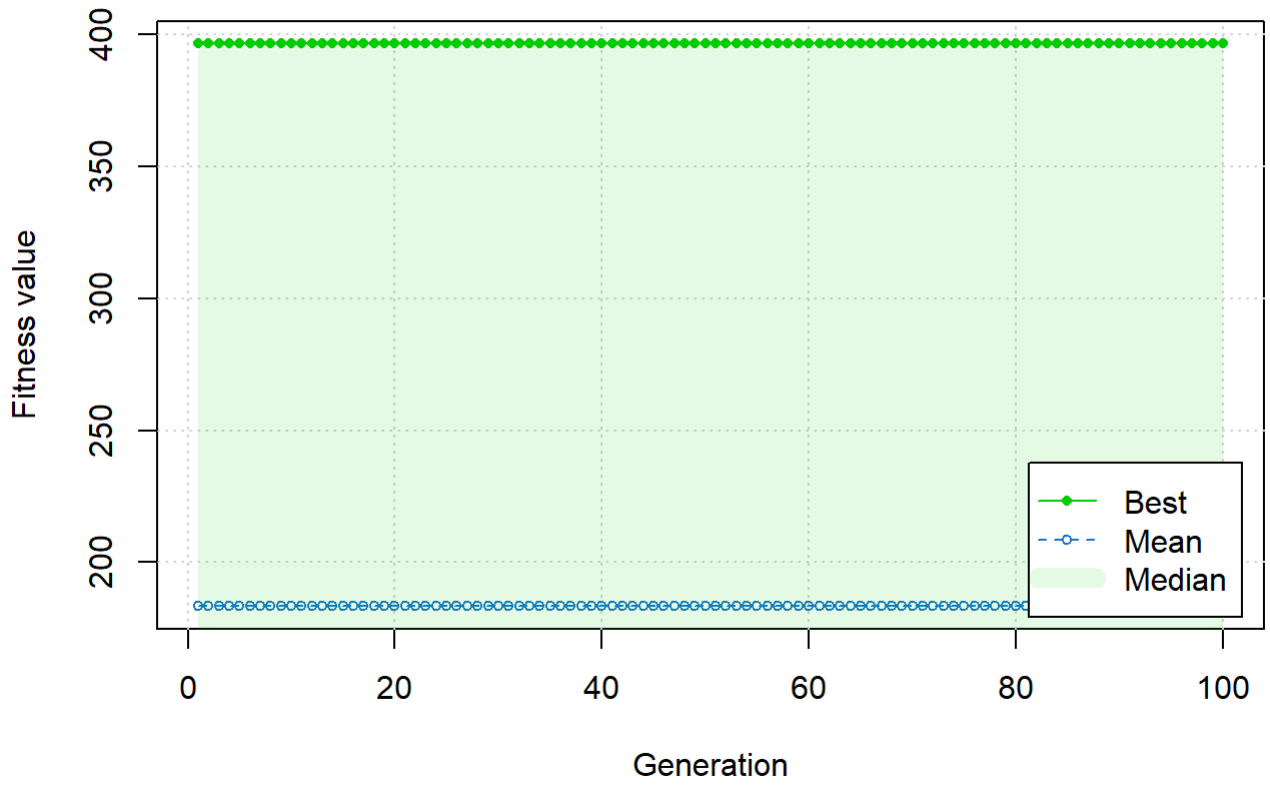


$\mu = 0.05$

```
GA <- ga(type = "binary",
         fitness =  function(theta) -fitness(theta),suggestions = runs,
         popSize = 20, maxiter = 100,elitism = 20,pmutation = 0.05,
         nBits=m,seed=2022,selection = ga_lrSelection,pcrossover = 1)
summary(GA)
```

```
## -- Genetic Algorithm -------------------
##
## GA settings:
## Type                  =  binary
## Population size       =  20
## Number of generations =  100
## Elitism               =  20
## Crossover probability =  1
## Mutation probability  =  0.05
## Suggestions =
##       x1 x2 x3 x4 x5 x6 x7 x8 x9 x10  ...  x26 x27
## 1      0  0  1  0  0  1  1  1  0   1         1   0
## 2      0  0  1  0  1  1  1  0  0   0         0   0
## 3      1  1  1  0  0  0  0  1  0   1         0   0
## 4      0  0  1  0  1  0  1  1  1   0         0   1
## 5      1  0  1  0  0  0  0  0  0   0         0   0
## 6      1  0  1  1  1  0  0  0  0   0         1   1
## 7      0  1  0  1  1  0  1  1  0   1         0   1
## 8      1  0  0  0  0  0  0  1  1   0         1   0
## 9      1  0  1  1  0  0  1  1  1   1         1   1
## 10     1  0  0  1  1  0  1  0  1   0         1   0
##  ...
## 19     0  1  0  1  1  1  0  1  0   0         0   0
## 20     1  1  0  1  1  0  1  1  1   1         0   0
##
## GA results:
## Iterations             = 100
## Fitness function value = 396.6147
## Solution =
##      x1 x2 x3 x4 x5 x6 x7 x8 x9 x10  ...  x26 x27
## [1,]  0  1  0  1  1  0  1  1  0   1         0   1
```

```
plot(GA)
```



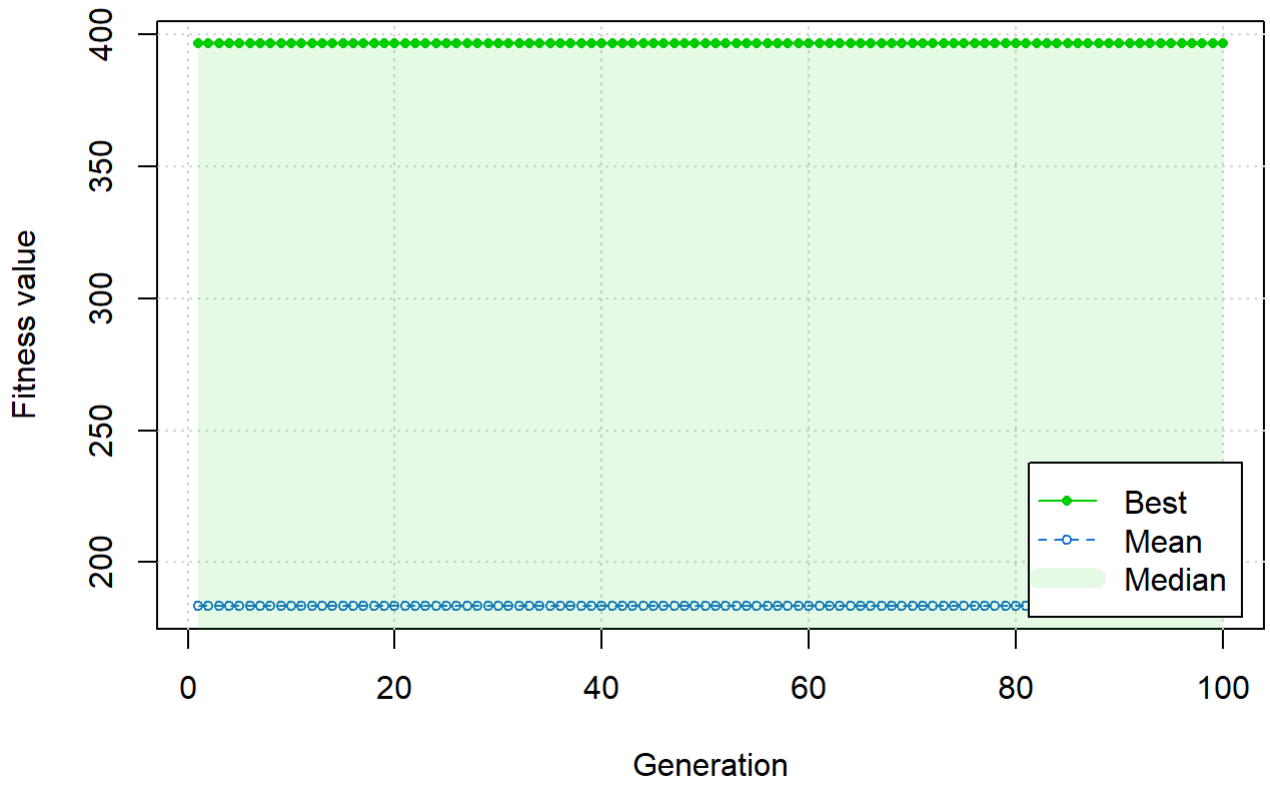
b.

$P = 20$

```
GA <- ga(type = "binary",
         fitness =  function(theta) -fitness(theta),suggestions = runs,
         popSize = 20, maxiter = 100,elitism = 20,pmutation = 0.01,
         nBits=m,seed=2022,selection = ga_lrSelection,pcrossover = 1)
summary(GA)
```

```
## -- Genetic Algorithm -------------------
##
## GA settings:
## Type                  =  binary
## Population size       =  20
## Number of generations =  100
## Elitism               =  20
## Crossover probability =  1
## Mutation probability  =  0.01
## Suggestions =
##       x1 x2 x3 x4 x5 x6 x7 x8 x9 x10  ...  x26 x27
## 1      0  0  1  0  0  1  1  1  0   1         1   0
## 2      0  0  1  0  1  1  1  0  0   0         0   0
## 3      1  1  1  0  0  0  0  1  0   1         0   0
## 4      0  0  1  0  1  0  1  1  1   0         0   1
## 5      1  0  1  0  0  0  0  0  0   0         0   0
## 6      1  0  1  1  1  0  0  0  0   0         1   1
## 7      0  1  0  1  1  0  1  1  0   1         0   1
## 8      1  0  0  0  0  0  0  1  1   0         1   0
## 9      1  0  1  1  0  0  1  1  1   1         1   1
## 10     1  0  0  1  1  0  1  0  1   0         1   0
##  ...
## 19     0  1  0  1  1  1  0  1  0   0         0   0
## 20     1  1  0  1  1  0  1  1  1   1         0   0
##
## GA results:
## Iterations             = 100
## Fitness function value = 396.6147
## Solution =
##      x1 x2 x3 x4 x5 x6 x7 x8 x9 x10  ...  x26 x27
## [1,]  0  1  0  1  1  0  1  1  0   1         0   1
```

```
plot(GA)
```

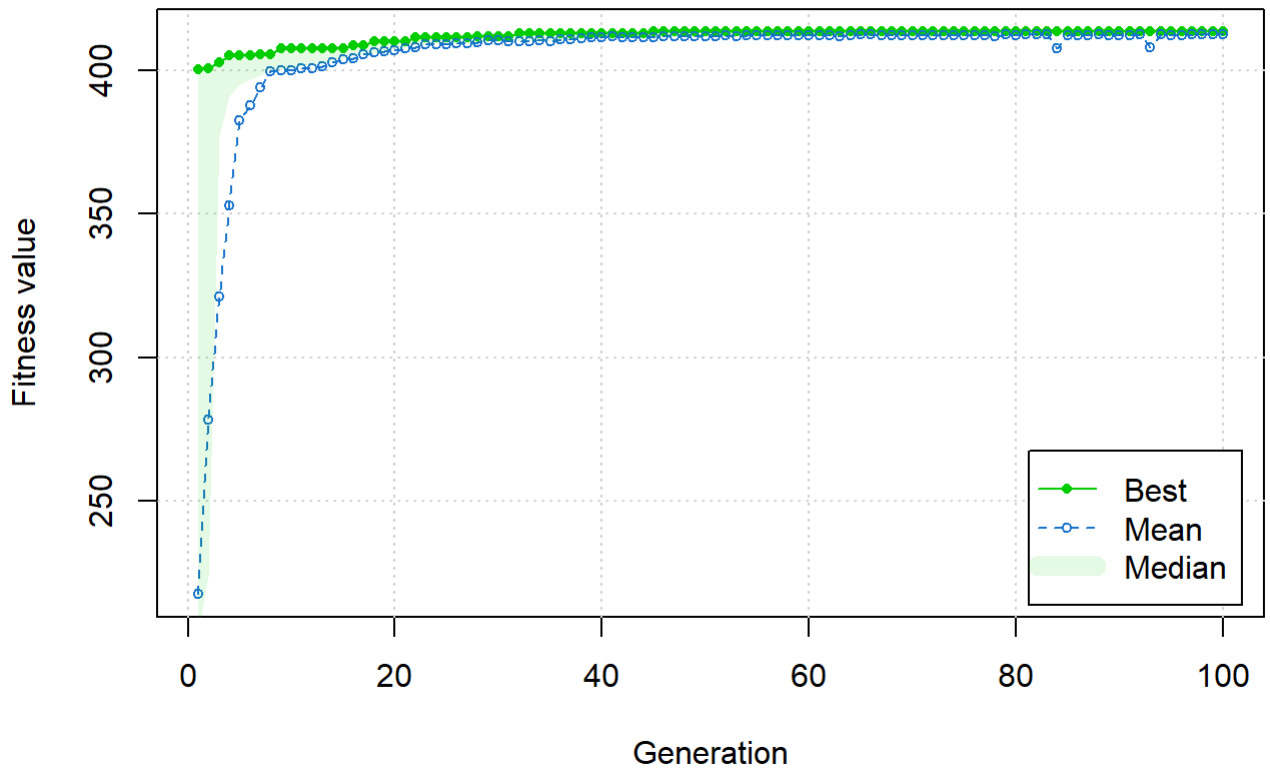


$P = 40$

```
GA <- ga(type = "binary",
         fitness =  function(theta) -fitness(theta),suggestions = runs,
         popSize = 40, maxiter = 100,elitism = 20,pmutation = 0.01,
         nBits=m,seed=2022,selection = ga_lrSelection,pcrossover = 1)
summary(GA)
```

```
## -- Genetic Algorithm -------------------
##
## GA settings:
## Type                  =  binary
## Population size       =  40
## Number of generations =  100
## Elitism               =  20
## Crossover probability =  1
## Mutation probability  =  0.01
## Suggestions =
##       x1 x2 x3 x4 x5 x6 x7 x8 x9 x10  ...  x26 x27
## 1      0  0  1  0  0  1  1  1  0   1         1   0
## 2      0  0  1  0  1  1  1  0  0   0         0   0
## 3      1  1  1  0  0  0  0  1  0   1         0   0
## 4      0  0  1  0  1  0  1  1  1   0         0   1
## 5      1  0  1  0  0  0  0  0  0   0         0   0
## 6      1  0  1  1  1  0  0  0  0   0         1   1
## 7      0  1  0  1  1  0  1  1  0   1         0   1
## 8      1  0  0  0  0  0  0  1  1   0         1   0
## 9      1  0  1  1  0  0  1  1  1   1         1   1
## 10     1  0  0  1  1  0  1  0  1   0         1   0
##  ...
## 19     0  1  0  1  1  1  0  1  0   0         0   0
## 20     1  1  0  1  1  0  1  1  1   1         0   0
##
## GA results:
## Iterations             = 100
## Fitness function value = 413.3895
## Solution =
##      x1 x2 x3 x4 x5 x6 x7 x8 x9 x10  ...  x26 x27
## [1,]  0  1  1  1  0  1  1  1  0   1         1   0
```

```
plot(GA)
```



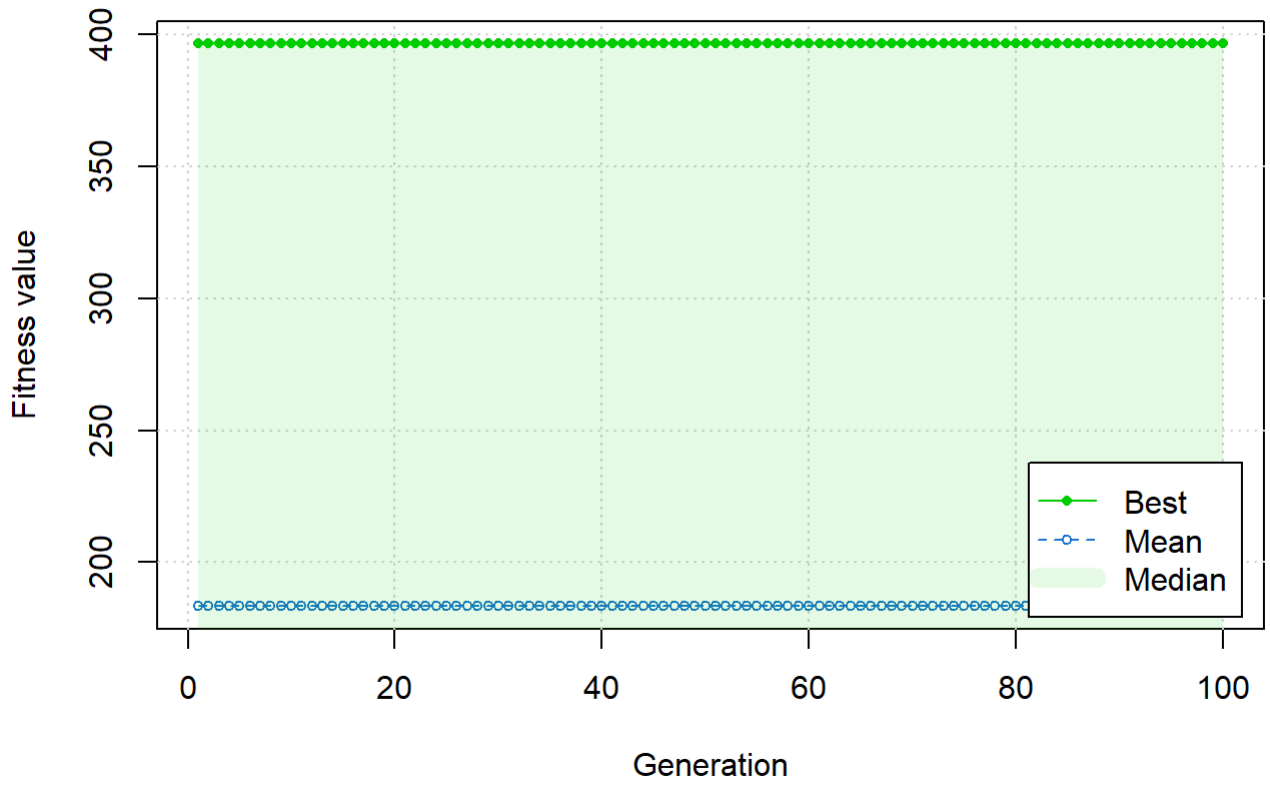
c.

i. ii.

```
GA <- ga(type = "binary",
         fitness =  function(theta) -fitness(theta),suggestions = runs,
         popSize = 20, maxiter = 100,elitism = 20,pmutation = 0.01,
         nBits=m,seed=2022,selection = ga_rwSelection,pcrossover = 1)
summary(GA)
```

```
## -- Genetic Algorithm -------------------
##
## GA settings:
## Type                  =  binary
## Population size       =  20
## Number of generations =  100
## Elitism               =  20
## Crossover probability =  1
## Mutation probability  =  0.01
## Suggestions =
##       x1 x2 x3 x4 x5 x6 x7 x8 x9 x10  ...  x26 x27
## 1      0  0  1  0  0  1  1  1  0   1         1   0
## 2      0  0  1  0  1  1  1  0  0   0         0   0
## 3      1  1  1  0  0  0  0  1  0   1         0   0
## 4      0  0  1  0  1  0  1  1  1   0         0   1
## 5      1  0  1  0  0  0  0  0  0   0         0   0
## 6      1  0  1  1  1  0  0  0  0   0         1   1
## 7      0  1  0  1  1  0  1  1  0   1         0   1
## 8      1  0  0  0  0  0  0  1  1   0         1   0
## 9      1  0  1  1  0  0  1  1  1   1         1   1
## 10     1  0  0  1  1  0  1  0  1   0         1   0
##  ...
## 19     0  1  0  1  1  1  0  1  0   0         0   0
## 20     1  1  0  1  1  0  1  1  1   1         0   0
##
## GA results:
## Iterations             = 100
## Fitness function value = 396.6147
## Solution =
##      x1 x2 x3 x4 x5 x6 x7 x8 x9 x10  ...  x26 x27
## [1,]  0  1  0  1  1  0  1  1  0   1         0   1
```

```
plot(GA)
```

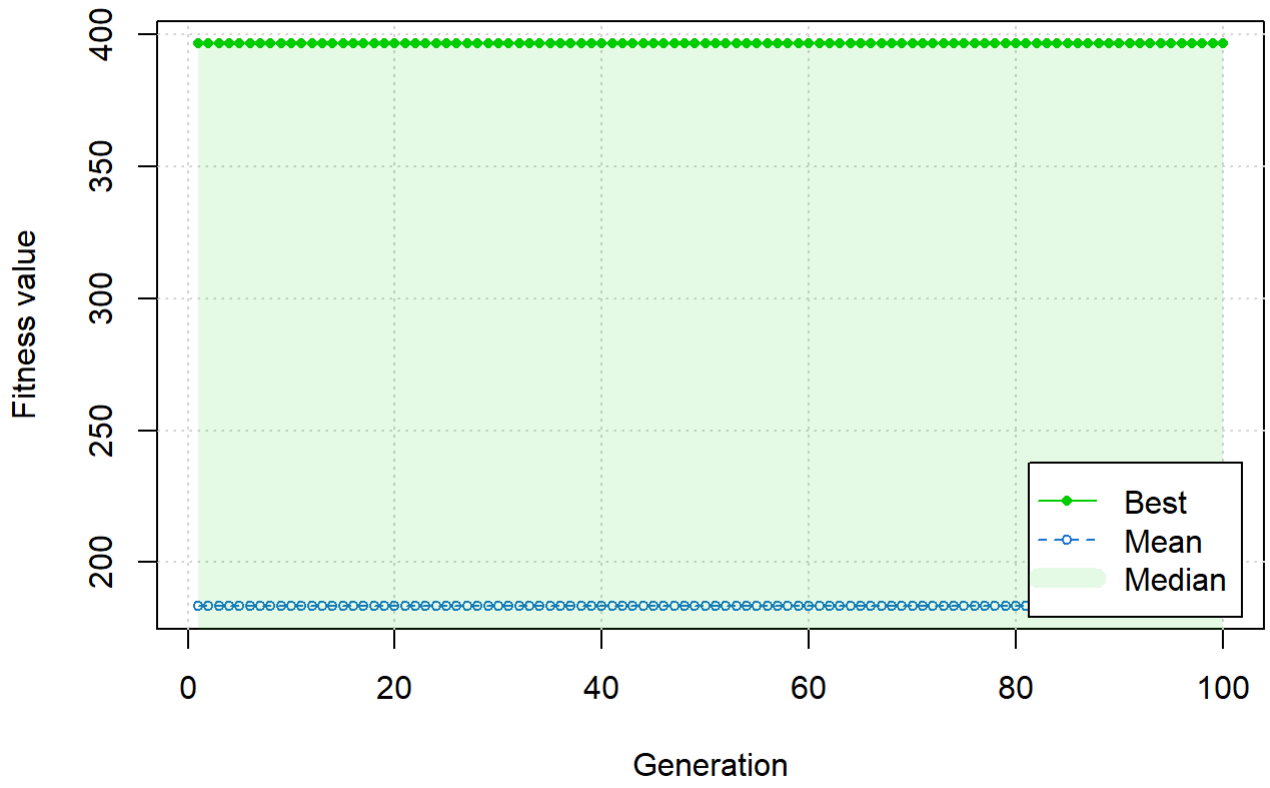


iii.

```
GA <- ga(type = "binary",
         fitness =  function(theta) -fitness(theta),suggestions = runs,
         popSize = 20, maxiter = 100,elitism = 20,pmutation = 0.01,
         nBits=m,seed=2022,selection = ga_tourSelection,
         pcrossover = 1)
summary(GA)
```

```
## -- Genetic Algorithm -------------------
##
## GA settings:
## Type                  =  binary
## Population size       =  20
## Number of generations =  100
## Elitism               =  20
## Crossover probability =  1
## Mutation probability  =  0.01
## Suggestions =
##       x1 x2 x3 x4 x5 x6 x7 x8 x9 x10  ...  x26 x27
## 1      0  0  1  0  0  1  1  1  0   1         1   0
## 2      0  0  1  0  1  1  1  0  0   0         0   0
## 3      1  1  1  0  0  0  0  1  0   1         0   0
## 4      0  0  1  0  1  0  1  1  1   0         0   1
## 5      1  0  1  0  0  0  0  0  0   0         0   0
## 6      1  0  1  1  1  0  0  0  0   0         1   1
## 7      0  1  0  1  1  0  1  1  0   1         0   1
## 8      1  0  0  0  0  0  0  1  1   0         1   0
## 9      1  0  1  1  0  0  1  1  1   1         1   1
## 10     1  0  0  1  1  0  1  0  1   0         1   0
##  ...
## 19     0  1  0  1  1  1  0  1  0   0         0   0
## 20     1  1  0  1  1  0  1  1  1   1         0   0
##
## GA results:
## Iterations             = 100
## Fitness function value = 396.6147
## Solution =
##      x1 x2 x3 x4 x5 x6 x7 x8 x9 x10  ...  x26 x27
## [1,]  0  1  0  1  1  0  1  1  0   1         0   1
```

```
plot(GA)
```

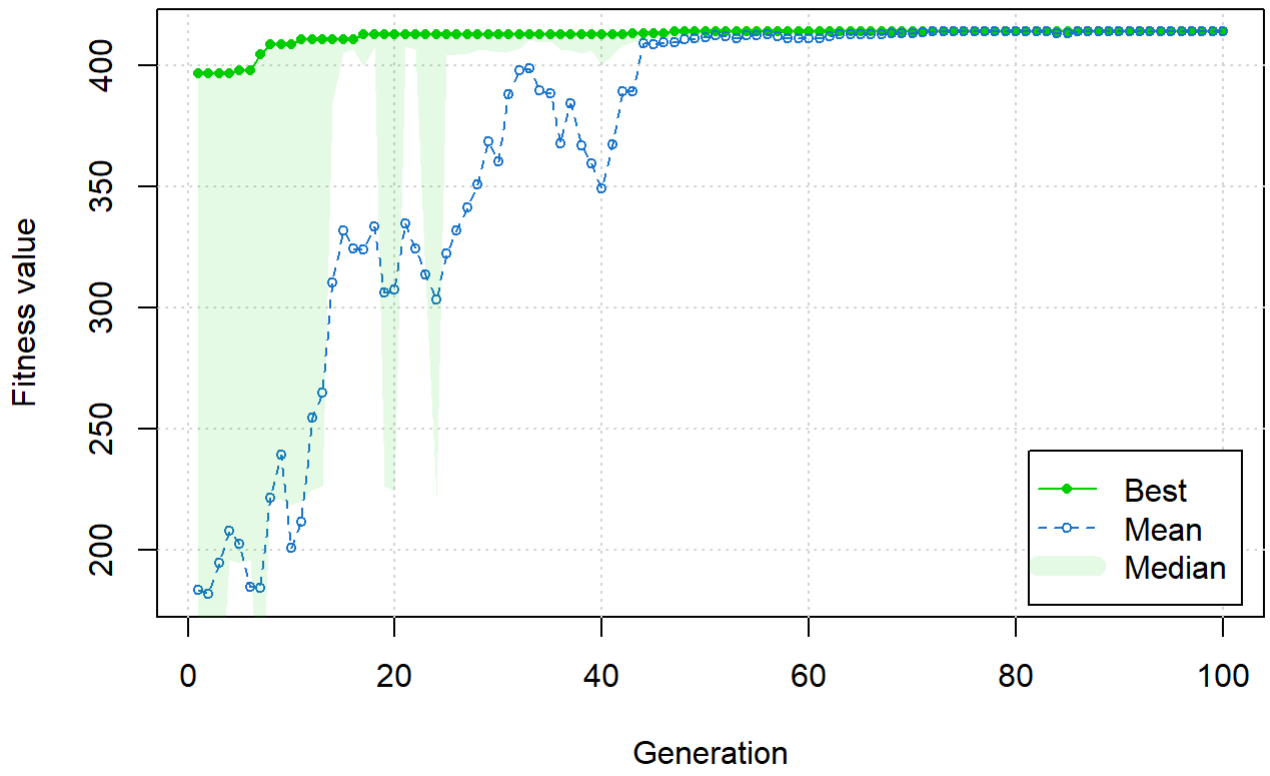


d.

```
GA <- ga(type = "binary",
         fitness =  function(theta) -fitness(theta),suggestions = runs,
         popSize = 20, maxiter = 100,elitism = 1,pmutation = 0.01,
         nBits=m,seed=2022,selection = ga_lrSelection,pcrossover = 1)
summary(GA)
```

```
## -- Genetic Algorithm -------------------
##
## GA settings:
## Type                  =  binary
## Population size       =  20
## Number of generations =  100
## Elitism               =  1
## Crossover probability =  1
## Mutation probability  =  0.01
## Suggestions =
##       x1 x2 x3 x4 x5 x6 x7 x8 x9 x10  ...  x26 x27
## 1      0  0  1  0  0  1  1  1  0   1         1   0
## 2      0  0  1  0  1  1  1  0  0   0         0   0
## 3      1  1  1  0  0  0  0  1  0   1         0   0
## 4      0  0  1  0  1  0  1  1  1   0         0   1
## 5      1  0  1  0  0  0  0  0  0   0         0   0
## 6      1  0  1  1  1  0  0  0  0   0         1   1
## 7      0  1  0  1  1  0  1  1  0   1         0   1
## 8      1  0  0  0  0  0  0  1  1   0         1   0
## 9      1  0  1  1  0  0  1  1  1   1         1   1
## 10     1  0  0  1  1  0  1  0  1   0         1   0
##  ...
## 19     0  1  0  1  1  1  0  1  0   0         0   0
## 20     1  1  0  1  1  0  1  1  1   1         0   0
##
## GA results:
## Iterations             = 100
## Fitness function value = 413.9199
## Solution =
##      x1 x2 x3 x4 x5 x6 x7 x8 x9 x10  ...  x26 x27
## [1,]  0  0  1  0  0  1  0  1  0   1         1   0
```

```
plot(GA)
```

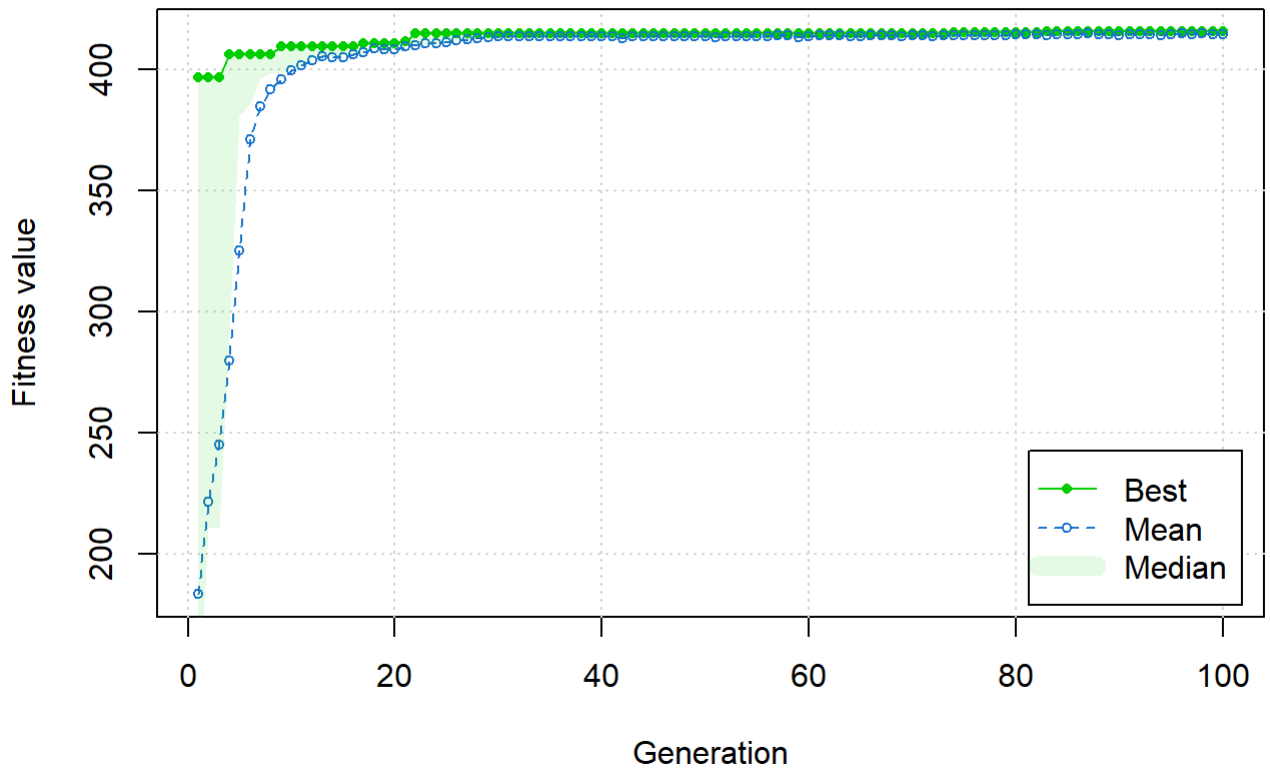


e.

```
GA <- ga(type = "binary",
         fitness =  function(theta) -fitness(theta),suggestions = runs,
         popSize = 20, maxiter = 100,elitism = 10,pmutation = 0.01,
         nBits=m,seed=2022,selection = ga_lrSelection,pcrossover = 0.5,
         crossover = gabin_uCrossover)
summary(GA)
```

```
## -- Genetic Algorithm -------------------
##
## GA settings:
## Type                  =  binary
## Population size       =  20
## Number of generations =  100
## Elitism               =  10
## Crossover probability =  0.5
## Mutation probability  =  0.01
## Suggestions =
##       x1 x2 x3 x4 x5 x6 x7 x8 x9 x10  ...  x26 x27
## 1      0  0  1  0  0  1  1  1  0   1         1   0
## 2      0  0  1  0  1  1  1  0  0   0         0   0
## 3      1  1  1  0  0  0  0  1  0   1         0   0
## 4      0  0  1  0  1  0  1  1  1   0         0   1
## 5      1  0  1  0  0  0  0  0  0   0         0   0
## 6      1  0  1  1  1  0  0  0  0   0         1   1
## 7      0  1  0  1  1  0  1  1  0   1         0   1
## 8      1  0  0  0  0  0  0  1  1   0         1   0
## 9      1  0  1  1  0  0  1  1  1   1         1   1
## 10     1  0  0  1  1  0  1  0  1   0         1   0
##  ...
## 19     0  1  0  1  1  1  0  1  0   0         0   0
## 20     1  1  0  1  1  0  1  1  1   1         0   0
##
## GA results:
## Iterations             = 100
## Fitness function value = 415.323
## Solution =
##      x1 x2 x3 x4 x5 x6 x7 x8 x9 x10  ...  x26 x27
## [1,]  0  0  1  0  0  0  0  1  0   1         0   1
```

```
plot(GA)
```



The results and computation time by GA package is similar to my own code.

The Himmelblau's function

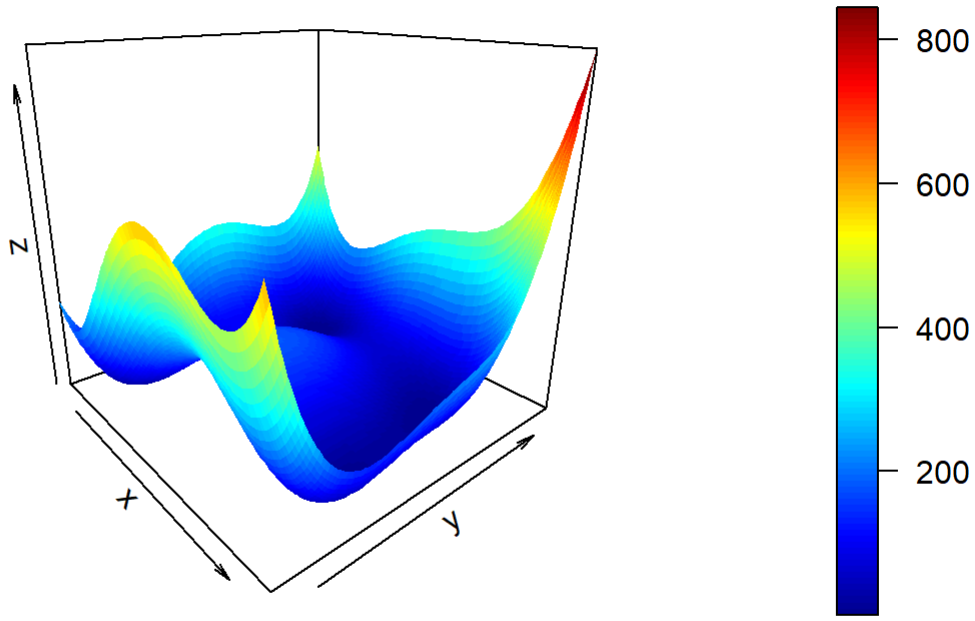
```
Himmelblau <- function(x,y){
  (x^2 + y - 11)^2 + (x + y^2 -7)^2
}

x <- seq(-5, 5, by = 0.1)
y <- seq(-5, 5, by = 0.1)

f <- outer(x, y, Himmelblau)
library(plot3D)
```

```
## Warning: 程辑包'plot3D'是用R版本4.1.3 来建造的
```

```
##
## 载入程辑包: 'plot3D'
```

```
## The following object is masked from 'package:GA':
##
##     persp3D
```

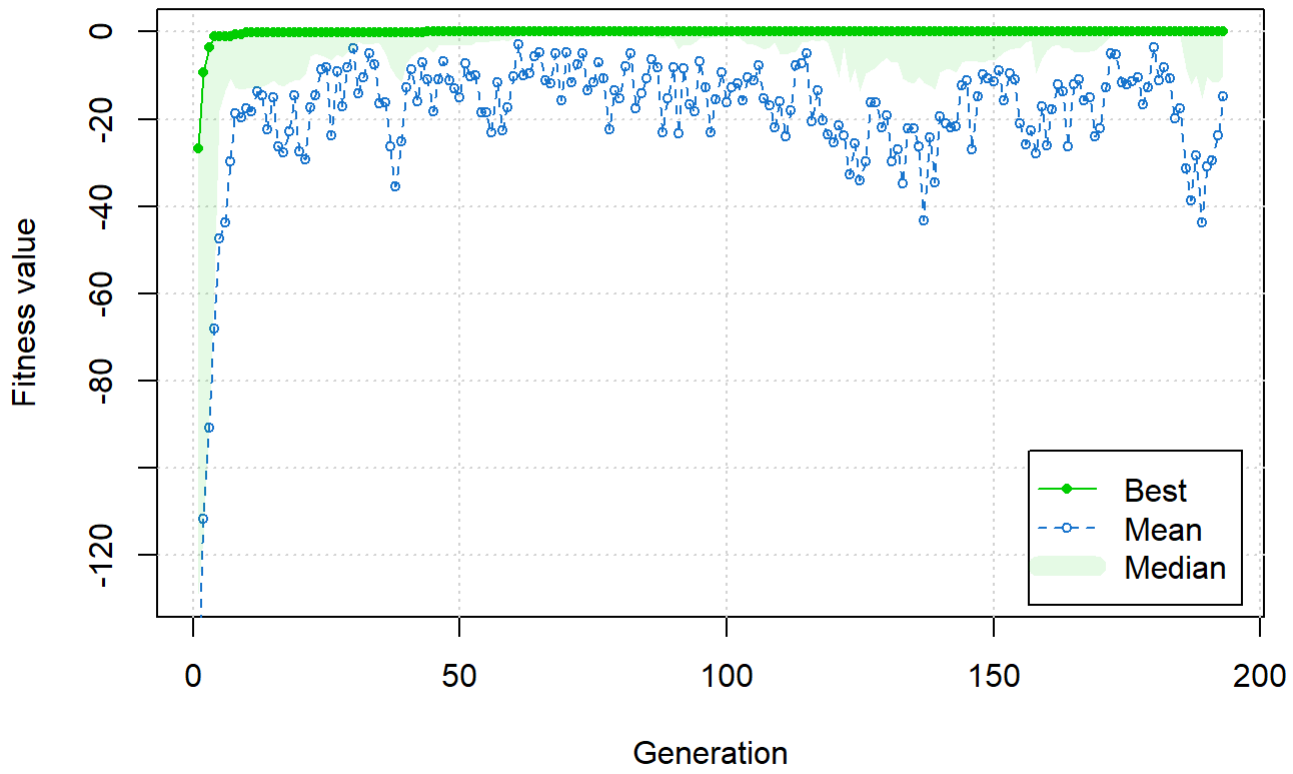
```
persp3D(x, y, f, theta = 50, phi = 20)
```




```
library(GA)
GA <- ga(type = "real-valued",
         fitness =  function(x) -Himmelblau(x[1], x[2]),
         lower = c(-5, -5), upper = c(5, 5),
         popSize = 50, maxiter = 1000, run = 100)
summary(GA)
```

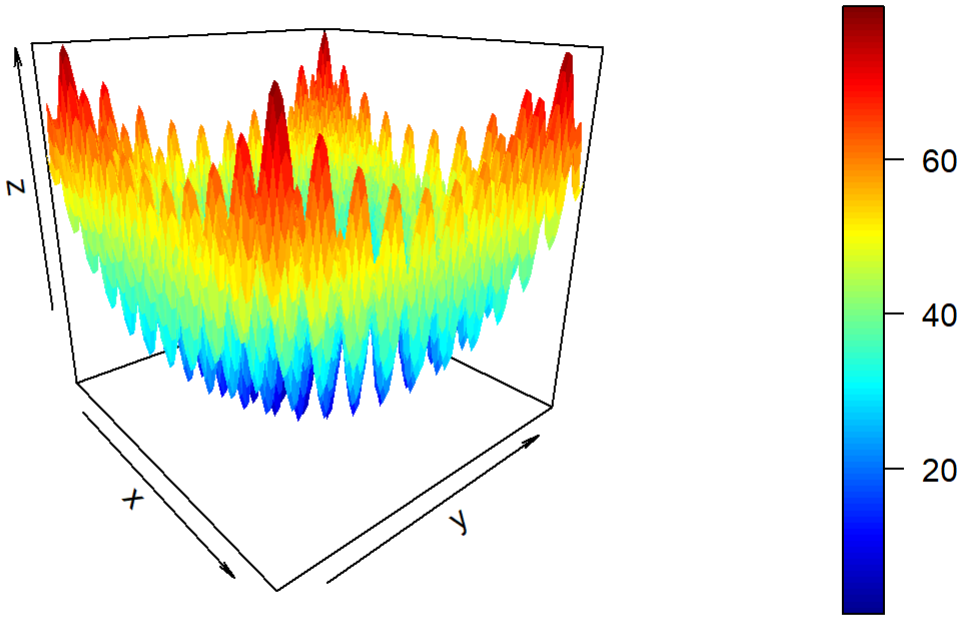
```
## -- Genetic Algorithm -------------------
##
## GA settings:
## Type                  =  real-valued
## Population size       =  50
## Number of generations =  1000
## Elitism               =  2
## Crossover probability =  0.8
## Mutation probability  =  0.1
## Search domain =
##       x1 x2
## lower -5 -5
## upper  5  5
##
## GA results:
## Iterations             = 193
## Fitness function value = -0.0004201568
## Solution =
##            x1      x2
## [1,] 2.999828 1.99513
```

```
plot(GA)
```



The Rastrigin function

```
Rastrigin <- function(x, y){
  20 + x^2 + y^2 - 10*(cos(2*pi*x) + cos(2*pi*y))
}

x <- seq(-5.12, 5.12, by = 0.1)
y <- seq(-5.12, 5.12, by = 0.1)

f <- outer(x, y, Rastrigin)
library(plot3D)
persp3D(x, y, f, theta = 50, phi = 20)
```




```
library(GA)
GA <- ga(type = "real-valued",
         fitness =  function(x) -Rastrigin(x[1], x[2]),
         lower = c(-5.12, -5.12), upper = c(5.12, 5.12),
         popSize = 50, maxiter = 1000, run = 100)
summary(GA)
```

```
## -- Genetic Algorithm -------------------
##
## GA settings:
## Type                  =  real-valued
## Population size       =  50
## Number of generations =  1000
## Elitism               =  2
## Crossover probability =  0.8
## Mutation probability  =  0.1
## Search domain =
##          x1    x2
## lower -5.12 -5.12
## upper  5.12  5.12
##
## GA results:
## Iterations             = 289
## Fitness function value = -8.926994e-08
## Solution =
##                 x1            x2
## [1,] -1.900616e-05 -9.419818e-06
```

```
plot(GA)
```