Assignment-6

TarakRam Nunna

22/11/2021

Setting working directory

```
getwd()

## [1] "C:/Users/TARAKRAM/OneDrive/Desktop/QMM_code/Assignment-6"

setwd("C:/Users/TARAKRAM/OneDrive/Desktop/QMM_code/Assignment-6")
```

Problem 1:

First, we will need to load the appropriate package and create the integer programming object. Rows would correspond to the number of nodes in the problem (9 in this case), and columns would correspond to the number of arcs (12 in this case).

```
# Load the "lpSolveAPI"

library(lpSolveAPI)

# Create an Integer Programming Object

cpm.ip <- make.lp(nrow = 9, ncol = 12)
```

Next, we will name each of the nodes and arcs.

```
# Name the arcs and nodes involved in this problem

arc.names <- c("X12", "X13", "X24", "X25", "X35", "X46", "X47", "X57", "X58",
"X69", "X79", "X89")
node.names <- c("Node1", "Node2", "Node3", "Node4", "Node5", "Node6",
"Node7", "Node8", "Node9")
```

The columns and rows in the object are renamed now.

```
# Rename the rows and columns in the interger programming object

rownames(cpm.ip) <- node.names
colnames(cpm.ip) <- arc.names
```

Now, the times for each arc must be entered into the objective function of the program. Additionally, the default value for the function is to find the minimum, so we will multiply the times by “-1”, so we find the largest negative value.

```
# Set the objective function for the problem

time <- c(5, 3, 4, 2, 3, 1, 4, 6, 2, 5, 4, 7)

set.objfn(cpm.ip, -1*time)
```

The constraints on the left hand side must now be entered into the problem.

```
# Set constraints on the left hand side of the problem

# Starting Node 1
set.row(cpm.ip, 1, c(1, 1), indices = c(1, 2))

# Intermediate Node 2
set.row(cpm.ip, 2, c(1, -1, -1), indices = c(1, 3, 4))

# Intermediate Node 3
set.row(cpm.ip, 3, c(1, -1), indices = c(2, 5))

# Intermediate Node 4
set.row(cpm.ip, 4, c(1, -1, -1), indices = c(3, 6, 7))

# Intermediate Node 5
set.row(cpm.ip, 5, c(1, 1, -1, -1), indices = c(4, 5, 8, 9))

# Intermediate Node 6
set.row(cpm.ip, 6, c(1, -1), indices = c(6, 10))

# Intermediate Node 7
set.row(cpm.ip, 7, c(1, 1, -1), indices = c(7, 8, 11))

# Intermediate Node 8
set.row(cpm.ip, 8, c(1, -1), indices = c(9, 12))

# Finish Node 9
set.row(cpm.ip, 9, c(1, 1, 1), indices = c(10, 11, 12))
```

Next, we need to set the constraint type

```
# Set constraint type

set.constr.type(cpm.ip, rep("="), 9)
```

Set the right hand side of the equation

```
# Set constraint values on the right hand side

rhs <- c(1, rep(0, 7), 1)
set.rhs(cpm.ip, rhs)
```

Now we will set the variables to all be binary

```
# Set all variables types to be binary

set.type(cpm.ip, 1:12, "binary")
```

Solve the integer programming problem with the following code

```
# Solve the integer programming problem

solve(cpm.ip)

## [1] 0

get.objective(cpm.ip)

## [1] -17

get.variables(cpm.ip)

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

To better visualize the results, we will add titles to the rows to illustrate which path to take for the problem.

```
# Place arc names with the values to get a better illustration of the path

cbind(arc.names, get.variables(cpm.ip))

##       arc.names
##  [1,] "X12"     "1"
##  [2,] "X13"     "0"
##  [3,] "X24"     "0"
##  [4,] "X25"     "1"
##  [5,] "X35"     "0"
##  [6,] "X46"     "0"
##  [7,] "X47"     "0"
##  [8,] "X57"     "1"
##  [9,] "X58"     "0"
## [10,] "X69"     "0"
## [11,] "X79"     "1"
## [12,] "X89"     "0"
```

Therefore, the critical path (longest path) is Node 1 to Node 2 to Node 5 to Node 7 to Node 9, which correspond to a time of 17 units.

Problem 2

First, we need to create a linear programming object. Based on our outline, this program will have 8 decision variables, 6 constraints, and minimum boundary conditions for each variable.

```
require(lpSolveAPI)

# Begin the lp problem with 4 constraints and 8 decision variables.

lprec <- make.lp(12,8)
```

Next, we must set the objective function into our program. As a default, the program sets the objective function to find the minimum; however, in this case we will need to find the maximum return, so we will also change that to look for the maximum value for the objective function.

```
# Set the objective function for the problem.

set.objfn(lprec, c(4,6.5,5.9,5.4,5.15,10,8.4,6.25))

# Change the direction to set maximization

lp.control(lprec, sense = "max")
```

```
## $anti.degen
## [1] "fixedvars" "stalling"
##
## $basis.crash
## [1] "none"
##
## $bb.depthlimit
## [1] -50
##
## $bb.floorfirst
## [1] "automatic"
##
## $bb.rule
## [1] "pseudononint" "greedy"       "dynamic"      "rcostfixing"
##
## $break.at.first
## [1] FALSE
##
## $break.at.value
## [1] 1e+30
##
## $epsilon
##       epsb       epsd      epsel     epsint epsperturb   epspivot
##      1e-10      1e-09      1e-12      1e-07      1e-05      2e-07
##
## $improve
## [1] "dualfeas" "thetagap"
##
## $infinite
## [1] 1e+30
##
```

```
## $maxpivot
## [1] 250
##
## $mip.gap
## absolute relative
##    1e-11    1e-11
##
## $negrange
## [1] -1e+06
##
## $obj.in.basis
## [1] TRUE
##
## $pivoting
## [1] "devex"    "adaptive"
##
## $presolve
## [1] "none"
##
## $scalelimit
## [1] 5
##
## $scaling
## [1] "geometric"   "equilibrate" "integers"
##
## $sense
## [1] "maximize"
##
## $simplextype
## [1] "dual"   "primal"
##
## $timeout
## [1] 0
##
## $verbose
## [1] "neutral"
```

All of the constraint values will also need to be added into the program. These are the total investment and diversification criteria called out in the problem statement. These will be input using the "set.row" command and defining which index to put the values at. This will prevent us from inputting a lot of "0" values manually.

```
# Set the constraint values row by row

# Total Investment Constraints:

set.row(lprec, 1, c(40,50,80,60,45,60,30,25))

# Diversification Constraints:
```

```
set.row(lprec, 2, c(40,50,80), indices = c(1,2,3))
set.row(lprec, 3, c(60,45,60), indices = c(4,5,6))
set.row(lprec, 4, c(30,25), indices = c(7,8))

# Minimum Investment Constraints:

set.row(lprec, 5, 40, indices = 1)
set.row(lprec, 6, 50, indices = 2)
set.row(lprec, 7, 80, indices = 3)
set.row(lprec, 8, 60, indices = 4)
set.row(lprec, 9, 45, indices = 5)
set.row(lprec, 10, 60, indices = 6)
set.row(lprec, 11, 30, indices = 7)
set.row(lprec, 12, 25, indices = 8)
```

Now, we will need to set the constraint values from the problem statement. In this case, these values correspond to maximum investment and diversification of each industry stocks.

```
# Set the right hand side values

rhs <- c(2500,1000,1000,1000,100,100,100,100,100,100,100,100)
set.rhs(lprec, rhs)
```

Next, we need to define the inequality values for the problem. All of these are going to be less than or equal to since the outlined problem stated the maximum capacity constraints.

```
# Set the constraint type

set.constr.type(lprec,
c("<=","<=","<=","<=",">=",">=",">=",">=",">=",">=",">=",">="))
```

For this problem, all stocks must have at least 1,000 units purchase. Therefore, this will be the lower bound for the problem.

```
# Set the boundary condition for the decision variables

set.bounds(lprec, lower = rep(0, 8))
```

We will also have to set all of the decision variables to an integer, which will allow us to satisfy the “multiples of 1000s” condition.

```
# Set the variable type to integer for all 8 variables

set.type(lprec, 1:8, "integer")
```

Print the integer programming object to review it for accuracy.

```
# Print the integer programming object to review it's accuracy

print(lprec)
```

```
## Model name:
##              C1    C2    C3    C4    C5    C6    C7    C8
## Maximize      4   6.5   5.9   5.4  5.15    10   8.4  6.25
## R1           40    50    80    60    45    60    30    25  <=  2500
## R2           40    50    80     0     0     0     0     0  <=  1000
## R3            0     0     0    60    45    60     0     0  <=  1000
## R4            0     0     0     0     0     0    30    25  <=  1000
## R5           40     0     0     0     0     0     0     0  >=   100
## R6            0    50     0     0     0     0     0     0  >=   100
## R7            0     0    80     0     0     0     0     0  >=   100
## R8            0     0     0    60     0     0     0     0  >=   100
## R9            0     0     0     0    45     0     0     0  >=   100
## R10           0     0     0     0     0    60     0     0  >=   100
## R11           0     0     0     0     0     0    30     0  >=   100
## R12           0     0     0     0     0     0     0    25  >=   100
## Kind        Std   Std   Std   Std   Std   Std   Std   Std
## Type        Int   Int   Int   Int   Int   Int   Int   Int
## Upper       Inf   Inf   Inf   Inf   Inf   Inf   Inf   Inf
## Lower         0     0     0     0     0     0     0     0
```

```
# The model can also be saved to a file

write.lp(lprec, filename = "Assignment-6-2.lp", type = "lp")
```

Finally, we will solve the integer programming problem and return the objective function value and variables

```
# Solve the integer programming model

solve(lprec)

## [1] 0

# Get the value for the objective function with the optimal values

get.objective(lprec)

## [1] 477.4

# Return the variable values for the optimal conditions

get.variables(lprec)

## [1]  3  5  2  2  3 12 29  5
```

Based on the returned values (w/ integer restriction):

Objective Function Total Return on Investment:

$477,400 (19.1% ROI)

Amount to Invest in Each Stock:

S1: 3,000 shares ($120,000 total) S2: 5,000 shares ($250,000 total) S3: 2,000 shares ($160,000 total) H1: 2,000 shares ($120,000 total) H2: 3,000 shares ($135,000 total) H3: 12,000 shares ($720,000 total) C1: 29,000 shares ($870,000 total) C2: 5,000 shares ($125,000 total)

Based on the returned values (w/o integer restriction):

Objective Function Total Return on Investment:

$487,152.80 (19.5% ROI)

Amount to Invest in Each Stock:

S1: 2,500 shares ($100,000 total) S2: 6,000 shares ($300,000 total) S3: 1,250 shares ($100,000 total) H1: 1,667 shares ($100,020 total) H2: 2,222 shares ($99,990 total) H3: 13,333 shares ($799,980 total) C1: 30,000 shares ($900,000 total) C2: 4,000 shares ($100,000 total)