Linear programming

The company produces two products, A and B, with a requirement that sales from product A comprise at least 80% of total sales($A + B$). There are 100 type A products in the saturated market. Producing one unit of A needs 2 kg of material, while B needs 4 kg. With a total of 240 kg material available, and profits of 20€ for A and 50€ for B, determine the optimal production strategy to maximize the company's profit within these constraints. Use linear programming.

Linear programming with lpSolve package

Lets' describe the above text with linear program.

Objective function:

Maxsimize: $20A + 50B$

Constraints:

$A \geq 0.8(A + B)$ - *80% of all sales are product A*

$A \leq 100$ - *market saturation*

$2A + 4B \leq 240$ - *materials limit*

$A, B \geq 0$ - *non zero constraint*

We will use the package called **lpSolve** which requires a different form for the first constraint:

$$
\begin{aligned}
A &\geq 0.8(A + B) \\
0.8(A + B) - A &\leq 0 \\
0.8B - 0.2A &\leq 0
\end{aligned}
$$

Now let's write this in R.

```
#optimization function
f.opt <- c(20, 50)

#constraints
f.con <- matrix(c(-0.2, 0.8, 1, 0, 2, 4, 1, 0, 0, 1), ncol = 2, byrow = TRUE)

#operators
f.dir <- c("<=", "<=", "<=", ">=", ">=")

#right hand side of constraints
f.rhs <- c(0, 100, 240, 0, 0)

library(lpSolve)
#function lp returns the value of the best solution
rez <- lp("max", f.opt, f.con, f.dir, f.rhs)
```

We can find the exact values of A and B.

```
rez$solution
```

```
## [1] 80 20
```

Or the maximum of the optimized function.

```
rez
```

```
## Success: the objective function is 2600
```

Linear program with lpSolveAPI package

Let's solve the same example with package **lpSolveAPI**.

The idea is that we create an empty linear problem and add constraint and settings later.

Let's create an empty linear problem.

```
# Load lpSolveAPI
require(lpSolveAPI)
```

```
## Loading required package: lpSolveAPI
```

```
# Set an empty linear problem
linProgram <- make.lp(nrow = 0, ncol = 2)
```

Now we can set objective function. The next few lines give output which is hidden here in this pdf.

```
# Set the linear program as a maximanization
lp.control(linProgram, sense="max")
# Set type of decision variables
set.type(linProgram, 1:2, type=c("real"))
# Set objective function coefficients vector C
set.objfn(linProgram, c(20, 50))
```

We can add constraints one by one.

```
# Add constraints
add.constraint(linProgram, c(-0.2, 0.8), "<=", 0)
add.constraint(linProgram, c(1, 0), "<=", 100)
add.constraint(linProgram, c(2, 4), "<=", 240)
add.constraint(linProgram, c(1, 0), ">=", 0)
add.constraint(linProgram, c(0, 1), ">=", 0)
```

Let's check the final linear program before running it.

```
# Display the LPsolve matrix
linProgram
```

```
## Model name:
##             C1    C2
## Maximize    20    50
## R1        -0.2   0.8  <=    0
## R2           1     0  <=  100
## R3           2     4  <=  240
## R4           1     0  >=    0
## R5           0     1  >=    0
## Kind       Std   Std
## Type      Real  Real
## Upper      Inf   Inf
## Lower        0     0
```

Run the program.

```
# Solve problem
solve(linProgram)
```

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

We can find the values of A and B.

```r
# Get the variables
get.variables(linProgram)
```

```
## [1] 80 20
```

And we can find the value of the objective function.

```r
# Get the value of the objective function
get.objective(linProgram)
```

```
## [1] 2600
```

Exercises

Maximal flow

You have a graph $G(V,E)$, where each edge (u,v) of E has a positive flow $c(u,v) \geq 0$. You also have a starting vertex s and terminal vertex t. Find the maximal flow from s to t.

The graph consists of vertices $V = s, t, v1, v2, v3, v4, v5$ and capacities:

$$\begin{array}{lll} c(s,v1) = 20, & c(s,v4) = 7, & c(v1,v2) = 15, \\ c(v1,v3) = 5, & c(v2,t) = 12, & c(v2,v5) = 5, \\ c(v3,v4) = 2, & c(v3,v5) = 4, & c(v4,v1) = 3, \\ c(v4,v5) = 8, & c(v5,t) = 17 & \end{array}$$

To describe this problem as a linear program we need a variable $f(u,v)$ on each edge that represent the current flow of the edge between u and v. With this in mind we can define objective function and constraints.

Objective function:

Maxsimize: $\sum\limits_{v \in V} f(s,v)$

We could also use:

Maxsimize: $\sum\limits_{u \in V} f(u,t)$

Constraints:

$f(u,v) \leq c(u,v) \quad \forall u, v$

$\sum\limits_{u \in V} f(u,v) = \sum\limits_{u \in V} f(v,u) \quad \forall v$, which equals $\sum\limits_{u \in V} f(u,v) - \sum\limits_{u \in V} f(v,u) \geq 0 \quad \forall v$

$f(u,v) \geq 0$

Let's write this program in R.

```r
f.obj <- c(1,1,0,0,0,0,0,0,0,0,0)

f.con <- matrix(c(1,0,0,0,0,0,0,0,0,0,0,
                  0,1,0,0,0,0,0,0,0,0,0,0,
                  0,0,1,0,0,0,0,0,0,0,0,0,
                  0,0,0,1,0,0,0,0,0,0,0,0,
                  0,0,0,0,1,0,0,0,0,0,0,0,
                  0,0,0,0,0,1,0,0,0,0,0,0,
                  0,0,0,0,0,0,1,0,0,0,0,0,
                  0,0,0,0,0,0,0,1,0,0,0,0,
                  0,0,0,0,0,0,0,0,1,0,0,0,
```

```
                  0,0,0,0,0,0,0,0,0,1,0,
                  0,0,0,0,0,0,0,0,0,0,1,
                  -1,0,1,1,0,0,0,0,-1,0,0,
                  0,0,-1,0,1,1,0,0,0,0,0,
                  0,0,0,-1,0,0,1,1,0,0,0,
                  0,-1,0,0,0,0,-1,0,1,1,0,
                  0,0,0,0,0,-1,0,-1,0,-1,1), ncol = 11, byrow = TRUE)
f.dir <- c(rep("<=", 11), rep("=", 5))
f.rhs <- c(20,7,15,5,12,5,2,4,3,8,17,0,0,0,0,0)
mf <- lp("max",f.obj, f.con, f.dir, f.rhs)
```

And check the solution.

```
mf
```

```
## Success: the objective function is 27
```

```
mf$solution
```

```
##  [1] 20  7 15  5 12  3  1  4  0  8 15
```

Shortest path

Write a function that will return the shortest path for graphs generated by `make_my_graph()`.

First let's load the required packages and `make_my_graph()` function. The packages are required for easier work with graphs and their visualization.

```
library(tidyverse)
library(lpSolveAPI)
library(igraph)
make_my_graph <- function(v, e){
  #first create a path so that the problem will always be solvable
  #number of vertices on path excluding 1 and v
  path_size <- ceiling(runif(1, min = 2, max = v-1))-1
  path_ind <- c(1, sample(2:(v-1), path_size), v)
  path_from <- path_ind[1:(length(path_ind)-1)]
  path_to <- path_ind[2:(length(path_ind))]
  path_from_to <- bind_cols(from = path_from, to = path_to)

  #print(path_from_to)

  from <- sample(1:v, e - (path_size + 1), replace = T) #create start vertices
  to <- sample(1:v, e - (path_size + 1), replace = T) #create end vertices

  #create indexes for all other edges
  from_to_other <- bind_cols(from = from, to = to)
  #combine and delete duplicates and self-loops
  from_to <- bind_rows(path_from_to, from_to_other) %>%
    distinct() %>%
    filter(from != to)
  print(from_to)
  edges <- as.vector(t(from_to))
  #the above part gives approximate number of edges
  #add randomly using while
  print(paste0("Current edges: ", length(edges)/2, " Desired edges: ", e))
  while(length(edges)/2 < e){
```

```
    #add one more edge (and check for self-loops and duplication)
    from_to <- bind_rows(from_to, c(from = sample(1:v,1), to = sample(1:v,1))) %>%
      distinct() %>% filter(from != to)
    edges <- as.vector(t(from_to))
  }
  print(paste0("Current edges: ", length(edges)/2, " Desired edges: ", e))
  make_empty_graph() %>%
    add_vertices(1, color = "green") %>%
    add_vertices(v-2, color = "red") %>%
    add_vertices(1, color = "green") %>%
    add_edges(edges) %>%
    set_edge_attr("cost", value = c(1,runif(length(edges)/2-2, 0.1, 2), 1))
}
```

First let's create a sample graph that we will work with.

```
set.seed(12345678)
g <- make_my_graph(30, 55)
```

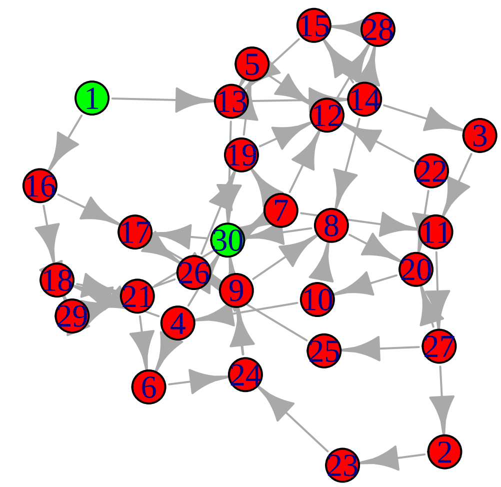
We can easily convert graph to a data frame.

```
head(as_data_frame(g))
```

```
##   from to      cost
## 1    1 16 1.0000000
## 2   16 17 1.1588181
## 3   17 26 1.2067410
## 4   26 19 0.7486961
## 5   19  5 0.6024305
## 6    5 12 0.3328848
```

Lets visualize this graph.

```
plot(g)
```



First, let's define the linear program analytically and then solve it using R.

We have a graph $G = (V, E)$ with edges (u, v) each with its own cost $c(u, v)$ and a starting vertex $s = 1$ and terminal vertex $t = n$. The solution mimics the Bellman-Ford algorithm. We can define a set of variables d_i for each vertex representing the shortest path cost to vertex i.

Objective function:

Maxsimize: d_t

Constraints:

$d_v \leq d_u + c(u, v) \quad \forall (u, v) \in E$

$d_s = 0$

$d_v \geq 0$

R solution.

```
#Finish the following function
return_shortest_path <- function(g){
  data <- as_data_frame(g)
  numOfVariables <- max(data[,1:2])
  lp <- make.lp(nrow = 0, ncol = numOfVariables)
  lp.control(lp, sense="max")


  # Set type of decision variables
  set.type(lp, 1:numOfVariables, type=c("real"))
  # Set objective function coefficients vector C
```

```
  set.objfn(lp, c(rep(0, numOfVariables-1), 1))

  # Add constraints
  add.constraint(lp, c(1, rep(0, numOfVariables-1)), "=", 0)
  for(i in 1:nrow(data)){
    newC <- rep(0, numOfVariables)
    newC[data[i,1]] <- -1
    newC[data[i,2]] <- 1
    add.constraint(lp, newC, "<=", data[i,3])
  }

  # Display the LPsolve matrix
  lp

  # Solve problem
  solve(lp)

  # Get the variables
  print("Selected variables")
  print(get.variables(lp))
  # Get the value of the objective function
  print("Final path cost")
  print(get.objective(lp) )
  lp
}
```

Let's look at the solution:

```
sol <- return_shortest_path(g)
```

```
## [1] "Selected variables"
##  [1] 0.0000000 0.0000000 0.0000000 1.5365389 0.5743080 0.0000000 0.6080444
##  [8] 0.5239623 0.0000000 0.3433549 0.0000000 0.0000000 0.9378576 0.0000000
## [15] 0.1578226 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## [22] 0.0000000 0.0000000 0.1600864 0.0000000 0.0000000 0.0000000 0.0000000
## [29] 0.0000000 1.9378576
## [1] "Final path cost"
## [1] 1.937858
```

The selected vertices on the path are all with non zero values.

```
c(1, which(get.variables(sol) > 0))
```

```
##  [1]  1  4  5  7  8 10 13 15 24 30
```

A smaller example:

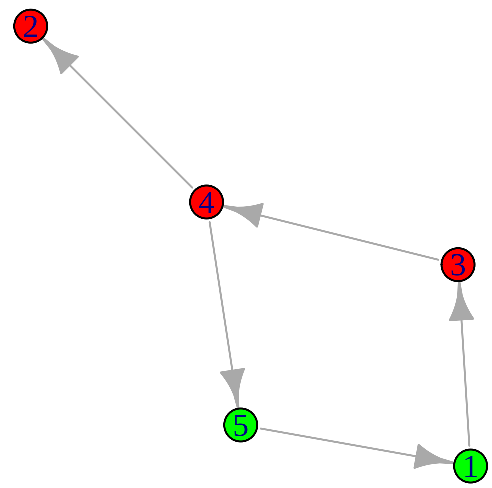
#example showing a simple path on a smaller graph

```
set.seed(1234)
g <- make_my_graph(5, 5)
```

```
## # A tibble: 4 x 2
##    from    to
##   <dbl> <dbl>
## 1     1     3
## 2     3     4
```

```
## 3     4     5
## 4     5     1
## [1] "Current edges: 4 Desired edges: 5"
## [1] "Current edges: 5 Desired edges: 5"
```

```
plot(g)
```



Find the solution.

```
sol <- return_shortest_path(g)
```

```
## [1] "Selected variables"
## [1] 0.000000 0.000000 1.000000 2.854524 3.509924
## [1] "Final path cost"
## [1] 3.509924
```

```
as_data_frame(g)
```

```
##   from to      cost
## 1    1  3 1.0000000
## 2    3  4 1.8545236
## 3    4  5 0.6554001
## 4    5  1 1.6908617
## 5    4  2 1.0000000
```

Path.

```
c(1, which(get.variables(sol) > 0))
```

```
## [1] 1 3 4 5
```

We can see the linear programm.

```
sol
```

```
## Model name:
##              C1    C2    C3    C4    C5
## Maximize      0     0     0     0     1
## R1            1     0     0     0     0   =                  0
## R2           -1     0     1     0     0  <=                  1
## R3            0     0    -1     1     0  <=   1.85452362012584
## R4            0     0     0    -1     1  <=  0.655400096485391
## R5            1     0     0     0    -1  <=    1.6908616934903
## R6            0     1     0    -1     0  <=                  1
## Kind        Std   Std   Std   Std   Std
## Type       Real  Real  Real  Real  Real
## Upper       Inf   Inf   Inf   Inf   Inf
## Lower         0     0     0     0     0
```