(SMM641) - Revenue Management & Pricing. Network Revenue Management Part 2 - R Supplement

Oben Ceryan

Contents

1 Required Packages and Data

1.1 Use Package lpSolve

Type the following code to install and activate the **lpSolve** package:

```
install.packages("lpSolve",repos = "http://cran.us.r-project.org")
```

```
##
## The downloaded binary packages are in
##  /var/folders/rx/_gm3py093nd25flx8hwzm1bm0000gp/T//RtmpoCxTg5/downloaded_packages
```

```
library(lpSolve)
```

```
## Warning: package 'lpSolve' was built under R version 3.5.2
```

2 Linear Programming - An Introductory Example

2.1 Setting up Linear Programming Formulation in R

Please see class slides for the problem description.

```
# Setting up the LP Formulation:

# This problem has two decision variables
# x1: Amount of Red Ripe to produce
# x2: Amount of Sassy Spicy to produce

# Objective Function Coefficients
# Profit contribution of Red Ripe is 3.
# Profit contribution of Sassy Spicy is 2.
obj.fun <- c(3,2);

# Constraint Coefficients
constr <- matrix(c(1,2,2,1,-1,1,0,1), ncol=2, byrow=TRUE)
# Alternatively, and especially for larger problems,
# you can build the constraint coefficient matrix in parts
# and combine them afterwards.  For example,
# Pepper usage: x1+2*x2
pepper<-c(1,2)
# Tomato usage: 2*x1+x2
```

```r
tomato<-c(2,1)
# Diversity: -x1+x2
diversity<-c(-1,1)
# Demand: x2
demand<-c(0,1)
constr<-rbind(pepper,tomato,diversity,demand)

# Constraint directions: Equality and/or Inequality
# For each row of constraint, indicate the
# constraint direction, "<=", ">=", or "=".
constr.dir <- c("<=", "<=", "<=", "<=")

# Constraint Right Hand Side
# For each row of constraint, indicate the
# corresponding right hand side value.
# First two are pepper and tomato availability:
# x1+2*x2 <= 6 and 2*x1+x2 <= 8
# The next is for diversity (-x1+x2 <= 1)
# The last is for demand (x2 <= 2)
rhs <- c(6,8,1,2)
```

2.2 Solving the LP

```r
# Solving the LP: (type "max" or "min" for first argument)
prod.sol <- lp("max", obj.fun, constr, constr.dir, rhs, compute.sens=TRUE)
```

2.3 Obtaining the Optimal Solution

```r
# Optimal Solution (Values for Decision Variables)
prod.sol$solution
```

```
## [1] 3.333333 1.333333
```

2.4 Obtaining the Optimal Objective Function Value

```r
# Optimal Objective Function Value
prod.sol$objval
```

```
## [1] 12.66667
```

2.5 Sensitivity Analysis

2.5.1 Changes in Objective Function Coefficients

Range for which the solution remains optimal. For details, please see slides.

```
# Sensitivity Analysis
# Changes in Objective Function Coefficients
prod.sol$sens.coef.from
```

```
## [1] 1.0 1.5
```

```
prod.sol$sens.coef.to
```

```
## [1] 4 6
```

2.5.2 Changes in Constraint Quantity

2.5.2.1 Shadow Price of Constraints

Shadow Price of Constraints, i.e., the price of each resource. For details, please see slides.

```
# Shadow (Dual) Value of Constraints
prod.sol$duals[1:length(constr.dir)]
```

```
## [1] 0.3333333 1.3333333 0.0000000 0.0000000
```

2.5.2.2 Ranges over which Shadow Prices Remain Valid

The ranges over which the shadow prices for constraints remain valid. For details, please see slides.

```
# The range over which the shadow price for constraints remain valid.
# Note, lpSolve only computes these ranges for nonzero shadow prices.
# If a shadow price is zero, and the right hand side is to be changed,
# lpSolve would require the LP to be run again with the new parameters.
prod.sol$duals.from[1:length(constr.dir)]
```

```
## [1]  4e+00  6e+00 -1e+30 -1e+30
```

```
prod.sol$duals.to[1:length(constr.dir)]
```

```
## [1] 7.0e+00 1.2e+01 1.0e+30 1.0e+30
```

3 Example: Convention Hotel

Please see class slides for the problem description.

3.1 Setting up Linear Programming Formulation

```
# Setting up the LP Formulation:

# The problem has 6 decision variables:
# x1: Allocations for convention goers for two nights
# x2: Allocations for convention goers for Friday only
# x3: Allocations for convention goers for Saturday only
# x4: Allocations for regular customers for two nights
# x5: Allocations for regular customers for Friday only
# x6: Allocations for regular customers for Saturday only

# Objective Function Coefficients
obj.fun <- c(225,123,130,295,146,152);

# Constraint Coefficients

# We will build the constraint coefficient matrix in parts
# and combine them afterwards.

# First, consider the demand constraints.
# Allocations for each type of reservation
# should be less than or equal to the anticipated demand.
# The coefficients for this constraint requires
# a diagonal matrix of size 6.
AllocateLessThanDemand<-diag(1, 6, 6)

# Second, consider the minimum allocation constraint
# The hotel needs to allocate at least half of its
# capacity to convention goers.
# Total Allocation for convention goers on Friday is x1+x2
AllocateAtLeast48onFriday<-c(1,1,0,0,0,0)
# Total Allocation for convention goers on Saturday is x1+x3
AllocateAtLeast48onSaturday<-c(1,0,1,0,0,0)
```

```
# Third, consider the maximum allocation constraint
# The hotel can allocate at most 96 rooms each day
# Total Allocation for Friday is x1+x2+x4+x5
AllocateLessThanCapacityFriday<-c(1,1,0,1,1,0)
# Total Allocation for Saturday is x1+x3+x4+x6
AllocateLessThanCapacitySaturday<-c(1,0,1,1,0,1)

# Bring all constraint coefficients together to form the
# constraint coefficient matrix
constr <- rbind(AllocateLessThanDemand,
                AllocateAtLeast48onFriday,AllocateAtLeast48onSaturday,
                AllocateLessThanCapacityFriday,AllocateLessThanCapacitySaturday);

# Constraint directions:
# For each row of constraint, indicate the
# constraint direction, "<=", ">=", or "=".
constr.dir <- c(rep("<=", 6),rep(">=", 2),rep("<=", 2));

# Constraint Right Hand Side
# For each row of constraint, indicate the
# corresponding right hand side value.
# First six are demand constraints
# The next two are minimum convention allocation of half capacity, 48 rooms
# The last two are maximum total allocation with capacity 96.
rhs <- c(40,20,15,20,30,25,48,48,96,96)
```

3.2 Solving the LP

```
# Solving the LP: (type "max" or "min" for first argument)
optconvention <- lp("max", obj.fun, constr, constr.dir, rhs, compute.sens=TRUE)
```

3.3 Obtaining the Optimal Solution

```
# Optimal Solution (Values for Decision Variables)
# The optimal allocations are:
optconvention$solution
```

```
## [1] 36 12 15 20 28 25
```

```
for (i in 1:6){
    print(paste("Allocate",optconvention$solution[i],"rooms for Reservation Type", i))
}
```

```
## [1] "Allocate 36 rooms for Reservation Type 1"
## [1] "Allocate 12 rooms for Reservation Type 2"
## [1] "Allocate 15 rooms for Reservation Type 3"
## [1] "Allocate 20 rooms for Reservation Type 4"
## [1] "Allocate 28 rooms for Reservation Type 5"
## [1] "Allocate 25 rooms for Reservation Type 6"
```

3.4 Obtaining the Optimal Objective Function Value

```
# The optimal revenue is:
optconvention$objval
```

```
## [1] 25314
```

```
print(paste("The optimal revenue is:",optconvention$objval))
```

```
## [1] "The optimal revenue is: 25314"
```

4 Network Revenue Management with Bid Price Heuristics

4.1 Problem Parameters

```
# Network Revenue Management with Dynamic Porgramming

N1=100; # Leg 1 seat availability
N2=120; # Leg 2 seat availability
TT=300; # Length of time horizon

arrivalprob=c(1/5, 4/15, 1/6, 4/15);

price=c(150,120,250,180);

totalarrivalprob=sum(arrivalprob);
noarrivalprob=1-totalarrivalprob;
```

4.2 Solving the corresponding LP with Expected Demand Values

```
# Airline DP with LP Heuristics

# Objective Function Coefficients
obj.fun <- c(150,120,250,180);
expdemands <- arrivalprob*300;
AllocateLessThanDLcap<-c(1,0,1,1)
AllocateLessThanLEcap<-c(0,1,1,1)
AllocateLessThanDemand<-diag(1, 4, 4)

constr <- rbind(AllocateLessThanDLcap,AllocateLessThanLEcap,
                AllocateLessThanDemand);

# Constraint directions:
constr.dir <- c(rep("<=", 2),rep("<=", 4));

# Constraint Right Hand Side
rhs <- c(100,120,expdemands)

# Solving the LP:
optairline <- lp("max", obj.fun, constr, constr.dir, rhs, compute.sens=TRUE)

# Optimal Solution (Values for Decision Variables)
optairline$solution
```

```
## [1] 60 80 40  0
```

```
# Optimal Objective Function Value
revenueLP<-optairline$objval
print(revenueLP)
```

```
## [1] 28600
```

4.3 Obtaining Bid Prices for Heuristics

```
# Bid Prices for Capacity Constraints
bidprices<-optairline$duals[1:2]

# The Bid Price of the first flight leg
```

```
print(paste("The Bid Price for the first flight leg:",bidprices[1]))
```

```
## [1] "The Bid Price for the first flight leg: 130"
```

```
# The Bid Price of the second flight leg
print(paste("The Bid Price for the second flight leg:",bidprices[2]))
```

```
## [1] "The Bid Price for the second flight leg: 120"
```

```
# We can set a heuristic accapetance rule as follows:
# Accept all Product 1 demand as price>=value of resources.
# Accept all Product 2 demand as price>=value of resources.
# Accept all Product 3 demand as price>=value of resources.
# Do not accept Product 4 demand as price<value of resources.
```

4.4 Implementing the Bid Price Heuristic

```
# Defining arrays with correct dimensions:
v=array(rep( 0, len=(N1+1)*(N2+1)*(TT+1)), dim=c(N1+1,N2+1,TT+1));
accept1=array(rep( 0, len=(N1+1)*(N2+1)*(TT+1)), dim=c(N1+1,N2+1,TT+1));
accept2=array(rep( 0, len=(N1+1)*(N2+1)*(TT+1)), dim=c(N1+1,N2+1,TT+1));
accept3=array(rep( 0, len=(N1+1)*(N2+1)*(TT+1)), dim=c(N1+1,N2+1,TT+1));
accept4=array(rep( 0, len=(N1+1)*(N2+1)*(TT+1)), dim=c(N1+1,N2+1,TT+1));

# Initialization / SettingTerminal Values:
for(i in 1:(N1+1)){
    for(j in 1:(N2+1)){
        v[i,j,1]=0;
    }
}

# The Value Function Recursions

for(t in 2:(TT+1)){ #2:TT+1
    for(i in 1:(N1+1)){ #1:N1+1
        for(j in 1:(N2+1)){ #1:N2+1

            # For no arrivals:
            vforarrival0=v[i,j,t-1];
```

```
        # For Product 1 arrival:
        # default not accept unless able to accept
        vforarrival1=v[i,j,t-1];
        accept1[i,j,t]=0;
        # If resource available:
        if(i>1){
            vforarrival1=price[1]+v[i-1,j,t-1];
            accept1[i,j,t]=1;
        }

        # For Product 2 arrival:
        # default not accept unless able to accept
        vforarrival2=v[i,j,t-1];
        accept2[i,j,t]=0;
        # If resource available:
        if(j>1){
            vforarrival2=price[2]+v[i,j-1,t-1];
            accept2[i,j,t]=1;
        }

        # For Product 3 arrival:
        # default not accept unless able to accept
        vforarrival3=v[i,j,t-1];
        accept3[i,j,t]=0;
        # If resources available:
        if(i>1){
            if(j>1){
                vforarrival3=price[3]+v[i-1,j-1,t-1];
                accept3[i,j,t]=1;
            }
        }

        # For Product 4 arrival:
        # Do not accept
        vforarrival4=v[i,j,t-1];
        accept4[i,j,t]=0;

        # Obtaining the overall value function from its parts:
        v[i,j,t]=noarrivalprob*vforarrival0+
```

```
                arrivalprob[1]*vforarrival1+
                arrivalprob[2]*vforarrival2+
                arrivalprob[3]*vforarrival3+
                arrivalprob[4]*vforarrival4;
        }
    }
}

# Bid Price Heuristic Revenue
revenueBidPriceHeuristic<-v[101,121,301]
print(revenueBidPriceHeuristic)
```

```
## [1] 28150.06
```