SMM 641
Revenue Management and Pricing

Week 5:

Bid Prices

Bid Price Heuristics

Buy-Up and Buy-Down Behaviour in Capacity Allocation

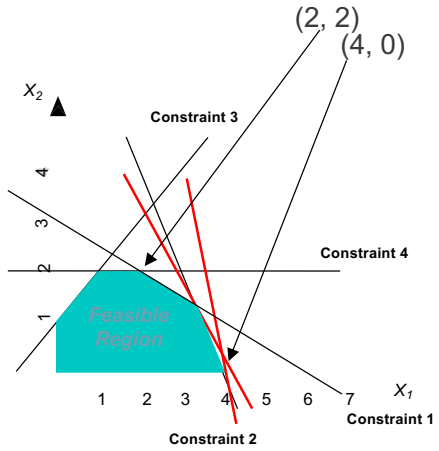
Overbooking

Recap: Range over which decisions remain optimal

- Range of parameters (e.g., prices, profit contributions, etc.) within which the optimal solution remains the same.

```
# Changes in Objective Function Coefficients
prod.sol$sens.coef.from
## [1] 1.0 1.5
prod.sol$sens.coef.to
## [1] 4 6
```

- When the profit margin of Sassy Spicy is within: [1.5,6], the optimal solution remains the same.
- If profit margin > 6, the optimal solution moves to: (2,2)
- If profit margin < 1.5, the optimal solution moves to: (4,0)



Recap: Shadow (Bid) Prices (Dual Values)

Shadow price: change in the objective function for a unit increase of the right-hand side of the constraint (e.g., the amount of resource available) while everything else remains the same.

Only **scarce resources are valuable**.

- If the *constraint is not binding, the shadow price is zero.*
- The shadow price is non-zero *only when a constraint is binding.*

Obtaining Shadow Prices in R:

```
# Shadow (Dual) Value of Constraints
prod.sol$duals[1:length(constr.dir)]
## [1] 0.3333333 1.3333333 0.0000000 0.0000000
```

Ex. The shadow price of the pepper availability constraint is 0.333333

Profit: 12.667 with 6 kgs of peppers

Profit: 13.0 with 7 kgs of peppers

Profit: 12.333 with 5 kgs of peppers

The shadow price of the maximum SS demand constraint is 0.

Profit: 12.667 with max demand of 2 kgs.

Profit: 12.667 with max demand of 3 kgs.

Recap: Change in the constraint quantity

- Once we solve an LP, the shadow prices we obtain for the constraints are only valid for certain ranges of the right-hand-side availability values.
- Range over which Shadow Price remains valid

```
# The range over which the shadow price for constraints remain valid.
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
```

Note: lpSolve only computes these ranges for nonzero shadow prices and displays -1e+30 (-infinity) and 1e+30 (infinity) as ranges if shadow price is zero. 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.

- For example, for the peppers constraint, if the availability of peppers is within 4 and 7 kgs,
  - the profit increases by 0.333 per unit increases of pepper availability.
- What happens if the change is beyond the range?
  - Need to re-solve the problem to find specifics.

Recap: Airline Network DP and LP based Heuristics

- An airline operates two flights, one from Dublin to London, and the other from London to Edinburgh.
- The capacity of the first flight (D-L) is 100 seats and the capacity for the second flight (L-E) is 120 seats.
- The Origin Destination Fares (ODF) and the anticipated demands are as follows:

	ODF	Fare	Per Period Arrival Probability
1	Dublin to London Full Fare	150	1/5
2	London to Edinburgh Full Fare	120	4/15
3	Dublin to Edinburgh Full Fare	250	1/6
4	Dublin to Edinburgh Discount Fare	180	4/15

- (The probability of no arrivals can be found as 1/10)

Recap: Network RM with Bid Price Heuristics

Part 1:

- Write down the Dynamic Programming formulation.
- Implement the Dynamic Programming algorithm to find the optimal value function starting at the given initial capacity.
- $V(x_1, x_2, t) = \lambda_0 * V(x_1, x_2, t-1)$ ($\lambda_0 = 0.1$ (from $1-(1/5 + \ldots + 4/15)$))
  $+ 1/5 * \max \{ 150 + V(x_1-1, x_2, t-1) , V(x_1, x_2, t-1) \}$
  $+ 4/15 * \max \{ 120 + V(x_1, x_2-1, t-1) , V_{t-1}(x_1, x_2, t-1) \}$
  $+ 1/6 * \max \{ 250 + V(x_1-1, x_2-1, t-1) , V(x_1, x_2, t-1) \}$
  $+ 4/15 * \max \{ 180 + V(x_1-1, x_2-1, t-1) , V(x_1, x_2, t-1) \}$
- Solving the DP with backward induction and defining the terminal values as
  - $V(x_1, x_2, 0) = 0$ for all x_1, x_2, we find:
- Optimal revenue at the beginning of the period is:
  - $V(100,120,300) = 28375.28$. (Please see R Supplement)

Network RM with Bid Price Heuristics

Part 2:

- From the arrival probabilities information in the table, find the mean demand for each product.
- Construct a Linear Program using the mean demand and the initial capacity. What is the optimal revenue?
- Obtain the bid prices for the constraints.
- From the arrival probabilities information in the table, find the mean demand for each product.

  $E[D_1] = 1/5 * 300 = 60$, $E[D_2] = 4/15 * 300 = 80$, etc.
- Let $y_1, y_2, \ldots, y_4$ be the decision variables (allocation)

  max $150\, y_1 + 120\, y_2 + 250\, y_3 + 180\, y_4$

  subject to $y_1 + y_3 + y_4 \le 100$ (available seats in leg 1)

  $y_2 + y_3 + y_4 \le 120$ (available seats in leg 2)

  $y_1 \le 60, y_2 \le 80, y_3 \le 50, y_4 \le 80$ (allocation $\le$ demand)

  $y_1 \ge 0, \ldots, y_4 \ge 0$ (nonnegative allocation)
- The optimal revenue from LP is 28600. Please see R Supplement. Note that this is an upper bound of what we can achieve as we assumed we knew the exact demand. Nevertheless, it is good to observe that the optimal DP revenue was very close to this upper bound.

Network RM with Bid Price Heuristics

```
# 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;
```

Solving the Corresponding LP based on Expected Demands

```
# 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
```

Obtaining LP Bid Prices

```
# 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.
```

Network RM with Bid Price Heuristics

Part 3:

- Run the DP recursions assuming an allocation decision based on LP bid prices rather than the optimal allocation. What is the expected revenue of the LP bid price heuristics?

Implementing the Bid Price Heuristics

```
# 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;
            }
```

Implementing the Bid Price Heuristics

```
            # 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;
                }
            }
```

Implementing the Bid Price Heuristics

```
            # 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
```

Network RM with Bid Price Heuristics

Part 3:

- Run the DP recursions assuming an allocation decision based on LP bid prices rather than the optimal allocation. What is the expected revenue of the LP bid price heuristics?
- The revenue from the bid-price heuristic is 28150.06
- Please see the R Supplement.

Part 4:

- Run the DP recursions assuming an allocation decision based on a first come first serve policy rather than the optimal allocation. What is the expected revenue of the LP bid price heuristics?
- The revenue from first come first serve is 25112.55

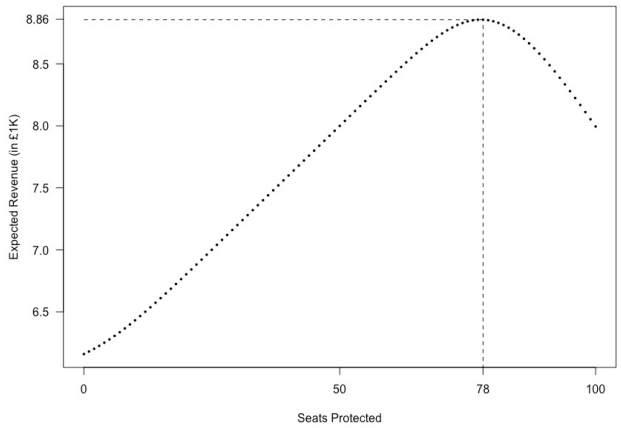
Compared to the potential gain from the optimal policy vs FCFS: 28375.28-25112.55=3,262.73, the bid price heuristic provided a gain over FCFS of 28150.06-25112.55=3,037.51. Hence the heuristic captured 93% (3037.51/3262.73) of the potential additional revenue.

Buy-up Behaviour in Capacity Allocation

- Recall the example from Week 1:
  - Capacity=100
  - $p_1 = 100$
  - $p_2 = 60$
  - D_1 is a Poisson random variable with mean 80.
  - D_2 is a Poisson random variable with mean 100.

Buy-up Behaviour in Capacity Allocation

- Optimal protection level for high fare is 78.



Buy-up Behaviour in Capacity Allocation

Suppose now that the demand for low-fare tickets includes a sub-segment of customers: those who prefer a low-fare ticket but are willing to buy up to a full-fare ticket if a low-fare ticket is not available (think of a price sensitive small business owner who must fly to a meeting).

- Whenever we stop selling low-fare tickets, some of the remaining low-fare passengers buy-up to full-fare.
- Specifically, assume that 10% of customers will be interested in buying a full-fare ticket if no low-fare ticket is available.

Identify the new optimal booking limit that maximizes the expected revenue in the presence of customers that are willing to buy-up.

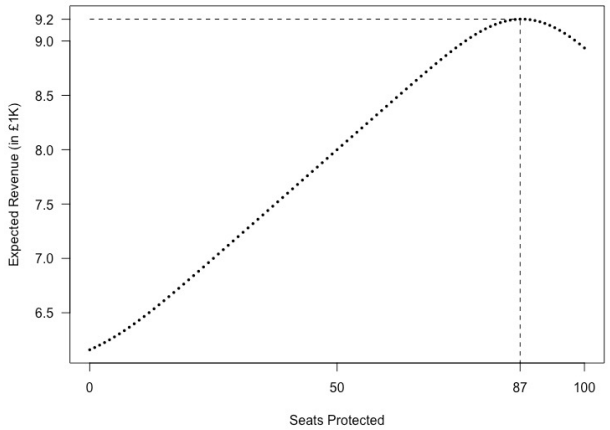
Buy-up Behaviour in Capacity Allocation

```
# Buy up Behaviour in Capacity Allocation

mL=100          # Mean Demand for Low-Fare, Poisson
mH=80           # Mean Demand for High-Fare, Poisson
pL=60           # Price for Low-Fare
pH=100          # Price for Low-Fare
capacity=100    # Capacity
qUp=0.1;          # Fraction Low Fare Buy Up
ExpRevenue=rep(0,capacity+1)
for (i in 1:(capacity+1)){
  protect=i-1
  availforLowFare=capacity-protect;
  ExpRevenue[i]=0;
  for(dL in 0:150){
    soldLowFare=min(availforLowFare,dL)
    unmetLowFare=dL-soldLowFare
    remainforHighFare=capacity-soldLowFare
    for(dH in 0:150){
      soldHighFare=min(remainforHighFare,dH+qUp*unmetLowFare)
      RevenueThisIter=pL*soldLowFare+pH*soldHighFare
      ExpRevenue[i]=ExpRevenue[i]+
        RevenueThisIter*dpois(dL,mL)*dpois(dH,mH)
    }
  }
}
Protectindexbest = which(ExpRevenue == max(ExpRevenue))
ProtectBest=Protectindexbest-1
OptimalExpRevenue=max(ExpRevenue)
print(paste("The Optimal Protection Level for High-Fare Demand:", ProtectBest))
```

```
## [1] "The Optimal Protection Level for High-Fare Demand: 87"
```

Buy-up Behaviour in Capacity Allocation

- Optimal protection level for high fare is now 87 (compared to 78).
- The Buy-up behavior has led the firm to protect more seats for the high fare class.



Buy-down Behaviour in Capacity Allocation

- Suppose now that the distinction between the two products is less clear (e.g., a restriction for the low fare is eliminated).
- As a result, a large proportion (40%) of the high fare business customers who were willing to buy the high-fare ticket would prefer to buy the low-fare ticket when it is available, i.e., buy down.
- Suppose the firm is unaware of this buy down behaviour.
  - The firm sets its protection level as usual.
  - It observes the high fare sales and misinterprets this as the new expected demand
  - It reconsiders and resets its protection level based on the updated demand.
  - It observes the high fare sales.
  - It reconsiders and resets its protection level based on the updated demand.
  - …

Buy-down Behaviour in Capacity Allocation

- $p_1 = 100$, $p_2 = 60$, Capacity=175 (higher capacity to help visualize)
- D_1 is Poisson with mean 80, D_2 is Poisson with mean 100.
- Ignoring the buy down behaviour, the protection level is 78 seats.

Buy-down Behaviour in Capacity Allocation

```
# Suppose the firm continues uses the original protection level of 78
# but there are customers in the market who buy down

mL=100          # Mean Demand for Low-Fare, Poisson
mH=80           # Mean Demand for High-Fare, Poisson
pL=60           # Price for Low-Fare
pH=100          # Price for Low-Fare
capacity=175    # Capacity
qDown=0.4;      # Fraction Low Fare Buy Up
ExpRevenue=rep(0,capacity+1)
ExpSoldHighFare=rep(0,capacity+1)
for (i in 79:79){ # incorporating protection level ignoring buy-down behaviour
    protect=i-1
    availforLowFare=capacity-protect;
    ExpRevenue[i]=0;
    ExpSoldHighFare[i]=0;
    for(dL in 0:150){
        for(dH in 0:150){
            lowBoughtLowFare=min(availforLowFare,dL)
            remainingCapatLowFare=availforLowFare-lowBoughtLowFare
            highBoughtLowFare=min(remainingCapatLowFare,qDown*dH)
            soldLowFare=lowBoughtLowFare+highBoughtLowFare

            remainingHighFareDemand=dH-highBoughtLowFare
            remainingCapforHighFare=capacity-soldLowFare
            soldHighFare=min(remainingHighFareDemand,remainingCapforHighFare)

            RevenueThisIter=pL*soldLowFare+pH*soldHighFare
            ExpRevenue[i]=ExpRevenue[i]+
                RevenueThisIter*dpois(dL,mL)*dpois(dH,mH)
            ExpSoldHighFare[i]=ExpSoldHighFare[i]+
                soldHighFare*dpois(dL,mL)*dpois(dH,mH)
        }
    }
}
```

Buy-down Behaviour in Capacity Allocation

```
print(paste("Expected Revenue:", ExpRevenue[i]))
```

```
## [1] "Expected Revenue: 13188.3472276875"
```

```
print(paste("Expected Observed High Fare Demand", ExpSoldHighFare[i]))
```

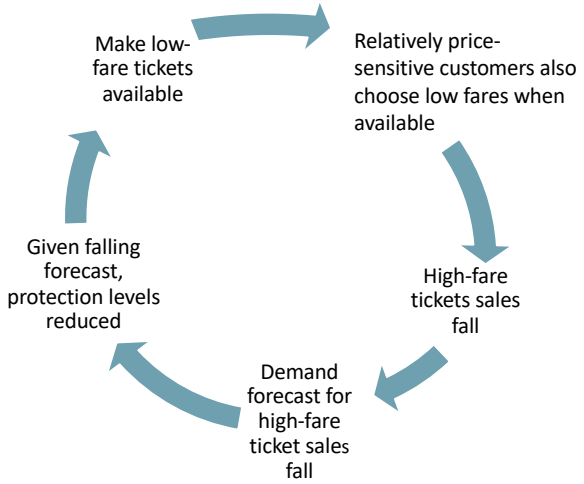
```
## [1] "Expected Observed High Fare Demand 73.6839316608279"
```

Buy-down Behaviour in Capacity Allocation

- If the firm uses a protection level of 78 in the presence of the buy down behaviour:
  - Received Revenue: 13188.
  - Expected High Fare Sales: 74.
- Iteration 1: Suppose the firm
  - updates the distribution of D_2 so that it has a mean of 74.
  - Obtains a new protection level based on the new D_2, which it computes as 72.
  - Using this protection level, its expected revenue is 13039.
  - It computes the expectation of High Fare Sales as 69.
- Iteration 2: Suppose the firm
  - updates the distribution of D_2 so that it has a mean of 69.
  - Obtains a new protection level based on the new D_2, which it computes as 67.
  - Using this protection level, its expected revenue is 12858.
  - It computes the expectation of High Fare Sales as 64.
- …

Buy-down Behaviour in Capacity Allocation

- Spiraling down of Profits:



Buy-down Behaviour in Capacity Allocation

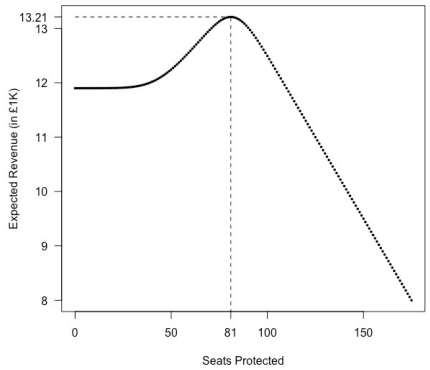
- If the firm took into account the buy-down behavior while determining the booking limit:

```
for (i in 1:(capacity+1)){
    protect=i-1
    availforLowFare=capacity-protect;
    ExpRevenue[i]=0;
    ExpSoldHighFare[i]=0;
    for(dL in 0:150){
        for(dH in 0:150){
            lowBoughtLowFare=min(availforLowFare,dL)
            remainingCapatLowFare=availforLowFare-lowBoughtLowFare
            highBoughtLowFare=min(remainingCapatLowFare,qDown*dH)
            soldLowFare=lowBoughtLowFare+highBoughtLowFare

            remainingHighFareDemand=dH-highBoughtLowFare
            remainingCapforHighFare=capacity-soldLowFare
            soldHighFare=min(remainingHighFareDemand,remainingCapforHighFare)

            RevenueThisIter=pL*soldLowFare+pH*soldHighFare
            ExpRevenue[i]=ExpRevenue[i]+
                RevenueThisIter*dpois(dL,mL)*dpois(dH,mH)
            ExpSoldHighFare[i]=ExpSoldHighFare[i]+
                soldHighFare*dpois(dL,mL)*dpois(dH,mH)
        }
    }
}
Protectindexbest = which(ExpRevenue == max(ExpRevenue))
ProtectBest=Protectindexbest-1
OptimalExpRevenue=max(ExpRevenue)
print(paste("The Optimal Protection Level for High-Fare Demand:", ProtectBest))
```

```
## [1] "The Optimal Protection Level for High-Fare Demand: 81"
```

Buy-down Behaviour in Capacity Allocation

- If the firm took into account the buy-down behavior while determining the booking limit:
- Then, it may pick a higher protection level of 81 compared to the original protection level of 78.
- Expected revenue is then 13211.



Buy-down Behaviour in Capacity Allocation

Takeaways:

- If buy-down behavior is ignored, booking limits settle to levels far above from optimal. This causes a spiraling down of profits. (This is not just an airline industry problem.)
- This can be avoided if the revenue management model takes into account more sophisticated customer behavior. This is a very active research area and such models are increasingly adopted in practice.

Overbooking

In many settings, customers are either allowed to cancel their reservations with short notice or they fail to show up to receive their service.

Overbooking: occurs when a seller with constrained capacity sells more units than he has available to protect themselves against unanticipated no-shows and cancellations.

- Example: When a flight is oversold, i.e., the # of passengers show up exceeds the seats on flight, the airline picks customers to 'bump', i.e., rebook on a later flight through:
  - Involuntary denied boarding
  - Voluntary denied boarding

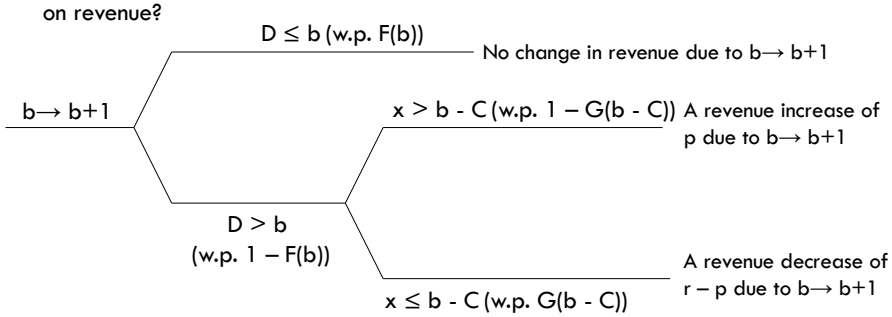
Overbooking

- Overbooking is applicable when
  - Capacity is constrained and perishable, and bookings are accepted for future use
  - Customers cancel or do not show up
  - Cost of denying service to a customer with a booking is relatively low
- Overbooking is
  - commonly used by airlines, hotels, rental car companies
  - generally not used by cruise lines, resort hotels, sporting events, theaters

Overbooking – A Simpler Setting

Assume the # of no-shows is independent of the number of total bookings. For capacity C, price p, denied boarding cost r (r>p), total demand D, number of no-shows x, let the probability that the total number of booking requests will be less than or equal to d be given by F(d), and the probability that the number of no-shows will be less than or equal to x be given by G(x).

Suppose we increase the booking limit b>C from b to b +1. What is the effect on revenue?



Overbooking

In summary, if we increase the booking limit from b to b +1, the expected change in revenue is:

$(1 - F(b)) \times (1 - G(b - C) \times p$ → Potential revenue increase from booking one more passenger without the need to bump a passenger

$-(1 - F(b)) \times G(b - C) \times (r - p)$ → Potential revenue decrease from booking one more passenger but bumping another passenger

At the optimal b:

$$(1 - F(b)) \times (1 - G(b - C) \times p = (1 - F(b)) \times G(b - C) \times (r - p)$$

$$\Downarrow$$

$$G(b - C) = p / r$$

The optimal booking limit is the smallest value of b such that $p / r \leq G(b - C)$

Example Application: Hotel Revenue Management

- Consider a hotel:
  - 118 rooms
  - Price, p=159
  - Bumped customer cost, r=509 (e.g., 159+350 compensation fee)
- The optimal booking limit is the smallest value of b such that $p / r \leq G(b - C)$
  - $p / r = 159/509 = 0.3124$

Example Application: Hotel Revenue Management

- If distribution of no-shows, (mean 8.5) is given by:

x	G(x)		x	G(x)
0	0.0002		10	0.7634
1	0.0019		11	0.8487
2	0.0093		12	0.9091
3	0.0301		13	0.9486
4	0.0744		14	0.9726
5	0.1496		15	0.9862
6	0.2562		16	0.9934
7	0.3856		17	0.9970
8	0.5231		18	0.9987
9	0.6530		19	0.9995

- The optimal booking limit is the smallest value of b such that $p / r \leq G(b - C)$
  - $p / r = 159/509 = 0.3124$
  - x = 7. Overbook, 7 customers, hence book 125 customers in total.

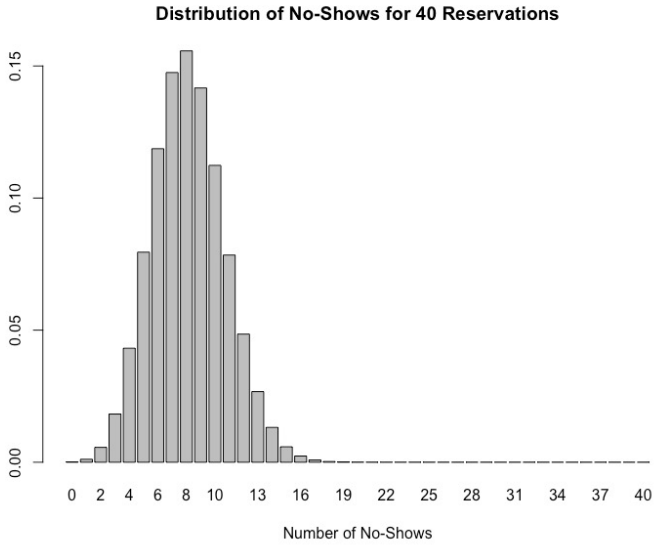
Example Application: Restaurant Revenue Management

- Consider a restaurant (Please see R Supplement):
  - 40 tables
  - Delaying cost, r=£1000 (e.g., loss of reputation, repeat visits by customers, free drinks, appetizers, discount on meal, etc.)
  - From data file "`Reservations.csv`"
    - Average revenue per table = £119.50
    - Show Up Probability = 0.79625 (i.e., no show prob=0.20375) Assume each reservation has a probability of 0.79625 for showing up, independent of whether other reservations show up or not.
    - The total number of shows is Binomial with a success probability of 0.79625 and a total trial size of the number of reservations taken.

Example Application: Restaurant Revenue Management

	ReservationID	Arrive	NoShow	Profit
1	1	1	0	73.22
2	2	1	0	104.19
3	3	1	0	104.32
4	4	1	0	68.00
5	5	1	0	99.34
6	6	1	0	15.16
7	7	0	1	0.00
8	8	0	1	0.00
9	9	1	0	107.29
10	10	1	0	114.23
11	11	1	0	136.56
12	12	1	0	161.82

Example Application: Restaurant Revenue Management



Example Application: Restaurant Revenue Management

```
ExpProfit=rep(0,overbookmax+1)

for (i in 1:(overbookmax+1)){
  reservationsTaken=cap+i-1;
  ExpProfit[i]=0;
  for (j in 0:reservationsTaken){
      arrive=j
      serve=min(arrive,cap)
      delay=arrive-serve
      ProfitThisIter=avgPrice*serve-penalty*delay
      ExpProfit[i]=ExpProfit[i]+
          dbinom(j,reservationsTaken,showProb)*ProfitThisIter
  }
}

overbookindexbest = which(ExpProfit == max(ExpProfit))
overbookBest=overbookindexbest-1
OptimalExpProfit=max(ExpProfit)
print(paste("The Optimal Overbooking Amount:", overbookBest))
```

```
## [1] "The Optimal Overbooking Amount: 6"
```

```
print(paste("The Optimal Number of Reservations to Take:",
            cap+overbookBest))
```

```
## [1] "The Optimal Number of Reservations to Take: 46"
```

Example Application: Restaurant Revenue Management

- Optimal number of tables to overbook: (Please see R Supplement):
  - Overbook 6 tables (i.e., take up to 46 reservations)
  - This brings an expected revenue of £4251 compared to the expected revenue from no overbooking of £3806, i.e., an 11.7% increase.