SMM 641

Revenue Management and Pricing

Network Revenue Management Part 1

Oben Ceryan

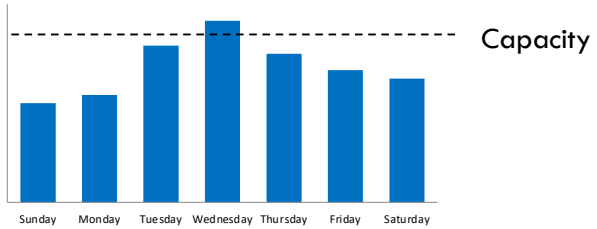
Network Revenue Management

- Network revenue management is relevant when
  - there is a set of constrained and perishable "resources" and
  - each "product" combines multiple resources.
- Examples:

	Resource	Product
Airline	Seat on a flight leg	Multi-leg itinerary
Hotel	Room night	Multi-night stay
Rental Car	Rental day	Multi-day rental
Train	Seat on a leg	Multi-leg trip

The need for sophistication

Consider a hotel that charges one of two rates: A full-rate of $200 per night and a discounted rate of $150 per night. Suppose the demand forecast for next week is as follows:



The manager's claim: Wednesday night is the only constrained resource. When accepting reservations, I will give priority to guests who are paying the full-rate on Wednesday night.

What can go wrong with this logic?

The greedy heuristic

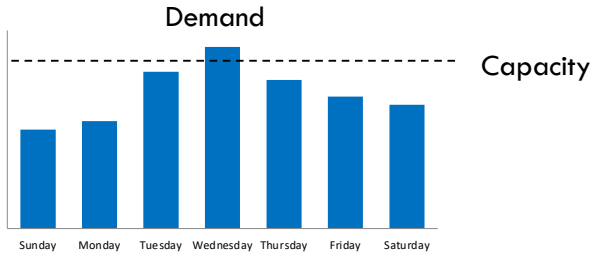
Let us sort in ascending order several classes of customers who might demand a room on Wednesday night:

Class	Rate	# Nights	Revenue Contribution
1	$150	1	$150
2	$200	1	$200
3	$150	2	$300
4	$200	2	$400
5	$150	3	$450
6	$200	3	$600
	$150	4	
7	$150	5	$750
...	...	...	...

Greedy heuristic: Accept Wednesday night reservations starting with the highest class and moving to the next class until we exhaust capacity.

When the greedy heuristic appropriate

If the demand pattern is as shown below, then the greedy heuristic is perfect:

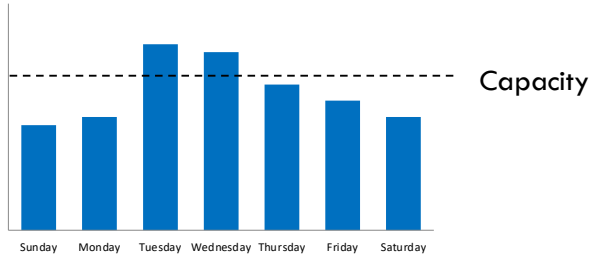


In this case, the booking requests that will be turned down are a subset of requests that involve a Wednesday night stay, and we are making sure that we pick the ones that yield the highest revenue contribution.

What is special about this example is that only one resource (Wednesday night) is constrained.

Shortcoming of the greedy heuristic

What if the demand is as shown below:



Now both Tuesday and Wednesday nights are constrained.

The recommendations of the greedy heuristic for Tuesday night may conflict with the recommendations of the greedy heuristic for Wednesday night.

Shortcoming of the greedy heuristic

- Even when the recommendations agree, they may lead to a suboptimal decision. Consider, for example, three guests: A, B and C.
  - A: will pay $150 per night for a two-night stay on Tuesday and Wednesday
  - B: will pay $200 per night for a one-night stay on Tuesday
  - C :will pay $200 per night for a one-night stay on Wednesday
- Tuesday-night greedy heuristic: favors A over B.
- Wednesday-night greedy heuristic: favors A over C.
- Both heuristics agree to rent the room to A. However, it would be better to admit B and C (for a total revenue of $400) instead of A (for a revenue of $300).

Network Revenue Management via Dynamic Programming

- *m* resources, *n* products
- Resource *i* has initial capacity c_i
- Product *j* has price p_j and a resource requirement of u_j, where $u_{ij} = 1$ if product *j* uses resources *i*, 0 otherwise.
- At most one request arrives in each period.
  - No request arrives with probability λ_0
  - Request for product *j* arrives with probability λ_j
  - $\lambda_0 + \lambda_1 + \ldots + \lambda_n = 1$

DP Formulation:

- In period t, state of the system is $x=(x_1,x_2, \ldots, x_m)$
- Action: $a_j = \{0,1\}$ whether to sell product *j* or not
- $V_t(x)=\lambda_0 * V_{t+1}(x) + \sum_{j=1:n} \lambda_j * \max_{a_j} E [p_j + V_{t+1} (x - a_j * u_j) , V_{t+1} (x)]$

Network Revenue Management via Dynamic Programming

Optimal Policy:

- If there is an arrival for product *j*:
- Sell product *j* if $p_j + V_{t+1}(x - a_j * u_j) > V_{t+1}(x)$ and if resources are sufficient, i.e., $x \geq a_j * u_j$
- Do Not sell product *j* otherwise.

Intuition:

- Sell a product only if its price exceeds the opportunity cost of losing future sales due to reduction in resource capacities.

Example: Network RM with DP

- An airline operates a flight network among 3 locations A, B and C, across itineraries (A, B), (B, C) and also (A, C); the latter requires reserving one seat on each leg.
- Assume that we can have at most one itinerary request at each time period. A part of the value function for time period t + 1 has been computed and given in the following table.

Remaining Capacity for B-C→				
Remaining Capacity for A-B ↓	0	1	2	3
0	0	600	900	1000
1	300	700	1000	1300
2	400	800	1100	1300
3	500	850	1100	1300

- The value function is given in the above table. For example, if the remaining capacity on flight leg (A-B) is 2 and the remaining capacity on flight leg (B-C) is 1, then the value of the value function at time period t + 1 is $V_{t+1}([2, 1]) = 800$.

* Source: Sid Banerjee (Cornell)

Example: Network RM with DP

Remaining Capacity for B-C→				
Remaining Capacity for A-B ↓	0	1	2	3
0	0	600	900	1000
1	300	700	1000	1300
2	400	800	1100	1300
3	500	850	1100	1300

- Assume that the remaining capacity on flight leg A-B is 1 and the remaining capacity on flight leg B-C is 2 at time period t. Suppose we get a request for a A – B flight for $150. Is it optimal to accept a this request? Why?

Example: Network RM with DP

Remaining Capacity for B-C→				
Remaining Capacity for A-B ↓	0	1	2	3
0	0	600	900	1000
1	300	700	1000	1300
2	400	800	1100	1300
3	500	850	1100	1300

- Assume that the remaining capacity on flight leg A-B is 2 and the remaining capacity on flight leg B-C is 3 at time period t. Is it optimal to accept a request for a A – C itinerary for $275 at time period t? Why?

Ex: Airline Network DP

- 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)

Network RM with DP

- Write down the Dynamic Programming (DP) 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 DP

```
# 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);

# R requires arrays be created at the beginning.
# Creating empty arrays of correct dimensions.
# For the value function v(x1,x2,t):
v=array(rep( 0, len=(N1+1)*(N2+1)*(TT+1)), dim=c(N1+1,N2+1,TT+1));
# To keep track of optimal decisions,
# e.g. acceptance decision for a product 1 arrival: accept1(x1,x2,t):
accept1=array(rep( 0, len=(N1+1)*(N2+1)*(TT+1)), dim=c(N1+1,N2+1,TT+1));
# remaining are similarly defined and created.
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));

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

# Terminal Values
for(i in 1:(N1+1)){
    for(j in 1:(N2+1)){
        v[i,j,1]=0; # All seats worthless at the end of horizon, i.e., t=1.
    }
}
```

Network RM with DP

```
# Dynamic Programming Algorithm

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/profitable to accept
            vforarrival1=v[i,j,t-1];
            accept1[i,j,t]=0;
            # If resource available:
            if(i>1){
                vforarrival1=max(price[1]+v[i-1,j,t-1],v[i,j,t-1]);
                # Recording the decision in the accept1 variable:
                if(price[1]+v[i-1,j,t-1]>v[i,j,t-1]){
                    accept1[i,j,t]=1;
                }
            }
```

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

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

Network RM with DP

```
            # For Product 4 arrival:
            # default not accept unless able/profitable to accept
            vforarrival4=v[i,j,t-1];
            accept4[i,j,t]=0;
            # If resources available:
            if(i>1){
                if(j>1){
                    vforarrival4=max(price[4]+v[i-1,j-1,t-1],v[i,j,t-1]);
                    # Recording the decision in the accept4 variable:
                    if(price[4]+v[i-1,j-1,t-1]>v[i,j,t-1]){
                        accept4[i,j,t]=1;
                    }
                }
            }

            # 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;
        }
    }
}
```

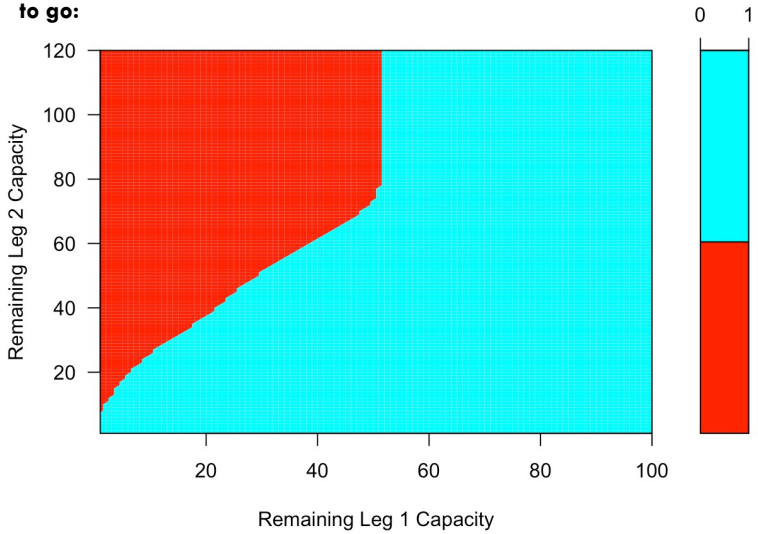
Network RM with DP

```
# Optimal Revenue Starting at
# Leg 1 seat availability: N1=100 and
# Leg 2 seat availability: N2=120
# with T=300 periods to go:
revenueDP<-v[101,121,301]
print(revenueDP)
```

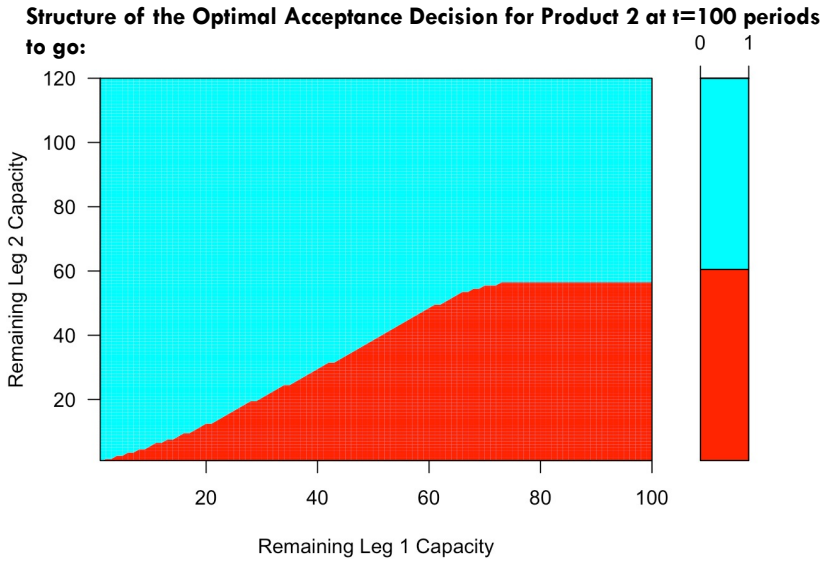
```
## [1] 28375.28
```

Recap: Network RM DP Policy Structure

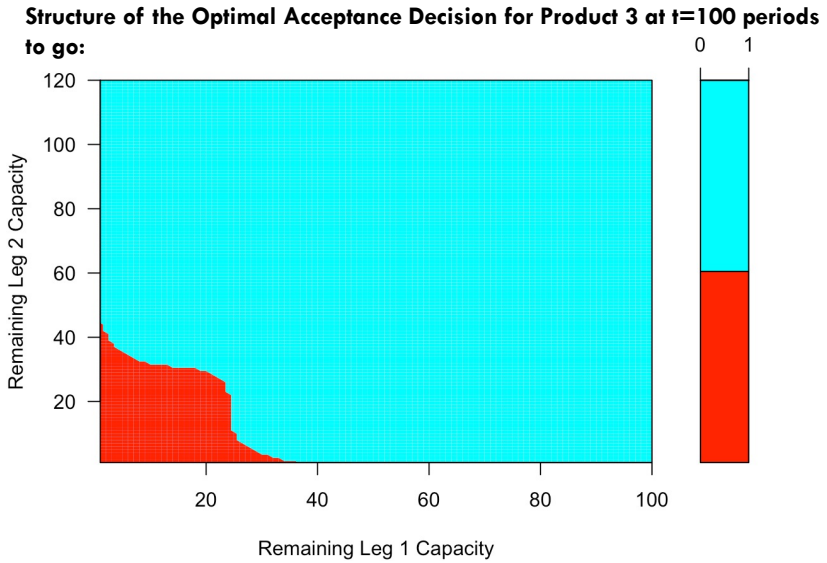
Structure of the Optimal Acceptance Decision for Product 1 at t=100 periods to go:



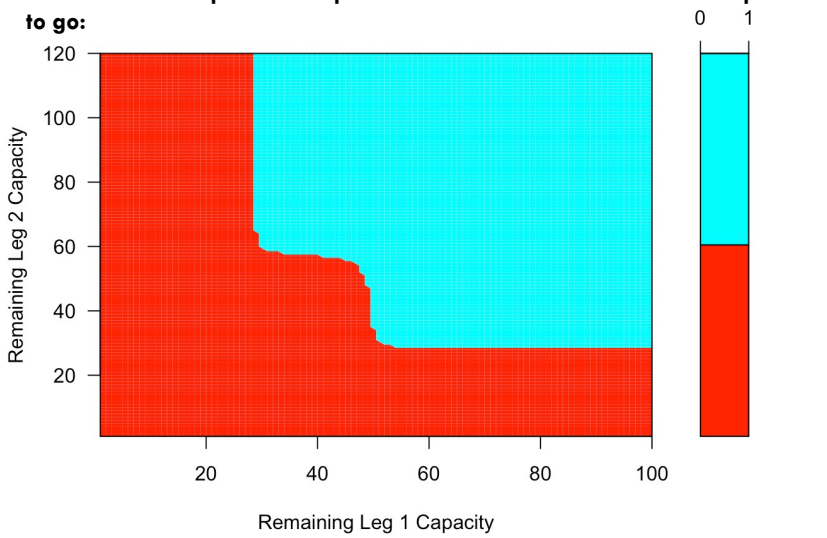
Recap: Network RM DP Policy Structure



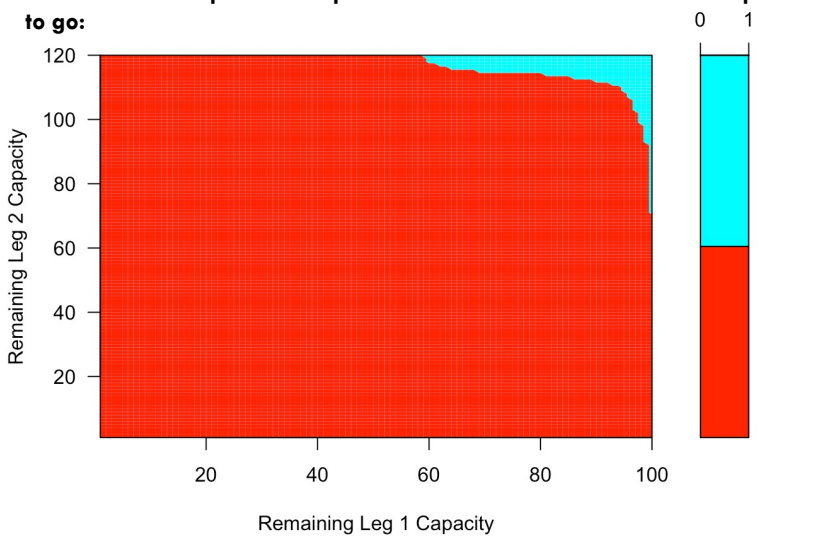
Recap: Network RM DP Policy Structure



Network RM DP Policy Structure



Recap: Network RM DP Policy Structure



Network RM with FCFS Allocation

First Come First Serve (FCFS) Allocation:

- Run the DP recursions assuming a first come first serve allocation decision rather than the optimal allocation. What is the expected revenue of the first come first serve strategy?
- (Please see the R Supplement.)

Network RM with FCFS Allocation

```
# Dynamic Programming Recursion

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;
            }
```

Network RM with FCFS Allocation

```
            # 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:
            # default not accept unless able to accept
            vforarrival4=v[i,j,t-1];
            accept4[i,j,t]=0;
            # If resources available:
            if(i>1){
                if(j>1){
                    vforarrival4=price[4]+v[i-1,j-1,t-1];
                    accept4[i,j,t]=1;
                }
            }

            # 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;
```

Network RM

First Come First Serve (FCFS) Allocation:

- Run the DP recursions assuming a first come first serve allocation decision rather than the optimal allocation. What is the expected revenue of the first come first serve strategy?
- (Please see the R Supplement.)
- The revenue from first come first serve is 25112.55.