

**e - c o m p a n i o n**

ONLY AVAILABLE IN ELECTRONIC FORM

Electronic Companion—“Simulation-Based Optimization of Virtual Nesting Controls for Network Revenue Management” by Garrett van Ryzin and Gustavo Vulcano, *Operations Research*, DOI 10.1287/opre.1080.0550.

---

# Simulation-Based Optimization of Virtual Nesting Controls for Network Revenue Management

## ONLINE APPENDIX

Garrett van Ryzin<sup>1</sup>

Gustavo Vulcano<sup>2</sup>

### A1 Derivatives of the revenue function

From equation (5), the deduction of the derivative of the revenue function with respect to  $y_{ic}$  proceeds as follows:

$$\begin{aligned} \frac{\partial}{\partial y_{ic}} R_t(x(t), y, \omega) &= r_{j_t} \frac{\partial}{\partial y_{ic}} u_{j_t}(x(t) - \zeta_t, y, q_t) + \sum_{k=1}^m \frac{\partial}{\partial x_k(t+1)} R_{t+1}(x(t+1), y, \omega) \frac{\partial}{\partial y_{ic}} x_k(t+1) \\ &\quad + \frac{\partial}{\partial y_{ic}} R_{t+1}(x(t+1), y, \omega) \end{aligned}$$

Take the column  $A_{j_t}$  in equation (6). It has a 1 in position  $k$  if and only if  $k \in A_{j_t}$ . Then,

$$\begin{aligned} \frac{\partial}{\partial y_{ic}} R_t(x(t), y, \omega) &= r_{j_t} \frac{\partial}{\partial y_{ic}} u_{j_t}(x(t) - \zeta_t, y, q_t) - \sum_{k \in A_{j_t}} \frac{\partial}{\partial x_k(t+1)} R_{t+1}(x(t+1), y, \omega) \frac{\partial}{\partial y_{ic}} u_{j_t}(x(t) - \zeta_t, y, q_t) \\ &\quad + \frac{\partial}{\partial y_{ic}} R_{t+1}(x(t+1), y, \omega) \\ &= \left( r_{j_t} - \sum_{k \in A_{j_t}} \frac{\partial}{\partial x_k(t+1)} R_{t+1}(x(t+1), y, \omega) \right) \frac{\partial}{\partial y_{ic}} u_{j_t}(x(t) - \zeta_t, y, q_t) \\ &\quad + \frac{\partial}{\partial y_{ic}} R_{t+1}(x(t+1), y, \omega). \end{aligned}$$

Now we have to solve for the derivative with respect to capacity, involved in the second term. Taking equation (5), we have that:

$$\begin{aligned} \frac{\partial}{\partial x_i(t)} R_t(x(t), y, \omega) &= r_{j_t} \frac{\partial}{\partial x_i(t)} u_{j_t}(x(t) - \zeta_t, y, q_t) \\ &\quad + \sum_{k=1}^m \frac{\partial}{\partial x_k(t+1)} R_{t+1}(x(t+1), y, \omega) \frac{\partial}{\partial x_i(t+1)} x(t+1) \end{aligned}$$

Now, consider again equation (6). According to the resources used by product  $j_t$ , we can rewrite the equation above as:

$$\frac{\partial}{\partial x_i(t)} R_t(x(t), y, \omega) = r_{j_t} \frac{\partial}{\partial x_i(t)} u_{j_t}(x(t) - \zeta_t, y, q_t)$$

---

<sup>1</sup>Graduate School of Business, Columbia University, 412 Uris Hall, New York, NY 10027, gjv1@columbia.edu.

<sup>2</sup>Stern School of Business, New York University, 44 West Fourth St., Suite 8-76, New York, NY 10012, gvulcano@stern.nyu.edu.

$$\begin{aligned}
& - \sum_{\substack{k \in A_{j_t} \\ k \neq i}} \frac{\partial}{\partial x_k(t+1)} R_{t+1}(x(t+1), y, \omega) \frac{\partial}{\partial x_i(t)} u_{j_t}(x(t) - \zeta_t, y, q_t) \\
& + \frac{\partial}{\partial x_i(t+1)} R_{t+1}(x(t+1), y, \omega) \left( 1 - \frac{\partial}{\partial x_i(t)} u_{j_t}(x(t) - \zeta_t, y, q_t) \mathbf{I}\{i \in A_{j_t}\} \right)
\end{aligned}$$

Finally, regrouping terms, we have:

$$\begin{aligned}
\frac{\partial}{\partial x_i(t)} R_t(x(t), y, \omega) & = \left( r_{j_t} - \sum_{k \in A_{j_t}} \frac{\partial}{\partial x_k(t+1)} R_{t+1}(x(t+1), y, \omega) \right) \frac{\partial}{\partial x_i(t)} u_{j_t}(x(t) - \zeta_t, y, q_t) \\
& + \frac{\partial}{\partial x_i(t+1)} R_{t+1}(x(t+1), y, \omega)
\end{aligned}$$

## A2 Displacement adjusted virtual nesting (DAVN)

We implemented *displacement adjusted virtual nesting* (DAVN) to obtain the indexing and the initial protection levels  $y$  in our numerical experiments. The method first uses a *deterministic linear program* (DLP) to compute a set of static bid prices  $\bar{\pi} = (\bar{\pi}_1, \dots, \bar{\pi}_m)$  as follows: Define  $D_j$  to be the aggregated demand for product  $j$  over the whole time horizon (i.e.  $D_j = \sum_{t=1}^T \mathbf{I}\{J_t = j\}$ ). The DLP is then

$$\max r^\top z \quad \text{s.t.} \quad Az \leq x, 0 \leq z \leq E[D]$$

Then, for all products  $j$  that use  $i$ , a *displacement adjusted revenue*  $\bar{r}_{ij}$  is computed using

$$\bar{r}_{ij} = r_j - \sum_{k \in A_j, k \neq i} \bar{\pi}_k$$

Roughly,  $\bar{r}_{ij}$  is an estimate of the net benefit of accepting product  $j$  over leg  $i$ , in which the revenue of  $j$  is reduced by the static bid price values of the other resources used by  $j$ . Note that  $\bar{r}_{ij}$  can be negative.

We then proceed to *cluster* or *index* the different products on a resource  $i$  into a fixed number of *virtual classes* or *buckets*. A particular virtual class is denoted  $v \in \{1, \dots, \bar{c} + 1\}$ . A variety of techniques can be used in this step. Ours is a rather simple clustering procedure that consists of evenly partitioning the range of displacement adjusted revenue values on each leg  $i$  – one for each virtual class – and assigning product  $j$  on leg  $i$  to virtual class  $c_i(j)$  if it falls in the corresponding interval. More precisely, define:

$$\bar{r}_i^{\min} = \min \{ \bar{r}_{ij} : j \in A^i \} \quad \text{and} \quad \bar{r}_i^{\max} = \max \{ \bar{r}_{ij} : j \in A^i \} + \epsilon, \quad \epsilon > 0$$

and the “width” of a virtual class on resource  $i$  to be  $w_i = (\bar{r}_i^{\max} - \bar{r}_i^{\min}) / (\bar{c} + 1)$ . The indexing is then described by the mapping:

$$c_i(j) = v \quad \text{if} \quad \bar{r}_{ij} \in [\bar{r}_i^{\min} + (v-1)w_i, \bar{r}_i^{\min} + vw_i)$$

Let  $c_i^{-1}(v) = \{j \in A^i : c_i(j) = v\}$  be the set of products on resource  $i$  that map to virtual class  $v$ . For all virtual classes  $v$  such that  $c_i^{-1}(v) \neq \emptyset$ , a representative revenue is computed by taking the revenue given by the mean demand weighted average:

$$R_i(v) = \left( \frac{\sum_{j \in c_i^{-1}(v)} \bar{r}_{ij} \mathbb{E}[D_j]}{\sum_{j \in c_i^{-1}(v)} \mathbb{E}[D_j]} \right)^+$$

By taking the positive part, we are guaranteed not to get a negative representative revenue for virtual class  $v$  over leg  $i$ , which could happen due to the  $\bar{r}_{ij}$ 's.

Define  $\bar{c}_i = |\{v : c_i^{-1}(v) \neq \emptyset\}| - 1$  (i.e.  $\bar{c}_i$  is the significant number of virtual classes over leg  $i$ , minus one). Virtual classes are then relabeled consecutively so that  $R_i(1) \geq R_i(2) \geq \dots \geq R_i(\bar{c}_i + 1)$ . Then, the distribution of total demand for a virtual class is computed by adding the means and variances of the demand of the involved products:

$$\mathbb{E}[D_i(v)] = \sum_{j \in c_i^{-1}(v)} \mathbb{E}[D_j] \quad \text{and} \quad \text{Var}[D_i(v)] = \sum_{j \in c_i^{-1}(v)} \text{Var}[D_j]$$

Next, we solve a multiclass, single-resource problem based on this data, using the EMSR-b heuristic of Belobaba [3]. This procedure yields a set of protection levels for the virtual classes at each resource  $i$ . The heuristic is based on the assumption that demand arrives from lowest to highest revenue in the order of virtual classes (i.e. demand for virtual class  $\bar{c}_i + 1$  arrives first, followed by demand for virtual class  $\bar{c}_i$ , and so on, until finally demand for virtual class 1 is realized). In particular, in stage  $k + 1$  in which demand for virtual class  $k + 1$  arrives and we want to set protection level  $y_{ik}$  for classes  $k$  and higher (in terms of revenue), DAVN computes the future aggregated demand to come

$$S_{ik} = \sum_{v=1}^k D_i(v)$$

Let the weighted average revenue from classes  $1, \dots, k$  over leg  $i$ , denoted  $\bar{R}_{ik}$ , be defined by

$$\bar{R}_{ik} := \frac{\sum_{v=1}^k R_i(v) \mathbb{E}[D_i(v)]}{\sum_{v=1}^k \mathbb{E}[D_i(v)]}$$

The EMSR-b protection level over leg  $i$  for virtual classes  $k$  and higher,  $y_{ik}$ , is chosen so that

$$P(S_{ik} > y_{ik}) = \frac{R_i(k+1)}{\bar{R}_{ik}}$$

In particular, if demand for each virtual class  $v$  is independent and normally distributed, then

$$y_{ik} = \mu_{S_{ik}} + z_\alpha \sigma_{S_{ik}}$$

where  $\mu_{S_{ik}} = \sum_{v=1}^k \mathbb{E}[D_i(v)]$ ,  $\sigma_{S_{ik}}^2 = \sum_{v=1}^k \text{Var}[D_i(v)]$  and  $z_\alpha = \Phi^{-1}(1 - R_i(k+1)/\bar{R}_{ik})$ . In case  $y_{ic} > x_i$ , we set  $y_{ic} = x_i$ . Finally, we round the vector  $y$  to end up with integer protection levels.

### A3 Pseudo-code for sample path gradient calculation

```
procedure getgrads(x,j,q,r,T,y);
/* input definitions
x[i]= remaining capacity on resource i (modified by program)
j[t]= product requested by customer t on sample path
q[t]= quantity requested by customer t on sample path
r[j]= revenue generated per unit sold of product j
T = total number of customer requests
y[i,c] = protection level for class c on resource i. We assume y[i,0]=0
/*

/* local variables
min_space[t] = minimum remaining capacity for customer t
u[t] = quantity accepted for customer t
gradX[i] = sample path subgradient with respect to x
gradY[i,c] = sample path subgradient with respect to y
/*

/* functions used */
GextNextRes(j,n); /* returns the n-th resource used by product j;
                  or -1 if less than n resources are used by j /*
Min(a,b);        /* returns minimum of a and b /*

/* initialize gradients to zero */
for (i:=1 to m)
  {gradX[i]:=0.0;
   for (c:=1 to cbar)
     gradY[i,c]:=0.0;
  }

/* forward pass */
for (t:=1 to T)
  {MIN:=INFY;
   j:=j[t];
   q:=q[t];
   n:=1;      /* compute minimum remaining capacity for j */
   while (i:=GetNextRes(j,n) NEQ -1)
     {if (x[i]-y[i,c[i,j]-1] < MIN)
        {if (x[i]-y[i,c[i,j]-1] >= 0)
            MIN:=x[i]-y[i,c[i,j]-1]; /* save the new minimum remaining capacity for j */
          else
            MIN:=0; /* protection level for resource i is beyond the remaining capacity */
        }
     }
  }
```

```

    n:=n+1;
  }
  min_space[t]:=MIN; /* save minimum remaining capacity */
  u[t]:= min(q,MIN); /* accepted amount is minimum of q and MIN */
  n:=1;
  while (i:=GetNextRes(j,n) NEQ -1) /* update remaining capacity */
    {x[i]:=x[i]-u[t];
     n:=n+1;
    }
}

/* backward pass */
for (t:=T down to 1)
  {j:=j[t];
   q:=q[t];
   MIN:=min_space[t]; /* retrieve minimum remaining capacity for j */
   SUM:=0.0;
   n:=1; /* go through resources used by product j */
   while (i:=GetNextRes(j,n) NEQ -1)
     {x[i]:=x[i]+u[t]; /* reconstruct capacity used by t on resource i */
      SUM:=SUM + gradX[i]; /* add up partial with respect to in x on path of j */
      n:=n+1;
     }
   if ( 0 < MIN <= q)
     {n:=1;
      while (i:=GetNextRes(j,n) NEQ -1)
        /* gradients affected only if resource i is binding */
        {if (x[i]-y[i,c[i,j]-1] == MIN)
          {gradX[i]:=gradX[i] + r[j] - SUM; /* update the gradX for next iteration */
           for (c:=1 to c[i,j]-1) /* search for protection levels for classes higher than c[i,j] */
             {if (x[i]-y[i,c] == MIN) /* update gradY if y[i,c] is binding */
               gradY[i,c]:=gradY[i,c] - r[j] + SUM;
             }
           }
          n:=n+1;
        }
     }
}
}

```