Implementation of the Lagrangian Relaxation Algorithm for Network Revenue Management

Hongzhang “Steve” Shao

May 18, 2025

1 The NRM Problem

Consider a **network revenue management problem** with:

- A set of resources (flight legs) $\mathcal{L}$, each with capacity c_i for $i \in \mathcal{L}$.
- A set of products (itineraries) $\mathcal{J}$, each with revenue f_j for $j \in \mathcal{J}$.
  - Each purchase of product j consumes a_{ij} units of capacity from resource i for each i.
- Discrete time horizon $\mathcal{T} = \{1, \ldots, \tau\}$.

In each period t:

- **At most one customer arrives**
- The customer requests product j with probability p_{jt}
- $\sum_{j \in \mathcal{J}} p_{jt} \leq 1$

Note that, by adding a dummy itinerary ψ with

$$\begin{aligned} f_\psi &= 0 \\ a_{i\psi} &= 0 && \forall i \in \mathcal{L} \\ p_{\psi t} &= 1 - \sum_{j \in \mathcal{J}} p_{jt} && \forall t \in \mathcal{T} \end{aligned}$$

It can be assumed that in each period t:

- One customer arrives
- The customer requests product j with probability p_{jt}
- $\sum_{j \in \mathcal{J}} p_{jt} = 1$

Let x_{it} be the remaining capacity of resource i at the start of period t. Let $x_t = \left(x_{1t}, x_{2t}, \ldots, x_{|\mathcal{L}|, t}\right)$ be the state vector. Let

$$C = \max_{i \in \mathcal{L}} c_i$$
$$\mathcal{C} = \{0, 1, \ldots, C\}$$

and let $\mathcal{C}^{|\mathcal{L}|}$ be the state space.

2 The LR Algorithm

Dynamic Programming Formulation:

- Let $u_{jt} \in \{0, 1\}$ indicate whether to accept (1) or reject (0) a request for product j.
- Let $V_t(x_t)$ be the maximum expected revenue from period t to τ given capacities x_t:

$$V_t(x_t) = \max_{u_t \in \mathcal{U}(x_t)} \left\{ \sum_{j \in \mathcal{J}} p_{jt} \left\{ f_j u_{jt} + V_{t+1} \left(x_t - u_{jt} \sum_{i \in \mathcal{L}} a_{ij} e_i \right) \right\} \right\} \tag{DP1}$$

where

$$\mathcal{U}(x_t) = \left\{ u_t \in \{0, 1\}^{|\mathcal{J}|} : a_{ij} u_{jt} \leq x_{it} \quad \forall i \in \mathcal{L},\ j \in \mathcal{J} \right\}$$

and e_i is the unit vector with a 1 in the i-th position and 0 elsewhere.

Equivalent Dynamic Program:

- Let $y_{ijt} \in \{0, 1\}$ indicate whether to accept (1) or reject (0) **resource** i when a **request for product** j arrives (e.g., it is allowed to partially accept some flight legs when an itinerary uses multiple legs).
- Let ϕ be a **fictitious resource** with infinite capacity.
- Let $y_t = \{y_{ijt} : i \in \mathcal{L} \cup \{\phi\},\ j \in \mathcal{J}\}$.
- Then, $V_t(x_t)$ can be computed as:

$$V_t(x_t) = \max_{y_t \in \mathcal{Y}(x_t)} \left\{ \sum_{j \in \mathcal{J}} p_{jt} \left\{ f_j y_{\phi jt} + V_{t+1} \left(x_t - \sum_{i \in \mathcal{L}} y_{ijt} a_{ij} e_i \right) \right\} \right\} \tag{DP2}$$
$$\text{subject to} \quad y_{ijt} = y_{\phi jt} \quad \forall i \in \mathcal{L},\ j \in \mathcal{J}$$

where

$$\begin{aligned} \mathcal{Y}_{it}(x_t) &= \left\{ y_{it} \in \{0, 1\}^{|\mathcal{J}|} : a_{ij} y_{ijt} \leq x_{it} \quad \forall j \in \mathcal{J} \right\} \quad i \in \mathcal{L} \\ \mathcal{Y}_{\phi t}(x_t) &= \left\{ y_{\phi t} \in \{0, 1\}^{|\mathcal{J}|} \right\} \\ \mathcal{Y}(x_t) &= \mathcal{Y}_{\phi t}(x_t) \prod_{i \in \mathcal{L}} \mathcal{Y}_{it}(x_t) \quad \text{(Cartesian product)} \end{aligned}$$

Lagrangian Relaxation:

- Let $\lambda = \{\lambda_{ijt} : i \in \mathcal{L},\ j \in \mathcal{J},\ t \in \mathcal{T}\}$ denote the Lagrangian multiplier.
- The Lagrangian relaxation $V_t^\lambda(x_t)$ is defined as:

$$V_t^\lambda(x_t) = \max_{y_t \in \mathcal{Y}(x_t)} \left\{ \sum_{j \in \mathcal{J}} p_{jt} \left[f_j y_{\phi jt} + \sum_{i \in \mathcal{L}} \lambda_{ijt} (y_{ijt} - y_{\phi jt}) + V_{t+1}^\lambda \left(x_t - \sum_{i \in \mathcal{L}} y_{ijt} a_{ij} e_i \right) \right] \right\} \tag{LR}$$

Lagrangian Relaxation Algorithm:

- **Goal**: The Lagrangian relaxation algorithm aims to find an optimal multiplier λ^* that solves

$$\min_{\lambda} V_1^{\lambda}(c_1)$$

As shown in [Topaloglu, 2009],

$$V_t(x_t) \leq V_t^{\lambda}(x_t) \quad \forall x_t \in \mathcal{C}^{|\mathcal{L}|},\ t \in \mathcal{T}$$

Therefore, $V_1^{\lambda^*}(c_1)$ provides a tight bound to $V_1(c_1)$.

- **Solving $V_1^{\lambda}(c_1)$ for a given λ**: It has been shown in [Topaloglu, 2009] that $V_1^{\lambda}(c_1)$ can be solved by concentrating on one resource at a time. Specifically, if $\{\vartheta_{it}^{\lambda}(x_{it}) : x_{it} \in \mathcal{C}, t \in \mathcal{T}\}$ is a solution to the optimality equation

$$\vartheta_{it}^{\lambda}(x_{it}) = \max_{y_{it} \in \mathcal{Y}_{it}(x_{it})} \left\{ \sum_{j \in \mathcal{J}} p_{jt} \left[\lambda_{ijt} y_{ijt} + \vartheta_{i,t+1}^{\lambda}(x_{it} - a_{ij} y_{ijt}) \right] \right\} \tag{SDP}$$

for all $i \in \mathcal{L}$, then

$$V_t^{\lambda}(x_t) = \sum_{t'=t}^{\tau} \sum_{j \in \mathcal{J}} p_{jt'} \left[f_j - \sum_{i \in \mathcal{L}} \lambda_{ijt'} \right]^+ + \sum_{i \in \mathcal{L}} \vartheta_{it}^{\lambda}(x_{it}), \tag{1}$$

where $[z]^+ = \max\{z, 0\}$.

- **Minimizing $V_1^{\lambda}(c_1)$ over λ**: It has also been shown in [Topaloglu, 2009] that the Lagrangian relaxation $V_1^{\lambda}(c_1)$ is convex in λ. Thus, the optimal multiplier λ^* can be found by using classical subgradient methods.

Control Policy:

- The control policy is to accept a request for product j at time t if and only if:

$$f_j \geq \sum_{i \in \mathcal{L}} \sum_{r=1}^{a_{ij}} \left[\vartheta_{i,t+1}^{\lambda^*}(x_{it} - r + 1) - \vartheta_{i,t+1}^{\lambda^*}(x_{it} - r) \right]$$

That is, a product is accepted if its revenue exceeds the opportunity cost of consumed resources. Specifically, the term $\vartheta_{i,t+1}^{\lambda^*}(x_{it}) - \vartheta_{i,t+1}^{\lambda^*}(x_{it} - 1)$ represents the bid price of resource i at time t.

3 The subgradient of $V_1^\lambda(\mathbf{c}_1)$

Computing the subgradient is important for updating λ_{ijt} when minimizing $V_1^\lambda(c_1)$. However, [Topaloglu, 2009] does not explicitly provide the subgradient of $V_1^\lambda(c_1)$. In this section, we show how to compute the subgradient of $V_1^\lambda(c_1)$ with respect to a specific Lagrange multiplier λ_{ijt}.

Single-Resource Value Function and Optimal Policy: Recall that the value function $\vartheta_{it}^\lambda(x_{it})$ for resource i with capacity x_{it} at time t is given by the Bellman equation:

$$\vartheta_{it}^\lambda(x_{it}) = \max_{y_{ijt}\in\{0,1\}} \left\{ \sum_{j\in\mathcal{J}} p_{jt} \left[\lambda_{ijt} y_{ijt} + \vartheta_{i,t+1}^\lambda(x_{it} - a_{ij} y_{ijt}) \right] \right\}$$

Let $y_{ijt}^{*\lambda}(x_{it})$ be the optimal decision for product j given capacity x_{it}. Then,

$$\vartheta_{it}^\lambda(x_{it}) = \sum_{j\in\mathcal{J}} p_{jt} \left[\lambda_{ijt} y_{ijt}^{*\lambda}(x_{it}) + \vartheta_{i,t+1}^\lambda(x_{it} - a_{ij} y_{ijt}^{*\lambda}(x_{it})) \right]$$

with terminal condition $\vartheta_{i,\tau+1}^\lambda(\cdot) = 0$. Thus, the value function $\vartheta_{i1}^\lambda(c_i)$ can be written as the expected sum of λ-weighted accepted products:

$$\vartheta_{i1}^\lambda(c_i) = \mathbb{E}\left[\sum_{t'=1}^{\tau} \sum_{j\in\mathcal{J}} p_{jt'} \lambda_{ijt'} y_{ijt'}^{*\lambda}(X_{it'}) \mid X_{i1} = c_i \right]$$

where $X_{it'}$ is the random capacity at time t'.

Capacity State Probability $\mu_{it}(x)$: To see the effect of λ_{ijt} on $\vartheta_{i1}^\lambda(c_i)$, define $\mu_{it}(x)$ as the probability that resource i has capacity x at time t, starting from c_i at $t = 1$ and following the optimal policy $y^{*\lambda}$. This probability mass function is computed recursively:

- **Base case ($t = 1$):** $\mu_{i1}(x) = \mathbb{I}\{x = c_i\}$.
- **Recursive step (for $s = 1, \dots, \tau - 1$):**

$$\mu_{i,s+1}(x') = \sum_{x=0}^{c_i} \mu_{is}(x) \sum_{j\in\mathcal{J}} p_{js} \mathbb{I}\left\{ x' = x - a_{ij} y_{ijs}^{*\lambda}(x) \right\}$$

The subgradient of $\vartheta_{i1}^\lambda(c_i)$ with respect to λ_{ijt}: As we can see, the change in $\vartheta_{i1}^\lambda(c_i)$ due to λ_{ijt} is the sum of local effects at time t, weighted by the probability $\mu_{it}(x_{it})$ of being in state x_{it}:

$$\frac{\partial \vartheta_{i1}^\lambda(c_i)}{\partial \lambda_{ijt}} = \sum_{x_{it}=0}^{c_i} \mu_{it}(x_{it}) \left(p_{jt} y_{ijt}^{*\lambda}(x_{it}) \right) = p_{jt} \sum_{x_{it}=0}^{c_i} \mu_{it}(x_{it}) y_{ijt}^{*\lambda}(x_{it})$$

Intuitively, the sum $\sum_{x_{it}=0}^{c_i} \mu_{it}(x_{it}) y_{ijt}^{*\lambda}(x_{it})$ is the expected optimal decision for product j at time t, given initial capacity c_i.

The subgradient of $V_1^\lambda(c_1)$ with respect to λ_{ijt}: Using (1), we have:

$$\frac{\partial V_1^\lambda(c_1)}{\partial \lambda_{ijt}} = \frac{\partial \vartheta_{i1}^\lambda(c_i)}{\partial \lambda_{ijt}} - p_{jt}\, \mathbf{1}\left\{ f_j - \sum_{k\in\mathcal{L}} \lambda_{kjt} \geq 0 \right\}$$

4 Implementation

Note: [Topaloglu, 2009] did not provide any specific implementation details or the choice of algorithms. Thus, I need to do my own implementation. What follows are my own implementation steps.

4.1 Subroutine: Solving the Single-Resource Dynamic Program

We solve the single-resource optimality equation using tabular backward induction:

Algorithm 1 Subroutine: Tabular Backward Induction for Single-Resource Dynamic Program

Require: Resource i, capacities $\mathcal{C}$, time periods $\mathcal{T}$, probabilities p_{jt}, consumption a_{ij}, multipliers λ_{ijt}

Ensure: Value functions $\vartheta_{it}^{\lambda}(x_{it})$ and optimal decisions $y_{ijt}^{*}(x_{it})$

Initialize $\vartheta_{i,\tau+1}^{\lambda}(x_{i,\tau+1}) \leftarrow 0$ for all $x_{i,\tau+1} \in \mathcal{C}$ ▷ Terminal condition
for $t = \tau$ **down to** 1 **do** ▷ Backward recursion
    for $x_{it} = 0$ **to** C **do** ▷ For each capacity level
        $\vartheta_{it}^{\lambda}(x_{it}) \leftarrow 0$
        $y_{ijt}^{*}(x_{it}) \leftarrow 0$ for all $j \in \mathcal{J}$ ▷ Initialize decision variables
        for $j \in \mathcal{J}$ **do** ▷ For each product
            if $a_{ij} \leq x_{it}$ **then** ▷ Check if capacity is sufficient
                $v_0 \leftarrow \vartheta_{i,t+1}^{\lambda}(x_{it})$ ▷ Value if reject
                $v_1 \leftarrow \lambda_{ijt} + \vartheta_{i,t+1}^{\lambda}(x_{it} - a_{ij})$ ▷ Value if accept
                if $v_1 > v_0$ **then**
                    $y_{ijt}^{*}(x_{it}) \leftarrow 1$ ▷ Accept product j at time t with capacity x_{it}
                end if
            end if
        end for
        $\vartheta_{it}^{\lambda}(x_{it}) \leftarrow \sum_{j \in \mathcal{J}} p_{jt} \Big[\lambda_{ijt}\, y_{ijt}^{*}(x_{it}) + \vartheta_{i,t+1}^{\lambda}\big(x_{it} - a_{ij} y_{ijt}^{*}(x_{it})\big) \Big]$
    end for
end for
return $\{\vartheta_{it}^{\lambda}(x_{it}) : x_{it} \in \mathcal{C}, t \in \mathcal{T}\}$ and $\{y_{ijt}^{*}(x_{it}) : j \in \mathcal{J}, x_{it} \in \mathcal{C}, t \in \mathcal{T}\}$

4.2 Subroutine: Computing State Probabilities

This subroutine is executed for each resource $i \in \mathcal{L}$. It computes the state occupancy probabilities $\mu_{is}(x)$ for resource i at each time $s \in \mathcal{T}$ and capacity level $x \in \mathcal{C}$. These probabilities indicate the likelihood of resource i having x units of capacity at the beginning of period s, given an initial capacity $c_{i,1}$ at $s = 1$ and following the optimal single-resource policies $y^{*\lambda}_{ijs}(x)$ derived from Algorithm 1.

Algorithm 2 Compute State Probabilities $\mu_{is}(x)$ for Resource i

Require: Resource i, initial capacity $c_{i,1}$, maximum capacity C, time periods $\mathcal{T}$, product set $\mathcal{J}$, arrival probabilities p_{js}, consumption a_{ij}, optimal policies $y^{*\lambda}_{ijs}(x)$
Ensure: State probabilities $\mu_{is}(x)$ for $s \in \mathcal{T}, x \in \{0, \ldots, C\}$

Initialize array $\mu_{i\cdot(\cdot)} : (\mathcal{T} \times \mathcal{C}) \to \mathbb{R}$ with all entries 0.0.
$\mu_{i1}(c_{i,1}) \leftarrow 1.0$ ▷ At $s = 1$, capacity is $c_{i,1}$ with prob. 1
for $s = 1$ **to** $\tau - 1$ **do** ▷ Forward in time
  for $x_{curr} = 0$ **to** C **do** ▷ For each capacity at s
    if $\mu_{is}(x_{curr}) > 10^{-9}$ **then** ▷ If x_{curr} is reachable
      for $j \in \mathcal{J}$ **do** ▷ For each product j
        $y^*_{curr} \leftarrow y^{*\lambda}_{i,j,s}(x_{curr})$ ▷ Optimal decision at (i, s, x_{curr})
        $x_{next} \leftarrow x_{curr} - a_{i,j} \cdot y^*_{curr}$
        if $x_{next} < 0$ **then** ▷ Negative capacity should not actually occur!
          $x_{next} \leftarrow 0$
        end if
        $\mu_{i,s+1}(x_{next}) \leftarrow \mu_{i,s+1}(x_{next}) + \mu_{is}(x_{curr}) \cdot p_{j,s}$
      end for
    end if
  end for
end for
return μ array (containing $\mu_{is}(x)$)

4.3 Subroutine: Computing ϑ-Subgradient

This subroutine uses the state probabilities and the optimal policies to calculate the subgradient of the single-resource total expected λ-weighted value $\vartheta_{i1}^{\lambda}(c_{i,1})$ with respect to each relevant Lagrange multiplier λ_{ijs}.

Algorithm 3 Compute ϑ-Subgradient G_{ijs} for Resource i

Require: State probabilities $\mu_{is}(x)$, optimal policies $y_{ijs}^{*\lambda}(x)$, arrival probabilities p_{js}, product set $\mathcal{J}$, time periods $\mathcal{T}$, maximum capacity C

Ensure: Subgradients $G_{ijs} = \frac{\partial \vartheta_{i1}^{\lambda}(c_{i,1})}{\partial \lambda_{ijs}}$ for $j \in \mathcal{J}, s \in \mathcal{T}$

Initialize array $G_{i\cdot\cdot} : (\mathcal{J} \times \mathcal{T}) \to \mathbb{R}$ with all entries set to 0.0.
for $s = 1$ **to** τ **do** ▷ For each time period s
    for $j \in \mathcal{J}$ **do** ▷ For each product j
        $\text{expected\_y\_star}_{ijs} \leftarrow 0$
        for $x = 0$ **to** C **do** ▷ For each capacity level x at time s
            if $\mu_{is}(x) > 10^{-9}$ **then** ▷ If state (i, s, x) is reachable
                $\text{expected\_y\_star}_{ijs} \leftarrow \text{expected\_y\_star}_{ijs} + \mu_{is}(x) \cdot y_{i,j,s}^{*\lambda}(x)$
            end if
        end for
        $G_{i,j,s} \leftarrow p_{j,s} \cdot \text{expected\_y\_star}_{ijs}$
    end for
end for
return G array (containing G_{ijs})

4.4 Final Subgradient Optimization

Note that as a Lagrangian multiplier, $\lambda_{ijt} \geq 0$. Thus, here we use a projected subgradient descent algorithm to optimize λ.

Algorithm 4 Projected Subgradient Descent for Lagrangian Multiplier Optimization

Require: Initial multipliers λ^0, step size α_0, tolerance ϵ, maximum iterations K
Ensure: Optimized multipliers λ^*

$k \leftarrow 0$
$V_{\text{prev}} \leftarrow \infty$
while $k < K$ **do**
  Compute $V_1^{\lambda^k}(c_1)$
  if $|V_1^{\lambda^k}(c_1) - V_{\text{prev}}| < \epsilon$ **then**
    break
  end if
  $V_{\text{prev}} \leftarrow V_1^{\lambda^k}(c_1)$
  Compute subgradient g^k of $V_1^{\lambda^k}(c_1)$ with respect to λ.
  $\alpha_k \leftarrow \frac{\alpha_0}{\sqrt{k+1}}$
  $\lambda^{k+1} \leftarrow \max\{0, \lambda^k - \alpha_k g^k\}$ ▷ Project onto feasible set
  $k \leftarrow k + 1$
end while
return λ^k

4.5 Algorithm: Computing Total Expected Revenue using Bid Prices

After obtaining optimal Lagrangian multipliers λ^* (e.g., via Algorithm 4), we define a bid price control policy. To evaluate this policy, we estimate total expected revenue through Monte Carlo simulation of customer arrivals and decisions. The algorithm below details this process.

Algorithm 5 Estimate Total Expected Revenue using Bid Price Policy (Monte Carlo Simulation)

Require: Precomputed value functions $\vartheta_{i,t'}^{\lambda^*}(x)$, initial capacities c, product set $\mathcal{J}$ (incl. dummy ψ), revenues f_j, resource consumptions a_{ij}, arrival probabilities p_{jt}, time horizon $\mathcal{T} = \{1, \dots, \tau\}$, number of simulations N_{sim}

Ensure: Estimated total expected revenue $E[R]$

Initialize $total\_rev \leftarrow 0.0$
for $s = 1$ **to** N_{sim} **do** ▷ For each simulation run s
    $run\_rev \leftarrow 0.0$
    $cap \leftarrow$ copy of c ▷ Reset capacities for this run
    for $t = 1$ **to** τ **do** ▷ For each time period t
        Sample product j using probabilities $\{p_{jt}\}_{j \in \mathcal{J}}$ for time t.
        if $f_j > 0$ **then** ▷ Process non-dummy products (dummy ψ has $f_\psi = 0$)
            $opp\_cost \leftarrow 0.0$
            $enough\_cap \leftarrow$ true
            ▷ Assess j: check capacity and calculate opportunity cost
            for $i \in \mathcal{L}$ such that $a_{ij} > 0$ **do** ▷ For each resource i consumed by j
                if $cap[i] < a_{ij}$ **then**
                    $enough\_cap \leftarrow$ false
                    break ▷ Insufficient capacity for j on resource i
                end if
                for $r = 1$ **to** a_{ij} **do** ▷ Sum bid prices for each unit of i consumed
                    $bp \leftarrow \vartheta_{i,t+1}^{\lambda^*}(cap[i] - r + 1) - \vartheta_{i,t+1}^{\lambda^*}(cap[i] - r)$
                    $opp\_cost \leftarrow opp\_cost + bp$
                end for
            end for
            if $enough\_cap$ and $f_j \geq opp\_cost$ **then** ▷ Accept j
                $run\_rev \leftarrow run\_rev + f_j$
                for $i \in \mathcal{L}$ such that $a_{ij} > 0$ **do** ▷ Update consumed capacities
                    $cap[i] \leftarrow cap[i] - a_{ij}$
                end for
            end if
        end if ▷ Capacities cap are now updated for period $t+1$
    end for ▷ End of time horizon $\mathcal{T}$ for run s
    $total\_rev \leftarrow total\_rev + run\_rev$
end for ▷ End of N_{sim} simulation runs
$E[R] \leftarrow total\_rev / N_{sim}$
return $E[R]$

5 Numerical Results

The exact results depend on the tuning of algorithm parameters, such as step length and other settings. Below is a comparison of our final results with those reported in [Topaloglu, 2009].

Problem	Upper Bound (Huseyin)	Upper Bound (Our Impl.)	Mean Revenue (Huseyin)	Mean Revenue (Our Impl.)	Std (Our Impl 1000 Samples)
rm_200_4_1.0_4.0	20,439	20,436	20,018	20,049	31.31
rm_200_4_1.0_8.0	33,305	33,261	32,226	32,821	62.70
rm_200_4_1.2_4.0	18,938	18,885	18,374	18,510	28.49
rm_200_4_1.2_8.0	31,737	31,651	30,852	31,271	64.73
rm_200_4_1.6_4.0	16,600	16,541	15,981	16,186	27.72
rm_200_4_1.6_8.0	29,413	29,247	28,381	28,978	63.80
rm_200_5_1.0_4.0	21,298	21,296	21,181	20,973	34.79
rm_200_5_1.0_8.0	34,393	34,377	34,271	34,068	69.42
rm_200_5_1.2_4.0	20,184	20,112	19,818	19,677	33.06
rm_200_5_1.2_8.0	33,165	33,051	32,766	32,620	68.80
rm_200_5_1.6_4.0	17,704	17,654	17,318	17,218	30.93
rm_200_5_1.6_8.0	30,594	30,492	30,107	29,980	66.84
rm_200_6_1.0_4.0	21,128	21,113	20,709	20,729	33.13
rm_200_6_1.0_8.0	34,178	34,102	33,466	33,664	66.86
rm_200_6_1.2_4.0	19,649	19,636	19,156	19,165	31.25
rm_200_6_1.2_8.0	32,566	32,520	31,808	31,993	67.36
rm_200_6_1.6_4.0	17,304	17,256	16,269	16,837	30.08
rm_200_6_1.6_8.0	30,170	30,061	29,320	29,599	65.70
rm_200_8_1.0_4.0	18,975	18,778	18,217	18,268	31.10
rm_200_8_1.0_8.0	30,490	30,275	29,453	29,716	66.44
rm_200_8_1.2_4.0	17,472	17,501	16,941	16,915	29.44
rm_200_8_1.2_8.0	28,908	28,889	28,130	28,236	61.56
rm_200_8_1.6_4.0	15,295	15,297	14,720	14,764	27.34
rm_200_8_1.6_8.0	26,661	26,555	25,701	25,988	63.75
rm_600_4_1.0_4.0	30,995	30,994	30,640	30,575	49.25
rm_600_4_1.0_8.0	50,444	50,406	49,862	49,872	107.31
rm_600_4_1.2_4.0	28,668	28,615	28,145	28,167	44.65
rm_600_4_1.2_8.0	48,054	47,947	47,162	47,541	101.67
rm_600_4_1.6_4.0	25,148	25,084	24,540	24,596	43.50
rm_600_4_1.6_8.0	44,555	44,357	43,547	44,003	102.95
rm_600_5_1.0_4.0	32,254	32,272	32,112	31,775	56.33
rm_600_5_1.0_8.0	52,071	52,056	51,275	51,668	118.57
rm_600_5_1.2_4.0	30,004	30,552	30,308	30,100	51.57
rm_600_5_1.2_8.0	50,282	50,162	49,899	49,629	114.59
rm_600_5_1.6_4.0	26,936	26,880	26,605	26,441	44.95
rm_600_5_1.6_8.0	46,497	46,355	46,070	45,858	107.20
rm_600_6_1.0_4.0	25,541	25,559	25,310	25,044	47.32
rm_600_6_1.0_8.0	41,412	41,262	40,849	40,753	102.27
rm_600_6_1.2_4.0	23,687	23,708	23,306	23,191	42.77
rm_600_6_1.2_8.0	39,307	39,270	38,704	38,799	100.42
rm_600_6_1.6_4.0	20,817	20,788	20,273	20,229	41.46
rm_600_6_1.6_8.0	36,381	36,261	35,631	35,867	101.19
rm_600_8_1.0_4.0	22,960	22,798	22,269	22,206	44.93
rm_600_8_1.0_8.0	36,933	36,718	36,046	36,071	95.73
rm_600_8_1.2_4.0	21,102	21,172	20,633	20,431	39.54
rm_600_8_1.2_8.0	34,831	34,939	34,277	34,168	88.63
rm_600_8_1.6_4.0	18,500	18,553	17,830	17,888	35.95
rm_600_8_1.6_8.0	32,247	32,180	31,317	31,411	90.19

Table 1: Comparison of bounds and revenues. Green: our value is better; Red: our value is worse.

6 Resources

This project is based on [Topaloglu, 2009]. Both the paper and its dataset are publicly available on Prof. Huseyin Topaloglu's website. You can access the paper directly here. The dataset can be downloaded from this page.

References

[Topaloglu, 2009] Topaloglu, H. (2009). Using lagrangian relaxation to compute capacity-dependent bid prices in network revenue management. *Operations Research*, 57(3):637–649.