Seat assignment with the social distancing

Dis· count

January 9, 2023

0.1 Branch And Price

Rather than solving IP directly to obtain the optimal integer solution, the commonly used method is to branch the fractional solution.

General branch is commonly as follows: $\sum_{k\in K(k^j)} x_k = \alpha$, where $K(p) = \{k \in K : k \geq p\}$ (column subsets) for $p \in Z_+^m$, α is fractional. $K = \{k \in N^m : \sum_{i=1}^m s_i k_i \leq S\}$ indicate all feasible patterns, and x_k is the number of times pattern k selected. Given a feasible solution x^* of LP that is not integral, take k^* to be any maximal element of the set $\{k \in K : x_k^* \notin Z_+\}$. Then the only fractional term in this sum $\sum_{k\in K(k^j)} x_k^*$ is $x_{k^*}^*$. Maximal means that the space left after cutting this pattern is less than the smallest size of the group.

Then the generic branching constraints will be: $\sum_{k\in K(k^j)} x_k \leq U^j, \forall j \in G^u$ and $\sum_{k\in K(k^j)} x_k \geq L^j, \forall j \in H^u$, where G^u and H^u are sets of branching constraints of the form

$$\sum_{k\in K(k)} x_k \leq \lfloor \alpha \rfloor \tag{1}$$

and

$$\sum_{k\in K(k)} x_k \geq \lceil \alpha \rceil \tag{2}$$

, respectively.

If we can always generate the maximal pattern, then any fractional column defines a branching set which contains only one member.

The corresponding restricted master problem will be reformulated as:

$$
\begin{aligned}
\max \quad & \sum_{k\in K}(\sum_{i=1}^{m}(s_i-1)t_i^k)x_k \\
\text{s.t.} \quad & \sum_{k\in K} x_k \le N \\
& \sum_{k\in K} t_i^k x_k \le g_i, \quad i=1,\ldots,m \\
& \sum_{k\in K(k^j)} x_k \le U^j, \forall j \in G^u \\
& \sum_{k\in K(k^j)} x_k \ge L^j, \forall j \in H^u \\
& x_k \ge 0, \quad k \in K
\end{aligned}
$$

Let (π, λ, μ, v) be optimal dual variables associated with the added constraints. The pricing problem(sub-problem) is:

$$
\begin{aligned}
\max \quad & [(s_i-1)-\lambda_i]\, y_i - \pi - \sum_{j\in G^u}\mu_j z^j - \sum_{j\in H^u} v_j z^j \\
\text{s.t.} \quad & \sum_{i=1}^{m} s_i y_i \le S \\
& z^j = 1 \text{ if } y \ge k^j; z^j = 0 \text{ otherwise} \\
& y_i \ge 0, \text{integer} \quad \text{for } i=1,\ldots,m.
\end{aligned}
$$

$Y_k = \{(y,z) : z = 1, \text{ if } y \ge k, z = 0 \text{ otherwise}\}$ can be formulated as MIPs.

The pricing problem is only affected when an upper bound has been placed on the value of the variable associated with the selected pattern. Because this variable could be a nonbasic variable for the LP. In this case, we have avoid regenerating this maximal pattern. Thus, we should solve a knapsack problem with a forbidden pattern set. Besides, the drawback of this method will be that the branch is unbalanced.

The procedure of branch and price is as follows:

1) Solve the restricted master problem with the initial columns.

2) Use the pricing problem to generate columns.

3) When the column generation method terminates, is the solution integral?

   Yes, update lower bound.

   No, update upper bound. And fathom node or branch to create two nodes.

4) Select the next node until all nodes are explored.

0.2 Column-and-cut generation

Recall that T is a subset of the set of all possible cutting patterns, thus we need to find more cutting patterns by the column generation.

Lemma 1. *When the group sizes are consecutive integers starting from 2, i.e.,* $[2, 3, \ldots, u]$*, T only need include the full or largest cutting patterns.*

We can always find a full pattern which accomodates more people than any non-full pattern except when the largest pattern is $(u, \ldots, u, 1)$.

Suppose that we have p cuts during some iteration, let $(\pi = [\pi_1, \pi_2, \cdots, \pi_p], \lambda)$ be optimal dual variables associated with the scenario constraints and row number constraint.

Then the pricing problem will be

$$\begin{aligned} \max \quad & \sum_{i=1}^{m} \left[(s_i - 1) - \sum_{p} \pi_p \alpha_i^p \right] y_i - \lambda \\ \text{s.t.} \quad & \sum_{i=1}^{m} s_i y_i \leq L \\ & y_i \geq 0, \text{integer} \quad \text{for } i = 1, \ldots, m \end{aligned} \tag{3}$$

where y_i indicate the number of times group type i placed in the new pattern.

By solving the pricing problem, we can obtain a new cutting pattern, then add it to T. Then solve the master problem to obtain (x^*, z^*) and continue the column-and-cut generation.

After we obtained x_1, add new optimality cuts and cutting patterns, then repeat the above procedure until the optimal value does not increase anymore.

0.3 Occupancy

For each pattern k, we use α_k, β_k to indicate the number of groups and the left space, respectively. Denote $(\alpha_k + \beta_k)$ as the loss for pattern k, $l(k)$.

For any pattern k, the occupancy is $\frac{S-l(k)}{S-1}$ and the largest occupancy is $\frac{S-l(k)}{S-1}, k \in I_1$. Thus, the largest occupancy for multiple rows is also $\frac{S-l(k)}{S-1}, k \in I_1$ when all rows are largest patterns. $l(k)$ can be represented in another way, then we have the following lemma.

Lemma 2. *Suppose the size of group types be* $s = [2, 3, \ldots, u]$*, the largest occupancy will be* $\frac{S-q-\text{sign}(r)}{S-1}$*, where* $q = \lfloor \frac{S}{u} \rfloor$*,* $\text{sign}(r) = 1$ *if* $r > 0$*,* $\text{sign}(r) = 0$ *if* $r = 0$*.*

Suppose the size of group types be $s = [2, 3, \ldots, u]$, the maximal group size be u and $S = u \cdot q + r$. When $r = 0$, the minimal loss is q, the largest occupancy will be $\frac{S-q}{S-1}$. When $r > 0$, the minimal loss will be $q + 1$, the largest occupancy will be $\frac{S-q-1}{S-1}$. Thus, the largest occupancy will be $\frac{S-q-\text{sign}(r)}{S-1}$, where $q = \lfloor \frac{S}{u} \rfloor$, $\text{sign}(r) = 1$ if $r > 0$, $\text{sign}(r) = 0$ if $r = 0$.

If we want the largest occupancy is no more than $\frac{1}{2}$, we have $\frac{S-\lfloor \frac{S}{u} \rfloor -1}{S-1} \leq \frac{1}{2} \Rightarrow \frac{S-1}{2} \leq \lfloor \frac{S}{u} \rfloor$. It is clear that when $u = 2$ this inequality holds. When $u = 3$, S should be no larger than 3. In other cases, this inequality doesnot hold.

When $uq \leq S \leq uq + (u-1)$, $\lfloor \frac{S}{u} \rfloor$ equals q and this ratio $\frac{S-q-1}{S-1}$ increases in S when q is fixed. Thus, when $S = uq + (u-1)$, the occupancy can reach maximum value, $\frac{S-q-1}{S-1}$.

When the social distancing is 2, the size of group types will be $s = [3, 4, \ldots, u]$, the occupancy will be $\frac{S-1-2*\lfloor \frac{S}{u} \rfloor}{S-1}$

2): The initial analysis about the loss between group type [2,3,. . . ,t-1] and [2,3,. . . ,t]. Suppose that $S \div t = q \ldots r$, then $S \div (t-1) = q \ldots q + r$ If $q + r < t - 1$, the loss of the largest pattern for group size $t, (t-1)$, will be the same. If $q + r = t - 1$, $S = (t-1) * (q+1)$, the loss of the largest pattern for group size $t, (t-1)$, will be the same.

In other cases, the quotient of $S/(t-1)$ will be larger than that of S/t. If the loss of the largest pattern for group size $(t-1)$ is l, the loss of the largest pattern for group size t will be $l + 1$.

3): If the size of group types is $s = [2, 3, 4]$, then we can obtain an optimal solution by the following way.

Suppose the demands for each group type are positive and d_2, d_3, d_4, respectively. The number of seats for each row is S. We have N rows at the beginning. Let $S = 4 \times q + r, q = \lfloor \frac{S}{4} \rfloor$.

If $r = 0$, the largest pattern will be $(4, 4, \ldots, 4)$;

If $r = 1$, the largest pattern will be $(4, 4, \ldots, 4, 3, 2)$;

If $r = 2$, the largest pattern will be $(4, 4, \ldots, 4, 2)$;

If $r = 3$, the largest pattern will be $(4, 4, \ldots, 4, 3)$;

Repeat to generate the largest pattern until we do not have enough demands. If any one of demands for group types runs out, the problem is reduced to the case of two group types, use the above method to solve this case.

Now suppose the rest demands are $d_2^{'}, d_3^{'}, d_4^{'}$ after deducting the demands of the largest patterns. Let $p = \lfloor \frac{d_4}{q} \rfloor$, then $d_4^{'} = d_4 - p \times q$. p is equal to the number of rows where we placed groups.

$d_4^{'}$ should be less than q, otherwise, we can continue to generate the largest pattern.

For the next row, we place all rest $d_4^{'}$ groups with size of 4. Then we will have $S^{'} = S - d_4^{'} \times 4$ empty seats. The problem can be reduced to the case of two group types. That is, place groups with size of 3 as many as possible, the rest seats can be taken by group with size of 2.

For the rest $(N - p - 1)$ rows, this problem is reduced to the case of two group types.

Extension: If the size of group types is $s = [2, 3, u]$, u is an integer and larger than 3, then we can obtain an optimal solution by the same way.

Remark: We can only obtain one optimal assignment, but this problem still has other optimal assignments.

0.4 Details about the initial cutting pattern

It seems that the initial cutting patterns are not crucial to the complexity of the whole problem.

How to generate the initial cutting patterns depends on the demands for scenarios.

At least, we know that the full or largest patterns are needed.

0.5 Complexity about the number of cutting patterns

We only know that the full pattern will be used.

Demonstrate the number of cutting patterns is large.

The number of cutting patterns is equal to the number of solutions to $\sum_{i=1}^{m} s_i y_i = L$. $\{y_i\}$

0.6 How to generate an integral demand from a fractional demand

If the optimal supply is $[2, 3.75, 4.25, 6.75]$, how should we deal with it?

Theorem 1. *An optimal integral supply can be derived from the fractional supply obtained by stochastic programming.*

For $[2, 3.75, 4.25, 6.75]$, change it to $[2, 3\frac{1}{3}, 4.25, 7]$, to $[1, 3, 5, 7]$.

When $S = 50$, $[2, 3, 4, 5]$, $N = 3$, the scenario demands generated from $[5, 15]$, then from the fractional demand $[6, 9\frac{1}{3}, 10, 14]$, the integral demands can be $[5, 10, 10, 14]$.

Let D be the polyhedron including all feasible demands with N rows, i.e., $D = \{d = (d_1, d_2, \ldots, d_n) : Tx = d, \sum_k x_k \leq N, x_k \geq 0, \text{integer}\}$. Denote by $d^i, i = 1, \ldots, I-1$, the planning demands, where I is the number of group types.

Then every d^i gives the maximum number of people under given scenarios and locates at the vertex of D.

We can obtain the patterns from the integral demand by solving the problem, which we mentioned in section . (From (D), we know how to check the feasibility of the integral demand) If some integral demand is not feasible, then delete it from the plannings.

For several plannings, we select the one that matches the request. (The plan has priority over rejection or use DP to compare the expected value.)

0.7 Two adjacent group types model

The model assumes two group types, group 1 and 2, with associated sizes $s_1 = s_2 + 1$.

It can help us accept or reject group 2 when the capacity of group 2 is not enough but the capacity of group 1 is sufficient.

Suppose that there are x quantities of group 1 remaining and we receive a request from group 2.

If we accept the request, $(s_2 - 1)$ people will be placed. If we reject it, we will place x quantities of group 1 if and only if demand for group 1 is x or higher. That is, if and only if $D_1 \geq x$. Thus, the expected number of people from reserving x units of capacity for group 1 is $(s_1 - 1)P(D_1 \geq x)$.

Therefore, we will accept a group 2 request if and only if $(s_2 - 1) \geq (s_1 - 1)P(D_1 \geq x) \Rightarrow P(D_1 < x) \geq \frac{1}{s_2}$.

If a continuous distribution $F(x)$ is used to model demand, D_1, then the optimal remaining capacity is given by the simpler expression, $x^* = F^{-1}(\frac{1}{s_2})$.

0.8 [a,b]

Lemma 3. *For* $[2,3]$*, we can obtain an optimal assignment by the policy.*

The following procedure can generate an optimal assignment.

Suppose the group sizes are $2,3$ and the demand is d_2, d_3, respectively. We have N rows initially.

If $S \div 3 = q \dots 1$, the largest patterns will be $(3,3,\dots,3,2,2),(3,3,\dots,3,1)$. If $S \div 3 = q \dots 2$, the largest pattern will be $(3,3,\dots,3,2)$. If $S \div 3 = q$, the largest pattern will be $(3,3,\dots,3)$.

According to the value of S, repeat the largest pattern until $q > d_3$ or $d_2 = 0$. When $q > d_3$, just place groups with 3 d_3 times, the rest space will be placed by groups with 2. When $d_2 = 0$, just place groups with 3 only. In the way, we can obtain the optimal scheme.

For example, $S \div 3 = q \dots 1$, each row will be divided by $(q-1) \times 3, 2 \times 2$, thus we need to find the smaller one, $\min\{\lfloor \frac{d_3}{q-1} \rfloor, \lfloor \frac{d_2}{2} \rfloor\}$, denoted by r. If $r \geq N$, choose the largest pattern N times. If $r < N$, when $\lfloor \frac{d_3}{q-1} \rfloor$ is smaller, the number of remaining groups with 3 is $d_3 - \lfloor \frac{d_3}{q-1} \rfloor * (q-1)$.

When $\lfloor \frac{d_2}{2} \rfloor$ is smaller, the number of remaining groups with 2 is $d_2 - \lfloor \frac{d_2}{2} \rfloor * 2(0 \text{ or } 1)$. Retain the group with 2, then place groups with 3 repeatedly.

The conclusion can be extended to any two types of group size. But we still need to consider the number of the largest pattern and the demands of group sizes.

As long as we can obtain the maximum number of the largest patterns with some demands, we will obtain an optimal assignment.

Lemma 4. *For* $[4,5]$*, only when* $r = 1$*, we have two largest patterns,* $(5,\dots,5,1)$ *and* $(5,\dots,5,4,4,4,4)$*.*

For $[3,5]$*, when* $r = 1$*, we have two largest patterns,* $(5,\dots,5,1)$ *and* $(5,\dots,5,3,3)$*. When* $r = 4$*, we have two largest patterns,* $(5,\dots,5,3,1)$ *and* $(5,\dots,5,3,3,3)$*, which depends on* D_3, D_5*.*

For $[2,5]$*, we don't have two largest patterns.*

For $[2,4]$*, we don't have two largest patterns.*

For $[3,4]$*, only when* $r = 1$*, we have two largest patterns,* $(4,\dots,4,1)$ *and* $(4,\dots,4,3,3,3)$*.*

For $[2,3]$*, only when* $r = 1$*, we have two largest patterns,* $(3,\dots,3,1)$ *and* $(3,\dots,3,2,2)$*.*

Theorem 2. *For two types of group size,*$[a,b], a < b \leq 5$*, we can obtain an optimal assignment by following our policy.*

Algorithm 1 Construct an optimal assignment for $[a,b], 2 \leq a < b \leq 5$

Step 1. Generate all the largest patterns, $k_i, i = 1, 2, \ldots, l$ when given S, $[a,b]$, $(d_a, d_b)_D$, n rows.

Step 2. According to the demands, obtain the priority of the largest patterns, suppose the priority be $k_1 \succeq k_2 \succeq \ldots \succeq k_l$.

Step 3. Fill the rows with the largest patterns until we cannot obtain the largest pattern. Suppose the number of rows is m. When $m \geq n$, we have obtained an optimal assignment. When $m < n$, continue Step 4.

Step 4. Generate the largest pattern in the greedy way. Place the corresponding groups in the $(m+1)$-st row. One of the group sizes will run out.

Step 5. Place groups with size of a or b in the remaining $n - (m+1)$ rows.

Why (3,3,3,1) will be worse than (3,3,2,2)? Just consider this case: We have 10 groups with 2, 6 groups with 3, the length for each row is 10. For the first policy, when we need 3 rows to place, the result will be (3,3,3,1)*2, (2,2,2,2,2). But the second policy will give a better result, (3,3,2,2)*3. The reason is 1 space will be wasted by (3,3,3,1).

Notice that the group with size of a can provide a profit of $(a-1)$, thus placing the group with the largest size as many as possible can make a larger profit.

Algorithm 2 Construct an optimal assignment for $[a,b,c], 2 \leq a < b < c \leq 5$

Step 1. Generate all the largest patterns, $k_i, i = 1, 2, \ldots, l$ when given S, $[a,b,c]$, $(d_a, d_b, d_c)_D$, n rows.

Step 2. According to the demands, obtain the priority of the largest patterns, suppose the priority be $k_1 \succeq k_2 \succeq \ldots \succeq k_l$.

Step 3. Fill the rows with the largest patterns until we cannot obtain the largest pattern. Suppose the number of rows is m. When $m \geq n$, we have obtained an optimal assignment. When $m < n$, continue Step 4.

Step 4. Generate the largest pattern in the greedy way. Place the corresponding groups in the $(m+1)$-st row. One of the group sizes will run out.

Step 5. We use Algorithm 2 to solve the case of two group sizes.

0.9 [a,b,c]

Theorem 3. *For any $[a,b,c], 2 \leq a < b < c \leq 5$, we can obtain an optimal assignment by the following policy.*

0.9.1 [2,3,4]

Theorem 4. *The following policy will give an optimal assignment for [2,3,4].*

Suppose the demands for each group type are positive and D_2, D_3, D_4, respectively. The number of seats for each row is S. We have n rows at the beginning. Let $S = 4 \times q + r, q = \lfloor \frac{S}{4} \rfloor$.

If $r = 0$, the largest pattern will be $(4, 4, \ldots, 4)$;

If $r = 1$, the largest pattern will be $(4, \ldots, 4, 1), (4, \ldots, 4, 3, 2), (4, \ldots, 4, 3, 3, 3)$; Let t be the number of 4 in pattern $(4, \ldots, 4, 3, 3, 3)$. When $\min\{\lfloor \frac{D_4}{t} \rfloor, D_2\} > \lceil \frac{D_3}{3} \rceil$, we select $(4, \ldots, 4, 3, 2)$; otherwise, we select $(4, \ldots, 4, 3, 3, 3)$.

If $r = 2$, the largest pattern will be $(4, \ldots, 4, 3, 3), (4, \ldots, 4, 4, 2)$, we will select $(4, \ldots, 4, 3, 3)$;

If $r = 3$, the largest pattern will be $(4, \ldots, 4, 3)$;

Repeat to generate the largest pattern until we do not have enough demands. If any one of demands for group types runs out, the problem is reduced to the case of two group types, use the above method to solve this case.

Now suppose the rest demands are d_2^l, d_3^l, d_4^l after deducting the demands of the largest patterns.

d_4^l should be less than q for $r = 0, 3$, less than $q - 1$ for $r = 1, 2$, otherwise, we can continue to generate the largest pattern.

For the next row, we place all rest d_4^l groups with size of 4. Then we will have $S^l = S - d_4^l \times 4$ empty seats. The problem can be reduced to the case of two group types. That is, place groups with size of 3 as many as possible, the rest seats can be taken by group with size of 2.

For the rest $(n - p - 1)$ rows, this problem is reduced to the case of two group types.

Lemma 5. *For the case $r = 2$, $(4, \ldots, 4, 3, 3)$ will give an assignment no worse than that of $(4, \ldots, 4, 2)$.*

(Proof of lemma 5)**.** *Let $d_i(n), i = 2, 3, 4$ indicate the number of demand which can be satisfied by the first policy when placing n rows. Denote by $d_i^{'}(n), i = 2, 3, 4$ the number of demand when we take the second policy.*

There are three stages to satisfy demands for n rows. The first one is to generate the largest pattern until we cannot get the largest pattern. The second one is to satisfy the group with largest size. The third one is to satisfy the left two types of demands.

If the first stage can cover n rows for both policies. These two policies will have no difference because they both give the largest pattern.

If the second policy cannot generate the largest pattern, but the first policy can generate the largest pattern, the first policy will give a better solution than the second one.

If the first policy cannot generate the largest pattern, $(4, \ldots, 4, 3, 3)$, but the second policy can generate the largest pattern, it means $D_3 - d_3 < 2$. Notice that the first policy will satisfy less groups with 4. Because the remaining demands, $D_4 - d_4$,$D_2 - d_2$ will be larger than those of the second one, the first policy can generate $(4, \ldots, 4, 2)$ as the largest pattern until both of them cannot generate the largest pattern.

If both policies cannot generate the largest pattern in stage two, for the second policy, it means $D_4 - d_4^{'} < q$ or $D_2 = d_2^{'}$.

When $D_4 - d_4^{'} \geq q$, the first policy can always generate $(4, \ldots, 4, 2)$, thus $D_4 - d_4^{'} < q$ and $D_4 - d_4 < q$ always hold.

In this way, D_4 will be satisfied in stage two. But $d_2 \leq d_2^{'}$, with the same rows, $d_3 \geq d_3^{'}$. Thus, the first policy will give a solution no worse than that of the second policy.

In stage three, $D_3 - d_3$ will be satisfied until all D_3 are satisfied. But before that, $d_2 \leq d_2^{'}$ and $d_3 \geq d_3^{'}$ will hold.

Therefore, the first policy will give a solution no worse than that of the second policy.

A case will show that. We have the demands, (10,6,4) for [2,3,4] respectively. The length of each row is 14. We need to arrange them on two rows. If we take $(4, \ldots, 4, 2)$, the solution will be (4,4,4,2) and (4,3,3,3). If we take $(4, \ldots, 4, 3, 3)$, the solution will be (4,4,3,3)*2, which is better than the former one.

Similarly, we have the following lemma.

Lemma 6. *For the case $r = 1$, $(4, \ldots, 4, 3, 2), (4, \ldots, 4, 3, 3, 3)$ will be better than $(4, \ldots, 4, 1)$. Let t be the number of 4 in pattern $(4, \ldots, 4, 3, 3, 3)$. When $\min\{\lfloor \frac{D_4}{t} \rfloor, D_2\} > \lceil \frac{D_3}{3} \rceil$, we select $(4, \ldots, 4, 3, 2)$; otherwise, we select $(4, \ldots, 4, 3, 3, 3)$.*

Example:

(10,3,7) for [2,3,4], length is 13, n=4. (4,4,3,2) is better than (4,3,3,3).

(10,2,8) for [2,3,4], length is 13, n=2. (4,3,3,3) is better than (4,4,3,2).

(Proof of Theorem 4)**.** *Suppose the demands for each group type are positive and D_2, D_3, D_4, respectively. The number of seats for each row is S. We have N rows at the beginning. Let $S = 4 \times q + r, q = \lfloor \frac{S}{4} \rfloor$. Let $d_i(n), i = 2, 3, 4$ indicate the number of demand which can be satisfied by the policy when placing n rows.*

There are three stages to satisfy demands for n rows. The first one is to generate the largest pattern until we cannot get the largest pattern. The second one is to satisfy the group with largest size. The third one is to satisfy the left two types of demands.

If the first stage can cover N rows for this policy. It is easy to know that this policy give an optimal assignment.

Suppose the number of rows can be covered in the first stage is m.

If $D_2 = d_2(m)$ or $D_3 = d_3(m)$, just follow the policy of two types of group in the second stage.

If $D_2 - d_2(m) > 0$ and $D_3 - d_3(m) > 0$, we know that $D_4 - d_4(m) < q$. Thus, one row will satisfy the left demand, $D_4 - d_4(m)$ and then place group with 3 as many as possible, if the left demand $D_3 - d_3(m)$ is also satisfied, then place group with 2 as many as possible.

D_4 will be satisfied in the second stage, and the demand, $d_3(m+1)$ will also be satisfied as many as possible. Thus, the policy gives the largest $d_4(m+1), d_3(m+1)$, which corresponds to an optimal solution.

In the third stage, follow the policy of two types of group, $d_4(n), d_3(n), N \geq n > m+1$ will still be the largest.

Thus, the policy will give an optimal solution.

0.9.2 [2,3,5]

For $[2, 3, 5]$, as long as we can gaurantee that d_5, d_3 are always largest during generating patterns, we can say our policy can give an optimal solution.

1) When $r = 1$, the largest patterns are $(5, \ldots, 5, 1), (5, \ldots, 5, 3, 3)$.

For this case, $(5, \ldots, 5, 3, 3)$ is always no worse than $(5, \ldots, 5, 1)$.

After generating all $(5, \ldots, 5, 3, 3)$, either the number of groups with 5 or 3 is not enough. Continue to place groups with 5 for each row until the group with 5 is not available. Because in this way, we can gaurantee the pattern generated for each row is no worse than others and d_3, d_5 are always largest.

2) When $r = 2$, the largest pattern is $(5, \ldots, 5, 2)$.

After generating all $(5, \ldots, 5, 2)$, continue to place groups with 5 for each row until the group with 5 is not available. In this way, we will generate $(5, \ldots, 5)$, which has the same profit with $(5, \ldots, 5, 3, 3)$. When the group with 5 is not available, continue to place the group with 3 for each row. d_3, d_5 will still be largest.

3) When $r = 3$, the largest pattern is $(5, \ldots, 5, 3)$

When the group with 3 is not available, generate $(5, \ldots, 5, 2)$.

When the group with 5 is not available, continue to place the group with 3 for each row.

All in all, always placing the groups with the largest size can make sure that d_3, d_5 are the largest.

4) When $r = 4$, the largest patterns are $(5, \ldots, 5, 3)$, $(5, \ldots, 5, 2, 2)$.

$(5, \ldots, 5, 3) \succeq (5, \ldots, 5, 2, 2)$.

Placing the groups with the available largest size can make sure that d_3, d_5 are the largest.

5) When $r = 5$, the largest pattern is $(5, \ldots, 5, 5)$.

When we cannot generate $(5, \ldots, 5, 5)$, place the group with 3 for each row. Filling the groups with 3 as many as possible into each row can always give the largest profit.

0.9.3 [2,4,5]

For $[2, 4, 5]$, as long as we can gaurantee that d_5, d_4 are always largest during generating patterns, we can say our policy can give an optimal solution.

1) When $r = 1$, the largest patterns are $(5, \ldots, 5, 1), (5, \ldots, 5, 4, 2), (5, \ldots, 5, 4, 4, 4, 4)$.

For $(\underbrace{5, \ldots, 5}_{t}, 4, 2)$, when $\min\{\lfloor \frac{D_5}{t} \rfloor, D_2\} > \lceil \frac{D_4}{4} \rceil$, we select $(5, \ldots, 5, 4, 2)$; otherwise, we select $(5, \ldots, 5, 4, 4, 4, 4)$.

$(5, \ldots, 5, 4, 4, 4, 4), (5, \ldots, 5, 4, 2) \succeq (5, \ldots, 5, 1)$.

When $(5, \ldots, 5, 4, 4, 4, 4)$ is not available, the number of groups with 4 is less than 4.

When all largest patterns are not available, place the groups with 5 as many as possible.

2) When $r = 2$, the largest patterns are $(5, \ldots, 5, 2), (5, \ldots, 5, 4, 4, 4)$.

For this case, $(5, \ldots, 5, 4, 4, 4) \succeq (5, \ldots, 5, 2)$.

3) When $r = 3$, the largest pattern is $(5, \ldots, 5, 4, 4)$.

For this case, when the group with 4 is less than 2, the second largest pattern $(5, \ldots, 5, 2)$ will be no worse than $(5, \ldots, 5, 4, 2, 2)$. When the group with 5 is not enough to generate $(5, \ldots, 5, 4, 4)$, d_4, d_5 will be largest

when placing the available largest pattern. Thus, placing the available largest pattern as many as possible will still give an optimal solution.

4) When $r = 4$, the largest pattern is $(5, \ldots, 5, 4)$. Continue to generate the second largest pattern, we can ensure $d_5(n), d_4(n)$ will be the largest when given n rows.

5) When $r = 5$, the largest pattern is $(5, \ldots, 5, 5)$.

0.9.4 [3,4,5]

For $[3, 4, 5]$, as long as we can gaurantee that d_5, d_4 are always largest during generating patterns, we can say our policy can give an optimal solution.

1) When $r = 1$, the largest patterns are $(5, \ldots, 5, 1), (5, \ldots, 5, 3, 3), (5, \ldots, 5, 4, 4, 3), (5, \ldots, 5, 4, 4, 4, 4)$.

In this case, $(5, \ldots, 5, 4, 4, 4, 4)$ will have the highest priority, because other patterns will have a lower occupancy.

$(5, \ldots, 5, 4, 4, 4, 4) \succeq (5, \ldots, 5, 4, 4, 3) \succeq (5, \ldots, 5, 3, 3)$.

2) When $r = 2$, the largest patterns are $(5, \ldots, 5, 4, 3), (5, \ldots, 5, 4, 4, 4)$.

For $(\underbrace{5, \ldots, 5}_{t}, 4, 3)$, when $\min\{\lfloor \frac{D_5}{t} \rfloor, D_3\} > \lceil \frac{D_4}{3} \rceil$, we select $(5, \ldots, 5, 4, 3)$; otherwise, we select $(5, \ldots, 5, 4, 4, 4)$.

Similar to the case, $[2, 3, 4]$.

3) When $r = 3$, the largest patterns are $(5, \ldots, 5, 3), (5, \ldots, 5, 4, 4)$

For this case, $(5, \ldots, 5, 4, 4) \succeq (5, \ldots, 5, 3)$. Because these two patterns are independent.

4) When $r = 4$, the largest pattern is $(5, \ldots, 5, 4)$. Continue to generate the second largest pattern, we can ensure $d_5(n), d_4(n)$ will be the largest when given n rows.

5) When $r = 5$, the largest pattern is $(5, \ldots, 5, 5)$. Similar to 4).

0.10 [2,3,4,5]

Example: $[2, 3, 4, 5] * (8, 8, 3, 7)_D, S = 16, n = 4$

The same policy will give the demand $(4, 3, 3, 7)_{D_p}$.

The optimal solution has the demand $(0, 7, 2, 7)_{D_o}$.

Theorem 5. *Let r be the remainder of S divided by 5, i.e., $S = 5 \times q + r, q = \lfloor \frac{S}{5} \rfloor$. The most preferred pattern is $(5, \ldots, 5, \underbrace{4, \ldots, 4}_{5-r}), r = 2, 3, 4$. When $r = 0$, The most preferred pattern is $(5, \ldots, 5)$. When $r = 1$, The most preferred pattern is $(5, \ldots, 5, 4, 2)$ or $(5, \ldots, 5, 4, 4, 4, 4)$, which depends on the demands.*

We can have the following 5 situations according to the remainder.

1) When $r = 1$, the largest patterns are $(5, \ldots, 5, 1), (5, \ldots, 5, 4, 2), (5, \ldots, 5, 3, 3), (5, \ldots, 5, 4, 4, 3), (5, \ldots, 5, 4, 4, 4, 4)$.

We have $(5, \ldots, 5, 4, 4, 4, 4) \succeq (5, \ldots, 5, 4, 4, 3) \succeq (5, \ldots, 5, 3, 3), (5, \ldots, 5, 4, 2) \succeq (5, \ldots, 5, 1)$.

Notice that $(5, \ldots, 5, 4, 4, 4, 4) + (5, \ldots, 5, 3, 3) = 2 * (5, \ldots, 5, 4, 4, 3)$, and $(5, \ldots, 5, 4, 4, 3)$ uses less the group with 3 than $(5, \ldots, 5, 3, 3)$, thus $(5, \ldots, 5, 4, 4, 3) \succeq (5, \ldots, 5, 3, 3)$.

For $(\underbrace{5, \ldots, 5}_{t}, 4, 2)$, when $\min\{\lfloor \frac{D_5}{t} \rfloor, D_2\} > \lceil \frac{D_4}{4} \rceil$, we select $(5, \ldots, 5, 4, 2)$; otherwise, we select $(5, \ldots, 5, 4, 4, 4, 4)$.

$(5,\ldots,5,1)$ will have the lowest priority because one empty seat is wasted.

2) When $r=2$, the largest patterns are $(5,\ldots,5,2),(5,\ldots,5,4,3),(5,\ldots,5,4,4,4)$.

For this case, $(5,\ldots,5,4,4,4),(5,\ldots,5,4,3)\succeq(5,\ldots,5,2)$.

For $(\underbrace{5,\ldots,5}_{t},4,3)$, when $\min\{\lfloor\frac{D_5}{t}\rfloor,D_3\}>\lceil\frac{D_4}{3}\rceil$, we select $(5,\ldots,5,4,3)$; otherwise, we select $(5,\ldots,5,4,4,4)$.

3) When $r=3$, the largest patterns are $(5,\ldots,5,3),(5,\ldots,5,4,4)$

For this case, $(5,\ldots,5,4,4)\succeq(5,\ldots,5,3)$.

4) When $r=4$, the largest pattern is $(5,\ldots,5,4)$. Continue to generate the second largest pattern, we can ensure $d_5(n),d_4(n)$ will be the largest when given n rows.

5) When $r=5$, the largest pattern is $(5,\ldots,5,5)$. Similar to 4).

Now we need to consider the situation where we cannot generate the largest patterns.

If all types of groups remain, i.e., $D_i-d_i>0,i=21,3,4,5$, we need to place the groups according to their sizes. Then D_5 will be satisfied and the problem is reduced to the case with $[2,3,4]$.

If $d_5=D_5$ initially, the problem will be reduced to the case with $[2,3,4]$.

If the group with the size of 5 remains and the demand of at least one other type of group is satisfied, i.e., we have $[2,3,5]$ or $[2,4,5]$ or $[3,4,5]$.

With the above conclusions, we can follow the policies to obtain an optimal solution.

0.10.1 Column-and-cut generation

Notice that T is a subset of the set of all possible cutting patterns, thus we need to find more cutting patterns by the column generation.

The linear relaxation of problem can be written as:

$$\begin{aligned}\max\quad & c'\mathbf{x}+f'\mathbf{y}\\ \text{s.t.}\quad & \mathbf{T}\mathbf{x}+\mathbf{V}_\omega\mathbf{y}_\omega=\mathbf{D}_\omega,\omega\in\Omega\\ & \mathbf{1}\mathbf{x}=N\\ & \mathbf{x},\mathbf{y}\geq\mathbf{0}\end{aligned}\tag{4}$$

Obtain the dual of the problem (4), the constraints are represented in the matrix form:

$$\begin{bmatrix}\mathbf{T} & \mathbf{T} & \cdots & \mathbf{T} & \mathbf{1}\\ \mathbf{W} & & & & \\ \mathbf{I} & & & & \\ & \mathbf{W} & \ddots & & \\ & \mathbf{I} & & & \\ & & \ddots & \mathbf{W} & \\ & & & \mathbf{I} & \end{bmatrix},$$

Let $(\pi = [\pi_{11}, \pi_{21}, \cdots, \pi_{m1}, \cdots, \pi_{1\omega}, \pi_{2\omega}, \cdots, \pi_{m\omega}], \lambda)$ be optimal dual variables associated with the scenario constraints and row number constraint.

Then the dual problem is:

$$\min \quad \sum_{\omega} D_\omega \pi_\omega + N\lambda \tag{5}$$

$$\text{s.t.} \quad \sum_{\omega} \mathbf{T}\pi_\omega + \mathbf{1}\lambda \geq (\mathbf{s}-1)\mathbf{T} \tag{6}$$

$$\mathbf{W}\pi_\omega \geq -\bar{\mathbf{s}}\, p_\omega,\ \omega \in \Omega \tag{7}$$

$$\mathbf{I}\pi_\omega \geq \mathbf{0},\ \omega \in \Omega \tag{8}$$

$\mathbf{s}$ is the vector of group sizes, s_i. $\bar{\mathbf{s}}_i = s_i - s_{i-1}$ with $s_0 = 1$.

From the constraints (7) and (8), we have

$$\begin{cases} 0 \leq \pi_{\omega,m} \leq (s_m - s_{m-1})p_\omega, \\ 0 \leq \pi_{\omega,m-1} \leq (s_{m-1} - s_{m-2})p_\omega + \pi_{\omega,m}, \\ \quad \vdots \quad \vdots \\ 0 \leq \pi_{\omega,1} \leq (s_1 - 1)p_\omega + \pi_{\omega,2}, \end{cases} \tag{9}$$

Similarly, according to the complementary slackness, when $y^+_{\omega,m} \neq 0$, $\pi_{\omega,m} = (s_m - s_{m-1})p_\omega$; when $y^-_{\omega,m} \neq 0$, $\pi_{\omega,m} = 0$.

When both $y^+_{\omega,i}, y^-_{\omega,i}$ are equal to 0, to minimize the objective function, $\pi_{\omega,i}$ should also be 0.

Therefore, when we obtain $\mathbf{x}, \mathbf{y}$ of the problem (4), we can immediately obtain π. And λ can be obtained by the strong duality.

The pricing problem will be

$$\begin{aligned} \max \quad & \sum_{i=1}^{m} \left[(s_i - 1) - \sum_{\omega} \pi_{i\omega} \right] y_i - \lambda \\ \text{s.t.} \quad & \sum_{i=1}^{m} s_i y_i \leq L \\ & y_i \geq 0, \text{integer} \quad \text{for } i = 1, \ldots, m, \end{aligned}$$

where y_i indicate the number of times group type i is placed in the new pattern.

By solving the pricing problem, we can obtain a new cutting pattern, then add it to T. Then solve the problem to obtain (x^*, z^*) and continue the Benders Decomposition.

0.11 Results of the column-and-cut method

The size of each row is 18.

# of scenarios number	# of group types	# of iterations	# of cutting patterns	time(s)
100	4	2	1	0.5
200	4	2	1	1
500	4	2	1	3
1000	4	2	1	5
2000	4	2	1	10
5000	4	2	1	24

Group types are $[2, 3, 4, 5]$.

The scenarios are generated from $[5, 15]$.

Given the number of rows is 3. The initial cutting pattern are gererated in the same way as before.

We can see that the number of cutting patterns and iterations is related with the size of row and group types.

The running time increases with the number of scenarios linearly.

Now Fix the number of scenarios at 500.

size of row	# of group types	# of iterations	# of cutting patterns	time(s)
10	4	1	0	0.8
20	4	3	0	3
30	4	4	2	6.3
40	4	5	1	9.6
50	4	7	3	16.5
90	4	5	1	9
100	4	1	0	1.29
200	4	1	0	1.26
500	4	1	0	1.25

One interesting thing is that the running time are higher when the size of row is 50. Then we fix the size of row at 50, just change the number of group types. (Starting from 2 and group types are consecutive integers starting from 2)

# of group types	size of row	# of iterations	# of cutting patterns	time(s)
2	50	1	1	1.26
3	50	4	2	6.3
4	50	7	3	16.1
5	50	5	5	9.6
6	50	5	4	9.7
7	50	4	4	6.5
8	50	4	3	6.2
9	50	3	1	4
10	50	3	1	4.2

0.11.1 Max regret

Let D be the polyhedron including all feasible demands with N rows, i.e., $D = \{d = (d_1, d_2, \ldots, d_n) : Tx = d, \sum_k x_k \leq N, x_k \geq 0, \text{integer}\}$.

We denote by $z^s(d)$ the value given by a demand $d \in D$ for scenario s. According to our assumption, the larger seats can be utilized by the smaller group. Then, we have $z^s(d) = \sum_{i=1}^{m-1}(s_i - 1)(Tx + y_{i+1}^+ - y_i^+) + (s_m -$

$1)(Tx - y_m^+)$.

When $s_i = i+1$, $z^s(d) = \sum_{i=1}^m iTx - \sum_{i=1}^m y_i^+$, $y_m^+ = (d_m - d_m^s)^+$, $y_i^+ = (d_i - d_i^s + y_{i+1}^+)^+, i = 1, \ldots, m-1$.

Let z_*^s be the optimal solution value under scenario s, i.e., the solution value of the following problem.

$$\begin{aligned} z_*^s = \max \quad & \sum_{k=1}^{K}\sum_{i=1}^{m}(s_i - 1)t_i^k x_k \\ \text{s.t.} \quad & \sum_{k=1}^{K} x_k \le N \\ & \sum_{k=1}^{K} t_i^k x_k \le d_i^s, \quad i = 1, \ldots, m \\ & x_k \ge 0, \text{integer} \quad k = 1, \ldots, K \end{aligned}$$

The regret associated with demand d, for scenario s, is then $r^s(d) = z_*^s - z^s(d)$.

Let S denote the set of all possible scenarios s, then $D \subset S$.

Definition 1. *We say d^{s_0} associated with s is the minimum demand if it satisfies $z_*^s = z_*^{s_0}$, $d^s \notin D$.*

Thus, $d_i^{s_0} \le d_i^s$.

The min-max regret problem is to find a feasible demand d such that the maximum regret is minimized.

$$\begin{aligned} \min_d \max_{s\in S} \quad & r^s(d) \\ \text{s.t.} \quad & \sum_{k=1}^{K} x_k \le N \\ & \sum_{k=1}^{K} t_i^k x_k \le d_i, \quad i = 1, \ldots, m \\ & x_k \ge 0, \text{integer} \quad k = 1, \ldots, K \end{aligned}$$

$\min_d \max_{s\in S} r^s(d) = \min_d \max_{s\in S}[(\max_{d^s\in S_0} \sum id^s) - \sum_{i=1}^m (id_i - y_i^+)]$

It is easy to find that the min-max regret is 0 when we take $d = d^s \in D$.

Thus, we only consider the maximum regret given a demand d^0.

Then the maximum regret is $\max_s \sum_i^m y_i^+ + \sum_i id_i^s - \sum_i id_i^0, y_m^+ = (d_m^0 - d_m^s)^+$, $y_i^+ = (d_i^0 - d_i^s + y_{i+1}^+)^+, i = 1, \ldots, m-1$.

Theorem 6. *The regret reaches its maximum if and only if d^s is at the vertex of D.*

When $s \notin S_0$, the increase of any element of d^{s_0} will keep the value of $\sum_i id_i^s$ unchanged and the value of $\max_s \sum_i y_i^+$ non-increasing.

When $s \in S_0$, if s is not at the vertex of D, then the increase of d^s will make the value of $\sum_i id_i^s + \max_s \sum_i y_i^+$ non-decreasing until d^s reaches the vertex.

Therefore, the regret reaches its maximum if and only if d is at the vertex of D.

Now, we consider to solve the following problem:

$$\begin{aligned}
&\max_s \sum_i y_i^+ + \sum_i i d_i^s - \sum_i i d_i^0 \\
&\text{s.t.} \quad d^s \in D = \{d = (d_1, d_2, \ldots, d_n) : Tx = d, \sum_k x_k \le N, x_k \ge 0, \text{integer}\} \\
&\qquad y_i^+ = (d_i^0 - d_i^s + y_{i+1}^+)^+, i = 1, \ldots, m-1 \\
&\qquad y_m^+ = (d_m^0 - d_m^s)^+.
\end{aligned} \tag{10}$$

Notice that every d^s can be expressed as $d^s = Tx$, then let $w_m = (d^0 - Tx)_m^+$, $w_i = ((d^0 - Tx)_i + y_{i+1}^+)^+, i = 1, \ldots, m-1$.

Then the problem (10) can be reformulated as

$$\begin{aligned}
&\max_{x,w} (\sum_i i(Tx)_i - \sum_i i d_i^0 + \min \sum_i w_i) \\
&\text{s.t.} \quad w_m \ge (d^0 - Tx)_m \\
&\qquad w_i \ge (d^0 - Tx)_{m-1} + w_{i+1}, i = 1, \ldots, m-1 \\
&\qquad w_i \ge 0, i = 1, \ldots, m \\
&\qquad \sum_k x_k \le N \\
&\qquad x_k \ge 0, \text{integer}.
\end{aligned} \tag{11}$$

We can introduce a new variable z to satisfy $z \le z(x) = \{\min \sum_i w_i : w_m \ge (d^0 - Tx)_m, w_i \ge (d^0 - Tx)_{m-1} + w_{i+1}, i = 1, \ldots, m-1, w_i \ge 0\}$. Then, we can obtain the dual:

$$\{\max \alpha'(d^0 - Tx) : 0 \le \alpha_1 \le 1, 0 \le \alpha_i \le \alpha_{i-1} + 1, i = 2, \ldots, m\}$$

Let $\alpha_0 = 0$, an optimal solution to the dual is given by

$$\begin{aligned}
&\alpha_i = 0, i = 1, \ldots, m \text{ if } d^0 - Tx < 0, \\
&\alpha_i = \alpha_{i-1} + 1, i = 1, \ldots, m \text{ if } d^0 - Tx \ge 0
\end{aligned}$$

The master problem will be:

$$\begin{aligned}
&\max_{x,z} \sum_i i(Tx)_i - \sum_i i d_i^0 + z \\
&\text{s.t.} \quad z \le \alpha'(d^0 - Tx) \\
&\qquad \sum_k x_k \le N \\
&\qquad x_k \ge 0, \text{integer}.
\end{aligned} \tag{12}$$

Notice that this problem is just one scenario version of problem , thus, we can use column-and-cut to solve it directly.

Algorithm 3 The column-and-cut algorithm

Step 1. Solve LP form of (12) with $\alpha^0 = \mathbf{0}$. Then, obtain the solution (x_0, z^0) and dual solution (π, λ).

Step 2. Set the upper bound $UB = c'x_0 + z^0 - \sum_i id_i^0$

Step 3. For x_0, we can obtain α^1 and $z^{(0)}$, set the lower bound $LB = c'x_0 + z^{(0)} - \sum_i id_i^0$

Step 4. If $(\alpha^1)'(\mathbf{d}^0 - \mathbf{T}x_0) < z^{(0)}$, add one new constraint, $(\alpha^1)'(\mathbf{d}^0 - \mathbf{T}x) \geq z$, to LP (12).

Step 5. Solve the pricing problem with (π, λ), add a new cutting pattern and update T.

Step 6. Solve the updated LP (12), obtain a new solution (x_1, z^1) and update UB.

Step 7. Repeat step 3 until $UB - LB < \epsilon$.(Or in our case, UB converges.)

0.11.2 Two-stage Robust optimization

We can find that the worst scenario is d^s when all elements d_i^s take their minimum values.

0.11.3 Details

Sample Average Approximation(SAA): generate the scenarios.

How to deal with the continuous situation?

Multi-cut, Single-Cut

How to know the valid cuts:

If we can find several solutions from theorem and these solutions do not dominate each other, we can add them into the cuts pool together during one iteration.

Generate enough cutting patterns at the beginning or use the column generation.

Disaggregated, Aggregated form

Benders Decomposition: relaxed master problem (x_k, z_ω), subproblems: add cuts

0.12 Occupancy by the greedy way

Recall that for any pattern k, the occupancy is $\frac{S-l(k)}{S-1}$ and the largest occupancy is $O^*(S) = \frac{S-l(k)}{S-1}, k \in I_1$, where $l(k)$ is the loss for pattern k.

For $[2, 3, 4]$, $S = 4q + r$, the largest loss,l^*, is $q + \text{sign}(r)$.

If $S = 20, q = 5, r = 0$, $l^* = 5$, $O^*(S) = \frac{15}{20} \approx 80\%$

If we only use 2 to fill one row, $O(S) = \frac{10}{19}$.

If we have six seats, 3 groups with size of 2 or 2 groups with size of 3.

The occupancy will increse from $\frac{3}{6}, 50\%$ to $\frac{4}{6}, 66.7\%$. On average, one more people for six seats if we allow groups with size of 3 join.

If we have 12 seats, 4 groups with size of 3 or 3 groups with size of 4.

The occupancy will increse from $\frac{8}{12}, 66.7\%$ to $\frac{9}{12}, 75\%$. On average, one more people for 12 seats if we allow groups with size of 4 join.

Suppose the demand, d_2, d_3, d_4 has a close ratio, for example, one to one to one, then according to the greedy way, we have $m_1 = \lceil \frac{d_4}{q} \rceil$ rows with the largest pattern at first. The corresponding occupancy is $\frac{S-q-\text{sign}(r)}{S-1}$.

The left demand, $d_4^{'}$ is $d_4 - m_1 q$, taking $4(d_4 - m_1 q)$ seats and having the occupancy of $\frac{3}{4}$.

The number of left seats is $S_1 = S - 4(d_4 - m_1 q)$. For S_1, $S_1 = 3q_1 + r_1$, the largest occupancy is $\frac{S_1 - q_1 - \text{sign}(r_1)}{S_1 - 1}$.

The left demand for group with size of 3 is $d_3^{'} = d_3 - q_1$. Continue...

0.12.1 Dynamic Programming

Besides, DP may be a possible method.

We can use DP to solve this problem for the fixed supply if we know the arrival rates.

To ensure optimality, we need to follow some rules:

1. When there are enough supplies, we will accept them directly.

2. The demand can be accepted by a larger-size supply when there is not enough corresponding-size supply.

3. The demand can only be satisfied by one larger-size supply.

Basic Bellman equation for network capacity: $V_t(\mathbf{X}) = E[\max_{\mathbf{u}}\{R_t\mathbf{u} + V_{t+1}(\mathbf{X} - A\mathbf{u})\}]$

$\mathbf{X}$ is the remaining capacity for each group type.

Boundary condition: $V_t(\mathbf{X} = 0) = 0, V_{T+1}(\mathbf{X}) = 0$.

Let $e_i = [\underbrace{0, \ldots, 0}_{i-1}, 1, 0, \ldots, 0]$.

When $\mathbf{X} > 0$(all elements is above 0), $\mathbf{u} = e_i, i \in I$. The value function $V_t(\mathbf{X}) = E[\max\{\mathbf{us} - 1 + V_{t+1}(\mathbf{X} - \mathbf{u})\}]$.

When some element of $\mathbf{X}$ is 0, say x_i, then here comes a group i, if we accept it by supply j, $\mathbf{u} = [-1, \underbrace{0, \ldots, 0}_{i}, 1, 0, \ldots, 0], [0, -1, \underbrace{0, \ldots, 0}_{i}, 1, 0, \ldots, 0], \cdots$. Here, $\mathbf{u}$ indicates all the decisions, i.e, group type i is served by supply j, $i < j$. Then the value function $V_t(\mathbf{X}) = E[\max\{\mathbf{us} - 1 + V_{t+1}(\mathbf{X} - \mathbf{u})\}]$, $\mathbf{us} = s_i$.

Example: $[0, 6, 8] \to [1, 6, 7]$ when 1 is served by 3.

Algorithm 4 Backward method to make decisions under fixed supply

Step 1. Obtain a supply, $\mathbf{X}^0 = [x_1, \ldots, x_J]$, from the stochastic programming.

Step 2. For the arrival group type i, if $x_i > 0$, accept it. $V_t(\mathbf{X}) = s_i - 1 + V_{t+1}(\mathbf{X} - e_i)$. If $x_i = 0$, $V_t(\mathbf{X}) = E[\max_{\mathbf{u}}\{\mathbf{us} - 1 + V_{t+1}(\mathbf{X} - \mathbf{u})\}]$.

Step 3. Set the boundary conditions: $V_t(\mathbf{X} = \mathbf{0}) = 0, V_{T+1}(\mathbf{X}) = 0$.

Step 4. Calculate $V_1(\mathbf{X}^0)$.

Complexity: $O(T \cdot |x_1| \cdot |x_2| \cdot \ldots \cdot |x_J|)$.

According to the rule 1, we only need to obtain $V_1(0, x_2, \cdots, x_J)$, $V_1(x_1, 0, \cdots, x_J), \ldots, V_1(x_1, \cdots, x_{J-2}, 0, x_J)$.

Use the same example to illustrate:

$V_1([3,3]) = p_2(1 + V_2([2,3])) + p_3(2 + V_2([3,2]))$

$V_1([3,3]) = 3 + V_4([0,3])$ when the first three arrivals are groups with 2.

$V_4([0,3]) = p_2 \max\{1 + V_5(0,2), V_5(0,3)\} + p_3(2 + V_5(0,2))$

.....

$V_9([0,2]) = p_2 \cdot (1 + V_{10}(0,1)) + p_3 \cdot (2 + V_{10}(0,1))$

$V_9([2,0]) = p_2 \cdot (1 + V_{10}(1,0))$

$V_{10}([0,1]) = p_2 \cdot 1 + p_3 \cdot 2$

$V_{10}([1,0]) = p_2 \cdot 1$

Notice that not all possible demands, $d \in D$, can be developed from $\mathbf{X}^0$. Thus, the stochatic information is still useful.

Continuous time

Richard and Yi solved the finite horizon stochastic knapsack problem. [9]

Group type, i, has an integer weight $s_i + 1$, a value s_i, and an arrival rate $\lambda_i(t)$.

Consider arrivals as a nonhomogeneous Poisson process over a continuous time horizon $[0, T]$ to a knapsack with capacity S.

Let $f(y,t)$ be the maximum expected sum of the benefits of the accepted objects given that y capacity and t seconds remaining.

$$\begin{aligned} f(y,t) &= \int_0^t \lambda(s) e^{-\int_s^t \lambda(\tau)\mathrm{d}\tau} \cdot \sum_{i=1}^{N} p_i(s) \max\left\{s_i + f\left(y - s_i - 1, s\right), f(y,s)\right\} \mathrm{d}s \\ &= \int_0^t \lambda(s) e^{-(\Lambda(t) - \Lambda(s))} \cdot \sum_{i=1}^{N} p_i(s) \max\left\{s_i + f\left(y - s_i - 1, s\right), f(y,s)\right\} \mathrm{d}s, \end{aligned}$$

By differentiation of the above, we obtain:

$$
\begin{aligned}
f'(y,t) =& \lambda(t) \sum_{i=1}^{N} p_i(t) \max\{s_i + f\left(y - s_i - 1, t\right), f(y,t)\} - \lambda(t) f(y,t) \\
=& \lambda(t) \sum_{i=1}^{N} p_i(t) \max\{0, s_i + f\left(y - s_i - 1, t\right) - f(y,t)\} \\
=& \sum_{i=1}^{N} \lambda_i(t) \max\{0, s_i + f\left(y - s_i - 1, t\right) - f(y,t)\}
\end{aligned}
$$

If the $\lambda_i(t) > 0$ for all i and t, then $f(y,t)$ is strictly monotone increasing in t for $y \geq \min_i s_i + 1$.

Need to solve a differential equation for every possible capacity.

In our problem, $f(1,t) = 0$.

$f'(2,t) = \lambda_1[1 - f(2,t)] \Rightarrow f(2,t) = 1 - e^{-\lambda t}$

$f'(3,t) = \lambda_1[1 - f(3,t)] + \lambda_2[2 - f(3,t)] \Rightarrow f(3,t) = \frac{\lambda_1+2\lambda_2}{\lambda_1+\lambda_2}[1 - e^{-(\lambda_1+\lambda_2)t}]$

$f'(4,t) = \lambda_1[1 + f(2,t) - f(4,t)] + \lambda_2[2 - f(4,t)] \Rightarrow f(4,t) = 2 - \frac{\lambda_1}{\lambda_2}e^{-\lambda_1 t} - (2 - \frac{\lambda_1}{\lambda_2})e^{-(\lambda_1+\lambda_2)t}$

$\delta(1,3,t) = 1 - f(3,t) = -\frac{\lambda_2}{\lambda_1+\lambda_2} + \frac{\lambda_1+2\lambda_2}{\lambda_1+\lambda_2}e^{-(\lambda_1+\lambda_2)t}$

$\delta(1,4,t) = 1 + f(2,t) - f(4,t) = \frac{\lambda_2-\lambda_1}{\lambda_2}e^{-\lambda_1 t} + (2 - \frac{\lambda_1}{\lambda_2})e^{-(\lambda_1+\lambda_2)t}$

Critical t', when $t < t'$, accept 1. $\lambda_1 \uparrow, t \uparrow$.

Divide $[0,T]$ into N subintervals to approximate $f(y,t)$:

$f(y,t_{n+1}) = f(y,t_n) + \Delta f(y,t_n)$ with $\Delta f(y,t_n) = f^{'}(y,t_n)\Delta t$, $\Delta t = t_{n+1} - t_n$.

Then use $f'(y,t_n) = \sum_k \lambda_k \max\{b_k + f(y - w_k, t_n) - f(y,t_n), 0\}$, $f(y,t_{n+1}) = f(y,t_n) + (t_{n+1} - tn)f'(y,t_n)$ to update.

1 Literature Review

[3] gives a method to check IRU and IRD property and give the conclusion that IRU holds for a given A if and only if IRU holds for C(A,w) for every fractional solution.

[4] gives a general decomposition property. We shall say that polyhedron $P(A,d,e)$ has the real decomposition property (RDP) if for any positive real T and any real $x \in TP(A,d,e)$, there exists an integer r, positive coefficients $\lambda_1, \ldots, \lambda_r$ and vectors $s^1, \ldots, s^r \in P(A,d,e)$ such that $x = \lambda_1 s^1 + \cdots + \lambda_r s^r$, $T = \lambda_1 + \cdots + \lambda_r$.

An important property: $Q(A,b)$ has the RDP iff is integral.

[2] gives that IRD holds for M(A matrix) if and only if P satisfies the integral decomposition property.

[7] demonstrates that CSP has the IRU property if and only if P, the corresponding knapsack polyhedron, has the integral decomposition property.

Cutting stock problems of the form $(a_1, a_2; b)$ have the IRU property.

[10] compares two branch-and-price approaches for the cutting stock problem. Both algorithms employ column generation for solving LP relaxations at each node of a branch-and-bound tree to obtain optimal integer solutions.

[11] transforms the integer knapsack subproblems into 0-1 knapsack problems and use branch-and-bound procedure to solve them.

[6] sloves CSP based on enumerating the possible cutting patterns.

[5] gives the well-known initial compact formulation.

[12] gives the branch and column generation for general IP.

[1] uses branch-and-price to solve huge integer programs.

[8] carries out the computational experiments with a lot of randomly generated instances of the one-dimensional cutting stock problem.

And shows that for all instances an integer solution fulfills the MIRUP(Modified Integer Round-Up Property). Moreover, in most cases the optimal value equals the round-up optimal value of the corresponding LP relaxation.

References

[1] Cynthia Barnhart, Ellis L Johnson, George L Nemhauser, Martin WP Savelsbergh, and Pamela H Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations research*, 46(3):316–329, 1998.

[2] Stephen Baum and LE Trotter, Jr. Integer rounding for polymatroid and branching optimization problems. *SIAM Journal on Algebraic Discrete Methods*, 2(4):416–425, 1981.

[3] Stephen Baum and Leslie E Trotter. Finite checkability for integer rounding properties in combinatorial programming problems. *Mathematical Programming*, 22(1):141–147, 1982.

[4] Dominique de Werra. A decomposition property of polyhedra. *Mathematical programming*, 30(3):261–266, 1984.

[5] Paul C Gilmore and Ralph E Gomory. A linear programming approach to the cutting-stock problem. *Operations research*, 9(6):849–859, 1961.

[6] Constantine Goulimis. Optimal solutions for the cutting stock problem. *European Journal of Operational Research*, 44(2):197–208, 1990.

[7] Odile Marcotte. The cutting stock problem and integer rounding. *Mathematical Programming*, 33(1):82–92, 1985.

[8] Guntram Scheithauer and Johannes Terno. The modified integer round-up property of the one-dimensional cutting stock problem. *European Journal of Operational Research*, 84(3):562–571, 1995.

[9] Richard Van Slyke and Yi Young. Finite horizon stochastic knapsacks with applications to yield management. *Operations Research*, 48(1):155–172, 2000.

[10] Pamela H Vance. Branch-and-price algorithms for the one-dimensional cutting stock problem. *Computational optimization and applications*, 9(3):211–228, 1998.

[11] François Vanderbeck. Computational study of a column generation algorithm for bin packing and cutting stock problems. *Mathematical Programming*, 86(3):565–594, 1999.

[12] François Vanderbeck and Laurence A Wolsey. An exact algorithm for ip column generation. *Operations research letters*, 19(4):151–159, 1996.