Seat assignment with the social distancing

Dis·*count*

May 4, 2022

Abstract

Keywords: Social distancing, Cutting stock problem, Combinatorial optimization, COVID-19.

1 Introduction

Social distancing, a physical way to contain the spread of an infectious disease, has been broadly recognized and practiced. As a result, extensive research has emerged on social distancing with respect to its effectiveness and impact. What lags behind is operational guidance for implementation, an issue particularly critical to social distance measures of which the implementation involves operations details. One typical example is how to make social distancing ensured seating plans.

We will start by considering the seat plan with a given set of seats. In a pandemic, government may issue a minimum physical distance between people, which must be implemented in the seating plan. The problem is further complicated by the existence of groups of guests who will be seated together. To achieve such a goal, we develop a mechanism for seat planning, which includes a model to characterize the riskiness of a seating plan, and a solution approach to make the tradeoff between seat utilization rate and the associated risk of infection.

In this paper, we are interested in finding a way to implement a seating plan with social distancing constraint, instead of solving the IP model directly. After knowing the group portfolio structure we can obtain the minimum number of seat rows inspired by the cutting stock problem. And we can formulate the corresponding model with a given number of rows to maximize the capacity.

Our main contributions are summarized as follows.

First, this paper is the first attempt to introduce a brand new concept and consider ... Most literature on social distancing in seat assignments, highlight the importance of social distance in controlling the spread of the virus and focus the model too much, there is not much work on the operational significance behind the social distance [2] [6]. Recently, some scholars studied the effects of social distance on health and economics, mainly in aircrafts [11] [7]. Especially, Our study provides another perspective to help the authority adopt a mechanism setting seat assignments to control the spread of the virus.

Second, we establish the model to analyze the effects of ..., and we ... However, rather than .., ... Then, We develop the theorems to guide us to design effective algorithms to solve this problem. Next we consider the

dynamic form.

Third, as a generalization, we apply this method on ... and show the similar conclusions. With this new .., we illustrate how to assign the seats by the govenment/stakeholder to balance health and economic issues. In addition, we also provide managerial guidance for the government on how to publish the related policy to make the tradeoff between economic maintenance and risk management.

The rest of this paper is structured as follows. The following section reviews relevant literature. We describe the motivating problem in Section 3. In Section 4, we establish the model and analyze its properties. Section 5 demonstrates the dynamic form and its property. Section 6 gives the results. The conclusions are shown in Section 7.

2 Literature Review

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

[5] 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 $s^1, \ldots, s^r \in P(A, d, e)Z^n$ 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.

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

[10] 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.

[13] 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.

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

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

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

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

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

[12] 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.

3 Problem Description

3.1 Basic Concept

At first, we will introduce some preliminary knowledge about our problem as follows.

3.2 Inspired Example

4 Deterministic Model

4.1 Obtain Minimum Number of Rows to Cover Demand

We are given a demand, for example, $(d_1, d_2, d_3, d_4, d_5, d_6) = (3, 5, 7, 0, 10, 6)$, where d_i indicates the number of group containing i people. Suppose each group has to leave a seat to maintain social distancing with the adjacent groups. Regard the groups as items in the CSP, and rows as stocks to be cut. With considering the social distancing, the size of each group should be treated as the actual size plus one. Then each row of seats should also add a dummy(virtual) seat for the same reason, and the number of all seats in each row is called the length of the row.

To find the minimum number of rows to satisfy the demand, we can formulate this problem as a cutting stock problem form and use the column generation method to obtain an approximate solution.

Similar to the concept of pattern in the CSP, let the k-th placing pattern of a line of seats with length S into some of the m group types be denoted as a vector $(t_1^k, t_2^k, \ldots, t_m^k)$. Here, t_i^k represents the number of times group type i is placed in the k-th placing pattern. For a pattern $(t_1^k, t_2^k, \ldots, t_m^k)$ to be feasible, it must satisfy: $\sum_{i=1}^{m} t_i^k s_i \leq S$, where s_i is the size of group type i. Denote by K the current number of placing patterns.

This problem is to decide how to place a total number of group type i at least g_i times, from all the available placing patterns, so that the total number of rows of seats used is minimized.

Immediately we have the master problem:

$$\begin{aligned} \min \quad & \sum_{k=1}^{K} x_k \\ \text{s.t.} \quad & \sum_{k=1}^{K} t_i^k x_k \geq d_i \quad \text{for } i = 1, \ldots, m \\ & x_k \geq 0, \text{integer} \quad \text{for } k = 1, \ldots, K. \end{aligned}$$

If K includes all possible patterns, we can obtain the optimal solution by solving the corresponding IP. But it is clear that the patterns will be numerous, considering all possible patterns will be time-consuming.

Thus, we need to consider the linear relaxation of the master problem, and the optimal dual variable vector λ. Using λ as the value assigned to each group type i, the next problem is to find a feasible pattern $(y_1, y_2, \ldots, y_m)$ that maximizes the product of λ and y.

Then the corresponding sub-problem is:

$$\begin{aligned} \max \quad & \sum_{i=1}^{m} \lambda_i y_i \\ \text{s.t.} \quad & \sum_{i=1}^{m} w_i y_i \leq S \\ & y_i \geq 0, \text{integer} \quad \text{for } i = 1, \ldots, m. \end{aligned}$$

This is a knapsack problem, its solution will be used as an additional pattern in the master problem. We should continue to add new pattern until all reduced costs are nonnegative. Then we have an optimal solution to the original linear programming problem.

But note that column generation method cannot gaurantee an optimal solution. If we want to reach the optimal solution, we should tackle with the integer formulation.

$$\begin{aligned}
&\min \sum_{k=1}^{K} y_k \\
&\sum_{k=1}^{K} x_{ik} \geq d_i \quad i = 1, \ldots, n \\
&\sum_{i=1}^{n} a_i x_{ik} \leq S y_k \quad k = 1, \ldots, K \\
&y_k \in \{0, 1\} \quad k = 1, \ldots, K \\
&x_{ik} \geq 0 \text{ and integer } i = 1, \ldots, n; k = 1, \ldots, K
\end{aligned} \tag{1}$$

$y_k = 1$ if line k is used and 0 otherwise, x_{ik} is the number of times group i is placed in row k, and K is the upper bound on the number of the rows needed.

4.2 Provide The Maximal Supply When Given Rows

Let us review this problem in another way. In most cases where the number of rows is fixed, we hope to accommodate as many as people possible. That is, we should minimize the space loss.

Then minimizing $NS - \sum_{i=1}^{m} r_i(s_i - 1)$ equals to maximize $\sum_{i=1}^{m} r_i(s_i - 1)$ and maximze $\sum_{i=1}^{m}(g_i - d_i)(s_i - 1)$.

Notice that $\sum_{k=1}^{K} t_i^k x_k + d_i = g_i$, by substituting this equation we can obtain the objective function of the following master problem.

$$\begin{aligned}
\max \quad & \sum_{k=1}^{K}(\sum_{i=1}^{m}(s_i - 1)t_i^k)x_k \\
\text{s.t.} \quad & \sum_{k=1}^{K} x_k \leq N && (2) \\
& \sum_{k=1}^{K} t_i^k x_k \leq g_i, \quad i = 1, \ldots, m && (3) \\
& x_k \geq 0, \quad k = 1, \ldots, K
\end{aligned}$$

Similarly, we consider the linear relaxation of the master problem and the optimal dual variable vector λ, μ. Using λ as the value assigned to the first constraint (2) and μ to the second constraints (3). This master problem is to find a feasible pattern $(t_1^{k_0}, t_2^{k_0}, \ldots, t_m^{k_0})$ that maximizes the reduced cost. The corresponding reduced cost is $c_{k_0} - c_B B^{-1} A_{k_0}$, where $c_{k_0} = \sum_{i=1}^{m}(s_i - 1)t_i^{k_0}$, $c_B B^{-1} = (\lambda, \mu)$, $A_{k_0} = (1, t_1^{k_0}, t_2^{k_0}, \ldots, t_m^{k_0})^T$. Use y_i indicate the feasible pattern instead of $t_i^{k_0}$, we can obtain the sub-problem:

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

Use column generation to generate a new pattern until all reduced costs are negative.

And the IP formulation can be shown as below:

$$
\begin{aligned}
\max \quad & \sum_{j=1}^{m} \sum_{i=1}^{n} (s_i - 1) x_{ij} \\
\text{s.t.} \quad & \sum_{i=1}^{n} s_i x_{ij} \leq S, \quad j = 1, \ldots, m \\
& \sum_{j=1}^{m} x_{ij} \leq g_i, \quad i = 1, \ldots, n \\
& x_{ij} \geq 0 \text{ and integer}, \quad i = 1, \ldots, n, j = 1, \ldots, m
\end{aligned}
\tag{4}
$$

m indicates the number of rows. x_{ij} indicates the number of group type i placed in each row j.

For our new problem, the column generation will give the upper bound (LP relaxation) and lower bound (restricted IP). After obtaining the patterns with column generation, restricted LP equals LP relaxation, $LP^r = LP$, which provides an upper bound. Thus, we have the relation, $\max LP \geq \max LP^r \geq \max IP \geq \max IP^r$.

To obtain an optimal solution, we should implement branch and bound into column generation. This method is called branch-and-price.

4.3 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

$$
\sum_{k \in K(k)} x_k \leq \lfloor \alpha \rfloor \tag{5}
$$

and

$$\sum_{k \in K(k)} x_k \geq \lceil \alpha \rceil \tag{6}$$

, 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:

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

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

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

$Y_k = \{(y, z) : z = 1, \text{ if } y \geq k, z = 0 \text{ otherwise}\}$ 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.

4.4 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 1. *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.

4.5 The Property

In view of the complexity and uncertainty of branch scheme, we should analyze the property of this problem and use it to obtain a solution.

At first, we consider the types of pattern. 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, $l(k)$.

Let I_1 be the set of patterns with the minimal loss. Then we call the patterns from I_1 are largest. And the pattern with zero left space is called full pattern. Recall that we use the vector $(t_1, t_2, \ldots, t_m)$ to represent a pattern, where t_i is the size of group i. For example, take the length of each row be $S = 21$, the size of group types be $s = [2, 3, 4, 5]$. Thus these patterns, $(0, 0, 4, 1), (0, 0, 0, 4), (0, 2, 0, 3)$, belongs to I_1. Notice that the pattern, $(0, 0, 0, 4)$, is not full because there is one left space.

Now consider this special case, $[2, 3, \ldots, u]$, the group sizes are consecutive integers starting from 2. Then we can use the following greedy way to generate the largest pattern. Select the maximal group size,u, as many as possible and the left space is occupied by the group with the corresponding size. The loss is $q + 1$, where q is the number of times u selected. Let $S = u \cdot q + r$, when $r > 0$, we will have at least $\lfloor \frac{r+u}{2} \rfloor - r + 1$ largest patterns with the same loss. When $r = 0$, we have only one possible largest pattern.

Lemma 2. *If all patterns from an integral feasible solution belong to I_1, then this solution is optimal.*

This lemma holds because we cannot find a better solution occupying more space.

When the number of given rows is small, we can construct a solution in the following way. Every time we can select one pattern from I_1, then minus the corresponding number of group type from demand and update demand. Repeat this procedure until we cannot generate a largest pattern. Compare the number of generated patterns with the number of rows. If the number of rows is small, this method is useful.

Corollary 1. *When the left updated demand can form a largest pattern, the optimal solution is the combination of patterns from* I_1.

For example, when given the demand $d = (10, 11, 12, 10)$ and three rows. By column generation, we will obtain the solution $2.333 \times (0, 0, 0, 4), 0.667 \times (0, 0, 4, 1)$. But we can construct an integral solution $2 \times (0, 0, 0, 4), 1 \times (0, 1, 2, 2)$ or $3 \times (0, 0, 4, 1)$, that depends on which pattern we choose at the beginning.

But how could we know if the number of rows is small enough? We can consider the relation between the demand and the number of group types in patterns. Then we develop the following theorem:

Theorem 1. *When* $N \leq \max_{k \in I_1} \min_i \{\lfloor \frac{d_i}{b_i^k} \rfloor\}$, *select* k^**-th pattern from* I_1 *and it is the optimal solution.* N *is the number of rows,* $i = 1, 2, \ldots, m$, d_m *is the demand of the largest size,* b_m^k *is the number of group* m *placed in pattern* k.

In the light of the Theorem 1, when the number of given rows is small, we just need to select some patterns from I_1. Continuing with the above example, we just take $(0, 0, 0, 4), (0, 0, 4, 1), (0, 1, 2, 2)$ as the alternative patterns. For each k, $\min_i \{\lfloor \frac{d_i}{b_i^k} \rfloor\}$ will be $2, 3, 5$ respectively. So when $N \leq 5$, we can always select the pattern $(0, 1, 2, 2)$ five times as the optimal solution.

When column generation method gives an integer solution at the first time, we obtain the optimal solution immediately. Now suppose that we have a fractional solution. Divide the solution into a pure integral part and the fractional part. The fractional solution will have a corresponding integral supply occupying the same space size. When the rest groups can be placed in the rest rows(given rows minus the integral rows), then the total groups can be placed in the given rows.

Based on the above analysis, we can establish an algorithm below.

Algorithm 1 The algorithm to construct an optimal solution

Step 1. After the column generation gives the fractional LP solution $x^{'}$, calculate the supply quantity $q^{'} = (q_1, \ldots, q_m)$ for each group type. If each element is integral, keep it. If any element of this supply is not integral, we can construct an integer supply vector which can provide the largest integral profit.

Step 2. Construction: calculate the space occupied by fractional supply, increase the corresponding supply having the same space, delete the fractional part. If the space occupied by fractional supply is fraction, find the supply providing the same profit.

Step 3. Take this integral supply vector q as a new demand and obtain a new LP solution x^* with column generation. Divide x^* into a pure integral part x^I and fractional part x^F. Subtract the corresponding supply of the integral part x^I from the new demand q to obtain the rest groups($r = q - q^I$). N is the number of given rows. The number of rest rows equals to $N - \sum x^I$.

Step 4. Use subset sum problem to check if the rest groups can be placed in rest rows.

Step 5. If the rest cannot be placed, go to step 2 to construct a new supply.

Now we need to judge whether we need to tackle step 4 in some cases.

For the case $(2, 3, \ldots, u)$, if the rest space is 1, we can discard 1 or exchange 1 and 3 with two 2. Remember what we need to do is to construct a maxmimal integer supply.

Assume that the demand is large such that the given rows cannot accommodate it and the number of group containing 2 people is large.

When the demand is large, i.e. supply cannot cover the whole demand. The equation $\sum_{k=1}^{K} x_k \leq N$ will be always valid. The sum of all elements of the solution vector always equals to the given number of rows.

Then the rest space for each maximal pattern will be no more than 1 with the extra groups with the size of 2.

4.6 Definitions

For the group sizes belong to the subsets of $\{2, 3, 4, 5\}$, we have the following conclusions.

Suppose the demands are large enough, we consider a greedy way to generate a pattern.

Definition 1. *The greedy way to generate a pattern is described as follows. For each row, we will place the groups with the largest size as many as possible, then place the group whose size equals the number of the remaining seats. When the number of remaining seat is* 1 *or* 0*, we stop placing. In this way, we generate a greedy pattern.*

Definition 2. *Let* $[a, b, \ldots, d], a < b < \ldots < d$ *denote the group sizes. Let* $(d_a, d_b, \ldots, d_d)_D$ *denote the initial demands of the group sizes. Denote by* $(a, b, \ldots, d), a \geq b \geq \ldots \geq d$ *some pattern.*

Lemma 3. *The pattern generated by the greedy way will be the largest pattern.*

Proof 1. *For each pattern* $k \in I$*(denote by* I *all feasible patterns), we have two parts of loss. The first one results from the social distance between adjacent groups, we use* $\alpha(k)$ *to express its value. The second one is the empty seats which are not taken by any groups, use* $\beta(k)$ *to express its value.*

The largest patterns have the minimal loss. We need to prove the greedy pattern will also have the minimal loss.

Notice that the number of remaining seats will be less than 2. Denote by $g \in I$ *the greedy pattern. By placing the group with the largest size as many as possible, the pattern will have the minimal* $\alpha(g)$*,* $\alpha(g) \leq \alpha(k), k \in I$*. When the number of remaining seat is* 0*,* $\beta(g)$ *will be 0. In these cases, the total loss will be the smallest.*

When the number of remaining seat is 1*,* $\beta(g)$ *will be 1. Notice that any pattern* g_1 *will have at least* $\alpha(g_1) = \alpha(g) + 1$ *when placing another group, in this way, the loss,*$\alpha(g) + \beta(g) \leq \alpha(g_1) + \beta(g_1)$*, will still be the smallest. Thus, the greedy pattern will be the largest pattern.*

Corollary 2. *Other largest patterns can be generated by the greedy largest pattern.*

Denote by r the number of seats after placing the groups with the largest size as many as possible. We can generate a new largest pattern by decreasing the group with the largest size and increasing the group with the size of r. We can obtain all the largest patterns until the gap between the size of all placed groups is less than 2.

For example, when $S = 21$, group sizes are $[2, 3, 4, 5]$. The greedy pattern is $(5, 5, 5, 5, 1)$, which can develop the second largest pattern $(5, 5, 5, 4, 2)$, the third one $(5, 5, 5, 3, 3)$, the fourth one $(5, 5, 4, 4, 3)$, the fifth one $(5, 4, 4, 4, 4)$. Because the gap between 4 and 5 is 1 less than 2, we have obtained all the largest patterns.

Remark 1. $r = 1$ *will lead to the largest number of different largest patterns. Thus, the case,* $r = 1$*, will be the most complex part we need to consider.*

Our policy is to use the largest patterns as many as possible, then place the group with the largest remaining size in a greedy way. The core problem is how to obtain the maximum number of the largest patterns. But maybe we have several largest patterns, and their priority will affect the maximum number of largest patterns

when given the demands. Once we can determine their priority, then use the specific largest patterns according to their priority, we can obtain an optimal assignment for any given n rows.

Definition 3. *Given the number of seats in a row, S. The group sizes are $[a, b, c, d]$. Suppose we have several largest patterns $k_i, i \in \{1, 2, \ldots, l\}$.*

When the demands are enough to obtain pattern k_i, we say pattern k_i is available. If some largest pattern is not available, just skip it in order of precedence.

Without loss of generality, we assume that $k_i \succeq k_j, i, j \in \{1, 2, \ldots, l\}$ which means we prefer pattern k_i to k_j in our policy when they are both available.

When $k_1 \succeq k_2 \succeq \ldots k_l$ holds, we can follow this priority in our policy to obtain an optimal assignment for any demands d_a, d_b, d_c, d_d. We can say that k_i is preferred at least as much as $k_{i+1}, i = 1, 2, \ldots, l-1$.

If the priority between k_i and k_j depends on the demands. Suppose that when $D_4 \leq D_3$, $k_i \succeq k_j$. We will always choose the preferred pattern k_i until it is unavailable.

Theorem 2. *For any two types of group size,$[a, b], a < b \leq 5$, we can obtain an optimal assignment by following our 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 $S \div 3 = q \ldots 1$, the largest patterns will be $(3, 3, \ldots, 3, 2, 2), (3, 3, \ldots, 3, 1)$. If $S \div 3 = q \ldots 2$, the largest pattern will be $(3, 3, \ldots, 3, 2)$. If $S \div 3 = q$, the largest pattern will be $(3, 3, \ldots, 3)$.

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, $S \div 3 = q \ldots 1$, 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, $(5, \ldots, 5, 1)$ and $(5, \ldots, 5, 4, 4, 4, 4)$.*

For $[3, 5]$, when $r = 1$, we have two largest patterns, $(5, \ldots, 5, 1)$ and $(5, \ldots, 5, 3, 3)$. When $r = 4$, we have two largest patterns, $(5, \ldots, 5, 3, 1)$ and $(5, \ldots, 5, 3, 3, 3)$, 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, $(4, \ldots, 4, 1)$ and $(4, \ldots, 4, 3, 3, 3)$.

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

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

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.

Theorem 3. *Follow the priority in our policy will give an optimal assignment.*

Theorem 4. *We can obtain the optimal assignment when given n rows in our policy. The first k rows of n rows will be an optimal assignment when given k rows,* $1\leq k<n$*.*

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.

4.7 [a,b,c]

Theorem 5. *When* $c\leq 5$*, for any* $(a,b,c), a<b<c$*, we can obtain an optimal assignment by the following policy.*

4.7.1 [2,3,4]

Theorem 6. *The following policy will give an optimal solution 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 a solution no worse than that of $(4, \ldots, 4, 2)$.*

Proof 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 3 (Proof of Theorem 6)**.** *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 solution.

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.

4.7.2 [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.

4.7.3 [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 $(5,\ldots,5,5)$.

4.7.4 [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).

4.8 [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 7. *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.

5 Dynamic Model

We also study the dynamic seating plan problem, which is more suitable for commercial use. In the problem, customers come dynamically and the seating plan needs to be made without knowing the number and composition of future customers. This becomes a sequential stochastic optimization problem where conventional methods fall into the curse of dimensionality due to the large number of seating plan combinations. To avoid this complexity, we develop an approach which aims directly to the final seating plans. Specifically, we define the concept of target seating plans which are deemed as satisfactory. In making the dynamic seating plan, we will try to maintain the possibility of achieving one of the target seating plans as much as possible.

5.1 Nested Structure

Firstly, suppose that $1 + V_T(x) \geq V_T(x+1)$ holds. Then we have $2\lambda_2 \leq 1$, $3\lambda_3 \leq 1$, $\cdots$.

Use the inductive method to prove $1 + V_t(x) \geq V_t(x+1)$.

Suppose that $1 + V_t(s) \geq V_t(s-1)$, then we prove $1 + V_t(s+1) \geq V_t(s)$

$V_t(s) = \sum_i \lambda_i(\max\{i + V_{t+1}(s-i-1), V_{t+1}(s)\})$

$V_t(s+1) =$

Then we can develop the Theorem 8.

Theorem 8. *what*

6 Results

To construct the whole diagram of ..., we need to obtain ...

As stated previously, ...

As we all know, ... Now, the question is how to solve the .... problem mentioned above efficiently.

Lemma 7.

Now we know the problem can be solved in ... and the ...

7 Conclusion

We mainly focus on how to provide a way to ...

In our study, we stressed....

Our main results show that ...

Moreover, our analysis provides managerial guidance on how to place the seats under the background of pandemic.

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] Michael Barry, Claudio Gambella, Fabio Lorenzi, John Sheehan, and Joern Ploennigs. Optimal seat allocation under social distancing constraints. *arXiv preprint arXiv:2105.05017*, 2021.

[3] 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.

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

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

[6] Martina Fischetti, Matteo Fischetti, and Jakob Stoustrup. Safe distancing in the time of covid-19. *European Journal of Operational Research*, 2021.

[7] Elaheh Ghorbani, Hamid Molavian, and Fred Barez. A model for optimizing the health and economic impacts of covid-19 under social distancing measures; a study for the number of passengers and their seating arrangements in aircrafts. *arXiv preprint arXiv:2010.10993*, 2020.

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

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

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

[11] Mostafa Salari, R John Milne, Camelia Delcea, Lina Kattan, and Liviu-Adrian Cotfas. Social distancing in airplane seat assignments. *Journal of Air Transport Management*, 89:101915, 2020.

[12] 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.

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

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

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

Proof

Proof 4 (Theorem 1)**.** □

Proof 5 (Lemma 1)**.** □

Proof 6 (Lemma 2)**.** □

Proof 7 (Theorem 2)**.** □

Proof 8 (Theorem 2)**.** □

Proof 9 (Theorem 3)**.** □

Proof 10 (Theorem 4)**.** □

Proof 11 (Theorem 5)**.** □

Proof 12 (Theorem 6)**.** □

Proof 13 (Lemma 2)**.** □

Proof 14 (Theorem 7)**.** □

Proof 15 (Lemma 4)**.** □