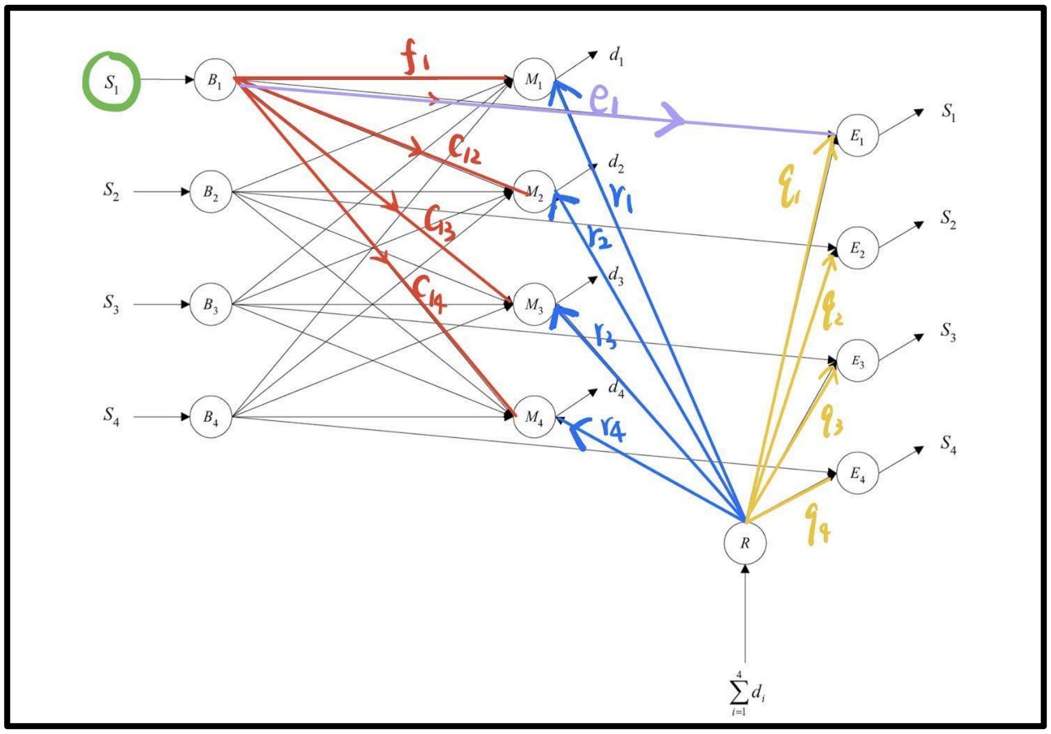
UNIVERSITY OF SOUTHERN CALIFORNIA
DEPARTMENT OF INDUSTRIAL ENGINEERING

ISE 533: INTEGRATIVE ANALYTICS
Spring 2021

ISE 533 FINAL PROJECT PORTFOLIO

(Team 1)



Miao Xu, miaox@usc.edu
Yiyi Wang, wangyiyi@usc.edu
Jacob Andreesen, andreese@usc.edu
Jeff Chen, chen364@usc.edu

MAY 12, 2021

Table of Contents

1 PORTFOLIO INTRODUCTION

1.1 *Structure*

In this portfolio, the reports for four projects we have studied over the course of Spring 2021 are provided. Each report follows a similar structure as detailed below:

1. **Introduction:** Section provides outline for the purpose of the project being performed, the specific problem being investigated, the scope and assumptions made for the project, the testing approach implemented, and an overview of the results
2. **Model Overview:** Section provides a detailed understanding of the linear programs used in each project. This includes theory behind the programs, coverage of the LPs notation, and the LP formulation including the objective functions and constraints
3. **Technical Architecture:** Section provides the detailed technical process and approach implemented for each project. The steps within each project vary and creative project extensions are included. Notice that the specific code utilized will not be in the architecture, but the programming notebooks are accessible through each projects *Reference Section*
4. **Results:** Section provides quantitative outcomes for the problem being investigated as well as a discussion to interpret the results that were obtained
5. **Discussion:** Section includes references to lessons learned during the project, specific challenges encountered, and future work the team would like to perform
6. **References:** Short list of sources referenced while implementing our solutions. The coding portion of each project and associated presentation is the 1st and 2nd reference of each project

1.2 *Project Abstracts*

Abbreviated abstracts for each project are provided below:

1.2.1 Meal Planning

In Project 1, we designed a restaurant recommendation application that utilizes a mixed integer program to find the five best restaurants based on a group's collective preferences. During the project we scraped data from Yelp, allowed for user inputs, and outputted a decision of 5 restaurants as well as a secondary cocktail bar recommendation. The app performed as expected with accurate results outputted. A range of additional features are currently being implemented.

1.2.2 Transshipment

In Project 2, we wanted to utilize simulation methods in coordination with stochastic optimization to solve problems with distributed uncertainty. We set out to solve the "multilocation transshipment problem" where a supply chain relies operates with transshipments, which is the practice of shipping inventory between retailers of the same class rather than the retailers relying on inventory from the central supplier alone. In order to solve the linear program, we developed, we used two alternate optimization algorithms (Stochastic Gradient Descent (SGD) and Bender's Decomposition) and compared their performance. Results showed that Bender's operated more efficiently in this formulation however we hypothesized that the outcome may be different when more scenarios are introduced.

1.2.3 LEO-Wyndor

In Project 3, we set out to investigate the behavior and performance of Predictive Stochastic Programming (PSP) where informed predictions of uncertainty can be made based on "covariate" sets of responses and predictors. During this project, a PSP formulation wthat the Simple Recourse model performs better than the Deterministic model in the long run.as applied to the popular 'Wyndor Glass Company' production problem. To solve the Wyndor PSP, a methodology referred to as Learning Enabled Optimization (LEO) was applied and is walked through in the *Technical Architecture Section*. To understand the benefit of the LEO protocol as applied to PSPs, two statistical models were utilized and compared: Deterministic Forecast (DF) and SAA-Empirical Additive Error (EAE/SAA). Ultimately, we determined that LEO-PSP poses a number of modeling advantages, the validation approach established is useful and accurate in assessing model validity, and the EAE/SAA model is appropriate for PSP models whereas Deterministic is not due to respective error considerations

1.2.4 Covid-19

In project 4, we try to implement the Deterministic and Simple Recourse model in stochastic optimization to find the optimal allocation of ventilators among U.S. During the COVID-19 pandemic, many states in the U.S. have faced a shortage of medical resources, so if the medical resources(ventilators) can be transferred across states and the Strategic National Stockpile, ventilators will be allocated to the states in need, and the shortage of ventilators will be alleviated. The objective of our problem is to minimize the unmet demand for ventilators by optimally distributing them across all states. We build two models to solve this problem, one is the deterministic (point-forecast) model without data uncertainty, and another is the two-stage simple recourse model considering three different scenarios for each state's demand. The result shows the simple recourse model performs better than the deterministic model in the long run, which means the simple recourse model can meet the actual demand with a smaller shortfall after considering three possible low, mean, and high expected demand for each state. Also, unlike stochastic optimization we did in project 2&3, in this project, we observe that the decision of one period will affect the start point of the next period. Therefore, the simple recourse model performs better over time.

2 RESTAURANT RECOMMENDER REPORT

2.1 Introduction

The objective of this project was to develop a personalized restaurant recommendation application that facilitates the selection of a set of restaurants based on the unique tastes and preferences of a group of students. Specifically, the app was required to maximize the sum of the “likes” of the group by selecting five different restaurants with consideration towards each member’s budget, dietary preferences, and travel distances during the period.

The solution our team implemented is called “Plate Mate” and utilizes a deterministic mixed integer program (MIP) to select the five highest rated Los Angeles restaurants that adhere to the users’ constraints. As we will see, upon opening our application, each member of the group is first directed to enter their address, budget, and food preferences. Next, users can enter additional optional preferences such as the minimum number of Yelp reviews a restaurant must have, the maximum distance they would be willing to travel, and specific LA neighborhoods that would be preferred. Following the entry of this information, the MIP (as described in *Section 2.2)* is run, and five restaurants are outputted in the order they will be visited. In addition, the application was extended such that a secondary suggestion for a local cocktail bar is provided to the group of users along with the list of restaurants.

In order to simplify the problem and account for differences in user preferences, a number of assumptions were made prior to solving. First, we assumed users’ restaurant budgets to be their total budget over five months rather than on a per restaurant basis, allowing for additional variability in the price of individual restaurants. Second, to ensure group members traveled similar distances to restaurants, the percent difference between the max and min distance traveled by members was constrained by an α of 50%. Finally, to maximize the sum of likes, we assumed that the best restaurant choices would be the most highly rated restaurants that satisfied the user’s constraints.

2.2 Model Overview

In this section, the Mixed Integer Program utilized to select restaurants that satisfy the group’s preferences and maximize the sum of collective likes will be described. In *Section 2.2.1,* user and restaurant matrix notation will be defined, representing the inputs provided to the optimization problem. Additionally, a number of indices are provided which are used to track specific users, restaurants, and the visitation month through the program. In *Section 2.2.2,* the “restaurant selection” decision variables are introduced along with their integer constraints. Finally in *Section 2.2.3,* the complete MIP formulation is provided through further description of the program’s objective function and linear constraints.

2.2.1 Dataset Matrices & Notation

The model utilized in our solution requires a dataset to be obtained from both the group of users and from Los Angeles restaurants. In this section we will detail what information is gathered from users and restaurants, how that data is stored, and specific notation that will be utilized for our program. Later, in *Section 2.3,* we will outline the specific methods used to acquire the data that fills these matrices.

Three indices are used to identify and track the specific user, month, and restaurant within our mixed integer program. These indices are provided below:

$$\begin{array}{l} \text{Let } k \in K \text{ reference a specific user within the group} \\ \text{Let } m \in M \text{ index the specific month in which a restaurant would be visited} \\ \text{Let } n \in N \text{ index a specific restaurant in Los Angeles} \end{array} \tag{1}$$

Note that in our application, the sets referenced are:

$$\begin{array}{l} K = \{1,2,3,4\} \text{ as the group size is taken to be four people} \\ M = \{1,2,3,4,5\} \text{ as the period is taken to be 5 months from January – May} \\ N = \{1,2,\ldots,q\} \text{ where q is the total number of LA restaurants in our dataset} \end{array} \tag{2}$$

In order to store the dataset of Los Angeles restaurants we denote the restaurant set matrix χ of form $\chi = \{x_1, x_2, \ldots, x_q\}$. Within χ, specific restaurants and their corresponding attributes are stored in matrices x_n which take the form shown below:

$$x_n = \begin{bmatrix} \text{Rest. Name} = \{\boldsymbol{String}\}, \\ \text{Address} = \{\boldsymbol{String}\}, \\ \text{Neigborhood} = \{\boldsymbol{String}\}, \\ \text{num. reviews} = \{0 \leq \boldsymbol{Integer} < \infty\}, \\ \text{rating} = \{0 < \boldsymbol{Float} \ \leq 5.0\}, \\ \text{price} = \{\$, \$\$, \$\$\$, \$\$\$\$\}, \\ \text{Cuisine} = \{Breakfast, Korean, \ldots Halal\} \end{bmatrix} \tag{3}$$

As seen in the matrix x_n, seven attributes are gathered for each restaurant including its name, address, neighborhood, number of Yelp reviews, Yelp rating, price range, and cuisine type. Notice that price range is symbolic, however as we will see later, the ‘$’ symbols can be converted to numeric values for our purposes.

User information is stored within the user matrix U of form $U = \{u_1, u_2, \ldots, u_k\}$.Within U, individual users and their corresponding attributes are stored in matrices u_k which take the form shown below:

$$u_k = \begin{bmatrix} \text{Name} = \{\boldsymbol{String}\}, \\ \text{Address} = \{\boldsymbol{String}\}, \\ \text{Budget} = \{0 < \boldsymbol{Float} < \infty\}, \\ \text{Cuisine} = \{Breakfast, Korean, \ldots Halal\}, \\ \\ \text{min. reviews} = \{0 \leq \boldsymbol{Integer} < \infty\} \\ \text{rating} = \{0 < \boldsymbol{Float} \ \leq 5.0\} \\ \text{neigborhood} = \{\boldsymbol{String}\} \end{bmatrix} \tag{4}$$

As seen in *Equation 4*, seven attributes are gathered for each user, with the first four attributes being required and the last three being optional. It is important to note that multiple entries can be made for each user's cuisine preference as we will see in *Section 2.3.*

One final expression that is important to note prior to introducing our decision variables and MIP formulation is the function

$$d(L_1, L_2) \rightarrow \ \mathbb{R}_0^+ \tag{5}$$

This function d takes in the latitude and longitude for two location (L_1, L_2) and utilizes the haversine distance formula to calculate the distance between the two addresses. As we will see, this function was useful in ensuring distance traveled by users was approximately equal and satisfied the defined constraints.

2.2.2 Decision Variables

The decision made by our application and outputted to the group of users is a selection of five LA restaurants which maximize the preferences of the group. Thus, in our MIP, we designate decision variables as ω_n, indicating whether or not restaurants x_n will be visited. These decisions are constrained to be integer values contained in the set as shown below:

$$\omega_n \in \mathbb{Z}_2 = \{0,1\} \tag{6}$$

As such, when $\omega_n = 0$, restaurant x_n is said to not be visited, and when $\omega_n = 1$, restaurant x_n will be visited.

2.2.3 Mixed Integer Program

After constructing the restaurant dataset and gathering preferences from the group of users, the information is used to solve the Mixed Integer Program provided below and the five optimal restaurants for the group are outputted:

$$\underset{\boldsymbol{\omega}}{\textbf{maximize}} \quad \omega^T \cdot (\chi_{rating}) \tag{7a}$$

$$\textit{subject to} \quad \sum_{i=1}^{n} \omega_n = 5 \tag{7b}$$

$$\sum_{i=1}^{n} (\omega_n x_n.price) \leq u_k.budget \;\forall\, n \in N, k \in K \tag{7c}$$

$$\omega_n \left(1 - \frac{\min\limits_{n,k} d(x_n.address, u_k.address)}{\max\limits_{n,k} d(x_n.address, u_k.address)} \right) \leq \alpha \;\forall\, n \in N, k \in K \tag{7d}$$

$$\omega_n u_k.min.reviews \leq x_n.num.reviews \;\forall\, n \in N, k \in K \tag{7e}$$

$$u_k.min.rating \;\leq \omega_n x_n.rating \;\forall\, n \in N, k \in K \tag{7f}$$

$$\omega_n \in \{0,1\} \text{ indicating whether restaurant } x_n \text{ was visited} \tag{7g}$$

The objective function (*Equation 7a)* of the program is the product of two vectors, the decision vector ω and the restaurant rating vector χ_{rating} . In this way, we define the optimal solution of the problem as the selection of the five highest rated restaurants that satisfy the constraints and preferences of the group of users.

The first constraint *(7b)* limits the number of selected restaurants to five. Constraint (*7c)* is the budgetary constraint ensuring that the total price for all five restaurants is less than or equal to the restaurant budget for each group member. Constraint *(7d)* is the travel distance constraint. The portion inside the parentheses calculates the percent difference in travel distance between the group member that travels the shorts distance and the member who travels the farthest distance for each restaurant using function $d(L_1, L_2)$. These percentages are multiplied by the decision vector and compared to $\alpha = 0.5$ thereby constraining the difference in travel distance for the selected restaurants by group members to be less than or equal to 50%. Constraint *(7e)* and (*7e)* are used for the optional user inputs ensuring that selected restaurants have more than the minimum desired number of reviews and higher ratings than the minimum desired respectively. Finally, constraint *(7g)* constrains the decision variables ω_n to integer values of 0 or 1 indicating “not visiting” and “visiting” respectively.

2.3 Technical Architecture

In this section, the technical architecture of Plate Mate and the process flow through the app are described in detail. To program our application, our group used Python as the primary language and the Pyomo Library and ‘GLPK’ solver to formulate the MIP. The section starts by outlining data acquisition for the restaurant and user matrices through Yelp Web Scraping (*2.3.1)* and prompted user inputs *(2.3.2)*. Next, efforts to filter and prepare the restaurant data for the MIP using the user preferences is described *(2.3.3)*. Finally, the section concludes by discussing the output from the MIP and the providing of secondary activity suggestions.

2.3.1 Yelp Web Scraping and Cleaning

In order to obtain Los Angeles restaurant data, initially our team developed and modified a web scraping tool to acquire key attributes from Yelp's website. After initial web scrapes, we came across a number of problems with this web scraping method. With each pass, key attributes were often missing or incorrect in the obtained dataframe, and a selection of restaurant attributes were only revealed through button clicks. Thus, to obtain that data we would have had to utilize a browser automation software to simulate website clicks and grab the data from the html afterwards.

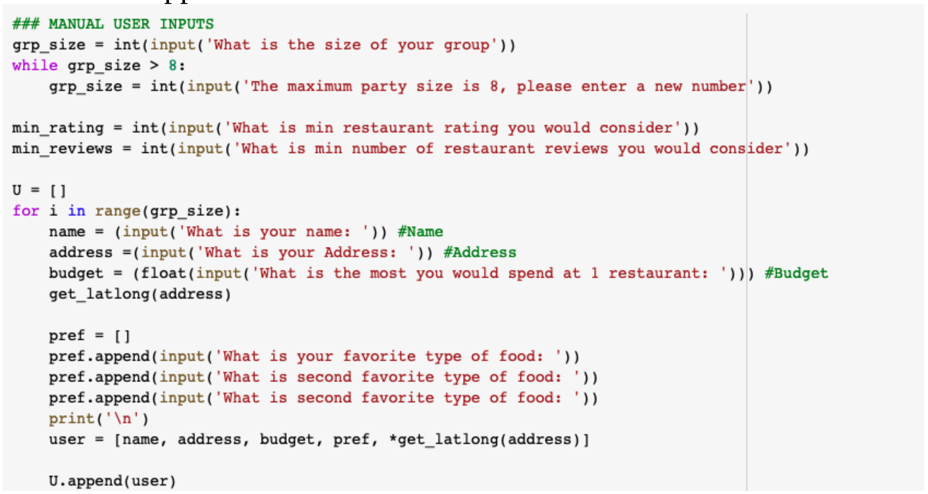
Instead of performing this tedious process, our team discovered the API endpoint 'https://www.yelp.com/search/snippet' which we were able to utilize to grab the required data for LA restaurants. Using this method, we constructed a dataframe containing 240 restaurants.

Next, initial data-preprocessing was performed beginning with the removal of unnecessary columns from the dataframe, such as the 'businessUrl' and 'isAd.' Data quality was another concern for our team. A number of restaurants had null values for attributes, specifically for price range. While contemplating how to repair these missing values, we ultimately concluded that it be best to make note of the restaurants with missing values and remove them from the dataset.

Further processing occurred through the manipulation of a number of key dataframe columns. The restaurant neighborhood column was converted from list object types to strings in order to conform to the required matrix definition. The 'priceRange' column was manipulated to allow numerical price ranges rather than solely numeric (where "$" = 0-10, "$$" = 11-30, "$$$" = 31-60, "$$$$" = 60-100). A locator function was used to convert street addresses into latitudes and longitudes. Following these manipulations, counts were taken for the neighborhood location and cuisine type of the remaining restaurants to ensure variability within the dataset.

2.3.2 User Matrix Inputs

Following the collection of restaurant data, users were prompted to input their preferences as described by the code snippet shown below:



```
### MANUAL USER INPUTS
grp_size = int(input('What is the size of your group'))
while grp_size > 8:
    grp_size = int(input('The maximum party size is 8, please enter a new number'))

min_rating = int(input('What is min restaurant rating you would consider'))
min_reviews = int(input('What is min number of restaurant reviews you would consider'))

U = []
for i in range(grp_size):
    name = (input('What is your name: ')) #Name
    address =(input('What is your Address: ')) #Address
    budget = (float(input('What is the most you would spend at 1 restaurant: '))) #Budget
    get_latlong(address)

    pref = []
    pref.append(input('What is your favorite type of food: '))
    pref.append(input('What is second favorite type of food: '))
    pref.append(input('What is second favorite type of food: '))
    print('\n')
    user = [name, address, budget, pref, *get_latlong(address)]

    U.append(user)
```

Figure 1:User Data Matrix in notebook

Essentially, the code shown gathers the preferences of users and inputs into a matrix of the form $\mathcal{U} = \{u_1, u_2, \ldots, u_k\}$ as described in the prior section. To simplify this process and ensure the restaurant dataset was not over-filtered, a number of preferences such as “min_reviews” and “min_rating” were designated as group decisions not individual member decisions. It is also important to note that the application prompts each user for a minimum of three cuisine preferences to ensure that selected restaurants will not all be of one type (i.e., the MIP is prevented from selecting five restaurants of 1 cuisine).

2.3.3 Secondary Restaurant Filtering

Following users’ inputs to the application, a secondary filtering is performed on the restaurant matrix χ based on the preferences not included in the MIP. Restaurants that do not serve a cuisine type from the collective list of the group’s cuisine preferences are filtered out. As expressed mathematically:

$$x_n.cuisine \subseteq \mathcal{U}.cuisine \tag{8}$$

Additionally, the optional neighborhood user preference may be used to filter out restaurants not located in the specified neighborhoods.

2.3.4 MIP Output & Activity Suggestion

The filtered restaurant matrix and user matrix are inputted into the mixed integer program as described by *Equation 7* and the program is then optimized. To solve the program, the model described was first implemented using Pyomo and the GLPK solver was utilized.

The user is outputted a dataframe containing the five optimal restaurants and their corresponding attributes that satisfy the groups preferences. The optimal objective value is also outputted to the user and scored from a maximum value of 25 which would indicate all five selected restaurants have ratings of five stars.

As an extension to the primary functionality offered by the application, our team added an additional “activity” recommendation that the group of users would be provided following the solution of the MIP. By conducting a Yelp scrape in a similar manner as described in *2.3.1,* we were able to construct a dataframe containing a list of cocktail bars in the Los Angeles area. This cocktail bar dataset was then cleaned and filtered such that all bars had a rating of 3.5 starts or above. After selecting the five optimal restaurants for the group, the distance function is utilized to select the closest bar to each restaurant and those bars are outputted to the group. This secondary suggestion provides the user with another activity in their area that they be interested in following their dinner at the restaurants.

2.3.5 Additional Optional Features

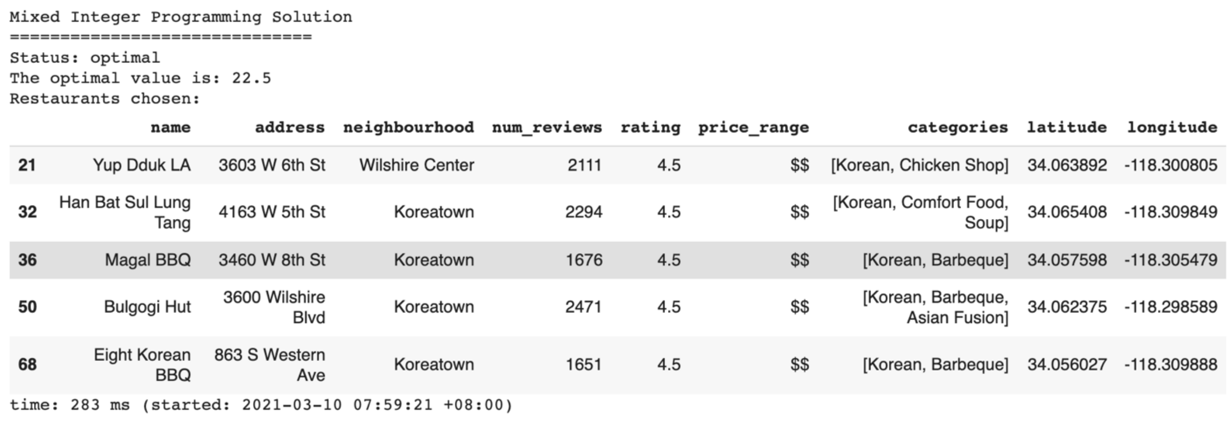
In addition to the primary functionality offered by the application, there were a number of additional features the team began implementing to enhance the results. While not all were all completed due to the project’s time frame, they will be mentioned in this section.

The primary extension we implemented as we saw in the prior section was the additional cocktail bar recommendation offered by the app. By scraping additional data from the Yelp, such as movie theaters, clubs, games, events, etc., we would like to provide a range of different recommendations that are connected to each restaurant. In this way the app will operate more as a nightlife application rather than just restaurants. Another interesting feature Plate Mate will incorporate is clustering of the restaurants gathered from Yelp. By performing K-Means clustering, we can determine which restaurants are most similar and assign them cluster labels. An additional constraint is then added to the optimization problem that restricts the number of restaurants selected for each cluster to be less than two. This ensures that the 5 restaurants we are selecting match the preferences of the group but are not so similar that it feels like we are going to the same restaurant five different times. Similar to providing additional activities with each restaurant, our team would like to add more attributes to each restaurant recommendation we provide. Details such as “do they deliver?”, parking spaces, service hours, and reservation links are all pieces of information that people tend to reference when making restaurant decisions. Finally, we are working to add additional filters to the problem that provide for more detailed preferences for user, such as allergies, driving vs walking, and time availability.

2.4 Results

In this section, example outputs of our application are provided. In order to generate these outputs, a series of five test cases were constructed based on the preferences from the four members of our project group. With each test case the user matrix was updated, and the MIP problem was resolved to yield different results.

The table below provides the five restaurant recommendations that the program found to be optimal based off the preferences from our first test case.



```
Mixed Integer Programming Solution
==============================
Status: optimal
The optimal value is: 22.5
Restaurants chosen:
```

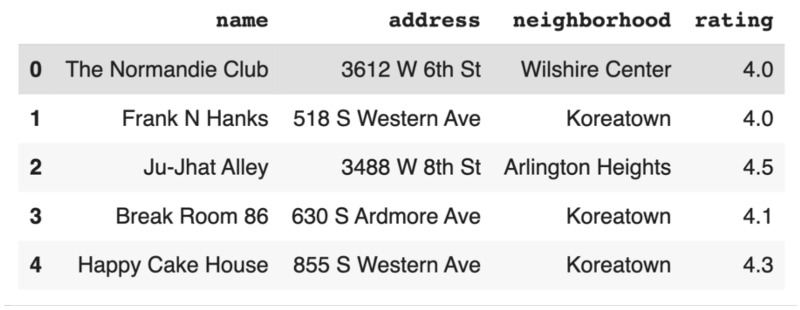
	name	address	neighbourhood	num_reviews	rating	price_range	categories	latitude	longitude
21	Yup Dduk LA	3603 W 6th St	Wilshire Center	2111	4.5	$$	[Korean, Chicken Shop]	34.063892	-118.300805
32	Han Bat Sul Lung Tang	4163 W 5th St	Koreatown	2294	4.5	$$	[Korean, Comfort Food, Soup]	34.065408	-118.309849
36	Magal BBQ	3460 W 8th St	Koreatown	1676	4.5	$$	[Korean, Barbeque]	34.057598	-118.305479
50	Bulgogi Hut	3600 Wilshire Blvd	Koreatown	2471	4.5	$$	[Korean, Barbeque, Asian Fusion]	34.062375	-118.298589
68	Eight Korean BBQ	863 S Western Ave	Koreatown	1651	4.5	$$	[Korean, Barbeque]	34.056027	-118.309888

```
time: 283 ms (started: 2021-03-10 07:59:21 +08:00)
```

Figure 2: Restaurant Decisions based on test preferences

Notice that no two restaurants are repeated in the dataframe and all restaurants fall within a similar area (primarily Koreatown as our test case addresses are located in the area). The collective price range of all restaurants satisfies the inputted user constraints, and the cuisine type is a subset of the user cuisine preference list. The recommended restaurants received a score of 22.5/25 with all restaurants having a rating of 4.5 stars.

The secondary “activity” recommendations are provided in the table below.



	name	address	neighborhood	rating
0	The Normandie Club	3612 W 6th St	Wilshire Center	4.0
1	Frank N Hanks	518 S Western Ave	Koreatown	4.0
2	Ju-Jhat Alley	3488 W 8th St	Arlington Heights	4.5
3	Break Room 86	630 S Ardmore Ave	Koreatown	4.1
4	Happy Cake House	855 S Western Ave	Koreatown	4.3

Figure 3: Secondary recommendations

Each row of this secondary table provides the closest cocktail bar to the corresponding row from the restaurant recommendations. As stated, cocktail bars with reviews lower than 3.5 stars were filtered from the full cocktail bar dataset ensuring that the recommended bars had high ratings.

2.5 Discussion

2.5.1 Software & Results Validation

At the beginning of this project, our team set out to develop a restaurant recommendation engine to help groups of friends find local restaurants to visit that satisfied each individual's preferences. By using various data acquisition, manipulation, and cleaning strategies and through the development of a mixed integer program, we were able to construct a fully functional application with a set of basic features allowing users to accomplish our goal.

When gathering results, we performed a series of optimizations based on five designated test cases. For each test case, we found that the application performed as expected, generating a list of five restaurants that met the users' constraints and maximized the sum of the rating of the five restaurants, Additionally, for each test case, five different cocktail bars were recommended to the group as secondary activities.

A number of interesting observations were made from the gathered results. When user's offered similar cuisine preferences, the restaurant matrix occasionally became over-filtered with remaining restaurants either very similar to one another or with poor overall ratings. Additionally, the travel distance constraint also limited restaurant recommendations as restaurants in between users were most often chosen even when better options existed slightly outside of the constraints. In these ways, the software implemented was very rigid, and was not capable of addressing changes to preferences and "borderline" restaurant cases.

2.5.2 Lessons Learned & Future Work

While the program we designed operated as intended, this project was not without obstacles and we learned many lessons along the way. Through this project, our team was able to recognize the real-world value linear programs, and specifically mixed integer programs, offer in solving complex problems. In this project, user preferences were deterministic in nature, therefore uncertainty did not have to be addressed in our model, and decisions were made based on an assumed accurate portrayal of reality.

One of the important lessons we learned as a group was the importance of time management and proper planning when working through a data science project. In the beginning, we had established an optimistic schedule that did not accurately represent some of the challenges we would face along the way, such as the problems faced during web-scraping and modifications made to the MIP. Due to the technical challenges encountered and the lengthening of time to complete tasks, some of the additional functionality we mentioned in our initial presentation had to be left out.

In the future, our team plans to enhance the functionality of the application in a number of ways. First, we would like to improve the user interface by building a front end to the application as it currently exists solely as backend software. Second, we would like to make the software more dynamic by enabling the group to swap out restaurants they did not prefer and offering additional recommendations ranked in three distinct tiers. The dynamics of Plate Mate can also be improved through additional optional filters and preferences that can be toggled on or off. Finally, we would like to expand our secondary recommendation from cocktail bars alone to more events, clubs, and activities in the Los Angeles area. In this way the app can serve not only as a restaurant recommendation engine but a nightlife planner as well.

2.6 References

[1] ***Code* - *https://nbviewer.jupyter.org/github/jeffchenchengyi/ISE-533/blob/main/project1.ipynb***

[2] ***Ppt* - *https://nbviewer.jupyter.org/github/jeffchenchengyi/ISE-533/blob/main/ISE%20533%20Project%201%20Restaurant%20Recommendation.pdf***

[3] A. Ng, “Intro to Yelp Web Scraping using Python,” *Towards Data Science*, 12-Jun-2020. [Online]. Available: https://towardsdatascience.com/intro-to-yelp-web-scraping-using-python-78252318d832. [Accessed: 12-May-2021].

[4] J. Xu, “Meal Planning for the New Millenium,” in ISE533 lecture, 02-Feb-2021.

[5] J. Xu, “MnM_MIP Jupyter Notebook,” *MnM_MIP*. .

3 TRANSSHIPMENT PROBLEM REPORT

3.1 Introduction

In Project 2, our team sought to better understand and solve classes of decision models involving independent uncertainty. Specifically, we wanted to utilize simulation methods in coordination with stochastic programming and optimization to solve problems with uncertainty.

The problem we set out to solve is referred to as the “multilocation transshipment problem” and was introduced in a paper by Herer [3]. In this problem (detailed further in *Section 3.2*) we will be considering a supply chain involving a single central supplier and four distinct retailers, each with different cost structures and demand parameters. The supply chain relies on an operating model that involves transshipments, which is the practice of shipping inventory between retailers of the same class rather than the retailers relying on inventory from the central supplier alone. This method is already practiced in industry as it enables distributers to reduce the amount of inventory at physical locations and provides an effective mechanism for managing fluctuating demand and reducing cost. In our case, transshipments will occur proactively, therefore these shipments will be planned in advance rather than in reaction to problems in real-time.

In order to solve the two-stage transshipment linear program we develop; we will introduce two alternate optimization algorithms: Stochastic Gradient Descent (SGD) and Bender’s Decomposition. Traditional solution methods for Stochastic Programs (SPs) can have substantial computational demands due the vast number of potential scenarios, thus in this project we will seek to understand the performance of these two algorithms and the benefit they offer in solving SPs.

This report begins by describing the transshipment network flow and corresponding two-stage linear program in depth. Next, we will walk through the technical architecture our team employed in our solution, integrating SGD and Bender’s. Finally, we will present our results, and have a discussion about the outcomes and potential future work.

3.2 Transshipment Model Overview

3.2.1 Network Flow Across Period

Prior to formulating the linear program to represent the transshipment problem, it is necessary to understand the order of events that occur within a given period and the objective our model will seek to optimize. During a single period, the described events occur in the following order:

1. In the first phase of the problem, we assume that the initial inventory level at each retailer is unknown
2. Replenishment orders placed with the supplier from the previous period arrive
3. Any outstanding backlogged orders (or shortage) is met through a portion of the replenishment. The remaining replenishment raises the inventory level at each retailer
4. Demand is realized at each retailer. Demand represents the only uncertain event during the period
5. After demand is observed, the supply system solves the reallocation problem, and transshipments are made between each retailer

6. Retailers meet as much demand as possible following transshipments. If retailers cannot meet all of the demand, they experience a shortage and are assigned backlogged orders. Backlog and inventory are updated for each location and new orders are placed with the supplier.

As mentioned in the prior section, the goal of our model will be determining the optimal order-up-to (replenishment) and transshipment quantities that minimize the expected long-run average cost and based on as series of assumptions made in the Herer paper.

<u>*3.3.2 Notation & Decision Variables*</u>

Based on the supply chain behavior and events described, the following notation was utilized in describing the transshipment system. As a reminder the system we will model has $N = 4$ retailers and a single central supplier. Transshipments only occur between the retail locations and retailers cannot ship inventory back to the central supplier.

In the first stage of the linear program, two variables will need to be understood. Those variables and a brief description are provided below:

$S = (s_1 \ldots s_N)$	Vector of order-up-to quantities
$D = (d_1 \ldots d_N)$	Vector of demand distributions at retailer i

As we will see in the model, S is the first stage decision being made and $\widetilde{D}$ will denote a vector of random variables, containing discrete demand distributions for each retail location.

After solving for the first-stage decision vector S and observing an outcome D from the vector of random demand variables, the second-stage problem is solved using the following decision variables:

e_i	Ending inventory at retailer i
f_i	Stock at retailer i used to satisfy demand at retailer i
q_i	Stock at retailer i increased by supplier replenishment
r_i	Amount of shortage met after replenishment at retailer i
t_{ij}	Stock at retailer i shipped to retailer j to meet j's demand

In the second-stage, cost is minimized under the know S and D outcomes. The retailer specific costs associated with transshipments and the events outlined are described by the variables below:

h_i	Cost per unit to hold inventory at retailer i
c_{ij}	Cost per unit of transshipment from retailer i to retailer j
p_i	Penalty cost at retailer i for unit backlogged/shortage

Using the notation defined and the series of events within a given period, a network flow diagram (as seen below) can be used to visualize the movement of inventory within the system.

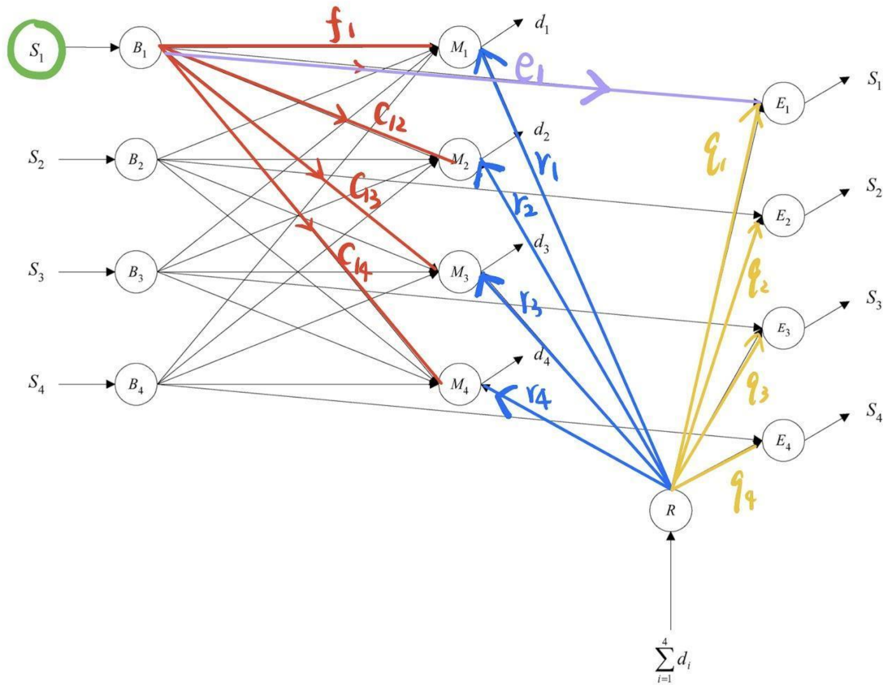


Figure 4: Transshipment Network Flow

Focusing on the flow pertinent to retailer $i = 1$, we see that in the beginning of the period B_1 an initial inventory s_1 is defined. This inventory is used to fulfill demand at retailer 1 (f_1), redistributed to other retailers through transshipments ($c_{12}..c_{14}$), and the remainder is the remaining inventory at 1 (e_1). After demand is realized during the period M_1, the variables $f_1, c_{21}, c_{31}, c_{41}$ and the replenishment for shortage (r_1) are used to satisfy demand d_1. The ending inventory (and starting for the next period) is the sum of e_1 and replenishment that increases inventory q_1.

3.2.3 Two Stage Formulation

Taking the variables and network flow described in the prior section, we can translate the information into a two-stage stochastic linear program. The expressions defining the program are provided below.

First Stage:

$$\min_{S\geq 0} \mathbb{E}\left[h\left(S,\widetilde{D}\right)\right] \tag{1a}$$

Second Stage:

$$h(S,D) = \quad \min \sum_i h_i e_i + \sum_{i\neq j} c_{ij} t_{ij} + \sum_i p_i r_i \tag{2a}$$

$$s.t. \quad f_i + \sum_{j\neq i} t_{ij} + e_i = s_i, \forall i \tag{2b}$$

$$f_i + \sum_{j\neq i} t_{ij} + r_i = d_i, \forall i \tag{2c}$$

$$\sum_i r_i + \sum_i q_i = \sum_i d_i \tag{2d}$$

$$e_i + q_i = s_i, \forall i \tag{2e}$$

$$e_i, f_i, q_i, r_i, s_i, t_{ij} \geq 0, \forall i, j \tag{2f}$$

In the first-stage, we are choosing vector S such that the expectation $\mathbb{E}$ of long run average cost $h\left(S,\widetilde{D}\right)$ is minimized where $\widetilde{D}$ is the vector of the random demand variable distributions.

In the second-stage, the objective function minimizes the transshipment cost when S and D are known quantities. The first constraint defines the use of the initial inventory. The second constraint is used for demand fulfillment. The third constraint defines the distribution of the supplier's replenishment shipments. And *Constraint 2e* determines the next day starting inventory. Notice that all decision variables are constrained to positive values.

3.3 Technical Architecture

In this section, we will walk through the technical procedure we followed when implementing our solution. First, we will touch on the data provided for the transshipment problems and the discrete demand distributions developed using the data. Next, we will explain how each algorithm was implemented when solving the two-stage linear program. Within these final sections, adaptations, and extensions we made to the algorithms to improve results will be explained. Third, the indicators utilized to compare the Stochastic Gradient Descent and Bender's Decomposition algorithms are detailed.

3.3.1 Random Retailer Demand Distributions

As mentioned in the prior sections, the supply chain being designed is proactive, thus we are making decisions for a period prior to realizing the actual demand for the period. Thus, uncertainty is introduced in the problem through our rand demand vector $\widetilde{D}$. Within $\widetilde{D}$, discrete demand distributions for each retailer are stored, which each of those distributions containing five demand values. Therefore, within the transshipment problem, 5^4 or 625 different scenarios are possible (4 retailers with 5 different demand possibilities each) due to the random variable.

In order to construct the discrete distribution for each retailer, we began with the normal demand (μ_i) and demand standard deviation (σ_i)data provided for each location. This data is shown in the table below:

Table 1: Normal Demand and Demand Standard Deviation for each retailer

Retailer (i)	Normal Demand (units)	Std. Dev (units)
1	100	20
2	200	50
3	150	30
4	170	50

From this data, a normal distribution was constructed with five discrete demand values for each retailer with points at $(\mu_i - 2\sigma_i, \mu_i - 1\sigma_i, \dots, \mu_i + 2\sigma_i)$. Additionally, each point within the retailers' demand distributions was assigned a probability corresponding to their Z-Score (the number of σ from the mean of the distribution). The assigned probabilities are provided in the table below:

Table 2: Probabilities associated with normal distribution points

Dist. Point	Z-Score	Probability P
1	-2	0.023
2	-1	0.14
3	0	0.68
4	1	0.14
5	2	0.023

When solving the two-stage stochastic LP for and considering only a single scenario (or sample = 1), the demand realization vector D utilized one point from each retailer's distribution selected at random.

3.3.2 Stochastic Gradient Descent

The first algorithm we employed to solve the SLP was stochastic gradient descent. To explain SGD, we will begin by defining the general form for an SLP problem as:

$$\begin{aligned} \min_{x \in \mathbb{R}_n} f(x) = c^T x + \sum_s p_s h_s(x) \\ s.t\ Ax = b \\ x \geq 0 \end{aligned} \tag{3}$$

where

$$\begin{aligned} h_s(x) &= \min d^T y \\ st. \qquad Dy &= [\zeta_s - C_s x] \\ y &\geq 0 \end{aligned} \tag{4}$$

In this formulation C, D, and ξ are functions given a specific scenario s. x is the first-stage decision variable vector and y is the second-stage decision contingent on the outcome of x. p_s is the probability of scenario s occurring, and $h_s(x)$ is the optimal value of the second-stage problem. $c^T x$ is a cost function of the first-stage.

Solving for the subgradient of $f(x)$, we arrive at the expression

$$v = -c - \sum_{s} p_s(-\pi_s^T C_s) \tag{5}$$

Where v is the subgradient and π is the scenario specific dual multiplier from the second-stage.

The SGD algorithm calls for solving the second-stage problem $h_s(x)$ for a sample of scenarios, updating the subgradient v of $f(x)$, and then using the subgradient to take a "step" towards the x decision that minimizes the first-stage objective. This process is iterative, thus with each step we approach the minimum value until the value of the subgradient is within a specified tolerance range.

Applying these steps to the general SLP form, to implement SGD on the transshipment problem we used the following procedure. Note that the samples used in SGD is a random sample of length M taken from the random variable distribution.

1. Given an initial x value (minimum of bounds), we first set $v = -c$
2. Using the initial x, we calculated $C_s x$ and updated the right-hand side of the second-stage constraints in $Equation$ 4.
3. Next, we solved $h_s(x)$ in $Equation$ 4 for each scenario s within the sample
4. Using these solutions, we gathered the dual multipliers π_s for each scenario
5. We then update the subgradient such that $v^+ \leftarrow v - p_s(\pi_s^T C_s)$
6. After solving all M scenarios, the first-stage decisions are updated such that $x^+ \leftarrow P_B(x - \alpha_k v^+)$ where α_k is the learning rate with momentum and P_B is a projection from the bounds.
7. $x^+$and v^+ are the new starting first-stage decisions and subgradient in the next iteration

The process detailed is repeated until the v^+ is within a specified tolerance. In our implementation, we tested the transshipment problem against three different values for M (10,50,100) and used a tolerance range of $[-0.1,\ 0.1]$ for the subgradient. By utilizing a momentum function in the learning rate α_k we were able to reach convergence in fewer iterations than without.

3.3.4 Bender's Decomposition

When implementing Bender's Decomposition, we start by referring back to the general SLP defined in the prior section. During Bender's we will say that $\sum_s p_s h_s(x)$is replaced with η is the first-stage problem, therefore it becomes

$$\min_{x \in X} c^T x + \eta \tag{6}$$

For x^t (x at iteration t) we include the constraint

$$\eta \geq \sum_{s} p_s[h_s(x^t) + v_s^T(x - x^t)] \tag{7}$$

Where v_s is again the subgradient of h_s at x^t. In this way, the constraint we have defined adds a cut to the function that reduces the feasible region of the optimal solution and moves us closer to the minimum. During Bender's we continue to iterate through the function, obtaining new x^t

values and adding additional constraints, until reaching the condition where the functions upper bound minus lower bound is within a specified tolerance.

In our implementation and solution of the transshipment problem using Benders, we applied the steps described as follows:

1. We began with defining the initial first stage and second stage as
$$\min_{S} \eta$$
$$\mathrm{h(S, D)} = \min \sum_{i} h_i e_i + \sum_{i \neq j} c_{ij} t_{ij} + \sum_{i} p_i r_i \tag{8}$$
With the same constraints as defined in *Section 3.2.3.* We also defined our initial upper bound UB = 1000 and lower bound LB = -1000

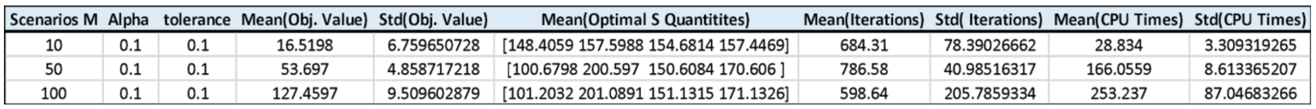
2. Next, we solved the first stage, η , to yield an S_0 value and new lower bound.
3. Given S_1, we solved the second-stage for all 5^4 scenarios to yield a new upper bound, S_2, and the coefficients of the hyperplane defined by the cut, α_1 and β_1.
4. The new constraint $\eta \geq \alpha_1 + B_1 S^1$ is added to the first stage LP.

Steps 2 through 4 are then repeated until we find that UB-LB$\leq \varepsilon$ where $\varepsilon = 0.1$ is the assigned tolerance for convergence.

3.3.2 Algorithm Performance & Comparison
In order to compare the performance of the SGD and Bender's algorithms, we measured two key metrics: CPU time and number of iterations before converging. Based on the structure of each algorithm, we predicted that convergence under SGD would occur faster (shorter CPU time) than under Bender's as SGD only requires a small sample of scenarios whereas Bender's considers and solves the second-stage of all 625 scenarios for each cut added. With respect to the number of iterations, we predicted that Bender's would require fewer as with each cut, the feasible region and bounds are significantly reduced. However, utilization of a momentum function on our learning rate in SGD will reduce the number of iterations for the algorithm.

3.4 Results
Results for the two-stage SLP using SGD are shown in the table below:

Table 3: SGD Results



Scenarios M	Alpha	tolerance	Mean(Obj. Value)	Std(Obj. Value)	Mean(Optimal S Quantitites)	Mean(Iterations)	Std(Iterations)	Mean(CPU Times)	Std(CPU Times)
10	0.1	0.1	16.5198	6.759650728	[148.4059 157.5988 154.6814 157.4469]	684.31	78.39026662	28.834	3.309319265
50	0.1	0.1	53.697	4.858717218	[100.6798 200.597 150.6084 170.606]	786.58	40.98516317	166.0559	8.613365207
100	0.1	0.1	127.4597	9.509602879	[101.2032 201.0891 151.1315 171.1326]	598.64	205.7859334	253.237	87.04683266

As mentioned, SGD was performed for samples of size 10, 50, and 100 scenarios. For each sample size, 100 trials were performed yielding mean and standard deviations for parameters. From the data it appears as though SGD under $M = 50$ scenarios yielded optimal S quantities and objective values viable for this problem. Despite yielding the best results, under $M = 50$, SGD experienced the highest number of average iterations (approx. 787) and the second highest CPU Time (166.06 seconds). This high iteration value likely drove the CPU Time and was a result of using $\alpha = 0.1$ as the initial learning rate for the problem. Additionally, altering the tolerance bounds to larger values may yield better performance characteristics.

Results for the two-stage SLP using Bender's Decomposition are shown in the table below:

Table 4: Benders Decomposition Results

Scenarios Sampled	Obj. Value	Optimal S Quantities	Iterations	CPU Time
625	80.52	[108.51, 214.07, 159.72, 187.7]	18	38.7504

As mentioned, Bender's solves all 625 scenarios in the second stage prior to making a cut. From the table, we see that obtaining the optima objective value and S decisions, the program required 18 iterations (i.e., 18 cuts were constructed), and it had a CPU time of 38.75 seconds.

We observe that Stochastic Gradient Descent with 50 sampled demands has comparable results with Bender's decomposition. The optimal objective of SGD was less than that of Benders yet the Optimal S quantities were in line with one another. In terms of performance SGD had a significantly higher CPU Time than Bender's to achieve comparable results. Additionally, while difficult to compare the number of iterations between the algorithms due to their operation, the number of iterations was very significant in SGD whereas Bender's only had to loop 18 times.

Under the Transshipment Problem, it appears as though Bender's is the more appropriate algorithm. This result, however, is highly dependent on the number of potential scenarios within a problem and not an absolute result across all problems. The transshipment problem had 625 scenarios which is relatively small for a stochastic problem, but traditional distribution and supply chains for larger retailers may be exponentially larger as more retailers are integrated and additional demand distributions are realized. As the problem size increases, our team would expect that the SGD under the same conditions as in this problem would experience relatively similar performance metrics, whereas Bender's performance would be significantly impacted due to the need to solve all scenarios. In the future we would like to implement a larger problem with significantly more scenarios to test this hypothesis.

3.5 Discussion

2.5.1 Lessons Learned & Project Challenges

In this Project, our team sought to better understand solution methods for Stochastic-Linear Programs. We utilized the "multilocation transshipment problem" to model an initial two-stage problem with uncertainty in demand realizations. To solve the two-stage program, we implemented two alternate stochastic optimization algorithms: Stochastic Gradient Descent (SGD) and Bender's Decomposition. By measuring and comparing the algorithms on two performance metrics, we were able to understand their applicability to stochastic optimization problems, and their ability to reduce computational requirements

We learned that SGD and Bender's are two comparable methods to go about solving stochastic optimization problems, with results yielding similar outcomes. The appropriateness of algorithm choice, however, is highly dependent on the size of problem (specifically the number of scenarios) that is at hand. For problems with a small magnitude of scenarios to loop over such as this problem, we find that Bender's was significantly more efficient. For larger problems with exponentially more scenarios, Bender's might prove to be infeasible thus additional testing is required.

During the project, our team encountered a number of challenges. Implementing the programs using Python and PySP proved to be a challenge in itself, but after gathering results we wanted to ensure we were getting the best possible performance from each of the algorithms. Tuning the hyperparameters was key in optimizing these results, and specifically with SGD, integrating momentum into the learning rate was able to allow convergence to occur under tighter tolerance windows. It was also challenging to compare results for the algorithms. Each algorithm operates significantly different from one another and the test case we constructed did not allow for larger numbers of scenarios to see how Bender's would change.

2.5.2 Future Work

To confirm the benefits of SGD on problems of large sizes, we want to explore increasing the number of retailers (from what we currently have of 4). Furthermore, we wanted to generalize the Stochastic Gradient Descent code to provide support for projections onto more general boundaries in the First-Stage problem using Euclidean norm minimization or Dykstra's algorithm so that the Stochastic Gradient Descent (according to how we currently implemented it) can be used on more general cases instead of only on the multi-shipment problem where we perform a nonnegative projection (it's required that decision variables be nonnegative). To improve on Stochastic Gradient Descent, we'd also like to implement SGD with momentum by decreasing the learning rate as we approach a minimum to avoid overshooting the minimum point. One of the assumptions we've made in this project is that demands are independent across retailers which might not be closest to reality. We want to explore the possible correlation between different demands/scenarios (maybe dependent demand) and using joint distribution sampling, i.e., sample the demands from a Multivariate Gaussian as opposed to sampling each of the retailer's expected demand independently.

3.6 References

[1] ***Code* - https://nbviewer.jupyter.org/github/jeffchenchengyi/ISE-533/blob/main/project2.ipynb**

[2] ***Ppt* - https://nbviewer.jupyter.org/github/jeffchenchengyi/ISE-533/blob/main/ISE%20533%20Project%202%20Multi-location%20Transshipment.pdf**

[3] Y. T. Herer, M. Tzur, and E. Yücesan, "The multilocation transshipment problem," *IIE Transactions*, vol. 38, no. 3, pp. 185–200, Apr. 2006.

[4] L. Zhao and S. Sen, "A Comparison of Sample-Path-Based Simulation-Optimization and Stochastic Decomposition for Multi-Location Transshipment Problems," *Proceedings of the 2006 Winter Simulation Conference*, 2006.

[5] "Stochastic Programming and Benders Decomposition," 2010, pp. 254–259.

4 LEO-WYNDOR REPORT

4.1 Introduction

In Project 3, we set out to investigate the behavior and performance of Predictive Stochastic Programming (PSP). Unlike ordinary stochastic programming (SP) where uncertainty is modeled using distributions and scenarios are unpredictable, PSP models are used for problems where there exists correlation between the response of a dataset to a series of predictors. In such cases, statistical learning may be integrated into traditional stochastic programming to allow for informed predictions of uncertainty based on "covariate" sets. This integration forms the basis for PSP.

During this project, a PSP formulation was applied to the popular 'Wyndor Glass Company' problem. In this production problem, Wyndor is a glass manufacturer which utilizes three production plants to assemble two different types of doors: aluminum-framed doors (y_A) and wood-framed doors (y_b). The sale of each door generates different profit to Wyndor, and each plant has different production limitations for the doors as described in the table provided below (Notice, the '--' indicates door type cannot be produced at location):

Table 5: Plant Production

Plant	A Production Time (Hours/Batch)	B Production Time (Hours/Batch)	Total Useable Hours
Plant 1	1	--	8
Plant 2	--	2	24
Plant 3	3	2	36
Profit	\$3000	\$5000	

Additionally, Wyndor generates sales for their products through an advertisement campaign that utilizes television (x_1) and radio (x_2) ads. Television ad slots cost Wyndor \$100 and radio slots \$500. Wyndor can purchase a maximum of 200 advertising time slots to market their products.

Wyndor's potential product sales information is uncertain for both doors, however sales are dependent on the marketing strategy adopted (as shown in the table below). In this way, sales and marketing form a covariate set where the amount Wyndor is capable of selling is the response (W) to the marketing strategy predictors (Z). As we will see in this report, the Wyndor problem lends itself to a two stage PSP where the first stage decisions are the marketing strategy $(x_1$ and $x_2)$ and the second stage decisions are the number of each door to produce $(y_a$ and $y_b)$ in order to maximize profit. The first stage decisions are thereby used to predict Wyndor's sales information which is incorporated into the second stage problem.

Table 6: Advertising Data Set

	TV	Radio	Sales
1	230.1	37.8	22.1
2	44.5	39.3	10.4
3	17.2	45.9	9.3
…	…	…	…
200	232.1	8.6	13.4

To solve the Wyndor PSP in this report, a methodology referred to as Learning Enabled Optimization (LEO) was applied. *Section 4.3* of this report will walk through each step of the LEO protocol as applied to the Wyndor problem, beginning with data preparation and linear regression, moving to statistical learning, stochastic optimization, and concluding with validation of the models. To understand the benefit of the LEO protocol as applied to PSPs, two statistical models will be utilized and compared: Deterministic Forecast (DF) and SAA-Empirical Additive Error (EAE/SAA). Ultimately, we would like to determine the modeling advantages LEO-PSP provides, assess the proposed validation approach established, and identify valid statistical models for this problem.

4.2 PSP Model Overview

4.2.1 Stochastic Programming v. Predictive Stochastic Programming

Prior to formulating the Wyndor problem as a two-stage PSP, it is important to understand the primary difference between Stochastic Programming and Predictive Stochastic Programming with decision dependent uncertainty. In Project 2, the Transshipment problem was modeled as an SP and incorporates random variables defined by discrete distributions, whereas this project focuses on uncertainty dependent on first-stage decisions.

The expression provided below provides the general form of a two-stage SP optimization problem:

$$\min_{x \in \boldsymbol{X}} c(x) \; + \; \mathbb{E}_{\widetilde{\omega}}[\, h(x, \widetilde{\omega})] \tag{1}$$

where $x \in \boldsymbol{X} \subseteq \mathbb{R}^{n_1}$is the vector of decision variables across a feasible set. $c(x)$ is defined as a cost function and h the outcome of the optimized second stage. In this case uncertainty is modeled by the random variable $\widetilde{\omega}$ where for SP, as we saw in project 2, is defined over a continuous or a discrete distribution. $\mathbb{E}$ denotes the expectation of the h outcome, and as we have seen can be approximated through various statistical methods such as the Sample Average Approximation (SAA).

In contrast, a two-stage PSP optimization is expressed in the general form shown below:

$$\min_{x \in \boldsymbol{X}} c(x) \; + \; \mathbb{E}_{\tilde{\varepsilon}}[\, h(x, \tilde{\varepsilon} | Z = z)] \tag{2}$$

In this case, $\tilde{\varepsilon}$ replaces $\widetilde{\omega}$ in modeling uncertainty and is defined as the decision dependent error random variable. This random error variable is a response to the predictor set $Z = z$ made during the first-stage decision. The outcomes h are defined using the outcomes of $\tilde{\varepsilon}$ and is assumed to be a convex function.

4.2.2 Wyndor Glass Co Formulation

Now that the general form of a PSP optimization problem is understood, the Wyndor scenario can be formulated using *Equation 2*. Below, the first stage and second stage Wyndor Model is provided:

First Stage: Marketing Strategy

$$f(x) = -0.1x_1 - 0.5x_2 + \mathbb{E}[H(x,\ \tilde{\varepsilon}_i \mid Z = z = x)] \tag{3a}$$

$$\text{s.t.} \quad x_1 + x_2 \leq 200 \tag{3b}$$

$$x_1 - 0.5x_2 \geq 0 \tag{3c}$$

$$L_1 := 0.7 \leq x_1 \leq U_1 := 200, L_2 := 0 \leq x_2 \leq U_2 := 50 \tag{3d}$$

Second Stage: Production Schedule

$$h(x,\ \boldsymbol{\varepsilon_i} \mid Z = z = x) \quad = \quad \text{Max} \quad 3y_A + 5y_B \tag{4a}$$

$$\text{s.t.} \quad y_A \leq 8 \tag{4b}$$

$$2y_B \leq 24 \tag{4c}$$

$$3y_A + 2y_B \leq 36 \tag{4d}$$

$$y_A + y_B \leq \boldsymbol{m_i(z, \varepsilon_i)} \tag{4e}$$

$$y_A, y_B \geq 0. \tag{4f}$$

As discussed, the first stage of the formulation regards the marketing strategy, with first stager decisions being the number of TV (x_1) and Radio (x_2) ad slots to purchase. The objective function of the first stage seeks to maximize Wyndor's profits as the money spent (in thousands) on x_1 and x_2 is subtracted from the expectation from the second stage. Notice that $\tilde{\varepsilon}_i$ is the error random variable derived from linear regression error values as will be discussed later in this report. *Equation 3b* constrains the number of slots purchased to less than or equal to 200 slots, and *Equation 3c* ensures that the number of TV ads purchased is at least half as many as the number of Radio ad purchased. The expressions in *Equations 3d* are upper and lower limits on the first stage decisions and are derived from the range of dataset in Table 6 which will be used for to construct regression.

The second stage of the Wyndor Model involves determining the company's product schedule with decisions made on the number of doors y_a and y_b to produce. The objective function, or $h(x, \varepsilon_i | Z = x)$, is the amount of profit generated from the production of the doors and is connected back to the objective of the first stage. Notice that this objective function is in response to a random error ε_i selected from the distribution. *Equations 4a - 4d* constrain the production of doors based on the plant production capabilities described in Table 5. In *Equation 4e*, $m_i(z, \varepsilon_i)$ is the amount of sales predicted from the first-stage decisions and random error variable using one of the three statistical models described in the following sections. This constrain ensures that the total number of doors produced is less than or equal to the total number of doors Wyndor is expected to sell based on the marketing strategy. *Equation 4f* constrains the number of doors produced to positive values.

In the following section we will utilize this Wyndor PSP developed in coordination with the data from Table 6 to perform the LEO protocol. The PSP model will serve as a test bed to introduce the two alternate statistical models and gather comparable results and validation intervals.

4.3 Technical Architecture

Learning Enabled Optimization is an iterative protocol that follows four primary steps as described in the illustration provided below:

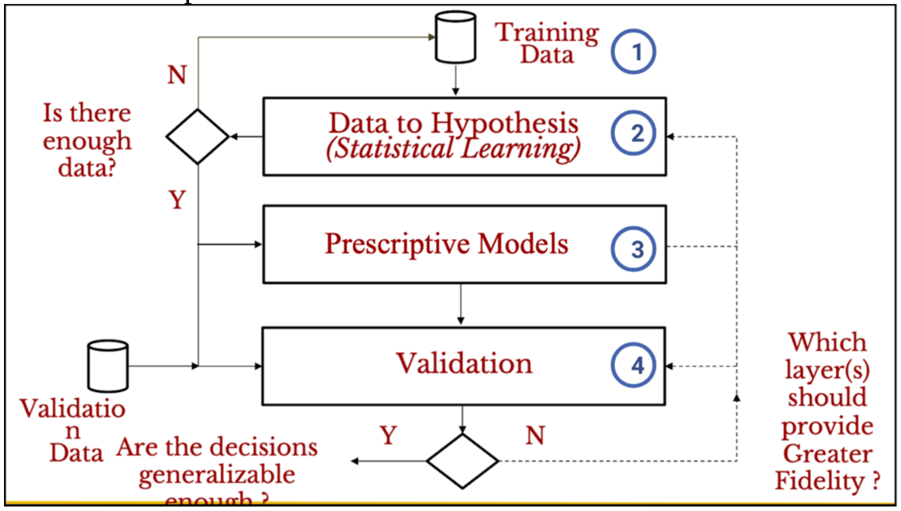

Training Data
1
N
Is there enough data?
Data to Hypothesis (Statistical Learning)
2
Y
Prescriptive Models
3
Validation
4
Validation Data
Y
N
Are the decisions generalizable enough ?
Which layer(s) should provide Greater Fidelity ?

Figure 5: LEO Protocol

In the first phase of the protocol, the dataset is split into training and validation subsets and multi-dimensional linear regression is conducted to obtain training and validation error terms. In the second step statistical models are implemented using the coefficients and error terms from the linear regression. In Step 3, these statistical models are integrated into a PSP model which is optimized to obtain the optimal objective value and the corresponding decisions. Finally, in the fourth step, the Model Validation Sample Average Estimate (MVSAE) is used to validate each model against a 95% confidence interval. In this section, we will walk through each step of the protocol as applied to the Wyndor problem.

Implementation of the LEO protocol and PSP models utilized Python as the primary programming language and the Pandas library for data preparation. Regressions and modeling were conducted using the scikit learn library and Pyomo Optimization package. Two solvers were tested including the GLPK solver and CBC solver.

4.3.1 Data Preparation, Linear Regression, & Error Terms

In the first step of the LEO-Wyndor protocol, the advertising dataset from Table 6 was split into two equally sized subsets: one for training and the second for validation. After splitting the data, a multi-dimensional linear regression was performed on the training dataset following the form shown below:

$$m(Z_i, \varepsilon) = \beta_0 + \sum_j \beta_j Z_{ij} + \varepsilon \tag{5}$$

According to this linear regression, the predicted response value is a function of the set of predictors Z_i and error term ε. The regression performed on our dataset was used to predict sales as a response to two predictors: $x_1$and x_2.

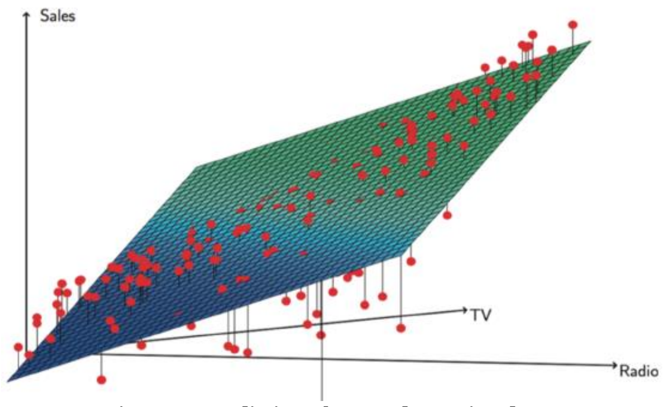


Figure 6: Prediction plane and associated error

The figure above shows the plane predicted by the initial linear regression plotted against the actual sales data. As seen in the plot, the error between the predicted and actual sales values is the vertical distance from the actual data to the prediction plane. Following the fitting of this regression, the linear regression was used to calculate error in the training set and validation set respectively. Below the expressions used to calculate the error terms are shown.

$$\varepsilon_{ti} = \omega_i - \beta_0 - \beta_1 x_{1i} - \beta_2 x_{2i} \tag{6}$$

$$\varepsilon_{vi} = \omega_i - \beta_0 - \beta_1 x_{1i} - \beta_2 x_{2i} \tag{7}$$

In these expressions i is an index to refer to each observation from the training and validation sets. ε_{ti} and ε_{vi} refer to the error value for a specific training or validation observation respectively. The β coefficients are the same coefficients from the performed linear regression.

Following the conduction of the linear regression, the validation and training errors were plotted on a QQ plot as shown in the graph below (with validation terms on the x and training on the y)

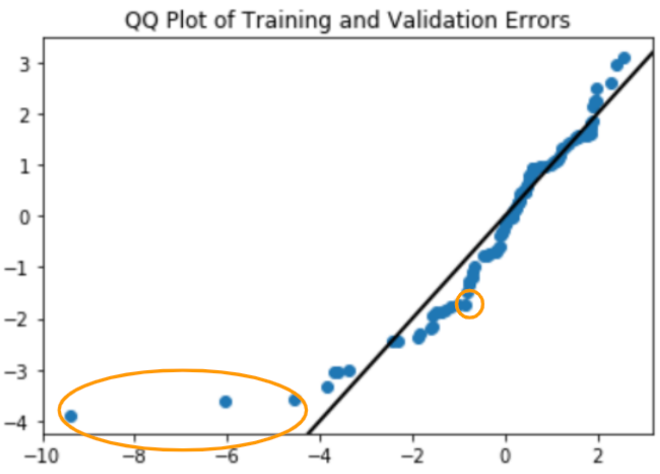


Figure 7: QQ Plot of Training vs. Validation Errors

Using this plot and by isolating error values greater than two standard deviations from the mean training error, we identified error outliers (circled in orange) and removed them from the training dataset. After removal, our team performed a second linear regression on the training set without outliers and recalculated the training and validation error terms. From the reperformed regression we gathered the following regression coefficients: $\beta_0 = 3.103, \beta_1 = 0.042, \beta_2 = 0.210$.

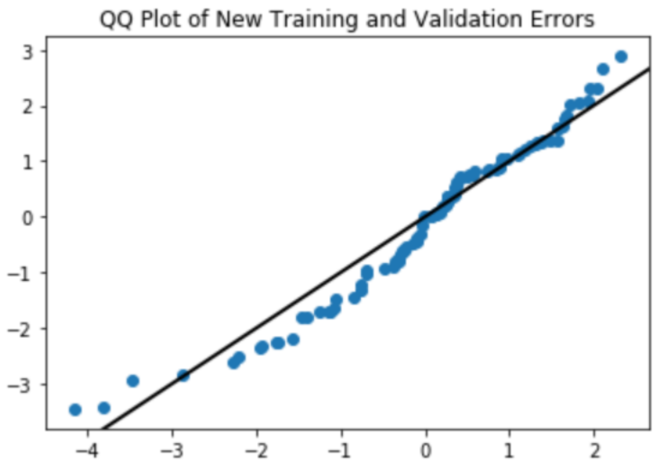


Figure 8: *QQ Plot of new training and validation errors. Notice all error values fall within 2SD of the mean.*

4.3.2 Deterministic Forecast & Wyndor Integration

The first statistical model used to gather computational results was the Deterministic Forecast (DF) Model. For DF, sales predictions are made according to the expression shown below:

$$m(x, \varepsilon) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 \tag{8}$$

This expression assumes that the training error used to develop the regression is equal to zero. Therefore, the model is the same equation as that of the regression plane and does not incorporate error into predictions.

In order to incorporate the deterministic forecast model into the Wyndor PSP, the DF expression is substituted into the right side of *Constraint 4e* of the second-stage problem. While implementing the two-stage Wyndor PSP using PySP, the original Stochastic Decomposition solver we utilized was not functioning correctly, thus our team pivoted to an All-In-One formulation and utilized Pyomo's solvers. The two-stage optimization problem is re-expressed as the equivalent All-In-One program shown below:

$$\text{Max} \; -0.1x_1 - 0.5x_2 + \; 3y_A + 5y_B \tag{9a}$$

$$\text{s.t.} \quad x_1 + x_2 \leq 200 \tag{9b}$$

$$x_1 - 0.5x_2 \geq 0 \tag{9c}$$

$$y_A \leq 8 \tag{9d}$$

$$2y_B \leq 24 \tag{9e}$$

$$3y_A + 2y_B \leq 36 \tag{9f}$$

$$-\beta_1 x_1 - \beta_2 x_2 + y_A + y_B \leq \beta_0 \tag{9g}$$

$$y_A, y_B \geq 0 \tag{9h}$$

$$L_1 \leq x_1 \leq U_1, \quad L_2 \leq x_2 \leq U_2 \tag{9i}$$

There are a few key features to make note of in the All-In-One formulation. The decision variables are the same as the those in the two-stage, and the constraints are largely unchanged. The expectation $\mathbb{E}$ in the objective function is no longer necessary, as again, the model assumes 0 uncertainty in predictions. Notice that *Constraint 9g* is the location where the DF sales prediction is incorporated, where decision variables are placed on the left side and the constant β_0 on the right.

4.3.3 EAE/SAA Model & Wyndor Integration

The second statistical model used to gather computational results was the Empirical Additive Error – SAA model. In this model, the predicted sales figure utilizes the expression of the regression and adds a generic outcome of the error random variable as seen in the expression below:

$$m(x, \varepsilon_0) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \varepsilon_{ti} \tag{10}$$

In this case, the error term ε_{ti} represents the stochastic uncertainty introduced into the model. For a single sample, one error term from the training error set is selected, such that each error term ε_{ti} generates a unique prediction of the sales figure.

As with the DF formulation, the Wyndor PSP incorporating the EAE/SAA model is shown below:

$$\text{Max } -0.1x_1 - 0.5x_2 + \frac{1}{N}\sum_{i=1}^{N} 3y_{Ai} + 5y_{Bi} \tag{11a}$$

$$\text{s.t.} \quad x_1 + x_2 \leq 200 \tag{11b}$$

$$x_1 - 0.5x_2 \geq 0 \tag{11c}$$

$$y_{Ai} \leq 8 \quad i = 1, \ldots, N \tag{11d}$$

$$2y_{Bi} \leq 24 \quad i = 1, \ldots, N \tag{11e}$$

$$3y_{Ai} + 2y_{Bi} \leq 36 \quad i = 1, \ldots, N \tag{11f}$$

$$-\beta_1 x_1 - \beta_2 x_2 + y_{Ai} + y_{Bi} \leq \beta_0 + \varepsilon_{ti} \quad i = 1, \ldots, N \tag{11g}$$

$$L_1 \leq x_1 \leq U_1, \quad L_2 \leq x_2 \leq U_2, y_{Ai}, y_{Bi} \geq 0. \tag{11h}$$

First notice in this program, as with the DF program, the EAE model is incorporate into *Constraint 11g.* Looking now at the objective function, the expectation of the second stage has been replaced by the Sample Average Approximation of the profit from the production of the doors. In this way, the model takes the average profit of N samples, with a random error variable selected from the training errors for each individual sample. In our experimentation, $N = 200$ samples were used, thus 200 error variables were selected from the error distribution with replacement. Finally, notice that y_A and y_B have been replaced by y_{Ai} and y_{Bi} in this program. The training error set has 96 datapoints (after outliers were removed), therefore 96 distinct pairs of y_{Ai} and y_{Bi} exist. The total number of decision variables is 194 (x_1,x_2, and 96 pairs of y_{Ai} and y_{Bi}).

4.3.4 Validation Approach

In order to validate the outcomes of the DF and EAE model, we utilized a validation methodology referred to as Model Validation Sample Average Estimate (MVSAE). The goal of this approach is to use the validation dataset to solve the PSP and construct a 95% confidence and determine if the training results were contained inside the interval.

In the first step of MVSAE, the DF and EAE models were solved according to *Sections 4.3.2 and 4.3.3*, and the optimal objective values and decision variables $\hat{x}_1$ and $\hat{x}_2$ for each model were noted. Next the respective DF and EAE $\hat{x}_1$ and $\hat{x}_2$ were fixed as known parameters in the second-stage program. The second-stage LP was then solved $M = 1000$ times using the fixed parameters and by selecting a single random validation error term for each replication. The mean objective value and standard deviation of the second-stage solution sample was calculated, and a 95% confidence interval was constructed. Finally, results from the training set solutions were compared against these confidence intervals

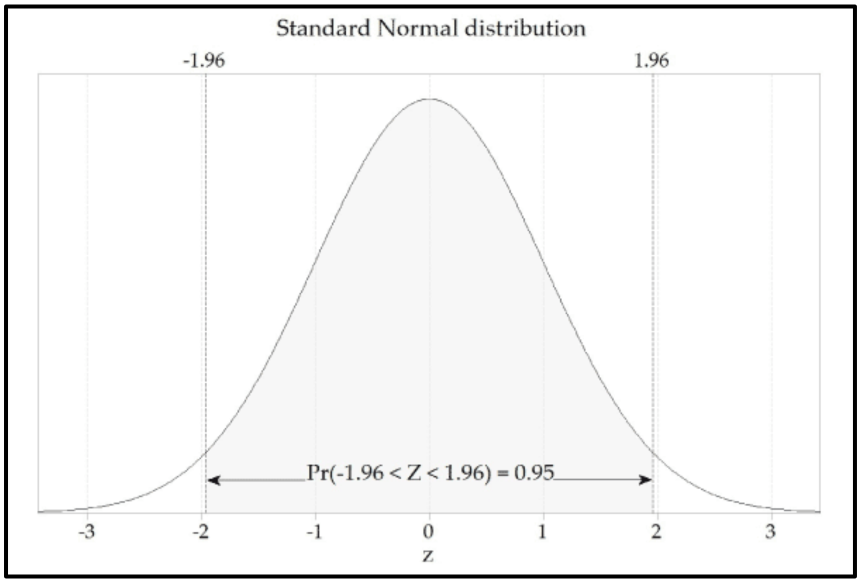


Figure 9: Standard Deviation values used to construct 95% CI

One of the challenges our team had to overcome when gathering results was determining the number of replications M to use for constructing the confidence intervals. By increasing the number of replications, the variance in the upper and lower boundaries of the interval were reduced. At $M = 1000$ replications, variance in the confidence interval was very slight with each incremental change, thus 1000 replications were used for both models' confidence intervals.

4.4 Results

The table below provides a summary of the LEO-Wyndor training and validation results calculated for the Deterministic and EAE/SAA statistical models:

Table 7: LEO-Wyndor Results for DF and EAE/SAA models

Stat. Model	x1 (TV)	x2 (Radio)	MPO (in $K)	MVSAE (in $K)
Deterministic (DF)	173.4341	26.5659	41.3736	[39.811, 40.136]
EAE/SAA	180.1731	19.8269	40.3400	[40.252, 40.723]

The x_1, x_2 and MPO columns provide the decisions made for the marketing strategy and the maximum profit after solving the two-stage problems using the training dataset. Notice that the DF model favored a higher proportion of radio ads than the EAE/SAA and found the max profit to be $41,373 whereas the EAE/SAA found the optimal objective to be $40,340.

The MVSAE column provides the validation confidence interval for both models. From the results, we can see that the predicted optimal objective value for the DF model was greater than the upper bound of its MVSAE confidence interval. This tells us, with 95%, that the outcome of the DF is not the accurate results for the LEO-Wyndor problem. By not accounting for error in predictions, the model did not prescribe accurate decisions, thus it should not be used as a viable statistical model for Predictive Stochastic Programming problems.

The predicted optimal objective value using the EAE/SAA model did fall inside the corresponding MSVAE confidence interval. This indicates that the EAE/SAA model was able to produce results and prescribe actions similar to those from the validation set. By incorporating uncertainty through the random error variable and using the LEO protocol, the EAE model proved to be a viable statistical model for Predictive Stochastic Programming problems.

4.5 Discussion

2.5.1 Lessons Learned and Project Challenges

In this report, we set out investigate the performance of Predictive Stochastic Programming, and better understand how integrating statistical learning and stochastic programming can enable more reliable predictions and solutions when uncertainty is present in covariate sets. During the project, we applied a PSP formulation to the 'Wyndor Glass Co.' problem and utilized a methodology know as Learning Enabled Optimization to study the correlation between uncertainty in the problem. Within LEO, we utilized two statistical models (DF & EAE/SAA) when measuring the effect of incorporating error. From the results, it was determined that for PSP models, it is crucial to account for error uncertainty during modeling, as the prescribed actions and results provide more reliable decisions than when it is unaccounted for.

Additionally, through our experimentation, we were able to prove the LEO-PSP protocol was a viable process in gathering and verifying computational results. LEO provides a step-by-step process useful in comparing independent models and testing validity of predictions. Alternate validation approaches may be used within LEO, but the MSVAE approach performed as expected and proved that the EAE/SAA model was statistically appropriate for the problem.

During the process, our team encountered a number of challenges, with some mentioned in the prior sections. During data preparation, some of the error values we initially encountered fell outside of what we deemed statistically acceptable. Thus, we had to identify outliers within the data and repeat the regression process. When implementing the two-stage model using PySP, the team encountered repeated problems and errors with the Stochastic Decomposition solver we planned to utilize leading to large delays. Our team was able to act creatively and adapt to the challenge by utilizing an All-In-One model to get computational results. Finally, during tuning of the number of samples during SAA and the repetitions during MSVAE, our team took an iterative, yet time-intensive, approach that ultimately allowed for powerful comparisons to be made.

2.5.2 Future Work Extension

In the future, our team would like to extend the project across additional statistical models to better understand their validity for PSP scenarios. The next model we plan to implement is the Normally Distributed Uncorrelated (NDU) model. In this model, sales are predicted under the condition that correlation between the regression's coefficients are ignored. The model takes the form shown below:

$$m(z,\ \varepsilon) = (\beta_0 + \varepsilon_0) + (\beta_1 + \varepsilon_1)x_1 + (\beta_2 + \varepsilon_2)x_2 \tag{12}$$

In this case, the error outcomes $(\varepsilon_0, \varepsilon_1, \varepsilon_2)$ are associated with NDU random error distributions with means of zero and organized into a variance matrix $\sum_B$. To determine the error outcomes for specific scenarios, the optimization function below is solved

$$\min_{\substack{\varepsilon \\ \varepsilon^T \sum_B^{-1}(\varepsilon) \le X_p^2(\alpha)}} [\,(\hat{\beta}_0 + \varepsilon_{i0}) + (\hat{\beta}_1 + \varepsilon_{i1})Z_{i1} + (\hat{\beta}_2 + \varepsilon_{i2})Z_{i2} - W_i]\text{^}2 \tag{13}$$

Where (Z_i, W_i) is a specific data point, α is used in determining the probability level and $(\varepsilon_{i0}, \varepsilon_{i1}, \varepsilon_{i2})$ the error associated with the specified data point.

4.6 References

[1] ***Code*** **-** ***https://nbviewer.jupyter.org/github/jeffchenchengyi/ISE-533/blob/main/project3.ipynb***

[2] ***Ppt*** **-** ***https://nbviewer.jupyter.org/github/jeffchenchengyi/ISE-533/blob/main/ISE%20533%20Project%203%20LEO%20Wyndor.pdf***

[3] Y. T. Herer, M. Tzur, and E. Yücesan, “The multilocation transshipment problem”, *IIE Transactions*, vol. 38, no. 3, pp. 185–200, Apr. 2006.

[4] J. Xu, “Learning Enabled Optimization(LEO) Software”, *Github*. USC3DLAB, 26-Jul-2018. https://github.com/USC3DLAB/learning-based-optimization

[5] Y. Deng, S. Sen, and J. Xu, “ISE 533 Notes: Integrative Analytics with Cross-Sectional Data”, *ISE533 Blackboard*, pp. 1–24, 2021.

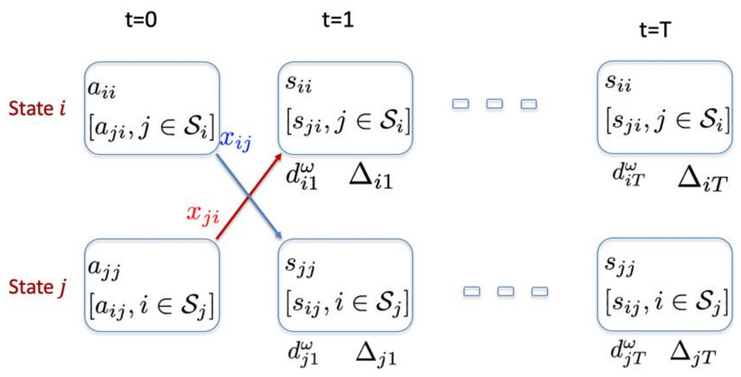
5 COVID – 19 REPORT

5.1 Introduction

In Project 4, we want to move beyond the deterministic optimization and make a decision in the face of uncertainty. In reality, we have to make decisions that involve uncertainty, and we are trying to optimize now for minimal future recourse. Different from projects we have done before, in project 4, the output we get from the first decision will influence the input we use in the next decision. Therefore, the better model has more accurate results over time.

During this project, we implement a deterministic and simple recourse model in stochastic optimization to make the decisions for nationwide ventilator allocation problems during the COVID-19 pandemic. In this allocation problem, many states in the U.S. have faced shortages of medical resources because of the surge in the number of patients suffering from COVID-19. As many projections indicate, the situation will be far worse in coming months. The upcoming challenge is not only due to the exponential growth in cases but also because of inherent uncertainty and lags associated with disease progression. In this project, we plan to build a two-stage stochastic optimization model (Simple Recourse model) and all-in-one optimization model (Deterministic model) to find the optimal allocation of ventilators among US.

In this problem, the distribution of the confirmed cases in the US is unbalanced, these confirmed cases accumulate in different areas at different times. It is reported that many states short medical resources during the previous surge. And the next surge is predicted to have a higher peak. Coordination between states is needed. It can reduce the unmet demand, but in the absence of coordination tools, there was no effective mechanism to help states meet their ventilator needs. The tools, or the allocation algorithms, would allow states to share ventilator inventory, thus minimizing shortages across the country.



t=0
t=1
t=T
State i
State j
a_{ii}
$[a_{ji}, j \in \mathcal{S}_i]$
a_{jj}
$[a_{ij}, i \in \mathcal{S}_j]$
x_{ij}
x_{ji}
s_{ii}
$[s_{ji}, j \in \mathcal{S}_i]$
s_{jj}
$[s_{ij}, i \in \mathcal{S}_j]$
d_{i1}^{ω} Δ_{i1}
d_{j1}^{ω} Δ_{j1}
d_{iT}^{ω} Δ_{iT}
d_{jT}^{ω} Δ_{jT}

Figure 9: here is a figure illustrating the allocation process.

First, we assume each state can only transfer the resources to its neighboring states. The SNS can send ventilators to all states. No state will send ventilators to the SNS. We have the initial state-owned ventilator data for all states, see 'ini_data.csv'. We specified the PCOV_ID = 0.8 as the percentage of ventilators will be used for COVID19 patients. The decisions for allocation will be implemented every week. The end condition will be the initial condition for the allocation problem in the following week.

In total, we will solve 10 allocation problems (from 03/25 to 06/02), where the initial condition is based on the decision in the past. The performance of our algorithm will be evaluated based on the data in 'eval' folder, which is the real demand data from 03/25 to 06/02.

5.2 Model Overview

5.2.1 Notation explanation

In our model, we consider an interval of time, say t=7 days. And totally we need to make decisions for 10 weeks(times). The objective of the model is to minimize the unmet demand for resources by optimally distributing them across all states, this is done based on a 7-day projection of demand across the 51 states (including Washington, D.C.)
We will begin with notation which spans all subsections, and then, proceed to notation which is specific to each particular section (or subsection).

5.2.1.1 Sets:

- $S = \{0,1,\ldots,50\}$ the index set of all states and Washington, D.C.
- $S^+ = S \cup \{51\}$, where 51 is the index for the Strategic National Stockpile. We assume that once SNS transfers the ventilators to a state, these ventilators belong to the state.
- $S_j = a\ subset\ of\ S$, which includes the indexes of the states able to ship resources to location j, $\forall j \in S^+$.
- $T = \{0,1,\ldots,T\}$ $T = 7$ the set of days which the plan covers.

5.2.1.2 Data:

In order to set up these allocation models, we need to have the data associated with the demand, supply, and policy considerations for each state. The supply data includes the number of ventilators that are available for COVID-19 patients in all states and in the Strategic National Stockpile. The policy definitions (or functions) reflect decision-rules which may be provided by decision-makers (e.g., governors or their surrogates). Such parameters may constrain the maximum number of resources shipped to neighboring states or set a minimum availability of resources. The data used for the decision-making models may be summarized as follows:

- a_{jj} = the initial total number of resources that are available at $\forall j \in S^+$, and belongs to state j.
- a_{ij} = the initial total number of resources that are available at $\forall j \in S^+$, and belongs to $i \in S_j$
- d_{jt} = the expected demand for resources at $j \in S$ on day $t \in T$
- U_{ji} = the maximum number of ventilators belonging to $j \in S^+$, and can be transferred to $i \in S$.
- G_j = the minimum number of ventilators belonging to state j which must be retained at location $j \in S$.
- θ_j = the maximum fraction of a_{jj} used for U_{ji} , that is $U_{ji} = \theta_j\, a_{jj}$
- ρ_j = the maximum fraction of a_{jj} used for G_j , that is $G_j = \rho_j\, a_{jj}$

5.2.1.3 Variables:

- x_{ij} = the number of ventilators shipped from location $i \in S^+$ to location $j \in S^+$at time 0. The flow matrix is X, with x_{ij} as the element at (i,j)

- s_{jj} = the total number of ventilators that are available in j ∈ S^+ after initial allocation and belongs to j.
- s_{ij} = the total number of ventilators that are available in j ∈ S^+ after initial allocation and belongs to $i \in S_j$.
- Δ_{jt}= unmet demand for ventilators at location $j \in S$ on day $t \in T$. The matrix representation is Δ.

5.2.2 Deterministic Model

When we start building the model, we first consider a deterministic decision model which seeks plans according to point-forecasts (i.e., assume that there are no errors in forecasting). This is unrealistic because prediction errors are inevitable. We aim to minimize the total amount of unmet demand across the country in the periods of 7 days (for 10 weeks).
The Deterministic Model provided as bellows:

$$\text{Min} \quad \sum_{j\in\mathcal{S},t\in\mathcal{T}} \Delta_{jt} + \lambda\left(\sum_{j\in\mathcal{S},i\in\mathcal{S}_j} s_{ij}\right) \tag{1a}$$

$$s.t. \quad s_{jj} = a_{jj} - \sum_{i\in\mathcal{S}_j} x_{ji} + x_{51,j}, \forall j \in \mathcal{S} \tag{1b}$$

$$s_{51,51} = a_{51,51} - \sum_{j\in\mathcal{S}} x_{51,j} \tag{1c}$$

$$s_{ij} = a_{ij} + x_{ij}, \forall j \in \mathcal{S}, i \in \mathcal{S}_j \tag{1d}$$

$$\Delta_{jt} \geq \max\{0, d_{jt} - s_{jj} - \sum_{i\in\mathcal{S}_j} s_{ij}\}, \forall j \in \mathcal{S}, t \in \mathcal{T} \tag{1e}$$

$$s_{jj} \geq G_j, \forall j \in \mathcal{S}^+ \tag{1f}$$

$$-a_{ij} \leq x_{ij} \leq U_{ij}, \forall i, j \in \mathcal{S}^+ \tag{1g}$$

The constraints include flow balance constraints (1b -1d) for all locations, policy constraints with Gj and Uij (1f -1g), and the unmet demand (1e), which is either zero or the difference between the demand and available ventilators. Additionally, in order to avoid unnecessary sharing, we need to penalize the number of ventilators which need to be borrowed from other states. Detailed explanation of all constraints will be shown in the technical architecture.

5.2.3 Simple Recourse Model

After building the deterministic model without uncertainty, now we take advantage of the forecast uncertainty, each demand has multiple possible paths. The reason why we need to consider the uncertain data is because the deterministic model with data certainty will exacerbate the bias due to optimization.

We replace point forecasts of the previous subsection with a collection of demand samples, where each sample represents a sample-path of potential demands for each day of a week for any given state.
First, we define the auxiliary variables

$$u_j = s_{jj} + \sum_{i \epsilon S_j} s_{ij}$$

Then, the problem will be presented by a two-stage stochastic optimization model as follows:

$$\text{Min} \quad \lambda \left(\sum_{j \in \mathcal{S}, i \in \mathcal{S}_j} s_{ij} \right) + \hat{\mathbb{E}}[h(s, X, \tilde{\omega})] \tag{2a}$$

$$s.t. (1b) - (1d), (1f) - (1g) \tag{2b}$$

The second stage formulation is the expectation of unmet demand with different scenarios, we have

$$h(s, X, \omega) = \text{Min} \sum_{j \in \mathcal{S}, t \in \mathcal{T}} \Delta_{jt}^{\omega} \tag{3a}$$

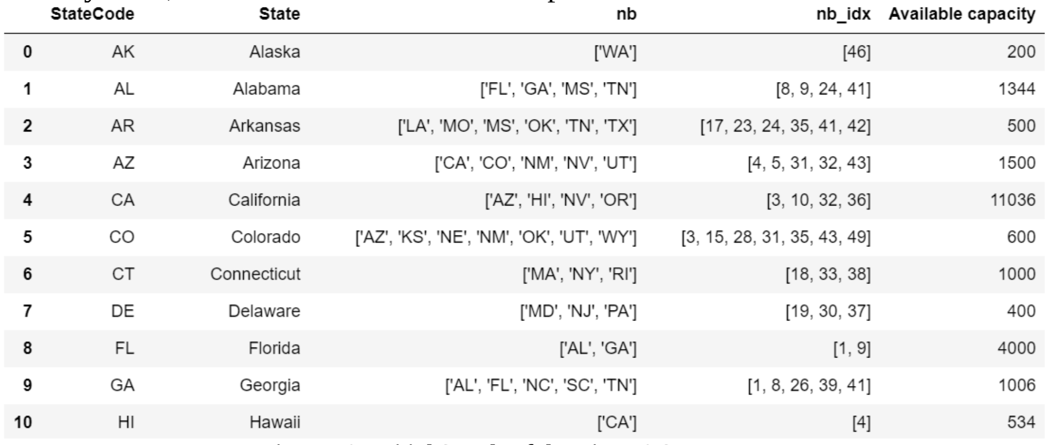
$$s.t. \quad \Delta_{jt}^{\omega} \geq \max\{0, d_{jt}^{\omega} - s_{jj} - \sum_{i \in \mathcal{S}_j} s_{ij}\}, \ \forall j \in \mathcal{S}, t \in \mathcal{T} \tag{3b}$$

With the formulation of two-stage stochastic optimization with lots of scenarios, we use the classic Benders Decomposition algorithm to evaluate the subgradients of h_j.

5.3 Technical Architecture

5.3.1 Data Structure

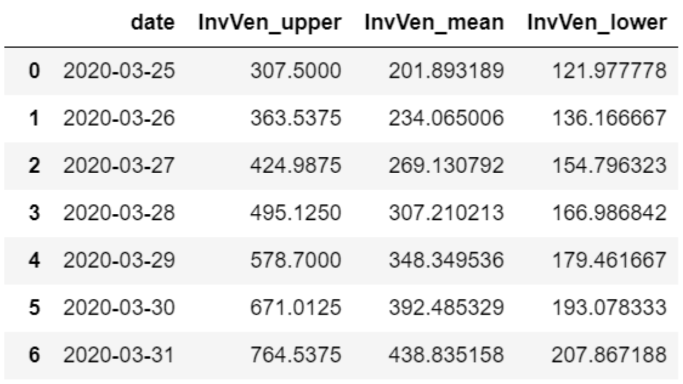
As mentioned in the previous section, there are three datasets that we will use in this project. The supply data includes the initial number of ventilators that are available for COVID-19 patients. The prediction data gives us the potential possible demand for ventilators for each day of a week for any given state, and the demand has three paths, lower, mean, and upper estimates. The real demand data, which includes the actual number of ventilators needed for COVID-19 patients every day in every state, is used for evaluation of the performance of our models.



	StateCode	State	nb	nb_idx	Available capacity
0	AK	Alaska	['WA']	[46]	200
1	AL	Alabama	['FL', 'GA', 'MS', 'TN']	[8, 9, 24, 41]	1344
2	AR	Arkansas	['LA', 'MO', 'MS', 'OK', 'TN', 'TX']	[17, 23, 24, 35, 41, 42]	500
3	AZ	Arizona	['CA', 'CO', 'NM', 'NV', 'UT']	[4, 5, 31, 32, 43]	1500
4	CA	California	['AZ', 'HI', 'NV', 'OR']	[3, 10, 32, 36]	11036
5	CO	Colorado	['AZ', 'KS', 'NE', 'NM', 'OK', 'UT', 'WY']	[3, 15, 28, 31, 35, 43, 49]	600
6	CT	Connecticut	['MA', 'NY', 'RI']	[18, 33, 38]	1000
7	DE	Delaware	['MD', 'NJ', 'PA']	[19, 30, 37]	400
8	FL	Florida	['AL', 'GA']	[1, 9]	4000
9	GA	Georgia	['AL', 'FL', 'NC', 'SC', 'TN']	[1, 8, 26, 39, 41]	1006
10	HI	Hawaii	['CA']	[4]	534

Figure 10: Initial Supply of the First 10 States

The supply data has 52 rows and 5 columns. The columns are state abbreviation, state name, neighbor states, index for the neighbor states, and available number of ventilators at the state. Each row represents the information for the specific state. For example, on March 24, 2020, we use the data in ‘ini_data.csv’ as the initial condition. The number of ventilators owned by Alaska and are available in Alaska on March 24, 2020 is 200*0.8=160. It has one neighbor state Washington which has an index of 46.

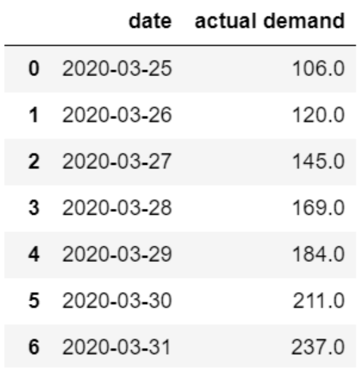


	date	InvVen_upper	InvVen_mean	InvVen_lower
0	2020-03-25	307.5000	201.893189	121.977778
1	2020-03-26	363.5375	234.065006	136.166667
2	2020-03-27	424.9875	269.130792	154.796323
3	2020-03-28	495.1250	307.210213	166.986842
4	2020-03-29	578.7000	348.349536	179.461667
5	2020-03-30	671.0125	392.485329	193.078333
6	2020-03-31	764.5375	438.835158	207.867188

Figure 11: Actual Demand of the first week in state California

Here is an example of the predicted demand of the first week from California. On the first day March 25th, 2020, the predicted upper number of ventilators needed in California is 307.5; the predicted average number of ventilators needed in California is 202; the predicted lower number of ventilators needed in California is 122. On each day, the demand can follow any one of the three paths, which gives us the uncertainty when incorporating the prediction data. Suppose California is corresponding to j = 4. Then for the first path we have $d_{41}(\omega_4^1) = 307.5$, $d_{42}(\omega_4^1) = 363.5375, \ldots\ldots$ For the second path, $d_{41}(\omega_4^2) = 201.893189,\ d_{42}(\omega_4^2) = 234.065006, \ldots\ldots$ Thus, we have the data for

$$d_{jt}\left(\omega_j^r\right), \forall\, j \in S, \forall t \in T, r = 1,2,3.$$



	date	actual demand
0	2020-03-25	106.0
1	2020-03-26	120.0
2	2020-03-27	145.0
3	2020-03-28	169.0
4	2020-03-29	184.0
5	2020-03-30	211.0
6	2020-03-31	237.0

Figure 12: Actual Demand of the first week in state California

After conducting the allocation optimization, we evaluate the performance of these models by comparing the optimized supply with actual demand. Figure 3 shows part of the evaluation data of the first week in California. For example, on March 25th, California needed 106 ventilators for COVID-19 patients.

5.3.2 Point Forecasts – Solving Deterministic Model

The allocation problem of ventilators aims to minimize the total amount of unmet demand across the country on a weekly basis. In point forecasts, we incorporate only the mean demand path and treat it as a definite prediction in demand, ignoring the errors in forecast. Hence, our model for point forecasts will be a deterministic model. For this model, the decisions we need to make are the number of ventilators shipped from one location to another at t = 0.

$$\text{Min} \quad \sum_{j\in\mathcal{S},t\in\mathcal{T}} \Delta_{jt} + \lambda \left(\sum_{j\in\mathcal{S},i\in\mathcal{S}_j} s_{ij} \right) \tag{4a}$$

$$\text{s.t.} \quad s_{jj} = a_{jj} - \sum_{i\in\mathcal{S}_j} x_{ji} + x_{51,j}, \forall j \in \mathcal{S} \tag{4b}$$

$$s_{51,51} = a_{51,51} - \sum_{j\in\mathcal{S}} x_{51,j} \tag{4c}$$

$$s_{ij} = a_{ij} + x_{ij}, \forall j \in \mathcal{S}, i \in \mathcal{S}_j \tag{4d}$$

$$\Delta_{jt} \geq \max\{0, d_{jt} - s_{jj} - \sum_{i\in\mathcal{S}_j} s_{ij}\}, \forall j \in \mathcal{S}, t \in \mathcal{T} \tag{4e}$$

$$s_{jj} \geq G_j, \forall j \in \mathcal{S}^+ \tag{4f}$$

$$-a_{ij} \leq x_{ij} \leq U_{ij}, \forall i, j \in \mathcal{S}^+ \tag{4g}$$

Although one of the inequalities in the allocation problem involves a nonlinear function due to the “max” operator, the problem can be solved as a linear program by replacing the “max” operator with two inequalities requiring each term inside the “max” to be less-than-or-equal-to Δjt. Constraint (4b) indicates the balance of stock in each state, where the right-hand side is the initial stock minus the outflows plus the support from SNS. Constraint (4c) shows the stock in SNS, which equals the initial amount minus the support to states. Constraint (4d) denotes shipments of ventilators among states. Constraints (4f) and (4g) represent the policy restrictions on the number of ventilators that can be transferred to other states and retained with state.

The first step we took is to solve the LP in Pyomo for the first week to see if the implementation was correct. After obtaining a reasonable result, we applied a looping algorithm to solve the allocation problem for 10 weeks iteratively. Notice that, the amount of supply after allocation will be the initial stock condition for the coming week. For example, after solving the problem from 03/25 to 03/31, we are able to obtain the allocation result for the first week; then we will use this result as the input stock to solve the second allocation problem, which is from 04/01 to 04/07. This process is iterated for 10 weeks from 03/25 to 06/02. The result should show the allocation of ventilators each week in each state.

5.3.3 Forecast with Uncertainty – Solving Simple Recourse Model with BD

Compared with the previous model, in the simple recourse model, we accommodate the uncertainty associated with prediction of demand. In a situation where errors are inevitable in prediction, we assume that using such a stochastic model will reduce or eliminate the biases derived during the process of optimization, and therefore will be a better option than the deterministic model. The problem is now formulated in two stages as follows:

$$\text{Min} \quad \lambda \left(\sum_{j \in \mathcal{S}, i \in \mathcal{S}_j} s_{ij} \right) + \sum_{j \in \mathcal{S}} \mathbb{E}[h_j(u_j, \tilde{\omega}_j)] \tag{5a}$$

$$s.t. \quad (1b) - (1d), (1f) - (1g) \tag{5b}$$

The first stage objective becomes the penalty for the number of ventilators belonging to other states plus the expectation of total unmet demand under all scenarios.

$$h_j(u_j, \omega_j^r) = \text{Min} \sum_{t \in \mathcal{T}} \Delta_{jt}^{\omega_j} \tag{6a}$$

$$s.t. \quad \Delta_{jt}^{\omega_j} \geq \max\{0, d_{jt}(\omega_j^r) - u_j\}, \ \forall t \in \mathcal{T} \tag{6b}$$

In the second stage, the only constraint we have is the estimation of unmet demand. The unmet demand is either the shortfall between a specific demand path and the allocated supply decision on that day or zero if supply is greater than the predicted demand.

The approach we applied to solve this two-stage problem is Benders Decomposition. The first step is to solve the first stage problem with current constraints. We will treat the objective value as our lower bound. Then we solve the second stage problem with every scenario, in this case, it will be the 3 demand paths for 51 states for 7 days. From solving the second stage problem, we will collect the objective with the penalty as upper bound and the subgradient using the dual problem. Because of the special structure of the simple recourse model, the subgradient is simply the coefficient of *u*, which is -1 (negative one).

After acquiring the subgradient, it is important to add the Benders cuts back into the first stage problem, so that the feasible region of the decision variables narrows down and approaches the optimal value. One challenge we encountered is that the algorithm cannot converge due to the large gap between upper bound and lower bound. Later we found out that we did not express the Benders cuts correctly, which resulted in the lower bound and upper bound not moving closer to each other. After fixing the problem, the Benders Decomposition algorithm was able to converge at a tolerance level 0.001.

Like what we did in the deterministic model, we need to iteratively solve the two-stage problem using Benders Decomposition for the consecutive 10 weeks. We can then compare the allocation results with the actual demand data during the COVID-19 pandemic season in 2020.

5.4 Results

The table below provides a summary of the Deterministic and Simple Recourse model results for Unmet Demand:

Table 8: Deterministic and Simple Recourse results for Unmet Demand

Week	1	2	3	4	5	6	7	8	9	10	Total
Point Forecasts	781	4421	6427	5949	3855	1706	0	0	0	0	23139
Forecast Uncertainty - SR	781	3438	4078	4637	3077	923	0	0	0	0	16934

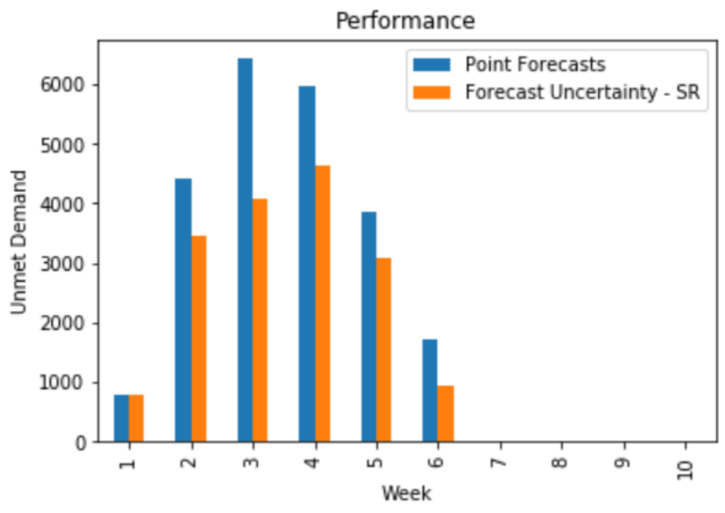


Figure 13 Unmet Demand for Deterministic and Simple Recourse Model

We observe that the Simple Recourse model performs better than the Deterministic model in the long run. We start off with no differences in unmet demand in the first week of running the optimizations. However, underestimations and misallocations add up over the course of the weeks and the increase in ventilator shortages become more significant for the deterministic model than for the simple recourse. We learn that because the simple recourse takes into account possible high and low expected demand for each state, it allocates ventilators to be able to accommodate this variance in the expected demand.

5.5 Discussion

5.5.1 Lessons Learned and Project Challenges

In this report, we set out to investigate how we can find an optimal allocation of ventilators across the different U.S. states to minimize the number of total ventilator shortages across time as well as the number of transfers of ventilators we'll need to perform across neighboring states. Firstly, we explored a deterministic model formulation which uses point forecasts of expected demand of ventilators in each state, which leads us to a simple constrained optimization problem, specifically a penalized linear programming problem. To improve on this model, we explored the possibility of variance in the expected demand and instead of minimizing the pointwise unmet demand, we minimize the expected unmet demand by accounting for possible high and low demand in each state. In the results section, we discover that the longer the model is run, the gap between the unmet demand achieved by the two model widens because suboptimal allocations of ventilators by the deterministic model add up much faster than the simple recourse model that accounts for the possibility of a variance in demand and prepares for that.
During the process, the primary challenge we encountered was when the simple recourse model had problems converging (getting the Lower Bound and Upper Bound to approach each other in Bender's). We realized that there was a bug when calculating the subgradients to be used inside of Bender's and corrected it, which allowed the model to converge successfully.

5.5.2 Future Work Extension

In the future, our team would like to proceed with improving both the demand forecast and subsequently, the algorithm being used to solve the optimization problem. Firstly, we could explore using time series regression with covariates to predict demand. Currently, data on expected high, low, mean demand has been provided to us. However, by introducing covariates such as the state case growth rate, prior unmet ventilator demand, and vaccination rate, we could use them to fit an actual continuous demand distribution (e.g., Gaussian) and instead of using Bender's to solve the optimization problem, we could sample demands from the fitted demand distribution and use Stochastic Gradient Descent instead. If we are unable to properly fit a Gaussian distribution for the demand, we could instead just use Sample Average Approximation and solve the entire problem in an All-In-One Linear Program.

5.6 References

[1] ***Code - https://nbviewer.jupyter.org/github/jeffchenchengyi/ISE-533/blob/main/project4-deterministic_model.ipynb***

[2] ***Code - https://nbviewer.jupyter.org/github/jeffchenchengyi/ISE-533/blob/main/project4-simple_recourse_model.ipynb***

[3] ***Ppt* – *https://nbviewer.jupyter.org/github/jeffchenchengyi/ISE-533/blob/main/ISE%20533%20Project%204%20Covid%20Ventilator%20Allocation.pdf***

[4] S. Sen and Y.Liu, "Mitigating Uncertainty via Compromise Decisions in Two-Stage Stochastic Linear Programming: Variance Reduction," *Operations Research*, Vol. 64, No. 6, pp. 1422–1437, Nov-Dec 2016. http://dx.doi.org/10.1287/opre.2016.1526

[5] "Stochastic Programming and Benders Decomposition," 2010, pp. 254–259.

[6] W. Zhang, "Coffee Machine Coin Stochastic Programming," *Github.* Optimization Metaheuristics, 05 May 2015. https://github.com/wesleyzh/Optimization-Metaheuristics/tree/master/Coffee%20Machine%20Coin%20Stochastic%20Programming

[7] Y. Deng, S. Sen, and J. Xu, "ISE 533 Project 4," *ISE533 Blackboard*, pp. 1–4, 08 Aril 2021.