VIETNAM NATIONAL UNIVERSITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

MATHEMATICAL MODELLING (CO2011)

Assignment

Stochastic Programming and Applications

HO CHI MINH CITY, DECEMBER 2023

Contents

1 Introduction

Stochastic programming is a branch of mathematical optimization that deals with problems involving uncertainty in the data or the objective function. It aims to find optimal decisions under various scenarios of uncertainty, such as randomness, variability, or incomplete information.

The history of stochastic programming can be traced back to the 1950s, when several researchers independently proposed methods for solving optimization problems with probabilistic constraints or objectives. Some of the pioneers of stochastic programming were George Dantzig, A. Charnes, W. Cooper, R. E. Bellman, L. V. Kantorovich, and A. N. Kolmogorov:

- Dantzig, who is widely regarded as the father of linear programming, introduced the concept of two-stage stochastic programming in 1955, where the decision maker first picks some variables and then adjusts the rest after seeing the data.
- Charnes and Cooper, who introduced chance-constrained programming, where the decision maker limits the probability of breaking any rule.
- Bellman, who used dynamic programming to solve stochastic problems, where the decision maker solves smaller problems step by step and updates the value based on the expected future outcomes.
- Kantorovich and Kolmogorov, who developed stochastic functional programming, where the decision maker chooses a function that optimizes the expected value of another function that depends on the data.

The history of stochastic programming shows how different methods were developed to deal with uncertainty in optimization problems. One way to understand these methods is to compare them with the concept of a stochastic model, which is a mathematical representation of a system or process that involves uncertainty:

One - stage stochastic model is a mathematical model used in optimization under uncertainty. This process aim to optimize objective function which directly includes uncertainty. By introducing random variables, every decision is made at a specific moment of scenario, using given probabilistic distributions to consider and optimize expected value.

Two - stage stochastic model is also a mathematical optimization used in decision-making process under uncertainty, in which uncertainty is considered as parameters along with probability. It involves two different stages, considering static and current information in first stage, and reconsider these information corresponding to different uncertainty revealed in second stage.

This assignment introduces two problems related to analysis, implementation and solving two - stage stochastic model related to Industry - Manufacturing model and Disaster - Evacuation plan model.

2 Theory-based

2.1 One - stage Stochastic Programming

One - stage stochastic linear programming (1 - SLP) is a linear programming (LP) model with one stage, no recourse, no penalize corrective actions, and all decision is made before specifying

uncertainty. There are two main approaches to solve this problem. Assuming there is a Linear Programming model LP that takes in α as a random vector.

$$Minimize\ g(x) = f(x) = c^T.x = \sum_{j=1}^{n} c_j x_j$$

$$s.t.\ \ Ax = b,\ (certain\ constraints)$$

$$Tx \geq h\ (stochastic\ constraints)$$

- Matrix T(α) and vector h(α) are unknown until all decisions $x = (x_1, x_2, x_3, ...x_n)$ have been made.
- Because all stochastic constraint is revealed only after all decisions are made, we tend to set lower and upper bound to decision as in a domain $\chi = \{x \in \mathbb{R}^n : l \leq x \leq u\}$

Approach 1: using Acceptable risk over a range of Chances

- $Tx \geq h$ is replace with $\mathbb{P}[Tx \geq h] \geq p$ with p is a prescribed reliability determined by the user.
- Explicitly define acceptable risk:

  $$r_x := \mathbb{P}[Not(Tx \geq h)] = \mathbb{P}[Tx \leq h] \leq (1 - p)$$

  with 1-p is maximal acceptable risk

As a result, our goal is

$$Minimize\ g(x) = f(x) = c^T.x = \sum_{j=1}^{n} c_j x_j, \quad c_j \in \mathbb{R}$$

$$s.t.\ \ Ax = b,\ (decision\ is\ made)$$

$$\mathbb{P}[Tx \leq h] \leq (1 - p)$$

Approach 2: Analyzing each scenario In general $T(\alpha)x \geq h(\alpha)$ Consider each scenario $(T^s, h^s), s = 1, 2, ..., S$ corresponding with each scenario, solve:

$$Minimize\ g(x) = f(x) = c^T.x = \sum_{j=1}^{n} c_j x_j, \quad c_j \in \mathbb{R}$$

$$s.t.\ \ Ax = b,$$

$$T^s x \leq h^s$$

In this case, after specify a scenario, it is now a linear problem. However, it is necessary to consider discrete distribution in each scenario, which leads to a mixed - interger linear programming problem.

2.2 Generic Stochastic Programming (GSP) with recourse

The recourse model is a paradigm for dealing with stochastic problems that involves making a decision now and then optimizing the expected outcomes of that decision based on the uncertainty that follows. The simplest version of this model has two stages: a first-stage decision, followed by the revelation of the uncertain parameters and a second-stage decision that can adjust to the new information. This model can be generalized to multiple stages, where a decision is made at each stage after some uncertainty is resolved, until the final stage. The goal is to minimize the expected costs of the decisions across all stages.
In this course, we focus on Stochastic program in two stages (generic 2-SP problem):

$$\min_x g(x) = \min_x \left(f(x) + E_\omega[v(x,\omega)]\right)$$

Where the mean $Q(x) := E_\omega[v(x,\omega)]$ of a function $v : \mathbb{R}^n \times \mathbb{R}^S \to \mathbb{R}$ upon influences of scenarios ω is the optimal value of a certain second-stage problem:

$$\min_{y \in \mathbb{R}^p} q(y) \quad \text{subject to} \quad T \cdot x + W \cdot y = h$$

In such formulation:

- $x = (x_1, x_2, \ldots, x_n)$ denote the first-stage decision variables.
- $y = (y_1, y_2, \ldots, y_p)$ denote the second-stage decision variables.
- q is the unit recourse cost vector, having the same dimension as y
- W is called m $\times$ p recourse matrix.
- h(ω) are the requirements.
- T represents the transition matrix

Note that the function $f(x)$ may be linear or nonlinear, a part of the grand objective function $g(x)$.

2.3 Two-stage Stochastic linear program: 2-SLP

A two-stage stochastic linear program with recourse is the most basic type of a stochastic program and a special case of generic 2-SP problem, where the objective function and the constraints are linear.

2.3.1 Two-stage SLP Recourse model - (simple form)

The two-stage stochastic linear program with recourse can be formulated as follows:

$$\min_{x \in X} c^T \cdot x + \min_{y(\omega) \in Y} E_\omega[q \cdot y]$$

or in general:

$$\min_{x \in X, y(\omega) \in Y} E_\omega[c^T \cdot x + v(x,\omega)] \tag{1}$$

where $v(x, \omega) := q \cdot y$ and subject to:

$$\text{First Stage Constraints: } Ax = b$$
$$\text{Second Stage Constraints: } T(\omega) \cdot x + W \cdot y(\omega) = h(\omega)$$

The problem consists of two stages: a here-and-now stage and a wait-and-see stage. In the first stage, a decision variable is chosen before the realization of the uncertain parameter is revealed. This decision is irreversible and affects the second-stage problem. In the second stage, after observing , a recourse variable is determined to optimize the second-stage objective function. The uncertainty is modeled by a set of scenarios , each with a probability. The goal is to find a first-stage decision that minimizes the expected total cost, which is the sum of the first-stage cost and the second-stage cost. For the second stage, we can apply the second approach:

Major Approaches - Scenarios analysis:

Consider the scenario where uncertain parameters adhere to a finite discrete distribution, and each scenario ω transpires with a probability $\mathbb{P}(\omega_s) = p_s$ for all $s = 1, 2, \ldots, S$ and $\sum_s p_s = 1$. Thus:

$$Q(x) = E_\omega[v(x, \omega)] = \sum_{s=1}^{S} p_s q y_s$$

This formulation encapsulates the expected value of the second state function across all scenarios weighted by their respective probabilities.

2.3.2 Two-stage SLP Recourse model - (canonical form)

The canonical 2-stage stochastic linear program with Recourse can be formulated as:

$$\min_x g(x) \quad \text{with} \quad g(x) := c^T \cdot x + v(y)$$

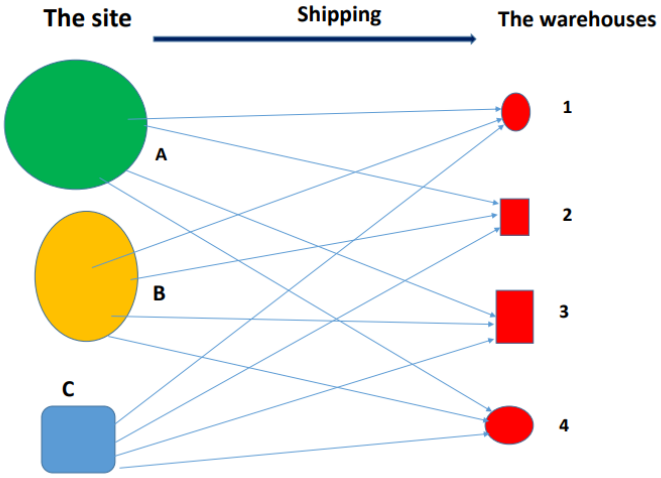
$$\text{subject to (s.t.) } Ax = b \text{ where } x \in X \subset \mathbb{R}^n,\ x \geq 0$$
$$\text{Here, } v(z) := \min_{y \in \mathbb{R}} py \text{ subject to } W \cdot y = h(\omega) - T(\omega) \cdot x =: z$$

Where $v(y) := v(x, \omega)$ is the second-stage value function.

3 Problem 1

3.1 Overview of Industry- Manufacturing problem

This problem considers a situation that a manufacturer produces n products and orders different parts from m 3rd-suppliers.
There are several variants of this kind of problem. In assignment's question, this image shows a transportation plan from site to warehouses. However, to adapt with formulas below and reference 1, we consider this image illustrates a simple production plan with **m=3, n=4**, which means a manufacturer will produce 4 products from 3 different parts.



Overall, to find optimal value x,y,z respectively, we are going to understand data vector b,l,q,s, the probability distribution of demand vector D and consider two-stage problem before and after knowing D demand vector.

3.2 Problem Analysis

Table 1 - Summaries of quantities in the problem:
This problem is an example of two-stage stochastic programming problem, which we divide its

Quantities	**Symbols**	**Description**
Number of products	n	
Number of different units	m	
Component of matrix A	$a_{i,j}$	A unit of products i requires $a_{i,j}$ units of part j
Random vector D	D	D_i, where i=1,2,...,n
Pre-oder cost b_j per unit of part j	b	b_j, where j=1,2,...,m
Additionally cost	l	l_i, where i=1,2,...,n
Selling price of this product i	q_i	q_i, where i=1,...,n, to satisfy demand for product i
Salvage values of part j	s_j	Selling price of the number of parts left in inventory
The number of parts ordered	x_j	x_j, where j=1,2,...,m
The number of products produced	z_i	z_i, where z=1,2,...,n
The number of parts left in inventory	y_j	y_i, where y=1,2,...,m

two stage:
In first stage, we denote $Q(x) := E[Z(z, y)] = E_\omega[x, \omega]$, which is the optimal value of second-stage after knowing D. Thus, we have to explain in detail the second-stage problem first for ease of understanding.
The second-stage problem:
For an observed value (a realization) $d = (d_1, d_2, \cdots, d_n)$, we can find the best production plan by solving the following Stochastic Linear Programming (SLP), which y and z are the vectors mentioned in **Table 1**.

$$\min_{z,y} Z = \sum_{i=1}^{n}(l_i - q_i)z_i \text{ - } \sum_{j=1}^{m} s_j y_j$$
$$\text{such that: } y_j = x_j \text{ - } \sum_{i=1}^{n} a_{i,j} z_i,\ j = 1, 2, ..., m \ (1)$$
$$0 \leq z_i \leq d_i,\ i = 1, 2, ..., n,\ y_j \geq 0,\ j = 1, 2, ..., m$$

Consider $c_i = \mathrm{l}_i$ - q_i are cost coefficient, then Z is the total cost a manufacturer spends to produce z products (Note that production $\geq$ demand). Therefore, min Z is the minimum cost lead to a best production plan for this manufacturer.
Because y,z depend on the realization d of the demand vector D and vector x, y,z are made after knowing random D, we consider y,z as "wait-and-see" decision variables.
Introducing the matrix A with entries $\mathrm{a}_{i,j}$, $i = 1, 2, ...n$ and $j = 1, 2, ..., m$, y,z are vectors so we can write above equations as follow:

$$\min_{z,y} \mathrm{Z} = \sum_{i=1}^{n}(l - q)^T z \text{ - } \sum_{j=1}^{m} s^T y$$
$$\text{such that: } y = x - A^T z,\ j = 1, 2, ..., m$$
$$0 \leq z \leq d,\ y \geq 0$$

The first-stage problem:
Before knowing the realization of D demand vector, as mentioned before, we denote Q(x) := E[Z(z, y)] = E_ω[x, ω] as the optimal value of (1). The quantity vector x, which is known as "here-and-now" decision variable should be made independently of the random vector demand D. Vector x is solved by this following optimization problem:

$$\min(g(x, y, z)) = b^T x + Q(x) = b^T(x) + E_\omega[x, \omega]$$
$$\text{where } Q(x) = \sum_{s=1}^{S} p_s (l - q)^T z^s - s^T y^s$$

The first part of the objective function $g(x, y, z)$ is the pre-ordering cost and the second part is the expected cost in equation (1).
In general, we consider this problem in various scenarios, denote S is the number of scenarios, s is the index, which means: $s = 1, 2, ..., S$.
We can consider two-stages problem in one large scale linear programming problem:

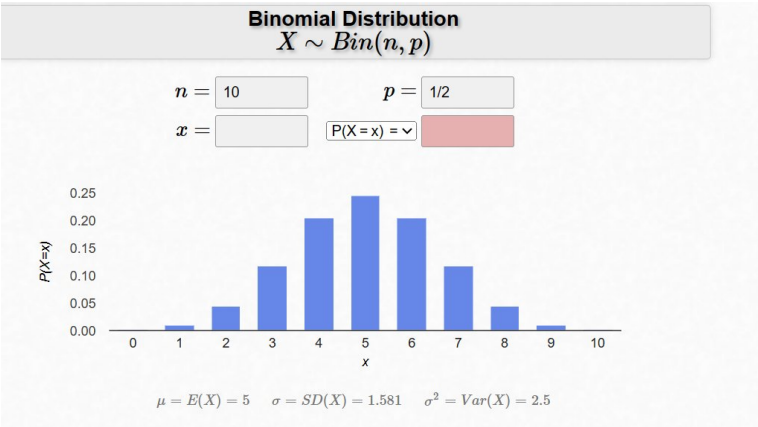
$$\min(g(x, y, z)) = b^T x + \sum_{s=1}^{S} p_s[(l - q)^T z - s^T y^s]$$
$$\text{such that: } y^s = x - A^T z^s,\ s = 1, 2, ..., S \ (3)$$
$$0 \leq z^s \leq d^s,\ y^s \geq 0,\ s = 1, 2, ..., S$$
$$x \geq 0$$

Note that, $p_1, p_2, ..., p_S$ is the probability that occur demand vector $d^1, d^2, ..., d^S$ then $\sum_{s=1}^{S} p_s = 1$.
Observing this problem, we can conclude that y_k, z_k depend on scenario k because demand vector d_k is different in each scenario k.
In conclusion, after analyze carefully two-stage problem, the objective function we have to solve

is (3) as a Linear Programming problem. Since x in (2) depends on expected cost and y,z in (1) depend on x and D, we can not solve two equations separately, which means we have to simulate a program could solve (3) to find the optimal value of x,y,z at the same time.

3.3 Requirements

To our specific requirement, we just have to solve problem in 2 scenarios ($S = 2$) and the probability of each scenario is $p_s = 1/2$.
Random vector D follow the binomial distribution Bin(10,1/2), as a way to format and limit demand D for ease of checking.
Binomial Distribution: In probability theory and statistics, the binomial distribution with parameters n and p is the discrete probability distribution of the number of successes in a sequence of n independent experiments, in which success (probability p) and failure (probability q = 1 - p)
Bin(10,1/2) relating to our problem 1:



Bin(10,1/2)

3.4 Generate code and simulate model

3.4.1 Introduction to GAMSPy

Base on the above theory, we created a program to solve Problem 1 in Python language with the help of GAMSPy library. GAMSPy is library of Python, which combines the high-performance GAMS execution system with the flexible Python language, creating a powerful mathematical optimization package. It is known as the powerful tool to solve LP and Optimization problem.

3.4.2 Explain code

The below block of code is used to import required library, including GAMSPy, as well as define some basic data for the program.

```
1 import numpy as np
2 import pandas as pd
3 import random
```

```
from gamspy import Container, Set, Parameter, Variable, Equation, Model, Sense,
    Problem, Options, Sum
#DEFINE DATA
n = 8  # number of products
S = 2  # number of scenarios
ps = [0.5, 0.5]  # scenario probabilities
m = 5  # number of units to be ordered before production
```

Next, we create a container that include all the functions and variables for the model and 2 sets represent products and materials, each set is label from 1 to 7 and 1 to 5 respectively.

```
# DECLARE CONTAINER
M = Container()
# CREATE SETS
product = Set(M, name = "product", records = ["product_1", "product_2", "product_3
    ","product_4", "product_5", "product_6", "product_7", "product_8"],
    description = "Product")
material = Set(M, name = "material", records = ["material_1", "material_2", "
    material_3", "material_4", "material_5"], description = "material")
```

Then, randomly create input vector for b,l,q,s,A,D1,D2.

```
b_records = [[f"material_{i}", round(random.uniform(30, 50), 2)] for i in range(1,
     6)]
s_records = [[f"material_{i}", round(random.uniform(10, 20), 2)] for i in range(1,
     6)]
l_records = [[f"product_{i}", round(random.uniform(1000, 2000), 2)] for i in range
    (1, 9)]
q_records = [[f"product_{i}", round(random.uniform(2000, 3000), 2)] for i in range
    (1, 9)]
D1_vector = np.random.binomial(10, 0.5, 8)  # demand vector
D2_vector = np.random.binomial(10, 0.5, 8)  # demand vector
A_matrix = np.random.randint(0, 11, size=(8, 5))  # create a matrix range from 0
    to 10
A_records = pd.DataFrame(A_matrix, index=[f"product_{i}" for i in range(1,9)],
    columns=[f"material_{j}" for j in range(1,6)])
```

In which:
- b represent the pre-order cost of each material and range from 30 to 50. All value of b have equal probability

```
b_records = [[ f "material_ {i}", round(random.uniform(30,50),2)]
for i in range (1 , 6) ]
```

- s is the savage cost of each material and range from 10 to 20, because s≤b. All value of s have equal probability

```
s_records = [[f"material_{i}", round(random.uniform(10, 20), 2)]
for i in range(1, 6)]
```

- q is the sale value of each product and range from 2000 to 3000 and all value of q have equal probability

```
q_records = [[f"product_{i}", round(random.uniform(2000, 3000), 2)]
for i in range(1, 9)]
```

- l is the additional cost for each created product and because l≤q, l range from 1000 to 2000 and all value of l have equal probability.

```
l_records = [[f"product_{i}", round(random.uniform(1000, 2000), 2)]
for i in range(1, 9)]
```

- Matrix A with dimension n×m represent the required material(or parts) to produce each product, each element of A is a random integer number between 0 and 10.

```
A_matrix = np.random.randint(0, 11, size=(8, 5))
A_records = pd.DataFrame(A_matrix, index=[f"product_{i}" for i in range(1,9)],
columns=[f"material_{j}" for j in range(1,6)])
```

- D1 and D2 are demand vector, show the demand for each product in scenario 1 and 2 respectively, each element of D1 and D2 is created based on the binomial distribution with Bin(10, 0.5).

```
D1 = Parameter(M, name = "D1",domain = [product],
description = "demand for scenario 1", records = D1_vector)
D2 = Parameter(M, name = "D1",domain = product,
description = "demand for scenario 2", records = D2_vector)
```

Put each of the above vector into Parameter with their respective domain.

```
# vector b represent pre-order cost for each parts
b = Parameter(M, name="b", domain=material, description="cost of materials",
    records = b_records)
# vector s represent savage cost for each parts
s = Parameter(M, name="s", domain=material, description="salvage value of
    materials", records = s_records)
# vector l represent additional cost
l = Parameter(M, name="l", domain=product, description="additional cost", records
    = l_records)
# vector q represent selling price for each product
q_records = [[f"product_{i}", round(random.uniform(2000, 3000), 2)] for i in range
    (1, 9)]
q = Parameter(M, name="q", domain=product, description="selling price", records =
    q_records)
# Matrix A represent the required materials for each product
A = Parameter(M,name = "A", domain = [product, material], description = "material
    for each product", records = A_records, uels_on_axes=True)
# Vector D1 for demand in scenerio 1
D1 = Parameter(M, name = "D1",domain = [product], description = "demand for
    scenario 1", records = D1_vector)
# Vector D2 for demand in scenerio 2
D2 = Parameter(M, name = "D2",domain = [product], description = "demand for
    scenario 2", records = D2_vector)
```

Print the created vector to the screen.

```
#PRINT RESULT
print("\n")
print("Obj func value: ", Model.objective_value)
print("Value for x: ")
print(x.records)
print("SCENARIO 1: ")
print("Value for z in scenario 1: ")
print(z1.records)
print("Value for y in scenario 1: ")
print(y1.records)
print("SCENARIO 2: ")
print("Value for z in scenario 2: ")
print(z2.records)
print("Value for y in scenario 2: ")
print(y2.records)
```

Define required variables and their lower bound.

```
# DECLARE VARIABLES
x = Variable(M, 'x', domain= [material], type ="free", description = "Number of
    pre-order materials")
y1 = Variable(M, 'y1', domain = [material], type ="free", description = "Number of
     leftover materials of scenerio 1")
```

```
z1 = Variable(M, 'z1', domain= [product], type ="free", description = "Number of
    sold products of scenerio 1")
y2 = Variable(M, 'y2', domain = [material], type ="free", description = "Number of
     leftover materials of scenerio 2")
z2 = Variable(M, 'z2', domain= [product], type ="free", description = "Number of
    sold products of scenerio 2")
#Set lower bound for variable
x.lo[material] = 0
y1.lo[material] = 0
z1.up[product] = 10
z1.lo[product] = 0
y2.lo[material] = 0
z2.up[product] = 10
z2.lo[product] = 0
```

In which:
- x is the number of pre-order materials, x≥0 and type"free" means x can be float.

```
x = Variable(M,'x',domain= [material], type ="free",
description = "Number of preorder materials")
```

- z is the number of sold products. Because the maximum value for the demand vector is 10, z also have an upper bound of 10. There are 2 z represent sale plan for each scenario.

```
z1 = Variable ( M,   z1   ,domain=[product], type ="free",
description ="Number of sold products of scenerio 1")
z2 = Variable ( M,   z2   , domain=[product], type = "free",
description = "Number of sold products of scenerio 2")
```

- y is the number of left-over materials, y≥0. There are 2 y for each scenario.

```
y1 = Variable ( M,   y1   , domain = [material], type = "free", description = "
    Number of leftover materials of scenerio 1")
y2 = Variable ( M,   y2   , domain = [material], type = "free", description = "
    Number of leftover materials of scenerio 2")
```

Define equation:

```
#DEFINE EQUATION
Eq1Sce1 = Equation(
                    M,
                    name = "Eq1Sce1",
                    domain = [product],
                    description = "Demand always higher than sales",
                    )
Eq1Sce1[product] = z1[product] <= D1[product]
Eq2Sce1 = Equation(
               M,
               name = "Eq2Sce1",
               domain = [material],
               description = "Define values for remaining materials",
               )
Eq2Sce1[material] = y1[material] == x[material] - Sum( product, A[product,material
    ]*z1[product])

Eq1Sce2 = Equation(
                    M,
                    name = "Eq1Sce2",
                    domain = [product],
                    description = "Demand always higher than sales",
                    )
Eq1Sce2[product] = z2[product] <= D2[product]
Eq2Sce2 = Equation(
               M,
```

```
                name = "Eq2Sce2",
                domain = [material],
                description = "Define values for remaining materials",
                )
Eq2Sce2[material] = y2[material] == x[material] - Sum( product, A[product,material
    ]*z2[product])
```

Define objective function

```
#DEFINE OBJECTIVE FUNCTION
obj = Sum( material, b[material]*x[material] ) + 0.5 * (Sum( product, (l[product]
    - q[product])* z1[product] ) - Sum( material, s[material]*y1[material])) + 0.5
     * (Sum( product, (l[product] - q[product])* z2[product] ) - Sum( material, s[
    material]*y2[material]))
```

Define model and solve model

```
#DEFINE MODEL
Model = Model( M, "hello", equations = [Eq1Sce1, Eq2Sce1,Eq1Sce2, Eq2Sce2],
    problem = "LP", sense = Sense.MIN, objective = obj)
#SOLVE
Model.solve()
```

Print the result to screen.

```
#PRINT RESULT
print("\n")
print("Obj func value: ", Model.objective_value)
print("Value for x: ")
print(x.records)
print("SCENARIO 1: ")
print("Value for z in scenario 1: ")
print(z1.records)
print("Value for y in scenario 1: ")
print(y1.records)
print("SCENARIO 2: ")
print("Value for z in scenario 2: ")
print(z2.records)
print("Value for y in scenario 2: ")
print(y2.records)
```

3.4.3 Final result and Discussion

- We random b,l,q,s in the above range for ease of calculating and considering result. The final result corresponding is:

```
Demand D1:
     product  value
0  product_1    6.0
1  product_2    7.0
2  product_3    5.0
3  product_4    3.0
4  product_5    5.0
5  product_6    2.0
6  product_7    5.0
7  product_8    7.0

Demand D2:
     product  value
0  product_1    3.0
1  product_2    4.0
```

```
2  product_3    7.0
3  product_4    4.0
4  product_5    5.0
5  product_6    5.0
6  product_7    5.0
7  product_8    5.0

Pre-order cost for each material:
     material  value
0  material_1  31.88
1  material_2  36.37
2  material_3  48.91
3  material_4  38.73
4  material_5  38.88

Additional cost for each product:
     product    value
0  product_1  1323.32
1  product_2  1038.26
2  product_3  1478.23
3  product_4  1920.95
4  product_5  1619.68
5  product_6  1566.78
6  product_7  1511.94
7  product_8  1945.37

Selling price for each product:
     product    value
0  product_1  2899.10
1  product_2  2960.93
2  product_3  2833.22
3  product_4  2826.81
4  product_5  2803.83
5  product_6  2782.19
6  product_7  2405.75
7  product_8  2647.34

Savage cost for each part:
     material  value
0  material_1  16.15
1  material_2  14.40
2  material_3  12.66
3  material_4  19.61
4  material_5  10.80

Objective function value  -8755.423124999998
```

Because we are considering LP problem, the final result is real numbers. However, we can round these results of b,l,q,s to the nearest integers. The number of sold products and pre-order parts of products are:

```
Matrix A for demanding materials:
material   material_1  material_2  material_3  material_4  material_5
product
product_1         2.0         8.0         3.0         3.0         5.0
product_2        10.0         0.0         9.0         0.0         8.0
product_3         5.0         9.0         5.0         5.0        10.0
product_4         4.0         9.0         7.0         4.0         3.0
product_5        10.0         2.0         5.0         6.0         8.0
product_6         8.0         2.0         5.0         8.0         2.0
product_7        10.0        10.0        10.0         7.0         0.0
```

```
product_8          2.0          7.0          3.0          9.0          9.0

Number of pre-order material (x):
     material    level
0  material_1  171.000
1  material_2  107.000
2  material_3  132.875
3  material_4  114.000
4  material_5  167.000

SCENARIO 1:

Number of sold product (z) in scenario 1:
     product  level
0  product_1  6.000
1  product_2  7.000
2  product_3  5.000
3  product_4  0.000
4  product_5  3.375
5  product_6  2.000
6  product_7  0.000
7  product_8  0.000

Number of leftover parts (y) in scenario 1:
     material  level
0  material_1  14.25
1  material_2   3.25
2  material_3   0.00
3  material_4  34.75
4  material_5   0.00

SCENARIO 2:

Number of sold product (z) in scenario 2:
     product  level
0  product_1    3.0
1  product_2    4.0
2  product_3    7.0
3  product_4    0.0
4  product_5    5.0
5  product_6    5.0
6  product_7    0.0
7  product_8    0.0

Number of leftover parts (y) in scenario 2:
     material  level
0  material_1  0.000
1  material_2  0.000
2  material_3  2.875
3  material_4  0.000
4  material_5  0.000
```

- Besides, the final result, min $g(x, y, z)$ is negative, which means that the production cost low and the manufacturer has more profit.

4 Problem 2

4.1 Overview of evacuation plan

Natural disasters may yearly strike a community with little warning and leave much damage and many casualties. Therefore, the main goal of this problem is to provide shelter and assistance to affected people as soon as possible.
However, in reality, it is difficult to predict the disaster magnitude and its impact on the road system because of travel time and capacity on the link are differently based on each scenario.
Therefore, this model utilizes two-stage stochastic programming framework to solve problem under uncertainty of nature disaster. Due to randomness along with other factors such as link capacity and travel time, it is broken down into two stages of stochastic models.
The objective is to find the robust a priori evacuation plan for all levels of disaster, with the expected overall evacuation time of each scenario's adaptive evacuation path and the probability of occurrence of each scenario s.
We divide the problem into two specific stages:
- In the first stage: It is assumed that affected people can not obtain information about the earthquake (travel times and capacities), therefore, they still follow priori plan before the time threshold $\widetilde{T}$.
- In the second stage: Affected people can obtain the accurate road network information through some real-time monitoring equipment, which mean they get travel time and capacities on each link to find the optimal path.

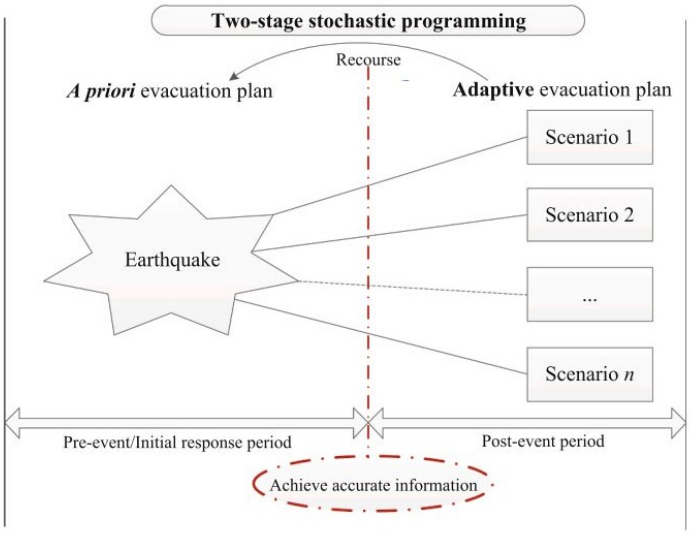


Figure 2: An Illustrative Example for Occurrence of Earthquake

We analysis clearly an example in research of Li Wang:

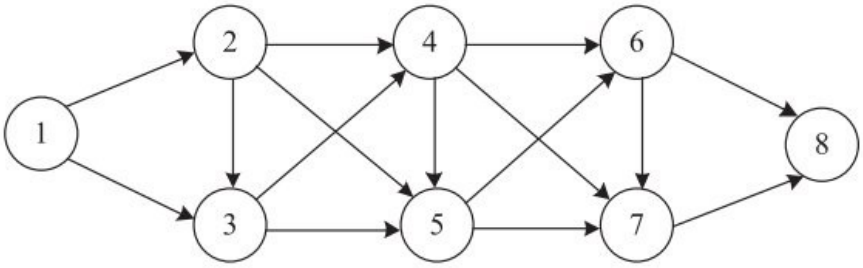


Figure 3: An illustrative evacuation road network.

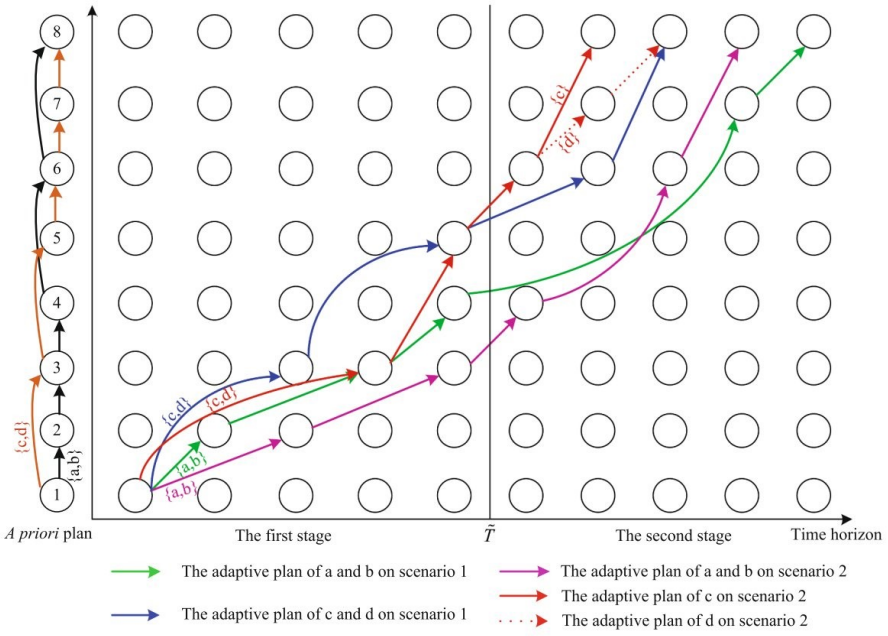


Figure 4: An example in time-dependent and stochastic two-stage evacuation planning models.

In this example, we have priori plan before time threshold $\widetilde{T}$ for cars a,b and cars c,d:
- **Cars a,b** : $1 \longrightarrow 2 \longrightarrow 3 \longrightarrow 4 \longrightarrow 6 \longrightarrow 8$
- **Cars c,d** : $1 \longrightarrow 3 \longrightarrow 5 \longrightarrow 6 \longrightarrow 7 \longrightarrow 8$

After knowing time threshold $\widetilde{T}$, which mean they obtain link capacity and travel time, cars a,b and cars c,d changed to adaptive plans based on two discrete scenarios:

Scenario 1:
- **Cars a,b** : $1 \longrightarrow 2 \longrightarrow 3 \longrightarrow 4 \longrightarrow 7 \longrightarrow 8$
- **Cars c,d** : $1 \longrightarrow 3 \longrightarrow 5 \longrightarrow 6 \longrightarrow 8$

Scenario 2:
- **Cars a,b** : $1 \longrightarrow 2 \longrightarrow 3 \longrightarrow 4 \longrightarrow 6 \longrightarrow 8$
- **Cars c** : $1 \longrightarrow 3 \longrightarrow 5 \longrightarrow 6 \longrightarrow 8$
- **Cars d** : $1 \longrightarrow 3 \longrightarrow 5 \longrightarrow 6 \longrightarrow 7 \longrightarrow 8$

4.2 Model formulation

System constraints:
- First stage:
In the first stage, a feasible evacuation should be determined from super-source to super-sink. The flow on each link should satisfy the flow balance constraint below:

$$\sum_{(i,j)\in A} x_{ij} - \sum_{(i,j)\in A} x_{ji} = d_i \tag{1}$$

where d_i represents the net demand or supply of a node i in a network flow problem with the following definition:

$$d_i = \begin{cases} v, & \text{if } i = s. \\ -d, & \text{if } i = t. \\ 0, & \text{otherwise} \end{cases}$$

Meanwhile, the flow on each link must also satisfy the capacity constraint, meaning that the amount of people that can move along each road should be between zero and the maximum capacity of the road, for every road in the network. This equation ensures that the roads are not overloaded or underused:

$$0 \leq x_{ij} \leq u_{ij}, \quad \forall (i,j) \in A \tag{2}$$

However, the first constraint may allow some paths that have loops or detours, which is not efficient for evacuation. To avoid these paths, we assign a cost to each road, which we want to minimize: the link penalty $p_{ij}, (i,j) \in A$ is particularly introduced. It is a measure of how inefficient the evacuation path is, based on the cost of each road. The cost of a road is higher if it is longer, more congested, or more dangerous. The penalty function is the sum of the costs of all the roads, multiplied by the amount of people moving along them.

With this consideration, the penalty function can be defined as:

$$f(X) = \sum_{(i,j)\in A} p_{ij} x_{ij} \tag{3}$$

where $X := \{x_{ij}\}_{(i,j)\in A}$.

- Second stage:
In first stage, the lack of information force everyone to follow a priori plan. After passing the time threshold, the plan is modified using second stage stochastic programming. In second stage, all information about disaster is determined. Therefore, appropriate evacuation plan is applied to get everyone save in minimal time.

$$y_{ij}^s(t) = x_{ij}, (i,j) \in A, s = 1, 2, ..., S \tag{4}$$

In advance of time threshold ($t \leq \widetilde{T}$), due to unknown information of disaster, the flow on a link in each scenario at as specific time $y_{ij}^s(t)$ is assumed to be the same as x_{ij}.

In the following, the goal is to minimize the overall evacuation time from dangerous zones to safe places.

$$Q(Y,s) = min \sum_{(i,j)\in A_s} c_{ij}^s(t).y_{ij}^s(t) \quad (5)$$

In aforemention assumption, each flow y_{ij}^s requires c_{ij}^s amount of time on each link (i,j). This amount of time can only be calculated at a deterministic time in whole process, so it is crucial to take the sum of all parts of times. To make the plan be more concise, equation (5) requires some following constrain

$$\sum_{(i_t,j_t)\in A_s} y_{ij}^s(t) - \sum_{(i_{t'},j_{t'})\in A_s} y_{ij}^s(t') = d_i^s(t), \forall i \in V, t \in \{0,1,2...,T\} \quad (6)$$

$$0 \leq y_{ij}^s(t) \leq u_{ij}^s(t), \forall (i,j) \in A, t \in \{0,1,2,...,T\}, s = 1,2,3,...,S \quad (7)$$

$$y_{ij}^s(t) = x_{ij}, (i,j) \in A, s = 1,2,...,S \quad (8)$$

Constraint (6) is the flow model balance, in which the difference of in and out of an area satisfy the balance variable $d_i^s(t)$ at a scenario s and a specific time t. (7) and (8) traffic capacity constraint and coupling constraint to no area is overloaded with people.

Evacuation planning mode:
Assume probability of each scenario is $\mu_s, s = 1,2,3,...,S$. Since second is limited with a number of scenarios, this two-stage stochastic model is time-dependent and to make sure there should not appear any loops or sub-tours, the model can rewrite as followed:

$$\begin{cases} min \sum_{(i,j)\in A} p_{ij}x_{ij} + \sum_{s=1}^{S} \mu_s.Q(Y,s) \\ \\ \text{in which,} \\ Q(Y,s) = min \sum\limits_{(i,j)\in A_s} c_{ij}^s(t).y_{ij}^s(t) \\ s.t. \\ \\ \sum\limits_{(i_t,j_t)\in A_s} x_{ij} - \sum\limits_{(i_{t'},j_{t'})\in A_s} x_{ji} = d_i, \forall i \in V \\ \\ 0 \leq x_{ij} \leq u_{ij}, \forall (i,j) \in A \\ \\ \sum\limits_{(i_t,j_t)\in A_s} y_{ij}^s(t) - \sum\limits_{(i_{t'},j_{t'})\in A_s} y_{ij}^s(t') = d_i^s(t), \forall i \in V, t \in \{0,1,2...,T\} \\ \\ 0 \leq y_{ij}^s(t) \leq u_{ij}^s(t), \forall (i,j) \in A, t \in \{0,1,2,...,T\}, s = 1,2,3,...,S \\ \\ y_{ij}^s(t) = x_{ij}, (i,j) \in A, s = 1,2,...,S \end{cases} \quad (9)$$

It is clear that this is a scenario-based problem, which implies that if a scenario is defined, this two-stage stochastic model is degenerated into a deterministic model. Additionally, if real-time

information is completely collected, this priori plan is not important since this problem turns into a wait and see model (WAS) and can be modelled as followed:

$$\begin{cases} min \sum_{s=1}^{S} \Big(\mu_s . min \sum\limits_{(i,j) \in A_s} c_{ij}^s(t) . y_{ij}^s(t) \Big) \\ \text{s.t.} \\ \sum\limits_{(i_t, j_t) \in A_s} y_{ij}^s(t) - \sum\limits_{(i_{t'}, j_{t'}) \in A_s} y_{ij}^s(t') = d_i^s(t), \forall i \in V, t \in \{0, 1, 2..., T\}, s = 1, 2, 3, ..., S \\ 0 \leq y_{ij}^s(t) \leq u_{ij}^s(t), \forall (i,j) \in A, t \in \{0, 1, 2, ..., T\}, s = 1, 2, 3, ..., S \end{cases} \tag{10}$$

This model constraints inherit all from the original model, which make WAS a lower bound of the original model. If ideally, every scenario is independent and no relation among them, this model can be further broken down into S sub-problem corresponding to S cases of scenario.

4.3 SubProblem 1 - Min-cost Flow Algorithm

The first sub-problem can be regarded as a min-cost flow problem, and its form is given as follows:

Minimize $SP1(\alpha) = \sum_{(i,j) \in A} (p_{ij} - \sum_{s=1}^{S} \sum_{t=1}^{\widetilde{T}} \alpha_{ij}^s(t)) \cdot x_{ij}$

Subject to:

$$\sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = d_i, \quad \text{for all } i \in V$$
$$0 \leq x_{ij} \leq u_{ij}, \quad \text{for all } (i,j) \in A$$

In this formulation, the parameters are as follows:

- x_{ij} represents the flow on arc (i, j).
- p_{ij} is the link penalty in case of potential loop.
- $\alpha_{ij}^s(t)$ represents Lagrangian multiplier for (4)
- d_i is the demand at node i.
- u_{ij} is the capacity of arc (i, j).
- A is the set of arcs in the network.
- V is the set of nodes in the network.
- S and $\widetilde{T}$ are the set of scenarios and time threshold respectively.

Objective Function:

The objective of this sub-problem is to minimize the total cost of the flow in the network. The cost of each arc (i, j) in the network is given by $p_{ij} - \sum \alpha_{ij}^s(t)$ and the total cost is then the sum of the cost of each arc times the amount of flow x_{ij} on that arc.

Constraints:

With the constraint in (4) adding to the objective function using Lagrangian multiplier, the problem is subject to only two sets of constraints:

Flow Conservation Constraints: For each node i in the network, the total flow out of the node minus the total flow into the node must equal the demand d_i at that node.
→ This ensures that the flow is conserved at each node - that is, all flow that enters a node must either leave the node or be consumed by the demand at that node.

Capacity Constraints: For each arc (i, j) in the network, the flow x_{ij} must be between 0 and the capacity u_{ij} of the arc. This ensures that the flow on each arc does not exceed its capacity.

Algorithm:
Sub-Problem 1 can be solved using the Successive Shortest Path Algorithm. This algorithm was first introduced by researchers like Jewell (1962), Iri (1960), Busacker, and Gowen (1961). The main aim of this algorithm is to find the flow in a network that has the least total cost. The network is represented as $G(V, A, C, U, D)$, where:

- V is the set of vertices or nodes.
- A is the set of arcs or edges.
- C is the cost function.
- U is the capacity function.
- D is the demand function.

The optimal value of Sub-Problem 1, found using this algorithm (provided in section 5.5), is denoted as $Z^*(SP1)$

Note: The current problem formulation does not consider the time-dependent factors, which will be the focal point of the subsequent section.

4.4 SubProblem 2 - Time-dependent Min-cost Flow Algorithm

Sub-problem 2 analyzes the min-cost flow with dependence of flow on each link on each scenario and time. In this problem, the optimal objective function is denoted as $Z_{SP2}(\alpha)$, with α is Lagrangian variable which helps break down the original problem into two sub-problem. In this case, sub-problem 2 is shown as follows:

$$\begin{cases} minSP2(\alpha) = \sum_{s=1}^{S} \sum_{(i,j)\in A} \left(\sum_{t\in\{0,1,2,...,T\}} .\mu_S.c_{ij}^s(t) + \sum_{t\leq T} \alpha_{ij}^S(t) \right) .y_{ij}^s(t) \\ \text{s.t.} \\ \sum_{(i_t,j_t)\in A_s} y_{ij}^s(t) - \sum_{(j_{t'},i_{t'})\in A_s} y_{ij}^s(t') = d_i^s(t), \forall i \in V, t \in \{0,1,2,...,T\}, s = 1,2,...,S \\ \\ 0 \leq y_{ij}^s(t) \leq u_{ij}^s(t), \forall (i,j) \in A, t \in \{0,1,2,...,T\}, s = 1,2,3,...,S \end{cases} \tag{11}$$

Because $Z_{SP2}(\alpha)$ is still a scenario-based problem, it can be broken down into S separated problem corresponding with S scenarios. In this way, each sub-problem is a min-cost flow problem

with time-dependent link travel times and capacity.

$$\begin{cases} minSP2(\alpha, s) = \sum_{(i,j)\in A} \left(\sum_{t\in\{0,1,2,...,T\}} .\mu_S.c_{ij}^s(t) + \sum_{t\leq T} \alpha_{ij}^S(t) \right) .y_{ij}^s(t) \\ s.t. \\ \sum_{(i_t,j_t)\in A_s} y_{ij}^s(t) - \sum_{(j_{t'},i_{t'})\in A_s} y_{ij}^s(t') = d_i^s(t), \forall i \in V, t \in \{0,1,2,...,T\}, s = 1,2,...,S \\ \\ 0 \leq y_{ij}^s(t) \leq u_{ij}^s(t), \forall (i,j) \in A, t \in \{0,1,2,...,T\}, s = 1,2,3,...,S \end{cases} \tag{12}$$

Obvious based on the matter of this problem, we introduce a variable called generalized cost g_{ij}^s. Since whole process is divided into two stages by time threshold $\widetilde{T}$, generalized cost is define as a piecewise function:

$$g_{ij}^s(t) = \begin{cases} \mu_S.c_{ij}^s(t) + \alpha_{ij}^S(t), & t \leq \widetilde{T} \\ \mu_S.c_{ij}^s(t), & \widetilde{T} < t \leq T \end{cases} \tag{13}$$

To gain a better insight and make more precise decision, it is necessary to redefine parameters A(y(t)), C(y(t)), and U(y(t)) in residual network N(y(t)) as follows: $A_s(y(t)) = \{(i_t, j_t) | (i_t, j_t) \in A_s, y_{ij}^s < u_{ij}^s\} \cup \{(j_{t'}, i_{t'}) | (j_{t'}, i_{t'}) \in A_s, y_{ij}^s > 0\},\ s = 1,2,...,S$

$$c_{ij}^s(y(t)) = \begin{cases} c_{ij}^S(t), (i_t, j_t) \in A_s, y_{ij}^s(t) < u_{ij}^s(t), t \in \{0,1,2,...,T\} \\ \\ -c_{ij}^S(t'), (j_{t'}, i_{t'}) \in A_s, y_{ij}^s(t) < u_{ij}^s(t), \forall \{t' \in \{0,1,2,...,T\} | y_{ij}^s(t') > 0\}, s = 1,2,...,S \\ \\ T, (j_{t'}, it') \in A_s, \forall \{t' \in \{0,1,2,...,T\} | y_{ij}^s(t') = 0\} \end{cases}$$

$$u_{ij}^s(y(t)) = \begin{cases} u_{ij}^s(t) - y_{ij}^s(t), (i_t, j_t) \in A_s, y_{ij}^s(t) < u_{ij}^s(t), t \in \{0,1,2,...,T\} \\ \\ y_{ji}^s(t'), (j_{t'}, i_{t'}) \in A_s, \forall \{t' \in \{0,1,2,...,T\} | y_{ij}^s(t') > 0\}, s = 1,2,...,S \\ \\ 0, (j_{t'}, it') \in A_s, \forall \{t' \in \{0,1,2,...,T\} | y_{ij}^s(t') = 0\} \end{cases}$$

Algorithm:
To get the optimal solution, Lagrangian multiplier α is introduced and applies the subgradient optimization algorithm to iterate whole process. This algorithm generally aim to reduce the gap between upper and lower bounds using iteration. Further information can be found in articles such as Yang and Zhou (2014, 2017) or Wang, Yang, and Gao (2016); Wang, Yang, Gao, Li, et al. (2016)

4.5 Algorithm 1 - Analysis

In this project, our primary focus will be on **implementing the two-stage stochastic evacuation planning model on a small grid network using the Experiment Design approach**. However, we will also analyze the algorithm for better understanding of the implementation.

The overall structure of the algorithm will be organized as follows::

Algorithm 1 Successive Shortest Path Algorithm for Min-Cost Flow Problem

Step 1: Select a feasible flow variable x between any OD pair that has the minimum delivery cost among feasible flows with the same flow value.
Step 2: The algorithm stops if the flow value of x reaches v or if there is no minimum cost path in the residual network $(V, (A_x), (C_x), (U_x), D)$. Otherwise:
while the flow value of x has not reached v and there exists a minimum cost path in the residual network **do**
    Calculate the shortest path with the maximum flow using a label-correcting algorithm. Define functions for the residual network:
    - $A(x) = \{(i,j) \mid (i,j) \in A, x_{ij} < u_{ij}\} \cup \{(j,i) \mid (j,i) \in A, x_{ji} > 0\}$
    - $C(x) = \begin{cases} x_{ij} & \text{if } (i,j) \in A, x_{ij} < u_{ij} \\ -x_{ij} & \text{if } (j,i) \in A, x_{ji} > 0 \end{cases}$
    - $U(x)_{ij} = \begin{cases} x_{ij} & \text{if } x = u_{ij} \\ x_{ji} & \text{if } x_{ji} \end{cases}$
    Step 3: Increase the flow along the minimum cost path. If the increased flow value does not exceed v, return to Step 2.
end while

In our implementation, we realize the first step which is finding the path with minimum cost by using a modified version of Bellman-Ford:

Algorithm 1.1 Shortest Path Faster Algorithm (SPFA)

```
procedure SPFA(n, v0, d, p)
d ← array of size n initialized with INF
d[v0] ← 0
inq ← array of size n initialized with false
q ← empty queue
q.push(v0)
p ← array of size n initialized with −1
    while q is not empty do
        u ← q.front()
        q.pop()
        inq[u] ← false
        for each vertex v adjacent to u do
            if capacity[u][v] > 0 and d[v] > d[u] + cost[u][v] then
                d[v] ← d[u] + cost[u][v]
                p[v] ← u
                if not inq[v] then
                    inq[v] ← true
                    q.push(v)
                end if
            end if
        end for
    end while
end procedure
```

The SPFA is an optimization of the Bellman-Ford algorithm for computing single-source shortest paths in a weighted directed graph. The algorithm operates by successively updating the shortest path estimates until they are optimal.

Initialization: The algorithm initializes by setting the shortest path estimate to the source vertex as zero, and to all other vertices as infinity. This is indicative of the fact that the shortest paths are unknown at the beginning.

Relaxation: The algorithm then performs a series of relaxation steps. In each relaxation step, the algorithm iterates over all edges in the graph. For each edge, it checks if the shortest path estimate to the destination vertex can be improved by going through the source vertex. If it can, the algorithm updates the shortest path estimate.

Convergence: The relaxation steps are repeated until the shortest path estimates converge, i.e., they cannot be improved anymore. At this point, the algorithm has found the shortest paths from the source vertex to all other vertices.

Negative Cycle Detection: If the algorithm can still improve the shortest path estimates after a certain number of iterations, it implies the presence of a negative cycle in the graph. A negative cycle is a cycle in the graph where the total weight of the edges in the cycle is negative.

Upon termination, the SPFA yields the shortest path estimates from the source vertex to all other vertices. In the presence of a negative cycle, the algorithm indicates that no shortest paths exist due to the possibility of indefinitely decreasing the path length by traversing the negative cycle. However, we will solely focus on a basic grid network for our experimentation. Therefore, this particular case will not be considered.

Min-cost Max-flow algorithm:

After finding the feasible flows based on minimum delivery cost, we then identifies paths in the residual network, and increases flows along the shortest path until reaching a maximum flow value or no further minimum cost path available. This is achieved by using Ford-Fulkerson method combining with shortest path found in step 1.

The Minimum Cost Flow (MinCF) algorithm, similar to the Edmonds-Karp algorithm for computing the maximum flow, relies on augmenting paths defined on the residual network G_f, consisting of all edges $e = u \to v$ with non-zero residual capacity $rc(e)$.

In the simplest case of MinCF, we assume the graph is oriented, and there is at most one edge between any pair of vertices. If (i, j) is an edge in the graph, then (j, i) cannot be part of it. Let C_{ij} be the capacity of an edge (i, j), A_{ij} be the cost per unit of flow along this edge, and F_{ij} be the flow along the edge (i, j). Initially, all flow values are zero.

We modify the network as follows: for each edge (i, j), we add the reverse edge (j, i) to the network with the capacity $C_{ji} = 0$ and the cost $A_{ji} = -A_{ij}$. Since, according to our restrictions, the edge (j, i) was not in the network before, we still have a network that is not a multigraph (a graph with multiple edges). Additionally, we always maintain the condition $F_{ji} = -F_{ij}$ true during the steps of the algorithm.

In the provided algorithm, N is the number of nodes, `edges` is a vector of edges where each edge has a "from" node, a "to" node, a cost, and a capacity, K is the maximum flow, s is the source node, and t is the target node. The algorithm calculates the shortest paths from the source

node to all other nodes, finds the maximum flow f that can be sent along the shortest path from s to t, updates the flow and the cost, and adjusts the capacities along the path. Finally, it returns the total cost. This algorithm implements a variant of the MinCF algorithm used to find the maximum flow in a flow network that also has costs associated with each edge, a common algorithm in network flow problems in graph theory.

During each iteration of the algorithm, we find the shortest path in the residual graph from s to t.

- If no path exists anymore, the algorithm terminates, and the flow F is the desired one.
- If a path is found, we increase the flow along it as much as possible. That is, we find the minimal residual capacity R of the path, increase the flow by R, and reduce the back edges by the same amount.
- If at some point the flow reaches the value K, we stop the algorithm. Note that in the last iteration of the algorithm, it is necessary to increase the flow by only such an amount so that the final flow value doesn't surpass K.

By setting the parameter K to infinity within the algorithm, the algorimth will solve the min-cost max-flow problem.

Algorithm 1.2 Minimum Cost Flow Algorithm

procedure MinCostFlow(N, edges, K, s, t)
Initialize adjacency list (adj), cost matrix (cost), and capacity matrix (capacity) for the graph with N nodes
Initialize $flow \leftarrow 0$, $total\_cost \leftarrow 0$, and vectors d, p
Set capacity and cost for each edge.
    while $flow < K$ **do**
Compute shortest paths from node s to all other nodes and store distances in d and predecessor nodes in p
        if $d[t] = \text{INF}$ **then**
Break the Loop
return -1
        end if
Find the maximum flow (f) on the shortest path
Initialize $f \leftarrow K - flow$
Set $cur \leftarrow t$
        while $cur \neq s$ **do**
Update f as $\min(f, capacity[p[cur]][cur])$
Set $cur \leftarrow p[cur]$
        end while
Apply the flow along the path
Increment $flow$ by f
Increment $total\_cost$ by $f \times d[t]$
Set $cur \leftarrow t$
        while $cur \neq s$ **do**
Update capacities: $capacity[p[cur]][cur] -= f$
Update capacities: $capacity[cur][p[cur]] += f$
Set $cur \leftarrow p[cur]$
        end while
    end while
    return $total\_cost$
end procedure

Foundation of the algorithm:

This algorithm is mainly based on Ford-Fulkerson Algorithm with the main difference being this algorithm seeks the path with the minimum cumulative cost instead of focusing on the minimum number of edges traversed.

The Ford-Fulkerson algorithm employs a greedy strategy to find the maximum achievable flow within a network or graph. In a flow network which contains vertices and edges with a source (S) and a sink (T), each vertex, excluding S and T, possesses an equal capacity to both receive and transmit material. Notably, vertex S can exclusively transmit, while vertex T is solely capable of receiving.

To comprehend this algorithm, envision a flow of liquid within a system of pipes, each with different capacities. Every pipe can handle a specific volume of liquid transfer at any given time. The goal of this algorithm is to determine the maximum amount of liquid that can be efficiently transferred from the source to the sink within this network of pipes.

The algorithm works as follow:

1. **Initialization Phase:** Set the flow in all edges of the graph to zero.
2. **Augmenting Path Identification:** Continuously examine the presence of augmenting paths between the source and the sink nodes within the graph.
3. **Flow Augmentation Process:** Upon identifying an augmenting path, incorporate this newly discovered path into the existing flow of the graph.
4. **Residual Graph Update:** After each augmentation, update the residual graph to reflect the alterations made to the flow distribution. This update involves adjusting the residual capacities of edges to accommodate the flow changes.
5. **Iteration Continuation:** Repeat this iterative process of augmenting paths and updating the residual graph until no further augmenting paths are found between the source and the sink nodes.
6. **Termination:** Terminate the algorithm when no more augmenting paths can be identified, signifying the achievement of maximum flow or saturation within the network.

4.6 Option 2 - solving the two-stage stochastic evacuation planning model with Experiment Design approach

Our group choose option 2 - solving the two-stage stochastic evacuation planning model with Experiment Design approach. It is assumed that there are 500 cars need to be evacuated to a safe area, and the duration of the disaster is 120 min. The unit growth interval is one minute, and thus the disaster period is divided into 120 time intervals.

4.6.1 Simulate data

According to the requirement, our group simulates a grid network with 25 nodes, 40 links and 4 scenarios. Initially, we random capacities and travel time on each link respectively, which capacities from 50 to 400 and travel time from 0 to 40 (corresponding to 500 cars and 120 time intervals).
After that, since we considered 4 scenarios, we create 4 for-loop to random capacities and travel time properly when a disaster happened. Range of capacities from 0 to initial capacity on this link because as requirement, there are some links will be damaged and unable to move. Travel time random from initial time to 50 minutes due to effect of disaster, it will take more time than in normal condition.
Note that the capacities and travel times of each link saved in corresponding matrix (pt.txt)

```
#SIMULATE DATA
def random_capacity_and_time_for_4_scenerio( capacity , cost):
    #capacity_sce_1 = [ np.random.randint(0, capacity[]) * n for _ in range(n)]
    # scenerio 1
    cost_sce_1 = [[0] * n for _ in range(n)]
    capacity_sce_1 = [[0] * n for _ in range(n)]
    # scenerio 2
    cost_sce_2 = [[0] * n for _ in range(n)]
    capacity_sce_2 = [[0] * n for _ in range(n)]
    # scenerio 3
    cost_sce_3 = [[0] * n for _ in range(n)]
```

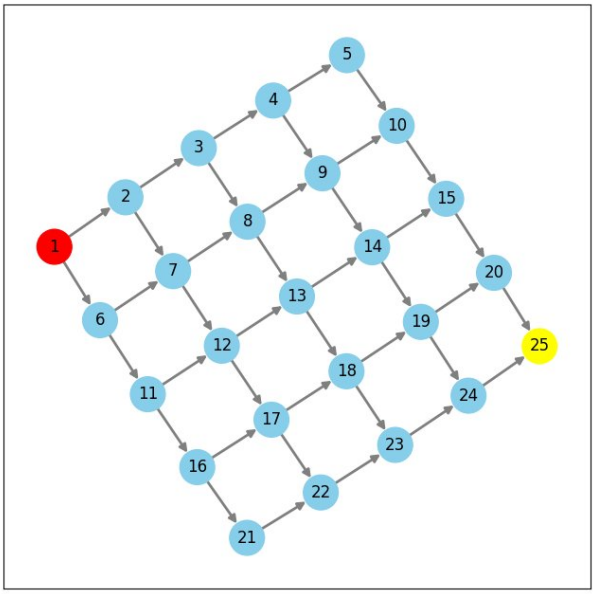
```
    capacity_sce_3 = [[0] * n for _ in range(n)]
    # scenerio 4
    cost_sce_4 = [[0] * n for _ in range(n)]
    capacity_sce_4 = [[0] * n for _ in range(n)]
    # random scenario 1,2,3,4 by for-loop
    return capacity_sce_1, cost_sce_1, capacity_sce_2, cost_sce_2, capacity_sce_3,
     cost_sce_3, capacity_sce_4, cost_sce_4
```

The capacity and travel time on each link will be displayed in a matching matrix. Because it is a grid network, each node will connect with 2 consecutive nodes.

```
Intial capacity:
[0, 193, 0, 0, 0, 272, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 267, 0, 0, 0, 273, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 181, 0, 0, 0, 283, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 291, 0, 0, 0, 269, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 282, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 261, 0, 0, 0, 366, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 289, 0, 0, 0, 267, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 193, 0, 0, 0, 261, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 258, 0, 0, 0, 397, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 391, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 282, 0, 0, 0, 177, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 158, 0, 0, 0, 256, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 291, 0, 0, 0, 347, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 260, 0, 0, 0, 262, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 177, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 255, 0, 0, 0, 353]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 298, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 299, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 357, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 262]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 271]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 270]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 285]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 395]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Initial cost:
[0, 21, 0, 0, 0, 16, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 29, 0, 0, 0, 26, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 33, 0, 0, 0, 28, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 32, 0, 0, 0, 25, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 25, 0, 0, 0, 19, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 32, 0, 0, 0, 28, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 15, 0, 0, 0, 22, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 24, 0, 0, 0, 12, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 19, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12, 0, 0, 0, 33, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 20, 0, 0, 0, 13, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 19, 0, 0, 0, 15, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 13, 0, 0, 0, 16, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 22, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12, 0, 0, 0, 42, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 31, 0, 0, 0, 29]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 15, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 20, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 35, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 18]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 14, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 22]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 29]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

The initial capacity and cost matrix - 25 nodes and 40 links



The initial grid network

4.6.2 Explain code

Pre-event evacuation plan:
After simulating data for 4 scenarios, to create a priori path, which is proper for almost levels of disasters, we calculate the average capacities and travel time for each link from 4 above scenarios based on its possibility, after that, use SSP (Successive Shortest Path - explain in theory) for finding the most reliable path before disaster.

```
# Generate random value for 4 scenerio
    random_values = np.random.rand(4)
    probabilities = random_values / np.sum(random_values)
```

Calculating the average cost and capacity for pre-event evacuation plan:

```
    for i in range (0,n):
      for j in range (0, n):
        cost_ave[i][j] = probabilities[0]*cost_sce_1[i][j] + probabilities[1]*
    cost_sce_2[i][j] + probabilities[2]*cost_sce_3[i][j] + probabilities[3]*
    cost_sce_4[i][j]
        capacity_ave[i][j] = probabilities[0]*capacity_sce_1[i][j] + probabilities
    [1]*capacity_sce_2[i][j] + probabilities[2]*capacity_sce_3[i][j] +
    probabilities[3]*capacity_sce_4[i][j]
        cost_ave[i][j] = round(cost_ave[i][j])
        capacity_ave[i][j] = round(capacity_ave[i][j])
```

Successive Shortest Path Algorithm:

```
def shortest_paths(n, v0, adj, capacity, cost):
    INF = float('inf')
    d = [INF] * n
    d[v0] = 0
    inq = [False] * n
```

```
    q = deque([v0])
    p = [-1] * n

    while q:
        u = q.popleft()
        inq[u] = False
        for v in adj[u]:
            if capacity[u][v] > 0 and d[v] > d[u] + cost[u][v]:
                d[v] = d[u] + cost[u][v]
                p[v] = u
                if not inq[v]:
                    inq[v] = True
                    q.append(v)

    return d, p
```

```
def min_cost_flow(N,adj, cost, capacity, edges, K, s, t):
    for e in edges:
      cost[e.to][e.from_node] = -e.cost
    flow = 0
    total_cost = 0
    d, p = [], []
    path = []

    while flow < K:
        d, p = shortest_paths(N, s, adj, capacity, cost)
        if d[t] == float('inf'):
            break
        vector_temp = []
        # find max flow on that path
        f = K - flow
        cur = t
        while cur != s:
            f = min(f, capacity[p[cur]][cur])
            cur = p[cur]

        # apply flow
        flow += f
        total_cost += f * d[t]
        cur = t
        print(d[t])
        print(cur + 1, end=" ")
        vector_temp.append(cur+1)
        while cur != s:
            print(p[cur] + 1, end=" ")
            vector_temp.append(p[cur] + 1)
            capacity[p[cur]][cur] -= f
            capacity[cur][p[cur]] += f
            cur = p[cur]
        path.append(vector_temp)
    #print("Max flow:", flow)
    return total_cost, path
```

In this step, we also consider time-dependence which time and capacity will be changed because of the magnitude of disaster.
The average cost and capacity after calculating related to possibility:

```
The average capacity matrix:
[0, 116, 0, 0, 0, 146, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 91, 0, 0, 0, 194, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 83, 0, 0, 0, 183, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 160, 0, 0, 0, 114, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 121, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 187, 0, 0, 0, 208, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 153, 0, 0, 0, 196, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 101, 0, 0, 0, 185, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 154, 0, 0, 0, 154, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 183, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 108, 0, 0, 0, 90, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 105, 0, 0, 0, 111, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 130, 0, 0, 0, 227, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 126, 0, 0, 0, 119, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 60, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 83, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 93, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 109, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 179, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 115]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 171]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 116]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 122]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 170]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

The average cost matrix:
[0, 29, 0, 0, 0, 30, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 33, 0, 0, 0, 41, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 40, 0, 0, 0, 47, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 36, 0, 0, 0, 43, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 26, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 32, 0, 0, 0, 40, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 43, 0, 0, 0, 38, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 33, 0, 0, 0, 40, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 33, 0, 0, 0, 32, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 35, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 34, 0, 0, 0, 40, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 36, 0, 0, 0, 37, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 34, 0, 0, 0, 32, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 33, 0, 0, 0, 33, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 34, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 36, 0, 0, 0, 43, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 37, 0, 0, 0, 37, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 37, 0, 0, 0, 43, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 33, 0, 0, 0, 39, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 37]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 28, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 27]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 43]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 43]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

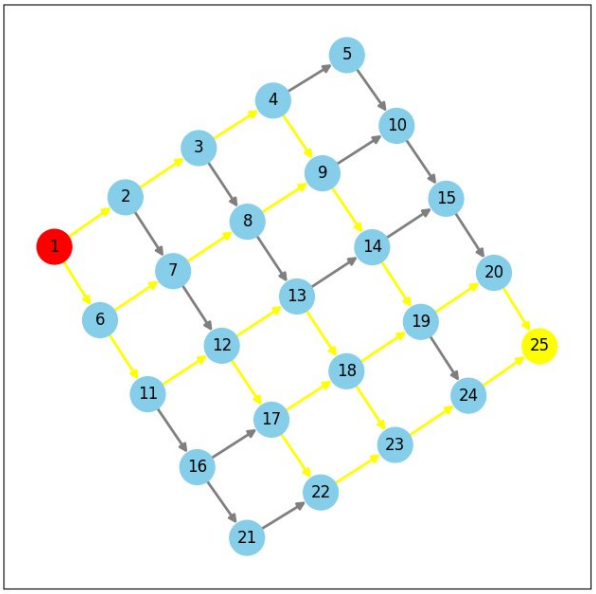
The average capacity and cost matrix

The priori path will be shown as follow:

```
Initial evacuation plan for all scenario:
 + [1, 6, 11, 12, 17, 22, 23, 24, 25]
 + [1, 6, 11, 12, 13, 18, 23, 24, 25]
 + [1, 6, 11, 12, 13, 18, 19, 20, 25]
 + [1, 6, 7, 8, 9, 14, 19, 20, 25]
 + [1, 6, 11, 12, 17, 18, 19, 20, 25]
 + [1, 2, 3, 4, 9, 14, 19, 20, 25]

The maximum number of people can be evacuted:  247
```

The pre-event evacuation plan



The initial evacuation plan is marked as the yellow line on the graph

After finding the most reliable path for a pre-event evacuation plan, we consider 4 scenarios as the time threshold $\widetilde{T} = 60$. At $t = 60$, the number of people in each node and link will be different, therefore, generally, we examine the priori plan for keeping track of affected people. Our idea is to create a super source that contains the sum of affected people of each node on the path to the sink at $\widetilde{T}$ and some dummy links from the super source to corresponding nodes. Thus, we utilize the algorithm Successive Shortest Path (SSP) without the loss of generality by changing the capacity from the source to corresponding nodes to 0 in the matrix and remaining the rest of the path to a specific scenario. In this case, we use node 1 as the super source and change capacities from 1 to 2 and 6 to 0, connecting super source (1) to matching nodes with corresponding capacities and time equals 0. Thus, the problem after the time threshold $\widetilde{T}$ can be solved by SSP as expected.

```
def create_super_source( adj, middle_node):
    adj[0] = []
    for i in middle_node:
         adj[0].append(i-1)
         adj[i-1].append(0)
    return adj

def change_capacity_cost (n, middle_flow, middle_node, capacity, cost ):
     for i in range(0, n):
       capacity[0][i] = 0

     for i in range (0, len(middle_node)):
        capacity[0][ middle_node[i] - 1] = middle_flow[i]
        cost[0][ middle_node[i] - 1] = 0
     return capacity, cost
```

Finally, using again SSP algorithm, we change problem after the time threshold $\widetilde{T}$ from the multiple source to a super source and return the different paths as follow.
Note that due to the number of nodes and links (25 nodes and 40 links), we could not use the

different color of each path and the information (capacity, travel time) on each link as expected. The movement of affected people to the sink:

```
New evacuation plan for scenario 1:
  + [12, 13, 18, 23, 24, 25]
  + [12, 17, 22, 23, 24, 25]
  + [12, 17, 22, 23, 18, 19, 24, 25]
  + [12, 17, 18, 19, 24, 25]
  + [12, 17, 18, 19, 20, 25]
  + [7, 8, 9, 10, 15, 20, 25]
  + [7, 8, 13, 18, 19, 20, 25]
The maximum number of people can be evacuated in scenerio 1:  220

New evacuation plan for scenario 2:
  + [12, 17, 22, 23, 24, 25]
  + [12, 13, 14, 19, 20, 25]
  + [12, 17, 18, 23, 24, 25]
  + [7, 8, 9, 14, 15, 20, 25]
  + [7, 8, 13, 14, 15, 20, 25]
  + [7, 8, 13, 18, 23, 24, 25]
The maximum number of people can be evacuated in scenario 2:  259

New evacuation plan for scenario 3:
  + [12, 17, 22, 23, 24, 25]
  + [12, 13, 14, 15, 20, 25]
  + [12, 17, 18, 23, 24, 25]
  + [12, 13, 18, 23, 24, 25]
  + [12, 13, 14, 19, 24, 25]
  + [7, 8, 9, 10, 15, 20, 25]
  + [7, 8, 9, 14, 19, 24, 25]
  + [7, 8, 13, 14, 19, 24, 25]
The maximum number of people can be evacuated in scenario 3:  140

New evacuation plan for scenario 4:
  + [12, 17, 18, 19, 20, 25]
  + [12, 17, 18, 19, 24, 25]
  + [12, 17, 22, 23, 24, 25]
  + [12, 13, 14, 19, 24, 25]
  + [7, 8, 9, 14, 19, 24, 25]
  + [7, 8, 9, 14, 19, 18, 23, 24, 25]
  + [7, 8, 13, 18, 23, 24, 25]
  + [7, 8, 13, 18, 17, 22, 23, 24, 25]
The maximum number of people can be evacuated in scenario 4:  317
```

Result in 4 scenarios

4.6.3 Final result and Discussion

Because we prioritize the minimum time, the number of people who move to the sink may be less than 500. Our result ultimately is not the optimal solution since we did not consider Lagrangian Relaxation.

Since we use two vectors for keeping track of the movements and total time of affected people on different paths, if people are traveling on the link at t = 60, we assume they will stay in the previous nodes.

Besides, since we calculate the priori plan based on the average capacities and travel time of 4 scenarios, the plan could not be reliable in some cases in which disasters happen with unexpected magnitude and levels.

5 Conclusion

Considering Stochastic Programming is a helpful chance for us to discover and realize the practical application of this field of mathematics in many aspects of life. As two problems we consider above, stochastic has diverse applications and help people could predict the production profit, the evacuation plan, etc . Although we could have some difficulties at the first time, this topic is quite interesting and could be study deeply for developing more advantageous tools in the future.

References

[1] A. Shapiro, D. Dentcheva, and A. Ruszczynski, Lectures on stochastic programming: modeling and theory. SIAM, 2021.

[2] S. W. Wallace and W. T. Ziemba, Applications of stochastic programming. SIAM, 2005.

[3] L. Wang, "A two-stage stochastic programming framework for evacuation planning in disaster responses," Computers Industrial Engineering, vol. 145, p. 106458, 2020.

[4] https://www.gams.com/blog/2023/11/introducing-gamspy/

[5] Minimum-cost flow - Successive shortest path algorithm

[6] Lectures on stochastic programming Modeling.pdf