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

MATHEMATICAL MODELING (CO2011)

Assignment Report
Stochastic Programming and Applications

Advisor: Nguyễn Văn Minh Mẫn
Trần Hồng Tài
Members: Nguyễn Thành Phát – 2252605 (Group CC02, Leader)
Ngô Bảo Ngọc – 2252536 (Group CC02)
Vương Khang – 2250008 (Group CC02)
Đào Nam Anh – 2252016 (Group CC02)
Võ Hoàng Phúc – 2252650 (Group CC02)

HO CHI MINH CITY, DECEMBER 2023

Member list & Workload

No.	Full name	Student ID	Task	Contribution
1	Nguyễn Thành Phát	2252605	Python code and writing report	100%
2	Ngô Bảo Ngọc	2252536	Search for algorithms, theoretical background for stochastic programming	100%
3	Vương Khang	2250008	Collecting datas, and researching models for data simulating in Python	100%
4	Đào Nam Anh	2252016	Testing the algorithms, theoretical background for problem 1	100%
5	Võ Hoàng Phúc	2252650	Implementing the model, researching theoretical basis and the model used in problem 2	100%

Weekly work progress

Week	Work progress
1	Finding out the topic and assigning work to each member
2	Researching the assignment and theoretical basis
3	Understanding the algorithms, deciding the programming language and software to simulate
4	Implementing the algorithms and testing the models
5	Researching the `GAMSPy` package, `pandas` library, `numpy` library and how to build data frame in Python
6	Building the model in Python and writing the report
7	Checking the report and Python code

Contents

1 Introduction

1.1 Stochastic linear program (SLP)

A Stochastic linear program (SLP) is
Minimize:

$$Z = g(x) = f(x) = c^T x, s.t Ax = b, and T_x \geq h$$

where $x = (x_1, x_2, \ldots, x_n)$ (decision variables),
certain real matrix A and vector b (for deterministic constraints),
and with random parameters T, h in $T_x \geq h$ define chance or probabilistic constraints.

1.2 One-Stage Stochastic linear programming - No recourse

1. Matrix T= $T(\alpha)$ and (vector) h = $h(\alpha)$ express uncertainty via stochastic constraints
$T(\alpha)x \geq h(\alpha) \rightarrow \alpha_1 x_1 + \cdots + \alpha_n x_n \geq h(\alpha)$
2. Values (T, h) not known: they are unknown before an instance of model occurs, $h(\alpha)$ depends only on random a_j;
3. Uncertainty is expressed by probability distribution of random parameters $(a_j) = \alpha$
so deterministic LP is the degenerate case of Stochastic LP when a_j are constant,
We deal with decision problems where the vector
$x = (x_1, x_2, \ldots, x_n) \in X$ of decision variables must be made before the realization of parameter vector $\alpha \in \Omega$ is known.
Often we set lower and upper bounds for x via a domain $X = \{x \in R^n : l \leqslant x \leqslant u\}$

1.3 Two-stage Stochastic LP With Recourse : 2-SLPWR

The Two-stage Stochastic linear program With Recourse (2-SLPWR) or precisely with penalize corrective action generally described as

$$2 \text{ - SLP} : min_{x \in X} c^T x + min_{y(\omega) \in Y} E_\omega [q.y]$$

or in general

$$2 \text{ - SLP} : min_{x \in X, y(\omega) \in Y} E_\omega [c^T x + v(x, \omega)] \text{ with } v(x, \omega) := q.y$$

2 Problem 1

2.1 Problem Definition

Consider an industrial firm **F** where a manufacturer produces n products.[2]
There are different parts (sub-assemblies) to be ordered from in total m 3rd-party suppliers (the sites).
This picture shows a transportation plan of the industrial firm **F** with $m = 5$ from suppliers and $n = 8$ production locations (products, or warehouses).
A unit of product i requires $a_{ij} \geq 0$ units of part j , where $i = 1, \ldots, n$ and $j = 1, \ldots, m$. The demand for the products is modeled as a random vector $\omega = D = (D_1, D_2, \ldots, D_n)$, where each ω_i with density p_i follows the binomial distribution $Bin(10, \frac{1}{2})$.

- For the second stage:

Denote $b = (b_1, b_2, \ldots, b_m)$ built by **preorder cost** bj **per unit of part** j (before the demand is known). After the **demand** D, the manufacturer may decide which portion of the demand is to be satisfied. Denote the price to satisfy a unit for demand of product i is l_i and the unit selling price of this product is q_i. **The parts not used** are assessed salvage values $s_j < b_j, j = 1, \ldots, m$
Suppose the **number of parts to be ordered before production** is `decision variable` $x = (x_1, x_2, \ldots, x_m)$. After knowing the **demand D**, for an observed value $d = (d_1, d_2, \ldots, d_n)$ of the **random vector** D, we denote `decision variable` $z = (z_1, z_2, \ldots, z_n)$ - **the number of units produced** and `decision variable` $y = (y_1, y_2, \ldots, y_m)$ - **the number of parts left in inventory**.

We can find the best production plan by solving the following **stochastic linear program (LSP)**:

$$minZ = \sum_{i=1}^{n}(l_i - q_i)z_i - \sum_{j=1}^{m} s_j y_j$$

$$s.t \begin{cases} y_j = x_j - \sum_{i=1}^{n}(a_i j\ x_i) \text{j=1,...,m} \\ 0 \le z_i \le d_i \text{,i=1,...,n;}\ \ y_j \ge 0 \text{,j=1,...,m} \end{cases} \tag{1}$$

Then, the whole model of the second-stage can be equivalently expressed as:

$$MODEL = \begin{cases} min_{z,y} Z = c^T z - s^T y \\ where\ c_i := l_i - q_i\ are\ cost\ coefficients \\ y = x - A^T,\ where\ A = [a_{ij}]\ is\ matrix\ of\ dimension\ n \times m, \\ 0 \le z \le d,\ y \ge 0 \end{cases} \tag{2}$$

- For the first stage:

Since the manufacturer may decide which portion of the demand is to be satisfied, the available numbers of parts are not exceed, which means *production* $\le$ *demand*. Following the distribution-based approach, $Q(x) := E(Z(z,y)) = E_\omega[Z,\omega]$ denote the optimal value of the problem (1). The quantities x_j are determined from the following optimization problem:

$$min\ g(x,y,z) = b^T x + Q(x) = b^T x + E[Z(z)]$$

where $Q(x) = E_\omega[Z] = \sum_{i=1}^{n} p_i c_i z_i$ is taken with respect to the probability distribution of $\omega = D$.
The first part of the objective function represents the **pre-ordering cost** and x. In contrast, the second part represents the **expected cost** of the **optimal production plan** (2), given by the updated order quantities z, already employing random demand $D = d$ with their density.
In the next part, we will build up numerical models by using library `GAMSpy`.

2.2 Data pre-processing

```
import pandas as pd
import numpy as np
import sys
import gamspy
from gamspy import Container, Set, Parameter, Variable, Equation, Model, Sum, Sense
```

Beside `GAMSPy`, the packages that we implement in this problem is:

- `pandas`: for building up and modifying our data.
- `numpy`: for dealing with numerical data
- `sys`: for printing our result to the console.

After that, from `GAMSPy` library, we import `Container`, `Set`, `Parameter`, `Variable`, `Equation`, `Model`, `Sum`, `Sense` which are necessary for building the model with our data.
In the next part, we will let user input the data into the data frame. If user's number is less than 0, they will be forced to input the data. However, if the number is not in the correct format, we will set this value to `NA` values.
Preorder cost

```
preorder_cost=pd.DataFrame([[1,0],[2,0],[3,0],[4,0],[5,0]],columns=["Supplier","Cost"]).
    set_index("Supplier")
## Set NA
i=1
while i<=5:
  preorder_cost.loc[i,"Cost"]=np.nan
  i=i+1
i=1
```

```
while i<=5:
  try:
    a=-1
    while a<0:
      a=float(input("Please input the preorder cost from the supplier: "))
      b=a
      if b<0:
        print("Wrong format of the input!\n")
  except:
      print("Wrong format of the input!\n")
  else:
    preorder_cost.loc[i,"Cost"]=a
  i=i+1
```

Salvage value

```
salvage=pd.DataFrame([[1,0],[2,0],[3,0],[4,0],[5,0]],columns=["Supplier","Cost"]).
    set_index("Supplier")
## Set NA
i=1
while i<=5:
  salvage.loc[i,"Cost"]=np.nan
  i=i+1
i=1
while i<=5:
  try:
    a=-1
    while a<0 or a>=preorder_cost.loc[i,"Cost"]:
      a=float(input("Please input the salvage value:"))
      b=a
      if b<0 or b>=preorder_cost.loc[i,"Cost"]:
        print("Invalid input!\n")
  except:
    print("Invalid input!\n")
  else:
    salvage.loc[i,"Cost"]=a
  i=i+1
```

Since the salvage value has the condition $s_j < b_j, j = 1 \ldots, m$, we must add this condition to `while` loop.

Cost to satisfy a unit of demand for product i

```
cost_produce=pd.DataFrame([[1,0],[2,0],[3,0],[4,0],[5,0],[6,0],[7,0],[8,0]],columns=["
    Location","Cost"]).set_index("Location")
## Set NA
i=1
while i<=8:
  cost_produce.loc[i,"Cost"]=np.nan
  i=i+1
i=1
while i<=8:
  try:
    a=-1
    while a<0:
      a=float(input("Please input the cost to produce:"))
      b=a
      if b<0:
        print("Wrong format input\n")
  except:
    print("Wrong format input\n")
  else:
    cost_produce.loc[i,"Cost"]=a
  i=i+1
```

Cost to sell a unit of product i

```
cost_sell=pd.DataFrame([[1,0],[2,0],[3,0],[4,0],[5,0],[6,0],[7,0],[8,0]],columns=["
    Location","Cost"]).set_index("Location")
## Set NA
i=1
while i<=8:
  cost_sell.loc[i,"Cost"]=np.nan
  i=i+1
i=1
```

```
while i<=8:
  try:
    a=-1
    while a<0:
      a=float(input("Please input the cost to sell:"))
      b=a
      if b<0:
        print("Invalid input!Try again.\n")
  except:
    print("Invalid input!\n")
  else:
    cost_sell.loc[i,"Cost"]=a
  i=i+1
```

The number of units of part j to form a unit of product i, where $i = 1 \ldots, n$ and $j = 1, \ldots, m$

```
assemble=pd.DataFrame([[1,1,0],[1,2,0],[1,3,0],[1,4,0],[1,5,0],[1,6,0],[1,7,0],[1,8,0],
                       [2,1,0],[2,2,0],[2,3,0],[2,4,0],[2,5,0],[2,6,0],[2,7,0],[2,8,0],
                      [3,1,0],[3,2,0],[3,3,0],[3,4,0],[3,5,0],[3,6,0],[3,7,0],[3,8,0],
                      [4,1,0],[4,2,0],[4,3,0],[4,4,0],[4,5,0],[4,6,0],[4,7,0],[4,8,0],
                      [5,1,0],[5,2,0],[5,3,0],[5,4,0],[5,5,0],[5,6,0],[5,7,0],[5,8,0],
                      ],columns=["Source","Destination","Number"]).set_index(["Source","
    Destination"])
## Set NA
i=1
while i<=5:
  j=1
  while j<=8:
    assemble.loc[(i,j)]=np.nan
    j=j+1
  i=i+1
i=1
while i<=5:
  j=1
  while j<=8:
    try:
      a=-1
      while a<0:
       a=int(input("Please input number of each assembler:"))
       b=a
       if b<0:
        print("Invalid input!")
    except:
      print("Invalid input!")
    else:
      assemble.loc[(i,j)]=a
    j=j+1
  i=i+1
```

In this part, the data type must be a positive integer, so any data that is not in the correct format will be set to `NA`.
Since we are given two scenario $S = 2$, which each has density $p_s = \frac{1}{2}$ and our random vector ω_i with density p_i follows the binomial distribution $Bin(10, \frac{1}{2})$, the first scenario we can choose is d_i equals to the **mean value**(which is 5) and the second scenario is that d_i reaches its maximum value, $d_i = 10$.
Demand in two scenario

```
demand_mean=pd.DataFrame([[1,5],[2,5],[3,5],[4,5],[5,5],[6,5],[7,5],[8,5]],columns=["
    Location","Number"]).set_index("Location")
demand_best=pd.DataFrame([[1,10],[2,10],[3,10],[4,10],[5,10],[6,10],[7,10],[8,10]],
    columns=["Location","Number"]).set_index("Location")
```

2.3 Data loading

```
m=Container()
j=Set(container=m,description="Supplier",name="j",records=preorder_cost.index)
i=Set(container=m,description="Location",name="i",records=cost_produce.index)
```

The `Container` is important since it can manage our data, sets, parameters, variables and constraints. `Set` in `GAMSPy` corresponds exactly to the indices in the algebraic representation of the models. Here, `i`, `j` is the set of products and parts (subassemblies) respectively.

```
b=Parameter(container=m,name="b",domain=j,description="Pre-order cost of the supplier j",
    records=preorder_cost.reset_index())
s=Parameter(container=m,name="s",domain=j,description="Salvage values in the supplier j",
    records=salvage.reset_index())
l=Parameter(container=m,name="l",domain=i,description="Cost to produce location i",
    records=cost_produce.reset_index())
q=Parameter(container=m,name="q",domain=i,description="Cost to sell at location i",
    records=cost_sell.reset_index())
d1=Parameter(container=m,name="d1",domain=i,description="Demand at first scenario",
    records=demand_best.reset_index())
d2=Parameter(container=m,name="d2",domain=i,description="Demand at second scenario",
    records=demand_mean.reset_index())
a=Parameter(container=m,name="a",domain=[j,i],description="Number of products at supplier
     j to produce one product at supplier i",records=assemble.reset_index())
```

`Parameter` is data structure that helps us handle exogenous model data. Here,we let the scalar variables such as **b,s,l,q,d1,d2,a** be the `Parameter`.

2.4 Solve the model

2.4.1 Variable

```
y1=Variable(container=m,name="y1",domain=j,type="integer",description="The number of
    parts left in inventory in scenario 1")
y2=Variable(container=m,name="y2",domain=j,type="integer",description="The number of
    parts left in inventory in scenario 2")
z1=Variable(container=m,name="z1",domain=i,type="integer",description="The number of
    units produced in scenario 1")
z2=Variable(container=m,name="z2",domain=i,type="integer",description="The number of
    units produced in scenario 2")
x=Variable(container=m,name="x",domain=j,type="integer",description="The number of
    ordered parts")
```

Since there are two scenarios, we must have two types of variable y and z. Specifically, $y1$ and $z1$ represents the number of parts left in inventory and units produced after knowing the Demand $d1$; whereas, $y2$ and $z2$ stands for the number of parts left in inventory and units produced after knowing the Demand $d2$. Before knowing the demand, let us denote x be the number of ordered parts.

2.4.2 Equation

```
eq_1=Equation(container=m,name="First_equation",domain=j,description="Base equation")
eq_2=Equation(container=m,name="Second_equation",domain=i,description="1st equation")
eq_3=Equation(container=m,name="Third_equation",domain=j,description="Base-1 equation")
eq_4=Equation(container=m,name="Forth_equation",domain=i,description="2nd equation")

eq_6=Equation(container=m,name="Sixth_equ",type="regular",description="3rd equ")
eq_7=Equation(container=m,name="Seventh_equ",type="regular",description="4th equ")

eq_8=Equation(container=m,name="Eighth_eq",domain=i,description="8th equ")
eq_9=Equation(container=m,name="Nineth_eq",domain=j,description="9th equ")
eq_10=Equation(container=m,name="Tenth_eq",domain=j,description="10th equ")
eq_11=Equation(container=m,name="Eleth_eq",domain=i,description="11th equ")
eq_12=Equation(container=m,name="Twelth_eq",domain=j,description="12th equ")

### Scenario 1
eq_1[j]=Sum(i,a[j,i]*z1[i])==x[j]-y1[j]
eq_2[i]=z1[i]<=d1[i]
### Scenario 2
eq_3[j]=Sum(i,a[j,i]*z2[i])==x[j]-y2[j]
eq_4[i]=z2[i]<=d2[i]
eq_6[...]=Sum(j,x[j])<=Sum(i,d1[i])
eq_7[...]=Sum(j,x[j])<=Sum(i,d2[i])
### Constraint for integer
eq_8[i]=z1[i]>=0
eq_9[j]=y1[j]>=0
eq_10[j]=x[j]>=0
eq_11[i]=z2[i]>=0
eq_12[j]=y2[j]>=0
```

From problem 2, we build up three equations in each scenario. Scalar equations like eq_6 and eq_7 are based on the condition $production \leq demand$. The final step is to build up model.

2.4.3 Model

```
pro=Model(m,name="Scenario_1",equations=[eq_1,eq_2,eq_6,eq_9,eq_8,eq_10],problem="MIP",
    sense=Sense.MIN,objective=Sum(i,(l[i]-q[i])*z1[i])-Sum(j,s[j]*y1[j]))
pro1=Model(m,name="Scenario_2",equations=[eq_3,eq_4,eq_7,eq_10,eq_11,eq_12],problem="MIP"
    ,sense=Sense.MIN,objective=Sum(i,(l[i]-q[i])*z2[i])-Sum(j,s[j]*y2[j]))
pro2=Model(m,name="Final_round",equations=[eq_1,eq_2,eq_6,eq_3,eq_4,eq_7,eq_8,eq_9,eq_10,
    eq_11,eq_12],problem="MIP",sense=Sense.MIN,objective=Sum(j,b[j]*x[j])+(1/2)*(Sum(i,(l
    [i]-q[i])*z2[i])-Sum(j,s[j]*y2[j]))+ (1/2)*(Sum(i,(l[i]-q[i])*z1[i])-Sum(j,s[j]*y1[j
    ])))
```

The first two models *pro* and *pro1* gathers all the equation needed for two demand in two scenarios and their objective equations are from problem 2.The final model *pro2* and its objective equation are derived from problem 2.1.

2.5 Data simulation

We will simulate the `Parameter` b, l, q, s, a.

Supplier	Cost
1	1.1
2	2.2
3	3.3
4	4.4
5	5.5

(a) *Preorder cost*

Supplier	Cost
1	0.1
2	1.2
3	2.3
4	3.4
5	4.5

(b) *Salvage value*

Location	Cost
1	1.0
2	2.0
3	3.0
4	4.0
5	5.0
6	6.0
7	7.0
8	8.0

(c) *Cost to product*

Location	Cost
1	4.40
2	5.50
3	6.60
4	7.70
5	8.80
6	9.90
7	10.01
8	10.02

(d) *Cost to sell*

Source	Destination	Number		Source	Destination	Number
1	1	1		4	6	0
1	2	1		4	7	1
1	3	0		4	8	1
1	4	1		5	1	1
1	5	1		5	2	1
1	6	0		5	3	1
1	7	0		5	4	1
1	8	0		5	5	0
2	1	0		5	6	0
2	2	1		5	7	0
2	3	0		5	8	1
2	4	1				
2	5	0				
2	6	1				
2	7	0				
2	8	1				
3	1	1				
3	2	1				
3	3	1				
3	4	0				
3	5	1				
3	6	0				
3	7	0				
3	8	0				
4	1	0				
4	2	1				
4	3	0				
4	4	0				
4	5	0				

(e) *Number of subassemblies to form the product*

With the data above, we will use the `GAMSPy` to solve and print out our optimized value.

```
pro.solve(output=sys.stdout)
pro1.solve(output=sys.stdout)
pro2.solve(output=sys.stdout)
### Print the result to the console
print(x.records)
print(y1.records)
print(z1.records)
print(pro.objective_value)

print(x.records)
print(y2.records)
print(z2.records)
print(pro1.objective_value)

print(x.records)
print(y1.records)
print(z1.records)
print(y2.records)
print(z2.records)
print(pro2.objective_value)
```

After running the code, the result is shown below (the "level" columns represent our result with respect to each variable after optimizing).

```
print(x.records)
print(y1.records)
print(z1.records)
print(pro.objective_value)
```

```
   j  level  marginal  lower  upper  scale
0  1    0.0       0.0    0.0    inf    1.0
1  2    0.0       0.0    0.0    inf    1.0
2  3    0.0       0.0    0.0    inf    1.0
3  4    0.0       1.1    0.0    inf    1.0
4  5   80.0       0.0    0.0    inf    1.0
   j  level  marginal  lower  upper  scale
0  1    0.0       4.4    0.0    inf    1.0
1  2    0.0       3.3    0.0    inf    1.0
2  3    0.0       2.2    0.0    inf    1.0
3  4    0.0       0.0    0.0    inf    1.0
4  5   80.0       0.0    0.0    inf    1.0
   i  level  marginal  lower  upper  scale
0  1    0.0      10.10    0.0    inf    1.0
1  2    0.0      17.90    0.0    inf    1.0
2  3    0.0       5.40    0.0    inf    1.0
3  4    0.0       9.80    0.0    inf    1.0
4  5    0.0       5.20    0.0    inf    1.0
5  6    0.0       0.60    0.0    inf    1.0
6  7    0.0       0.39    0.0    inf    1.0
7  8    0.0      10.38    0.0    inf    1.0
-360.0
```

(a) *Result of Scenario 1*

```
print(x.records)
print(y2.records)
print(z2.records)
print(pro1.objective_value)
```

```
   j  level  marginal  lower  upper  scale
0  1    0.0       0.0    0.0    inf    1.0
1  2    0.0       0.0    0.0    inf    1.0
2  3    0.0       0.0    0.0    inf    1.0
3  4    0.0       1.1    0.0    inf    1.0
4  5   40.0       0.0    0.0    inf    1.0
   j  level  marginal  lower  upper  scale
0  1    0.0       4.4    0.0    inf    1.0
1  2    0.0       3.3    0.0    inf    1.0
2  3    0.0       2.2    0.0    inf    1.0
3  4    0.0       0.0    0.0    inf    1.0
4  5   40.0       0.0    0.0    inf    1.0
   i  level  marginal  lower  upper  scale
0  1    0.0      10.10    0.0    inf    1.0
1  2    0.0      17.90    0.0    inf    1.0
2  3    0.0       5.40    0.0    inf    1.0
3  4    0.0       9.80    0.0    inf    1.0
4  5    0.0       5.20    0.0    inf    1.0
5  6    0.0       0.60    0.0    inf    1.0
6  7    0.0       0.39    0.0    inf    1.0
7  8    0.0      10.38    0.0    inf    1.0
-180.0
```

(b) *Result of Scenario 2*

```
print(y1.records)
print(z1.records)
print(y2.records)
print(z2.records)
print(pro2.objective_value)
```

```
   j  level  marginal  lower  upper  scale
0  1    0.0       0.0    0.0    inf    1.0
1  2   10.0       0.0    0.0    inf    1.0
2  3    0.0       0.0    0.0    inf    1.0
3  4    0.0       1.0    0.0    inf    1.0
4  5    0.0       1.0    0.0    inf    1.0
   j  level  marginal  lower  upper  scale
0  1    0.0       0.0    0.0    inf    1.0
1  2    0.0       1.0    0.0    inf    1.0
2  3    0.0       1.0    0.0    inf    1.0
3  4    0.0       0.0    0.0    inf    1.0
4  5    0.0       0.0    0.0    inf    1.0
   i  level  marginal  lower  upper  scale
0  1    0.0      2.750    0.0    inf    1.0
1  2    0.0      6.000    0.0    inf    1.0
2  3    0.0      2.600    0.0    inf    1.0
3  4    0.0      2.050    0.0    inf    1.0
4  5    0.0      0.300    0.0    inf    1.0
5  6   10.0      0.000    0.0    inf    1.0
6  7    0.0      0.195    0.0    inf    1.0
7  8    0.0      4.540    0.0    inf    1.0
   j  level  marginal  lower  upper  scale
0  1    0.0       1.0    0.0    inf    1.0
1  2    5.0       0.0    0.0    inf    1.0
2  3    0.0       0.0    0.0    inf    1.0
3  4    0.0       0.0    0.0    inf    1.0
4  5    0.0       0.0    0.0    inf    1.0
   i  level  marginal  lower  upper  scale
0  1    0.0      2.750    0.0    inf    1.0
1  2    0.0      5.000    0.0    inf    1.0
2  3    0.0      1.600    0.0    inf    1.0
3  4    0.0      2.050    0.0    inf    1.0
4  5    0.0      0.300    0.0    inf    1.0
5  6    5.0      0.000    0.0    inf    1.0
6  7    0.0      0.195    0.0    inf    1.0
7  8    0.0      3.540    0.0    inf    1.0
-10.25
```

(c) *Result of First-stage*

3 Problem 2

3.1 Theoretical basis

3.1.1 Introduction

In this problem, we are going to study about the two-stage stochastic programming framework for evacuation planning in disaster responses proposed by Li Wang [3]. The first stage decisions are robust pre-event evacuation plan for all level of disaster. The second stage decisions are conditional on the first stage, relating to the evacuation plan for affected people after the event in response to the scenario-based road conditions. This section represent the theoretical basis which is mainly based on Ref [3].

Extreme unnatural disasters (terrorist attacks, political unrest, war, etc.) and natural disasters (earthquakes, storms, fires, hurricanes, etc.) can strike a town without much notice and cause significant damage,and a large number of deaths. The primary objective of emergency response is to offer providing impacted individuals with refuge and aid as soon as possible. This study falls under the latter category and seeks to solve the problem of creating a generic modeling framework in the event of a disaster in order to plan the affected people's evacuation route.The evacuation problem is best understood as a stochastic programming problem in which capacity and travel time both contribute to randomness. Put another way, there are random link travel times and capacities as a result of potential damage to a particular road that would make it impossible for people to flee catastrophe areas.

Since the internet and contemporary information technology have advanced, users can typically access real-time road network information via a variety of sources following unforeseen situations. Thus, the subject of relevance in this study is how to determine the optimal evacuation route for impacted individuals under the assumption of real-time information availability. Moreover, recourse-based stochastic programming is used to identify non-anticipative decisions that need to be made first in order to determine the realizations of certain random variables in a way that minimizes the overall projected costs of potential recourse actions. In order to capture the randomness resulting from earthquake magnitude and impact, the evacuation problem in this work is modeled as a scenario-based two-stage stochastic programming model.

One of the key elements of emergency response following a disaster is the optimal evacuation plan for the impacted population. Apparently, not much research has been done on the method of determining the affected people's evacuation plan by combining a priori (pre-disaster) and adaptive (post-disaster) path selection, which can be accomplished by two-stage stochastic programming. There are a limited number of scenarios that can represent the randomness, and each one has a known probability. This approach to dealing with randomness is frequently used in a variety of disaster relief management situations. More specifically, during the event, a portion of the transportation network may be destroyed, resulting in travel times and capacities. Put differently, non-anticipative first-stage decisions are taken before uncertainty is realized. Following the realization of stochastic travel times and capacities, decisions are made for the second stage (recourse), which are contingent on the decisions made in the first stage. Because of this, the goal is to create the optimal pre-event evacuation plan possible in the first stage, which will be confronted with uncertain circumstances in the second stage.

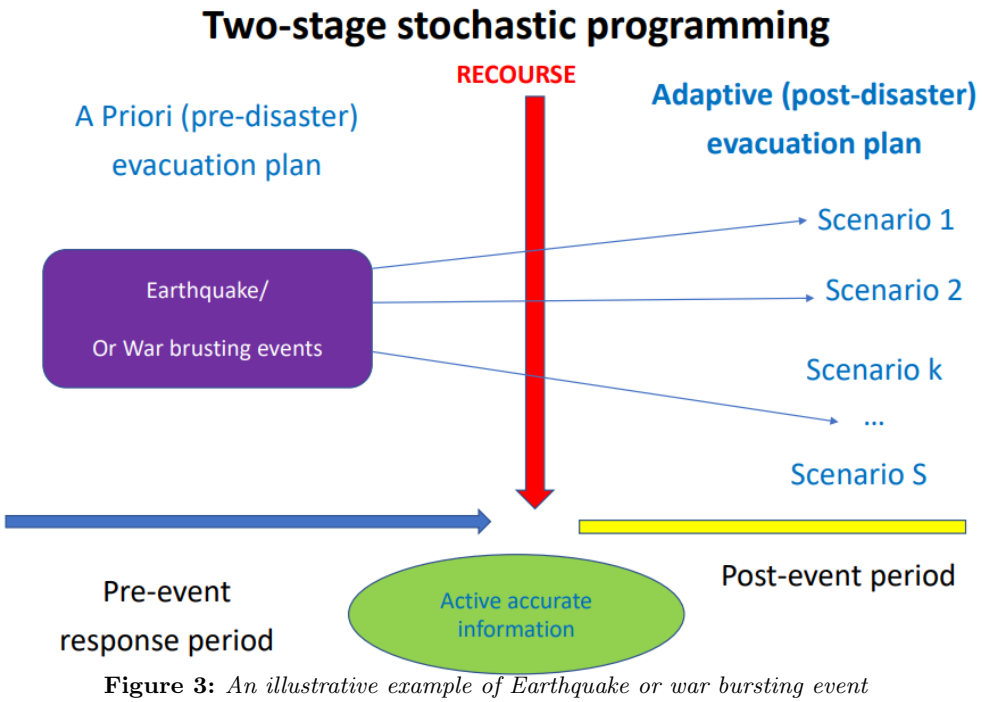

Figure 3: *An illustrative example of Earthquake or war bursting event*

Now, we will take earthquake into account to show the evacuation process of affected population. During the process, it is assumed that the impacted people can obtain early warning information and leave the hazardous area in their own cars. Initially, emergency response and evacuation are carried out without precise knowledge of the extent and degree of the affected area. Therefore the response should be dependent on a number of scenarios. As a result, the evacuation process ought to be split into two stages based on when precise information is acquired. In the first stage, the choice to evacuate must be made prior to the realization of any uncertain future events, and the second stage evacuation plan is decided upon following the discovery of any uncertain information. Generally speaking the objective is to make the optimal evacuation planning in the first stage under uncertainty to be faced in the second stage.

The concept of two-stage stochastic evacuation planning problem is specifically illustrated by a simple network with 8 nodes and 15 links. Nodes 1 and 8 are assumed to be disaster area and safe area respectively, and four cars, namely a, b, c and d, are needed to be evacuated to safe areas. In the priori plan, cars a and b are supposed to travel along the paths $1- > 2- > 3- > 4- > 6- > 8$, c and d travel along the path $1- > 3- > 5- > 6- > 7- > 8$. After the time threshold T, the latter part of the plan may be modified in an adaptive manner. From figure shown, it can be seen that before the time threshold, the sub-trip in an adaptive evacuation plan under 2 scenarios are the same as the priori plan, a and b travel along paths $1- > 2- > 3- > 4$ while c and d evacuate along $1- > 3- > 5$. When the precise data is accessible, that is after the time threshold, the two scenarios' adaptive evacuation plans differ from the a priori evacuation plan. For instance, car a travel along the path $7- > 8$ and $6- > 8$ in two scenarios instead of the path $6- > 8$ in the initial plan.

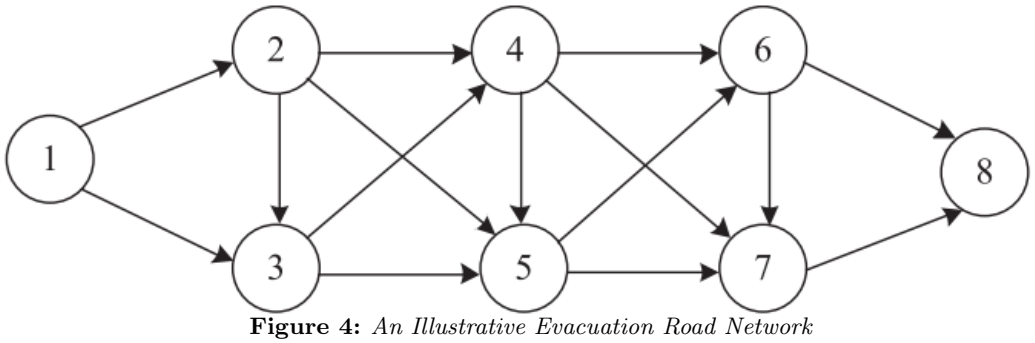

Figure 4: *An Illustrative Evacuation Road Network*

3.1.2 Multiple sources and sinks conversion

There are multiple sources and sinks in the real evacuation network in the two-stage stochastic evacuation problem. As a result, the physical network having several sources should be transformed into a comparable network with a single supersource, even to multiple sinks. In actuality, the network receives K as supersource. The dummy arc (k, i) has the travel time with value 0, $c_{Ki} = 0$. Let the capacity on the dummy arc be the value of the supply at node i, $u_{Ki} = d_i$ .Thus, the supply value at the supersource can be determined as $d_k = \sum_{i \in K} d_i$. Equivalently, the supersink l can be transformed in the same way.

The arc travel time and capacity will change when the departure time changes so a time-dependent network is designed. When adding a supersource, the travel times on the dummy arcs are $c_{Ki}(t) = 0, t \in 0, 1, \ldots, T$ and the capacity $u_{Ki}(t) = d_i(t)$. Since the time of arriving at each flow's sink perhaps distinct from one another, we first add a copy j' for each original sink $j, j \in D$. For each sink,we add the arc (j, j') to the network with infinite capacity, $u_{jj'}(t) = \infty$ and the travel time of the arc is assumed to be 0, $c^s_{jj'}(t) = 0$. Then, a self loop is added for each node j', where $u_{j'j'}(t) = \infty$ and $c^s_{j'j'}(t) = 1$. Moreover, the capacity of arc (j, l) is equal to the demand of node j at time T, $u_{j'l}(T) = b_j(T)$ and equal to 0 at other times, and the time travel of the arc is assumed to be 0 .

3.1.3 The description of the two-stage stochastic evacuation problem by the min cost flow problem

The main concern is to minimize the loss of life when a disaster strikes. The careful planning of people's movements from hazardous to safe areas is formulated in this study as a min-cost flow problem with random link travel times and capacities. The root goal is to transfer people in a capacity-cost network $G(V, A, C, U, D)$ from hazardous areas to safe areas in the shortest amount of time possible, where V is the set of nodes, A is the set of links with random travel time and capacity, $C(i, j)$ represents the travel time on link $(i, j) \in A$, denoted by c_{ij}, $U(i, j)$ represents the capacity of link (i, j), denoted by u_{ij} and $D(i)$ represents the flow at node $i \in V$, denoted by d_i. It is assumed that the number of impacted individuals has no effect on the link travel time. As far as the problem is concerned, sources denote the hazardous regions, sinks represent the safe regions, and other nodes are the network's intersections. The links stand in for the affected individuals' transportation roads. More specifically, the scenario where the first and second stages emerge in distinct time phases within the same evacuation network is represented by the two-stage stochastic programming model with recourse.

The impacted individuals must be evacuated from source nodes to other nodes before link travel times and capacities are realized in the second stage since link travel times and capacities are only probabilistically known in the first stage. In the second stage, as the link travel times and capacities are realized,the evacuation plan will be entirely decided. The decision for the second stage offers shorter evacuation times solved by the first stage, which is not feasible for the data obtained at that moment. In this case, the objective function will consist of the first-stage penalty costs and the expected value of the second stage recourse costs.

3.1.4 Model notations

Parameters:

Symbol	Defination
V	The set of nodes
A	The set of links
i,j	The index of nodes $i, j \in V$
(i, j)	The index of directed links $(i, j) \in A$
s	The index of scenario
S	Total number of scenarios
v	The supply value of source node
$T^{\sim}$	The time threshold
T	The total number of time intervals
u_{ij}	The capacity of physical link (i, j)

Symbol	Defination
$u_{ij}^s(t)$	The capacity of physical link (i,j) in scenario s at time t
$c_{ij}^s(t)$	The travel time of link (i,j) in scenario s at time t
μ_s	The probability in scenario s
d_i	Supply at node i
$d_i^s(t)$	Supply at node i in scenario s at time T
p_{ij}	The penalty of link (i,j)

3.1.5 Decision variable

Decision variable	Definition
x_{ij}	The flow on link (i,j)
$y_{ij}^s(t)$	The flow on link (i,j) in scenario s at time t

3.1.6 The first stage

In this stage, a feasible plan should be determined from super-source to super-sink.

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

where d_i is defined:

$$d_i = \begin{cases} v, i = s \\ -v, i = t \\ 0, otherwise \end{cases} \tag{3}$$

Meanwhile, the flow on each link must also satisfy the capacity constraint:

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

Since the flow balance constraint can potentially generate loops in evacuation network, we have to eliminate loops by adding a parameter called penalty. The penalty function can be defined as:

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

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

3.1.7 The second stage

In order to create a robust, ideal evacuation plan, the first stage is evaluated in the second stage, in which the impacted individuals will be provided with adaptive paths that require the least amount of time for evacuation. In advance of the time threshold, the evacuation still follows the initial plan in the first stage.

The coupling constraints for the evacuation plan of each scenario before the time threshold $T^{\sim}$ can be formulated as follows :

$$\sum_{t\leq T^{\sim}} y_{ij}^s(t) = x_{ij}, (i,j) \in A, s = 1,2,\ldots,S$$

A second-stage model is established with the ultimate goal of minimizing the overall time of the affected people evacuated from the dangerous area to the safe area in each scenario:

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

such that:

$$\begin{cases} \sum_{(i_t,j_{t'})\in A_s} y_{ij}^s(t) - \sum_{(j_{t'},i_t)\in A_s} y_{ij}^s(t') = d_i^s(t), \forall i \in V, t \in \{0,1,\ldots,T\}, s = 1,2\ldots,S \\ 0 \le y_{ij}^s(t) \le u_{ij}^s(t), \forall (i,j) \in A, t \in \{0,1,\ldots,T\}, s = 1,2\ldots,S \\ \sum_{t\le T^\sim} y_{ij}^s(t) = x_{ij}, (i,j) \in A, s = 1,2\ldots,S \end{cases} \tag{4}$$

Moreover, the penalty for the prior evacuation plan and the expected overall evacuation time of each scenario must be minimized:

$$min \sum_{i,j\in A} p_{ij}x_{ij} + \sum_{i,j\in A_s} c_{ij}^s(t).y_{ij}^s(t)$$

Since the second stage of the model has a limited number of scenarios, the above time-dependent function is equivalent to the following:

$$min \sum_{i,j\in A} p_{ij}x_{ij} + \sum_{s=1}^{S} \left(\mu_s \sum_{(i,j)\in A_s} c_{ij}^s(t).y_{ij}^s(t) \right)$$

where μ_s is the probability in scenario s.
Putting it all together, we have a following model:

$$\begin{cases} min \sum_{i,j\in A} p_{ij}x_{ij} + \sum_{s=1}^{S} \left(\mu_s \sum_{(i,j)\in A_s} c_{ij}^s(t).y_{ij}^s(t) \right) \\ s.t. \\ \sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ij} = d_i, \forall i \in V \\ 0 \le x_{ij} \le u_{ij}, \forall (i,j) \in A \\ \sum_{(i_t,j_{t'})\in A_s} y_{ij}^s(t) - \sum_{(j_{t'},i_t)\in A_s} y_{ij}^s(t') = d_i^s(t), \forall i \in V, t \in \{0,1,\ldots,T\}, s = 1,2\ldots,S \\ 0 \le y_{ij}^s(t) \le u_{ij}^s(t), \forall (i,j) \in A, t \in \{0,1,\ldots,T\}, s = 1,2\ldots,S \\ \sum_{t\le T^\sim} y_{ij}^s(t) = x_{ij}, (i,j) \in A, s = 1,2\ldots,S \end{cases} \tag{5}$$

3.1.8 Solution algorithm

Because of the complicated coupling constraint in this model, it cannot be solved in polynomial time. This section attempts to relax the coupling constraint into the goal function using the Lagrangian relaxation method. In particular, the original model may be easily solved by exact algorithms since it is divided into two problems: a standard min-cost flow problem and a time-dependent one.
The coupling constraint characterizes the relationship between selection of a physical link and corresponding scenario-based time-dependent arcs. Hence, we introduce the Lagrangian multiplier $\alpha_{ij}^s(t), (i,j) \in A, s = 1,2,\ldots,S, t \le T^\sim$ for this constraint leading this constraint to be relaxed into the objective function:

$$\sum_{s=1}^{S} \sum_{(i,j)\in A} \alpha_{ij}^s \left(\left(\sum_{t\le T^\sim} y_{ij}^s(t) \right) - x_{ij} \right)$$

As a result, we have a model

$$\begin{cases} min \sum_{i,j\in A} p_{ij}x_{ij} + \sum_{s=1}^{S} \left(\mu_s \sum_{(i,j)\in A_s} c_{ij}^s(t).y_{ij}^s(t) \right) + \sum_{s=1}^{S} \sum_{(i,j)\in A} \alpha_{ij}^s \left(\left(\sum_{t\le T^\sim} y_{ij}^s(t) \right) - x_{ij} \right) \\ s.t \\ \sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ij} = d_i, \forall i \in V \\ 0 \le x_{ij} \le u_{ij}, \forall (i,j) \in A \\ \sum_{(i_t,j_{t'})\in A_s} y_{ij}^s(t) - \sum_{(j_{t'},i_t)\in A_s} y_{ij}^s(t') = d_i^s(t), \forall i \in V, t \in \{0,1,\ldots,T\}, s = 1,2\ldots,S \\ 0 \le y_{ij}^s(t) \le u_{ij}^s(t), \forall (i,j) \in A, t \in \{0,1,\ldots,T\}, s = 1,2\ldots,S \end{cases} \tag{6}$$

The variables X and Y can be separated from each other in the above relaxed model. That is, by combining similar terms, the relaxed model is decomposed into two subproblems as follows:

3.1.8.a Subproblem 1: Min-cost Flow problem

$$\begin{cases} minSP1(\alpha) = \sum_{(i,j)\in A}\left(p_{ij} - \sum_{s=1}^{S}\sum_{t\leq T^{\sim}}\alpha_{ij}^{s}(t)\right)x_{ij} \\ s.t \\ \sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ij} = d_i, \forall i \in V \\ 0 \leq x_{ij} \leq u_{ij}, \forall (i,j) \in A \end{cases} \tag{7}$$

The objective function of subProblem 1 can be defined as $g_{ij} = p_{ij} - \sum_{s=1}^{S}\sum_{t\leq T^{\sim}}\alpha_{ij}^{s}(t)$ to present the generalized cost of each link.
This subproblem can be solved by the successive shortest path algorithm.

3.1.8.b Subproblem 2: Time-Dependent Min-cost Flow Problem

$$\begin{cases} minSP2(\alpha) = \sum_{s=1}^{S}\sum_{(i,j)\in A}\left(\sum_{t\in\{0,1,\ldots,T\}}\mu_s \cdot c_{ij}^{s}(t) + \sum_{t\leq T^{\sim}}\alpha_{ij}^{s}(t)\right)y_{ij}^{s}(t) \\ s.t \\ \sum_{(i_t,j_{t'})\in A_s} y_{ij}^{s}(t) - \sum_{(j_{t'},i_t)\in A_s} y_{ij}^{s}(t') = d_i^{s}(t), \forall i \in V, t \in \{0,1,\ldots,T\}, s = 1,2\ldots,S \\ 0 \leq y_{ij}^{s}(t) \leq u_{ij}^{s}(t), \forall (i,j) \in A, t \in \{0,1,\ldots,T\}, s = 1,2\ldots,S \end{cases} \tag{8}$$

Algorithm 1 Successive shortest path algorithm for min-cost flow problem

1: Take variable x as a feasible flow between any OD and it has the minimum delivery cost in the feasible flows with the same flow value.
2: The algorithm will terminate if the flow value of x reaches v or there is no minimum cost path in the residual network $(V, A(x), C(x), U(x), D)$; otherwise, the shortest path with the maximum flow is calculated by label-correcting algorithm, and then go to Step 3. The functions $A(x), C(x), U(x)$ in the residual network can be defined as:

$$A(x) = \{(i,j)|(i,j) \in A, x_{ij} < u_{ij}\} \cup \{(j,i)|(j,i) \in A, x_{ij} > 0\}$$

$$C(x) = \begin{cases} c_{i,j}, (i,j) \in A, x_{i,j} < u_{i,j} \\ -c_{j,i}, (j,i) \in A, x_{ji} > 0 \end{cases} \tag{9}$$

$$U_{ij}(x) = \begin{cases} u_{ij}, (i,j) \in A, x_{ij} < u_{ij} \\ x_{ji}, (j,i) \in A, x_{ji} > 0 \end{cases} \tag{10}$$

3: Increase the flow along the minimum cost path. If the increased flow value does not exceed v, go to Step 2.

Since the time period is split into two phases, the generalized cost is defined as a piecewise function :

$$g_{ij}^{s}(t) = \begin{cases} \mu_s \cdot c_{ij}^{s}(t) + \alpha_{ij}^{s}(t), t \leq T^{\sim} \\ \mu_s \cdot c_{ij}^{s}(t), T^{\sim} < t \leq T \end{cases} \tag{11}$$

By solving the 2 subproblems, the optimal objective value Z_{LR}^{*} for the relaxed model with Lagrangian multiplier can be defined as follow:

$$Z_{LR}^{*} = Z_{SP1}^{*}(\alpha) + Z_{SP2}^{*}(\alpha)$$

3.2 Analysis of the algorithm

For each vertex $i \in V$, let us denote π_i be the potential which associates with vertex i. Let c_{ij}^{π} be the reduced cost of an edge $(i,j), i,j \in V$ and the formula of c_{ij}^{π} is given:

$$c_{ij}^{\pi} = c_{ij} + \pi_i - \pi_j$$

Then, we define:

$$z(x,\pi) = \sum_{(i,j)\in E} c_{ij}^{\pi} \cdot x_{ij}$$

$$z(x,\pi) = \sum_{(i,j)\in E} (c_{ij} + \pi_i - \pi_j) \cdot x_{ij}$$

$$z(x,\pi) = z(x) + \sum_{(i,j)\in E} +\pi_i \cdot x_{ij} - \sum_{(i,j)\in E} \pi_j \cdot x_{ij}$$

$$z(x,\pi) = z(x) + \sum_{i\in V} \pi_i \cdot \sum_{j:(i,j)\in E} x_{ij} - \sum_{j\in V} \pi_j \cdot \sum_{i:(i,j)\in E} x_{ij}$$

$$z(x,\pi) = z(x) + \sum_{i\in V} \pi_i \left(\sum_{j:(i,j)\in E} x_{ij} - \sum_{i:(i,j)\in E} x_{ij} \right)$$

$$z(x,\pi) = z(x) + \sum_{i\in V} \pi_i \cdot d_i$$

Since $\sum_{i\in V} \pi_i \cdot d_i$ is constant, $z(x)$ minimizes if and only if $z(x,\pi)$ minimizes.
We also let the residual edge of the edge (i,j) be $r_{ij} = u_{ij} - x_{ij}$. This is useful for expressing the flows from vertex i to vertex j.

3.2.1 Cycle-canceling algorithm

This sub section describes the negative cycle optimal conditions and cycle-canceling algorithm. Firstly, we have some important theorem in [1]:

- `Theorem 1`:Let G be a transportation network. Suppose that G contains no uncapacitated negative cost cycle and there exists a feasible solution of the minimum cost flow problem. Then the optimal solution exists.
- `Theorem 2`: Let x^* be a feasible solution of a minimum cost flow problem. Then x^* is an optimal solution if and only if the residual network G_{x^*} contains no negative cost (directed) cycle.

Proof also can be found in [1].

Algorithm 2 Cycle-canceling algorithm

```
Find a feasible flow x in the network
while G_x contains negative cycle do
    Find the negative cycle C
    γ := min{(i,j) ∈ C|u_ij}
    Augment γ units of flow along the cycle C
    Update G_x
end while
```

3.2.2 Successive shortest path algorithm

The cycle-cancel algorithm is considered as a sub-task in the maximum flow problem. The successive shortest path algorithm search for the maximum flow and optimizes the objective function.
The idea is that we will send flow from the node s to node t. Then we find the argument path, update the residual network, find the shortest path and augment the flow until the residual network does not contain a path from s to t.
The successive shortest path algorithm can be used when G does not contain any negative cost cycles. Otherwise, we can not define the shortest path. When the value of current flow is zero, the transportation network G contains no negative cost cycle . Suppose that after some augmenting steps we have flow x and G_x still contains no negative cycles. If x is maximal then it is optimal, according to `Theorem 2`. Otherwise, let us denote the next successfully found shortest path in G_x by P.
In order to making all the edges' cost non-negative, we will use the Bellman-Ford's algorithm. Because

dealing with residual costs does not change the shortest path, we can rely on transformed network and implement Dijkstra's algorithm in order to find the successive path. In order to make our cost not less than zero, we will update the node potentials and reduce costs after the shortest path founded.

Algorithm 3 Reduce cost$(\pi_i, i \in E_x)$

1: **for** each $(i, j) \in E_x$ **do**
2: $\quad c_{ij} := c_{ij} + \pi_i - \pi_j$
3: $\quad c_{rev_{ij}} := 0$
4: **end for**

After reducing the cost of each arc, the shortest path from s to t arc has zero cost while the other arcs which do not involve in the shortest path to any vertex has a positive cost. Hence, we assign $c_{rev_{ij}} := 0$. The augmentation (along the found path) adds reversal arc (j,i) and due to the fact that (reduced) cost $c_{ij} = 0$ we make $c_{rev_{ij}} := 0$ beforehand. Hence we get $O(n^2mA)$ complexity of the successive shortest path algorithm.

Algorithm 4 Successive shortest path algorithm

1: Add the source and sink node.
2: Initialize the flow x to zero
3: Implement Bellman-Ford algorithm to form node potentials π
4: Reduce cost(π)
5: **while** G_x contains a path from s to t **do**
6: $\quad$ Implement Dijkstra's algorithm to find a shortest path P from s to t
7: $\quad$ Reduce cost(π)
8: $\quad$ Augment current flow x along P
9: $\quad$ Update G_x
10: **end while**

Notice that Bellman-Ford algorithm is used only once to make the cost non-negative and it takes $O(nm)$ (since there may be negative edges), the it takes $O(nA)$ for Dijkstra's algorithm, where A is an upper bound to the largest supply node.

3.3 Implementation of the algorithm

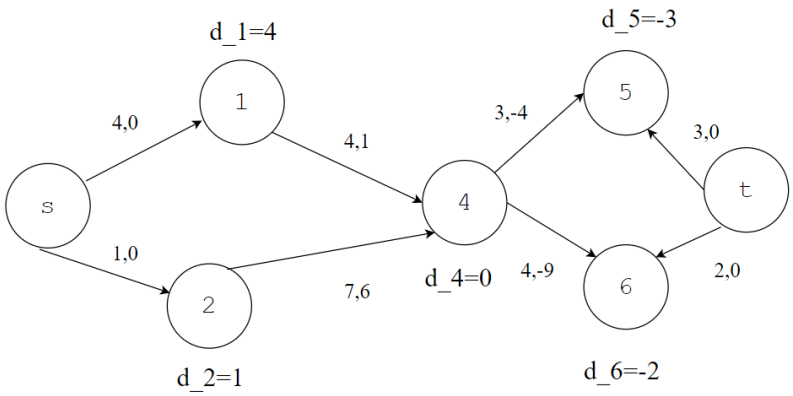

Figure 5: *An example of a small grid network.*

In Figure 5 let the capacity of the two disaster nodes such as 1, 2 and two safe nodes 5, 6 be d_1 and d_2, d_5 and d_6. We let $d_1 = 4, d_2 = 1, d_5 = -3, d_6 = -2$ and $d_4 = 0$. On every edge of the directed graph in Figure 5, we define the two number separated by the comma, the capacity and cost. We also add a source, sink node and define its capacity, cost as shown in Figure 5. Then we calculate the node potentials by using Bellman-Ford's algorithm as shown in the Table 4.

Node	s	1	2	4	5	6	t
$\emptyset$	0	∞	∞	∞	∞	∞	∞
s	0	0	0	∞	∞	∞	∞
1	0	0	0	1	∞	∞	∞
2	0	0	0	1	∞	∞	∞
4	0	0	0	1	-3	-8	∞
6	0	0	0	1	-3	-8	-8
5	0	0	0	1	-3	-8	-8

Table 4: *Bellman-Ford's algorithm to deal with negative cost*

After running Bellman-Ford algorithm, we get the result of the node potentials in Figure 6.

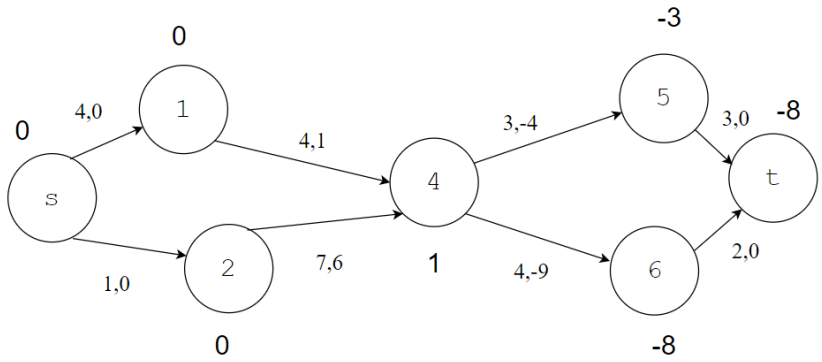

Figure 6: *Node potentials after line 3*

Then, we try to make all cost non-negative by implementing the `Reduce cost` function in Figure 7.

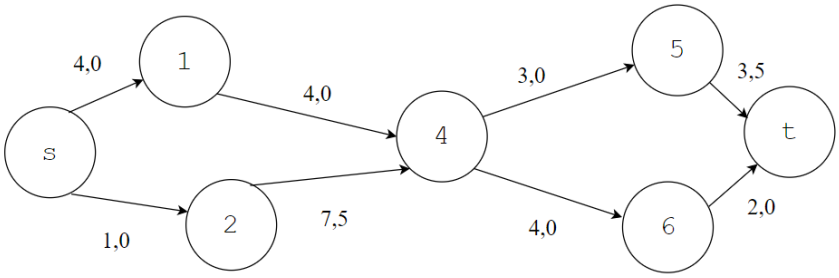

Figure 7: *Reduce cost after line 4*

Then, we use Dijkstra's algorithm to find the shortest path as shown in Table 5.

Node	s	1	2	4	5	6	t
$\emptyset$	0	∞	∞	∞	∞	∞	∞
s	0	0	0	∞	∞	∞	∞
1	0	0	0	0	∞	∞	∞
2	0	0	0	0	∞	∞	∞
4	0	0	0	0	0	0	∞
5	0	0	0	0	0	0	5
6	0	0	0	0	0	0	0
t	0	0	0	0	0	0	0

Table 5: *Dijkstra's algorithm in the first iteration*

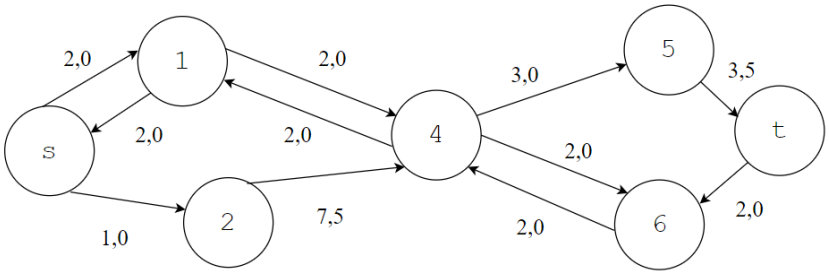

Figure 8: *The first augmenting path $s-1-4-6-t$ is found*

In the second iteration, we use Dijkstra's algorithm to find the shortest path as shown in Table 6.

Node	s	1	2	4	5	6	t
$\emptyset$	0	∞	∞	∞	∞	∞	∞
s	0	0	0	∞	∞	∞	∞
1	0	0	0	0	∞	∞	∞
2	0	0	0	0	∞	∞	∞
4	0	0	0	0	0	0	∞
5	0	0	0	0	0	0	5
6	0	0	0	0	0	0	5
t	0	0	0	0	0	0	5

Table 6: *Dijkstra's algorithm in the second iteration*

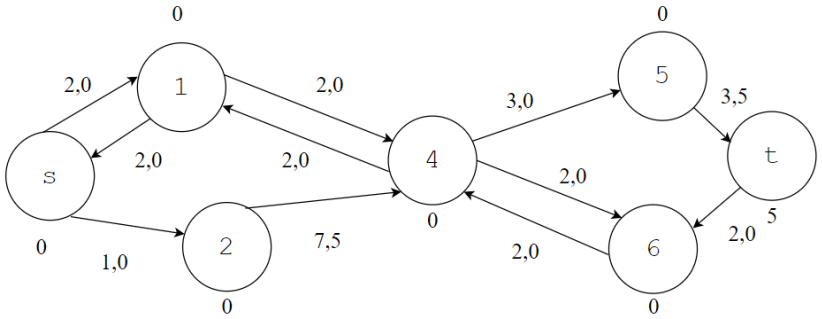

Figure 9: *New node potential is calculated*

After that, we reduce the cost in Figure 10.

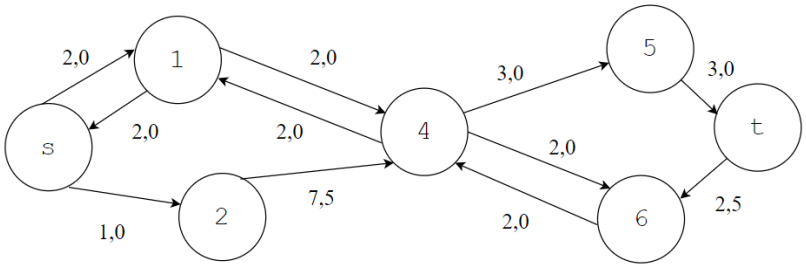

Figure 10: *Reduce cost in second iteration*

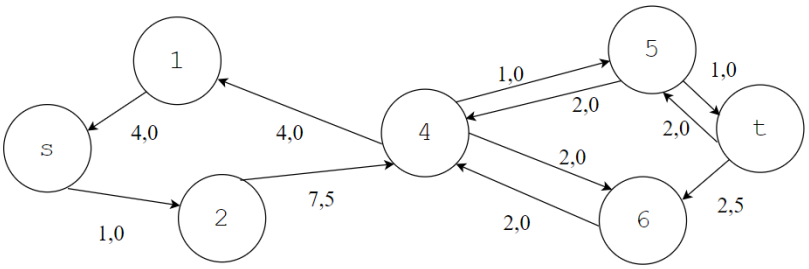

Figure 11: *The second augmenting path $s - 1 - 4 - 5 - t$ is found*

In the third iteration, we use Dijkstra's algorithm to find the shortest path as shown in Table 7.

Node	s	1	2	4	5	6	t
$\emptyset$	0	∞	∞	∞	∞	∞	∞
s	0	∞	0	∞	∞	∞	∞
2	0	∞	0	5	∞	∞	∞
4	0	5	0	5	5	5	∞
1	0	5	0	5	5	5	∞
5	0	5	0	5	5	5	5
6	0	5	0	5	5	5	5
t	0	5	0	5	5	5	5

Table 7: *Dijkstra's algorithm in the third iteration*

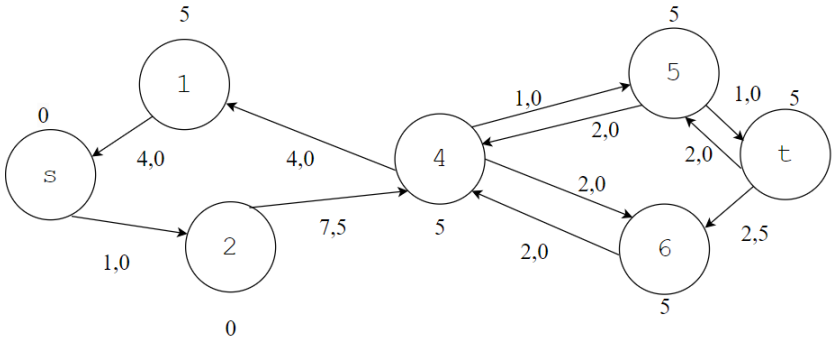

Figure 12: *New node potential is calculated*

After that, we reduce the cost in Figure 13.

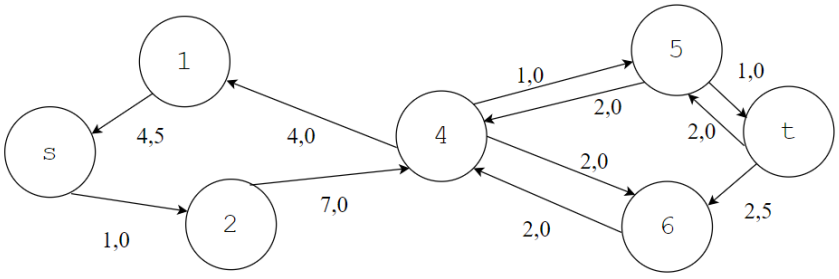

Figure 13: *Reduce cost in third iteration*

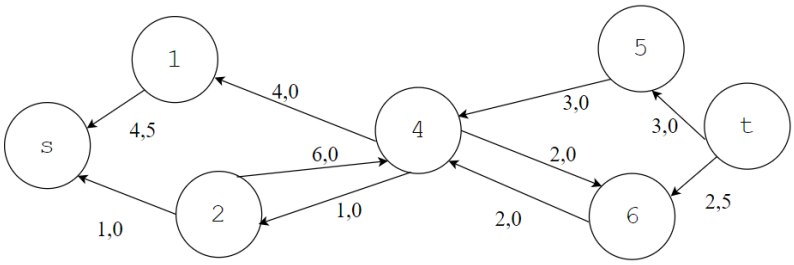

Figure 14: *The third augmenting path $s-2-4-5-t$ is found*

Since there is no augmenting path found, the program terminates with the optimal flow (which equals to 10) and the cost (which equals to -20) in Figure 15.

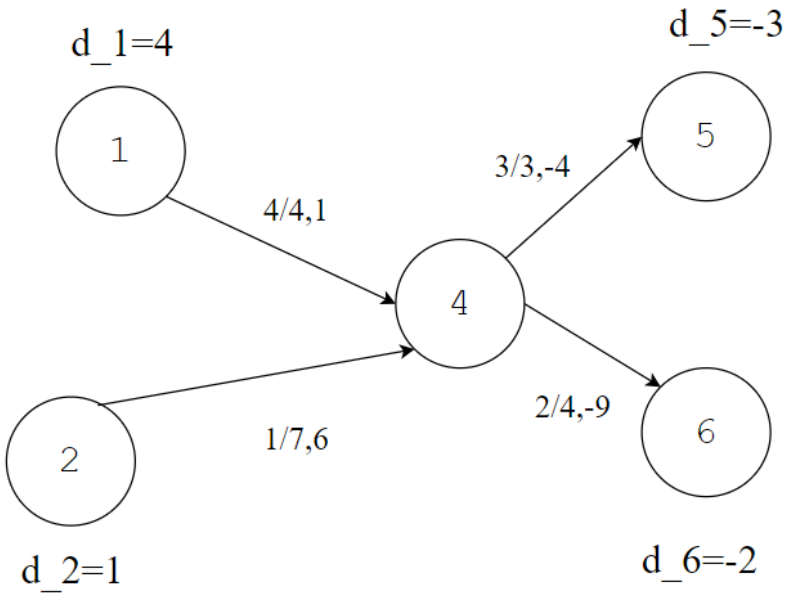

Figure 15: *Optimal result*

3.4 Simulation by GAMSPy

In this sub section, we will simulate the sub problem 1 with the given data in the previous sub section. First of all, libraries needed are similar to problem 1.

```
### In case of gampsy has not installed
### : ! pip install -q gamspy
import numpy as np
import pandas as pd
import sys
from gamspy import Container, Set, Parameter, Variable, Equation, Model, Sum, Sense,
    Alias
```

Then, we define the given data by using the data frame in Python:

```
cost=pd.DataFrame([["1","4",1],["4","5",-4],["2","4",6],["4","6",-9]],columns=["Source","
    Destination","Cost"]).set_index(["Source","Destination"])
capacity=pd.DataFrame([["1","4",4],["4","5",3],["2","4",7],["4","6",4]],columns=["Source"
    ,"Destination","Cost"]).set_index(["Source","Destination"])
### bien_d is the variable d in the balance constraint
bien_d=pd.DataFrame([["1",4],["2",1],["4",0],["5",-3],["6",-2]],columns=["Name","Value of
     d"]).set_index("Name")
```

Here, we let the string "1", "2", "3", "4" be the name of the node considered.

```
m=Container()
node_name=Set(m,name="Node_i",records=bien_d.index)
### We create set j that is similar to set node_name in order to represent the directed edge
j=Alias(m,name="Node_j",alias_with=node_name)
```

We define the set `j` to demonstrate the flow of the directed edge. For example, `node_name` equals to "1", `j` equals to "2" and the edge that goes from "1" to "2" is represented as [`node_name`,`j`]. Then we put the given data in `Parameter` in `GAMSPy` as below:

```
u=Parameter(m,name="u",domain=[j,node_name],description="Full capacity",records=capacity.reset_index())
g=Parameter(m,name="g",domain=[node_name,j],description="Generalized cost",records=cost.reset_index())
d=Parameter(m,name="d",domain=[node_name],description="Capac of the node",records=bien_d.reset_index())
```

Then, we define the variables and constraints as shown in sub problem 1:

```
x=Variable(m,name="x",domain=[node_name,j],type="integer",description="Variable needed")

eq=Equation(m,name="Balance",domain=[node_name],description="Balance equ")
eq1=Equation(m,name="Constrain",domain=[node_name,j],description="Constr equ")
eq2=Equation(m,name="Constrain1",domain=[node_name,j],description="Constr equ1")

eq[node_name]=Sum(j,x[node_name,j]-x[j,node_name])==d[node_name]
eq1[node_name,j]=x[node_name,j]<=u[node_name,j]
eq2[node_name,j]=x[node_name,j]>=0
```

Then we define our model, solve the model and print out the result:

```
pro=Model(m,name="Sub_1",equations=[eq,eq1,eq2],problem="MIP",sense=Sense.MIN,objective=Sum((node_name,j),g[node_name,j]*x[node_name,j]))
### Then we print the result to the console
pro.solve(output=sys.stdout)

print(x.records)
print(pro.objective_value)
```

In Figure 16, the directed edge is represented as [`Node_i`,`Node_j`] (from node i to node j) and the value of "level" columns is the value of x when the model is optimized.

```
print(x.records)
print(pro.objective_value)
```

	Node_i	Node_j	level	marginal	lower	upper	scale
0	1	1	0.0	-0.0	0.0	inf	1.0
1	1	2	0.0	-0.0	0.0	inf	1.0
2	1	4	4.0	1.0	0.0	inf	1.0
3	1	5	0.0	-0.0	0.0	inf	1.0
4	1	6	0.0	-0.0	0.0	inf	1.0
5	2	1	0.0	-0.0	0.0	inf	1.0
6	2	2	0.0	-0.0	0.0	inf	1.0
7	2	4	1.0	6.0	0.0	inf	1.0
8	2	5	0.0	-0.0	0.0	inf	1.0
9	2	6	0.0	-0.0	0.0	inf	1.0
10	4	1	0.0	-0.0	0.0	inf	1.0
11	4	2	0.0	-0.0	0.0	inf	1.0
12	4	4	0.0	-0.0	0.0	inf	1.0
13	4	5	3.0	-4.0	0.0	inf	1.0
14	4	6	2.0	-9.0	0.0	inf	1.0
15	5	1	0.0	-0.0	0.0	inf	1.0
16	5	2	0.0	-0.0	0.0	inf	1.0
17	5	4	0.0	-0.0	0.0	inf	1.0
18	5	5	0.0	-0.0	0.0	inf	1.0
19	5	6	0.0	-0.0	0.0	inf	1.0
20	6	1	0.0	-0.0	0.0	inf	1.0
21	6	2	0.0	-0.0	0.0	inf	1.0
22	6	4	0.0	-0.0	0.0	inf	1.0
23	6	5	0.0	-0.0	0.0	inf	1.0
24	6	6	0.0	-0.0	0.0	inf	1.0

```
-20.0
```

Figure 16: *The result of problem 2*

References

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows: Theory, algorithms, and applications*. Prentice-Hall, 1993.

[2] Alexander Shapiro, Darinka Dentcheva, and Andrzej Ruszczynski. *Lectures on stochastic programming: modeling and theory*. SIAM, 2021.

[3] Li Wang. "A two-stage stochastic programming framework for evacuation planning in disaster responses". In: *Computers & Industrial Engineering* 145 (2020), p. 106458.