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

MATHEMATICAL MODELING (CO2011)

Assignment

STOCHASTIC PROGRAMMING AND APPLICATIONS

Advisors:	Nguyễn Văn Minh Mẫn, Mahidol University
	Nguyễn An Khương, CSE - HCMUT
	Mai Xuân Toàn, CSE - HCMUT
	Trần Hồng Tài, CSE - HCMUT
	Nguyễn Tiến Thịnh, CSE - HCMUT
Students:	Trần Đăng Bảo - 2210270 *(Group L01 - Team TTBC)*
	Huỳnh Thái Minh Châu - 2210352 *(Group L01 - Team TTBC)*
	Nguyễn Minh Toàn - 2213533 *(Group L01 - Team TTBC*
	Nguyễn Hoàng Tú - 2213846 *(Group L01 - Team TTBC, **Leader**)*
	Trần Mạnh Tuấn - 2213807 *(Group L01 - Team TTBC)*

HO CHI MINH CITY, DECEMBER 2023

Contents

1 Member List & Workload

No.	Fullname	Student ID	Problems	Percentage
1	Trần Đăng Bảo	2210270	Stochastic Programming Problem 1 : description, analyzing the problem and example case.	21%
2	Huỳnh Thái Minh Châu	2210352	Problem 2 : problem, subproblem 1,2 description, example case of algorithm 1	21%
2	Nguyễn Minh Toàn	2213533	Problem 2 : algorithm 1 example case and remarks	16%
2	Nguyễn Hoàng Tú	2213846	Problem 2 : the MIP sample for the two-stage stochastic approach, the code sample for the algorithm 1 approach, algorithm 1 advantages and disadvantages	22%
2	Trần Mạnh Tuấn	2213807	Problem 1 : model building and example case	20%

2 Introduce

Readers familiar with the area of optimization can easily name several classes of optimization problems, for which advanced theoretical results exist and efficient numerical methods have been found. We can mention linear programming, quadratic programming, convex optimization, and nonlinear optimization. Stochastic programming sounds similar, but no specific formulation plays the role of the generic stochastic programming problem. The presence of random quantities in the model under consideration opens the door to a wealth of different problem settings, reflecting different aspects of the applied problem at hand. This report illustrates the main approaches that can be followed when developing a suitable stochastic optimization model. In this report, we will analyze examples of stochastic programming and the widespread application of stochastic programming in various fields. These examples are intended to help the reader build intuition on how to model uncertainty. They also reflect different structural aspects of the problems. In particular, we show the variety of stochastic programming models in terms of the objectives of the decision process, the constraints on those decisions, and their relationships to the random elements. In each example, we investigate the value of the stochastic programming model over a similar deterministic problem. We show that even simple models can lead to significant savings. These results provide the motivation to lead us into the following chapters on stochastic programs, solution properties, and techniques.

3 Stochastic Programming

3.1 Stochastic Programming - Concepts and Objectives

3.1.1 Programming? What is Stochastic Programming? Uncertainty?

Programming in mathematics is generally viewed as a collection of computational and mathematical methods used by engineers, investigators, managers, scientists ... to solve and make decision of optimization problems.
Mathematical Optimization is about decision making using mathematical methods.
Stochastic Programming (SP) is about decision making under uncertainty. View it as 'Mathematical Programming (Optimization) with random parameters'.
Stochastic linear programs (SLP) are linear programs (i.e. its objective function is linear) in which some problem data may be considered uncertain.
Recourse (Stochastic) programs are those in which some decisions or recourse (remake, modify) actions can be taken after uncertainty is disclosed.

3.1.2 Basic concepts, assumptions - Motivation

Definition 1 (Linear program (LP) with random parameters - SLP). A Stochastic linear program (SLP) is

$$\text{Minimize } Z = g(x) = f(x) = c^T.x, \quad \text{S.t. } Ax = b, \text{ and } Tx \geq h$$

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

What do we mean by solving this problem?
We can solve the problem but no waiting, i.e. we need to decide on $x = [x_1, x_2]$ before knowing the values of $\omega = [\omega_1, \omega_2]$?. We need to decide what to do about not knowing ω. We suggest (a) Guess at uncertainty, and (b) Probabilistic Constraints (see Definition 1).

(a) **Guess at uncertainty** We will guess few reasonable values for ω. Three (reasonable) suggestions – each of which tells us something about our level of 'risk'

- Unbiased: Choose mean values for each random parameters ω.
- Pessimistic: Choose worst case values for ω.
- Optimistic: Choose best case values for ω.

3.2 One-Stage Stochastic linear programming - No recourse (1-SLP)

Definition 2 (SLP with one-stage (No recourse) : 1-SLP). Consider the following program $LP(\alpha)$ that is parameterized by the random vector α:

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

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

$$and\ T\,x \geq h\ (stochastic\ constraints)$$

with assumptions that

1. matrix $T = T(\alpha)$ and (vector) $h = h(\alpha)$ express uncertainty via stochastic constraints

$$T(\alpha)x \geq h(\alpha) \Leftrightarrow \alpha_1 x_1 + ... + \alpha_n x_n \geq h(\alpha)$$

2. values (T, h) not known: they are unknown before an instance of model occurs, $h(\alpha)$ depends only on random α_j;
3. uncertainty is expressed by probability distribution of random parameters $(\alpha_j) = \alpha$ so deterministic LP is the degenerate case of Stochastic LP when α_j are constant,
   - We deal with decision problems where the vector $x = (x_1, x_2, ..., x_n) \in X$ 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 $X = x \in R_n : l \leq x \leq u$.

APPROACHES: In Stochastic Programming we utilize some assumption and facts.

Fundamental assumption: We know a (joint) probability distribution of data. Hence the first approach gives Probabilistic (Chance) Constraint LP.

The Scenario Analysis: not perfect, but useful, is the second approach. The scenario approach assumes that there are a finite number of decisions that nature can make (outcomes of randomness). Each of these possible decisions is called a scenario.

3.2.1 APPROACH 1 : use Chance constraint and Acceptable risk

- We can replace $Tx \geq h$ by probabilistic constraints $P[T\,x \geq h] \geq p^2$ for some prescribed reliability level $p \in (0.5, 1)$, (to be determined by problem owner.)
  The LP in **Definition (2)** above with random parameters $\alpha = [\alpha_1, \alpha_2, ...]$ then is called Probabilistic Constraint LP, or just 1-SLP.
- Risk then is taken care of explicitly, if define an

$$acceptable\ risk\ r_x := P[Not\,(Tx \geq h)] = P[Tx \leq h] \leq 1 - p$$

  then $(1 - p)$ is maximal acceptable risk. The chance constraint $T \leq h$ implies that the acceptable risk r_x is less than a specified maximal $1 - p \in (0, 1)$

Definition 3. Stochastic LP or 1-SLP with Probabilistic Constraints is defined by a random coefficients $\alpha = (\alpha_1, \alpha_2, ..., \alpha_n)$ in chance constraints, and a linear objective $f(x)$:

$$SP: \ \min_x Z, \quad Z = f(x) = c^T.x = \sum_{j=1}^{n} c_j x_j, \ \ c_j \in R$$

$$s.t. \begin{cases} Ax = b \quad (x = (x_1, x_2, ..., x_n) \ makes\ decision\ variables) \\ \qquad P[T\,x = \alpha.x \leq h] \leq (1 - p) \quad (0 < p < 1). \end{cases}$$

NOTE: we use parameter vector $\alpha = [\alpha_1, \alpha_2, ..]$ in general, and denote $\omega = [\omega_1, \omega_2, ..., \omega_S]$ specifically for states s called scenarios. We treat each scenario $\omega \in \omega$ possibly by a combination of many random parameters α_i at once in a SP.

3.2.2 APPROACH 2: for stochastic constraints $T(\alpha)x \leq h(\alpha)$

Use Scenario analysis of $T(\alpha)x \leq h(\alpha)$
For every scenario $(T^s; h^s), s = 1, ..., S$, solve

$$Minimize \;\; \{f(x) = c^T.x; \;\; Ax = b, \;\; T^s x \leq h^s\}$$

This kind of program targets a specific linear objective while accounting for a probability function associated with **various scenarios**. Hence, we find an overall solution by looking at the scenario solutions $x^s(s = 1, ..., S)$.
Advantage: Each scenario problem is an LP. / vs / **Disadvantage:** discrete distribution $\rightarrow$ mixed-integer LP model. (In general: possibly non-convex model).

3.3 Generic Stochastic Programming (GSP) with RECOURSE

PREVIEW of Generic Stochastic Programming (GSP) with RECOURSE
The above examples motivate the generic stochastic problem (GSP).

- We have a set of decisions taken without full information on some random events. These decisions are called first-stage decisions and are usually represented by a vector x. Later, full information is received on the realization of some random vector α.
- Then, the second-stage or corrective actions y are taken, so we represent the 2nd-stage decisions by vector y. Write 2-SP problem for two-stage stochastic program which allow infeasibilities w.r.t. random constraints $T.x \leq h, T.x + W.y = h$ or equivalently $W.y = h - T.x$
  We correct infeasibilities later, and must pay for corrections via y.
- Both α and y may use functional forms, as $\alpha(\omega)$, $y(\omega)$, or $y(s)$, to show explicit dependence on a scenario ω or an outcome s of the random experiment **e** coupled with the 2-SP modeling.

We focus on modeling and leave out details if not essential for understanding concepts. **Definition 4** (Stochastic program in two stages (generic 2-SP problem)). The two-stage stochastic program (2-SP) extended from Definition 2 has the form

$$2 - SP : \min_{x} \;\; g(x) \; with \; g(x) = f(x) + E\omega[v(x, \omega)]$$

where $x = (x_1, x_2, ..., x_n)$ is the first stage decision variables, $f(x)$ can be linear or not, a part of the grand objective function $g(x)$. * The mean $Q(x) := E\omega[v(x, \omega)]$ of a function

$$v : R^n \times R^n \rightarrow R$$

upon influences of scenarios $\omega.Q(x)$ is the optimal value of a certain second-stage problem

$$\min_{y \in R^p} \; q\,y \mid subject\ to\ T.x + W.y = h$$

Vectors $\alpha = \alpha(\omega)$ and $y = y(\omega)$ are named correction, tuning or recourse decision variables, only known after the experiment e. Briefly we Minimize total expected costs $g(x) = f(x) + Q(x)$ while satisfying

$$W.y(\omega) = h(\omega) - T(\omega).x$$

Here W is called mp recourse matrix, and we begin with simple case of $m = 1$, q is the unit recourse cost vector, having the same dimension as y, and $y = y(\omega) \in R^p$.
ELUCIDATION (On Recourse modeling issues)

- Our grand objective $g(x)$ is built up by $f(x)$ and $Q(x)$. Here y is the decision vector of a second-stage LP problem, value y depends on the realization of $(T, h) := (T(\omega), h(\omega))$. Recourse variables $y(\omega) \sim$ corrective actions e.g. use of alternative production resources (overtime...).
- Quantitative risk measure: size of deviations $h(\omega) - T(\omega).x$ is relevant.
- Here RISK is described by expected recourse costs $Q(x)$ of the decision x.

3.4 Two-stage Stochastic linear program: 2-SLP

We now treat the two-stage stochastic LP with recourse action.

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

Definition 5 (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

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

or in general

$$2 - SLP : \underset{x \in X, y(\omega) \in Y}{min}\, E_\omega[c^T.x + v(x, \omega)]$$

with $v(x, \omega) := q.y$
subject to

$$Ax = b \text{ First Stage Constraints,}$$

$$T(\omega).x + W.y(\omega) = h(\omega) \text{ Second Stage Constraints}$$

$$\text{or shortly } W.y = h(\omega) - T(\omega).x$$

This SLP program specify the above 2-SP (2) to the target - a specific random grand objective (function) g(x) having

1. the deterministic $f(x)$- being linear function, while accounting
2. for a probability function $v(x, \omega)$ associated with various scenarios ω.

$y = y(x, \omega) \in R^p_+$ is named recourse action variable for decision x and realization of ω.
Recourse actions are viewed as Penalize corrective actions in SLP.
The Penalize correction is expressed via the mean $Q(x) = E_\omega[v(x, \omega)]$.
Major Approaches- APPROACH 2: Scenarios analysis again

To solve system (4-4.1) numerically, approaches are based on a random vector α having a finite number of possible realizations, called scenarios.

Expected value $Q(x)$ obviously for a discrete distribution of ω!

So we take $\Omega = \omega_k$ be a finite set of size S (there are a finite number of scenarios $\omega_1, ..., \omega_S \in \Omega$, with respective probability masses pk).

Since $y = y(x, \omega)$ so the expectation of $v(y) = v(x, \omega) := q.y$ (one cost q for all y_k) is

$$Q(x) = E_\omega[(v(x, \omega)] = \sum_{k=1}^{S} p_k\, q\, y_k = \sum_{k=1}^{S} p_k\, v(x, \omega_k)$$

where

- p_k is the density of scenario ω_k, q is single unit penalty cost,
- and $qy_k = v(x, \omega_k)$ - the penalty cost of using y_k units in correction phase, depends on both the first-stage decision x and random scenarios ω_k.

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

We now fully characterize the system (4-4.1) in the linear case.
Definition 6 (Stochastic linear program With Recourse action (2-SLPWR)). The canonical 2-stage stochastic linear program with Recourse can be formulated as

$$2 \text{ - SLP: } \underset{x}{min}\ g(x) \text{ with}$$

$$g(x) := c^T.x + v(y)$$

$$\text{subject to: } Ax = b \text{ where } x \in X \subset R^n, x \geq 0$$

$$v(z) := \underset{y \in R^p}{min}\ q.y \text{ subject to } W.y = h(\omega) - T(\omega).x =: z$$

where $v(y) := v(x, \omega)$ is the second-stage value function, and $y = y(x, \omega) \in R^p+$ is a recourse action for decision x and realization of ω.

1. The expected recourse costs of the decision x is $Q(x) := E_\omega[v(x, \omega)]$ by Equation (5). [precisely expected costs of the recourse $y(\alpha)$, for any policy $x \in R^n$.] Hence overally we minimize total expected costs $min_{x \in R^n}, y \in R^p{}_+ c^T.x + Q(x)$.
2. We design the 2nd decision variables $y(\omega)$ so that we can (tune, modify, or) react to our original constraints (4.2) in an intelligent (or optimal) way: we call it recourse action!

$$x - - - - - - - - - - - - T, h, \omega - - - - - - - - - - - - > y$$

3. The optimal value of the 2nd-stage LP is $v = v(y)$, with $y = y(x, \omega)$ is its optimal solution, here $y \in R^p{}_+$. The total optimal value is $c^T.x^+ v(y)$.

4 Problem 1: Industry- Manufacturing

4.1 Two-Stage Model

Consider a situation where a manufacturer produces n products. There are in total m different parts (or subassemblies) which have to be ordered from third-party suppliers. A unit of product i requires aij units of part j, where $i = 1, ..., n$ and $j = 1, ..., m$. Of course, a_{ij} may be zero for some combinations of i and j. The demand for the products is modeled as a random vector $D = (D_1, ..., D_n)$. Before the demand is known, the manufacturer may preorder the parts from outside suppliers at a cost of c_j per unit of part j. After the demand D is observed, the manufacturer may decide which portion of the demand is to be satisfied, so that the available numbers of parts are not exceeded. It costs additionally li to satisfy a unit of demand for product i, and the unit selling price of this product is q_i. The parts not used are assessed salvage values $s_j < c_j$. The unsatisfied demand is lost.

Suppose the numbers of parts ordered are equal to $x_j, j = 1, ..., m$. After the demand D becomes known, we need to determine how much of each product to make. Let us denote the numbers of units produced by $z_i, i = 1, ..., n$, and the numbers of parts left in inventory by $y_j, j = 1, ..., m$. For an observed value (a realization) $d = (d_1, ..., d_n)$ of the random demand vector D, we can find the best production plan by solving the following linear programming problem:

$$\underset{z,y}{Min} \sum_{i=1}^{n}(l_i - q_i).z_i - \sum_{j=1}^{n} s_j.y_j$$

$$\text{s.t. } y_j = x_j - \sum_{i=1}^{n} a_{ij} z_i, \;\; j = 1, ..., m$$

$$0 \le z_i \le d_i, \;\; i = 1, ..., n, \;\; y_j \ge 0, \;\; j = 1, ..., m$$

Introducing the matrix A with entries a_{ij}, where $i = 1, ..., n$ and $j = 1, ..., m$, we can write this problem compactly as follows:

$$\underset{z,y}{Min}(l-q)^T z - s^T y$$

$$\text{s.t. } y = x - A^T z, \qquad (1)$$

$$0 \le z \le d, \;\; y \ge 0.$$

Observe that the solution of this problem, that is, the vectors z and y, depend on realization d of the demand vector D as well as on x. Let $Q(x, d)$ denote the optimal value of problem (1). The quantities x_j of parts to be ordered can be determined from the optimization problem

$$\underset{x \ge 0}{Min} \; c^T x + E[Q(x, D)], \qquad (2)$$

where the expectation is taken with respect to the probability distribution of the random demand vector D. The first part of the objective function represents the ordering cost, while the second part represents the expected cost of the optimal production plan, given ordered quantities x. Clearly, for realistic data with $q_i > l_i$, the second part will be negative, so that some profit will be expected.

Problem (1)˘(2) is an example of a two-stage stochastic programming problem, where (1) is called the second-stage problem and (2) is called the first-stage problem. As the second-stage problem contains random data (random demand D), its optimal value $Q(x, D)$ is a random variable. The distribution of this random variable depends on the first-stage decisions x, and therefore the first-stage problem cannot be solved without understanding of the properties of the second-stage problem.

In the special case of finitely many demand scenarios $d_1, ..., d_K$ occurring with positive probabilities $p_1, ..., p_K$, with $\sum_{k=1}^{K} p_k = 1$, the two-stage problem (1)˘(2) can be written as one large-scale linear programming problem:

$$Min \; c^T + \sum_{k=1}^{K} p_k[(l-q)^T z^k - s^T y^k]$$

$$\text{s.t. } y^k = x - A^T z^k, \;\; k = 1, ...K, \qquad (3)$$

$$0 \le z_k \le d_k, \;\; y_k \ge 0, \;\; k = 1, ..., K, \;\; x \ge 0$$

where the minimization is performed over vector variables x and z_k, y_k, $k = 1, ..., K$. We have integrated the second-stage problem (1) into this formulation, but we had to allow for its solution (z_k, y_k) to depend on the scenario k, because the demand realization d_k is different in each scenario. Because of that, problem (3) has the numbers of variables and constraints roughly proportional to the number of scenarios K.

It is worth noticing the following. There are three types of decision variables here: the numbers of ordered parts (vector x), the numbers of produced units (vector z), and the numbers of parts left in the inventory (vector y). These decision variables are naturally classified as the first- and the second-stage decision variables. That is, the first-stage decisions x should be made before a realization of the random data becomes available and hence should be independent of the random data, while the second-stage decision variables z and y are made after observing the random data and are functions of the data. The first-stage decision variables are often referred to as here-and-now decisions (solution), and second-stage decisions are referred to as wait-and-see decisions (solution). It can also be noticed that the second-stage problem (1) is feasible for every possible realization of the random data; for example, take $z = 0$ and $y = x$. In such a situation we say that the problem has relatively complete recourse.

4.2 Chance Constrained Model

Suppose now that the manufacturer is concerned with the possibility of losing demand. The manufacturer would like the probability that all demand be satisfied to be larger than some fixed service level $1 - \alpha$, where $\alpha \in (0, 1)$ is small. In this case the problem changes in a significant way.

Observe that if we want to satisfy demand $D = (D_1, ..., D_n)$, we need to have $x \geq A^T D$. If we have the parts needed, there is no need for the production planning stage, as in problem (1). We simply produce $z_i = D_i,\ i = 1, ..., n$, whenever it is feasible. Also, the production costs and salvage values do not affect our problem. Consequently, the requirement of satisfying the demand with probability at least $1 - \alpha$ leads to the following formulation of the corresponding problem:

$$\underset{x \geq 0}{Min}\ c^T x$$

$$\text{s.t. } Pr\{A^T D \leq x\} \geq 1 - \alpha \quad (4)$$

The chance (also called probabilistic) constraint in the above model is more difficult than in the case of the news vendor model considered, because it involves a random vector $W = A^T D$ rather than a univariate random variable.

Owing to the separable nature of the chance constraint in (4), we can rewrite this constraint as

$$H_W(x) \geq 1 - \alpha \qquad (5)$$

where $H_W(x) := Pr(W \leq x)$ is the cumulative distribution function of the n-dimensional random vector $W = A^T D$. Observe that if $n = 1$ and $c > 0$, then an optimal solution $\overline{x}$ of (5) is given by the left-side $(1 - \alpha)$-quantile of W, that is, $\overline{x} = H_W^{-1}(1 - \alpha)$. On the other hand, in the case of multidimensional vector W, its distribution has many "smallest (left-side) $(1 - \alpha)$-quantiles," and the choice of $\overline{x}$ will depend on the relative proportions of the cost coefficients c_j . It is also worth mentioning that even when the coordinates of the demand vector D are independent, the coordinates of the vector W can be dependent, and thus the chance constraint of (5) cannot be replaced by a simpler expression featuring one-dimensional marginal distributions.

The feasible set

$$\{x \in R^m_+ : Pr(A^T D \leq x) \geq 1 - \alpha\}$$

of problem (4) can be written in the following equivalent form:

$$\{x \in R^m_+ : A^T d \leq x,\ d \in \mathcal{D},\ Pr(\mathcal{D}) \geq 1 - \alpha\} \quad (6)$$

In the formulation (6), the set D can be any measurable subset of Rn such that probability of $D \in \mathcal{D}$ is at least $1-\alpha$. A considerable simplification can be achieved by choosing a fixed set $\mathcal{D}_\alpha$ in such a way that $Pr(\mathcal{D}_\alpha) \geq 1-\alpha$. In that way we obtain a simplified version of problem (4):

$$\underset{x \geq 0}{Min}\ c^T x \qquad (7)$$

$$\text{s.t. } A^T d \leq x,\ \forall d \in \mathcal{D}$$

The set $\mathcal{D}_\alpha$ in this formulation is sometimes referred to as the uncertainty set and the whole formulation as the robust optimization problem. Observe that in our case we can solve this problem in the following way. For each part type j we determine x_j to be the minimum number of units necessary to satisfy every demand $d \in \mathcal{D}_\alpha$, that is,

$$x_j = \underset{d \in \mathcal{D}_\alpha}{max} \sum_{i=1}^{n} a_{ij} d_i, \quad j = 1, ..., n.$$

In this case the solution is completely determined by the uncertainty set $\mathcal{D}_\alpha$ and it does not depend on the cost coefficients c_j .

The choice of the uncertainty set, satisfying the corresponding chance constraint, is not unique and often is governed by computational convenience. In this book we shall be mainly concerned with stochastic models, and we shall not discuss models and methods of robust optimization.

4.3 Multistage Model

Consider now the situation when the manufacturer has a planning horizon of T periods. The demand is modeled as a stochastic process D_t, $t = 1, ..., T$, where each $(D_t = D_{t_1}, ..., D_{t_n})$ is a random vector of demands for the products. The unused parts can be stored from one period to the next, and holding one unit of part j in inventory costs h_j. For simplicity, we assume that all costs and prices are the same in all periods.

It would not be reasonable to plan specific order quantities for the entire planning horizon T . Instead, one has to make orders and production decisions at successive stages, depending on the information available at the current stage. We use symbol $D_{[t]} := (D_1, ..., D_t)$ to denote the history of the demand process in periods $1, ..., t$. In every multistage decision problem it is very important to specify which of the decision variables may depend on which part of the past information.

Let us denote by $x_{t-1} = (x_{t-1}, 1, ..., x_{t-1}, n)$ the vector of quantities ordered at the beginning of stage t, before the demand vector D_t becomes known. The numbers of units produced in stage t will be denoted by z_t and the inventory level of parts at the end of stage t by y_t for $t = 1, ..., T$. We use the subscript $t-1$ for the order quantity to stress that it may depend on the past demand realizations $D_{[t-1]}$ but not on D_t, while the production and storage variables at stage t may depend on $D_{[t]}$, which includes D_t. In the special case of $T = 1$, we have the two-stage problem discussed in section 4.1; the variable x_0 corresponds to the first stage decision vector x, while z_1 and y_1 correspond to the second stage decision vectors z and y, respectively.

Suppose $T > 1$ and consider the last stage $t = T$, after the demand D_T has been observed. At this time, all inventory levels y_{T-1} of the parts, as well as the last order quantities x_{T-1}, are known. The problem at stage T is therefore identical to the second stage problem (1) of the two-stage formulation:

$$\underset{z_T, y_T}{Min}\ (l-q)^T z_T - s^T y_T$$

$$\text{s.t. } y_T = y_{T-1} + x_{T-1} - A^T z^T, \qquad (8)$$

$$0 \leq z_T \leq d_T, \;\; y_T \geq 0,$$

where d_T is the observed realization of D_T . Denote by $QT(x_{T-1}, y_{T-1}, d_T)$ the optimal value of (8). This optimal value depends on the latest inventory levels, order quantities, and the present demand. At stage $T-1$ we know realization $d_{[T-1]}$ of $D_{[T-1]}$, and thus we are concerned with the conditional expectation of the last stage cost, that is, the function

$$\mathcal{Q}_T(x_{T-1}, y_{T-1}, d_{[T-1]}) := E\{Q_T(x_{T-1}, y_{T-1}, D_T)|D_{[T-1]} = d_{[T-1]}.$$

At stage $T-1$ we solve the problem

$$\underset{z_{T-1},y_{T-1},x_{T-1}}{Min} \;\; (l-q)^T z_{T-1} + h^T y_{T-1} + c^T x_{T-1} + \mathcal{Q}_T(x_{T-1}, y_{T-1}, d_{[T-1]})$$

$$\text{s.t. } y_{T-1} = y_{T-2} + x_{T-2} - A^T z_{T-1}, \qquad (9)$$
$$0 \leq z_{T-1} \leq d_{T-1}, y_{T-1} \geq 0$$

Its optimal value is denoted by $Q_{T-1}(x_{T-2}, y_{T-2}, d_{[T-1]})$. Generally, the problem at stage $t = T-1, ..., 1$ has the form

$$\underset{z_t,y_t,x_t}{Min} \;\; (l-q)^T z_t + h^T y_t + c^T x_t + \mathcal{Q}_{t+1}(x_t, y_t, d_{[t]})$$

$$\text{s.t. } y_t = y_{t-1} + x_{t-1} - A^T z_t, \qquad (10)$$
$$0 \leq z_t \leq d_t, y_t \geq 0,$$

with

$$\mathcal{Q}_{t+1}(x_t, y_t, d_{[t]}) := E\{Q_{t+1}(x_t, y_t, D_{[t+1]})|D_{[t]} = d_{[t]}$$

The optimal value of problem (10) is denoted by $Q_t(x_{t-1}, y_{t-1}, d_{[t]})$, and the backward recursion continues. At stage $t = 1$, the symbol y_0 represents the initial inventory levels of the parts, and the optimal value function $Q_1(x_0, d_1)$ depends only on the initial order x_0 and realization d_1 of the first demand D_1. The initial problem is to determine the first order quantities x_0. It can be written as

$$\underset{x_0 \geq 0}{Min} c^T x_0 + E[\mathcal{Q}_1(x_0, D_1)]. \qquad (11)$$

Although the first-stage problem (11) looks similar to the first-stage problem (2) of the two-stage formulation, it is essentially different since the function $Q_1(x_0, d_1)$ is not given in a computationally accessible form but in itself is a result of recursive optimization.

4.4 Run Code

In the code below, we have built a simple yet powerful application that addresses a specific problem using the Python programming language. This code focuses on solving the first issue of reducing costs and optimizing profits when pre-ordering raw materials for manufacturing products from factories, with innovation and efficiency being top priorities. We utilize the NUMPY library and the GAMSPY library to address the problem, as these two libraries help us optimize the source code. This code is not only a simulation prototype but also a clear guide on how to solve the problem, with detailed comments to help the reader understand each step of the implementation. In this code, a comprehensive mathematical model for production and inventory management is crafted and solved using the "gamspy" library in Python. The model takes into account various decision-making factors, such as preorder costs, production costs, selling prices, and post-demand preorder costs, while considering different scenarios with randomly generated demand values. By formulating and solving a linear programming problem, the code optimizes decision variables, such as the number of parts to be ordered before production, the number of units produced, and the remaining parts after production. The resulting insights provide a systematic and data-driven approach to making informed decisions in the face of uncertain demand scenarios, demonstrating the flexibility and utility of the "gamspy" library for solving real-world optimization challenges in production and inventory settings.

4.4.1 Code Explanation

Here is the complete Python program that our team developed to solve the first issue presented in the problem.

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

#from helper import printOptimal, printScenerioResult

m=Container()
i=Set(container=m,name="i",records=["i1","i2","i3","i4","i5","i6","i7","i8"])
j=Set(container=m,name="j",records=["j1","j2","j3","j4","j5"])

#Preorder cost per unit before demand
b_input=input("b=")
b_list=b_input.split()
b=Parameter(container=m,name="b",domain=j,records=np.array(b_list,dtype=int))

#Cost
l_input=input("l=")
l_list=l_input.split()
l=Parameter(container=m,name="l",domain=i,records=np.array(l_list,dtype=int))

#Price
q_input=input("q=")
q_list=q_input.split()
q=Parameter(container=m,name="q",domain=i,records=np.array(q_list,dtype=int))

#Preoder cost per unit after demand
s_input=input("s=")
s_list=s_input.split()
s=Parameter(container=m,name="s",domain=j,records=np.array(s_list,dtype=int))

#Requirement
A_matrix=[]
for k in range (8):
    row_values=input(f"A{k+1}j=")
    A_matrix.append(row_values.split())
A=Parameter(container=m,name="A",domain=[i,j],records=np.array(A_matrix,dtype=int))

#First demand
d1=Parameter(container=m,name='d1',domain=[i],records=np.random.binomial(10,0.5,8))

#Second demand
```

```
d2=Parameter(container=m,name='d2',domain=[i],records=np.random.binomial(10,0.5,8))

#Variables
#The number of parts to be ordered before production
x=Variable(container=m,name="x",domain=[j],type="positive")
#The number of units produced
z1=Variable(container=m,name="z1",domain=[i],type="positive")
#Partleft
y1=Variable(container=m,name="y1",domain=[j],type="positive")
#The number of units produced
z2=Variable(container=m,name="z2",domain=[i],type="positive")
#Partleft
y2=Variable(container=m,name="y2",domain=[j],type="positive")

#Equation
Y1_eq=Equation(container=m,name="Y1_eq",domain=[j])
z1_bound=Equation(container=m,name="z1_bound",domain=[i])
Y2_eq=Equation(container=m,name="Y2_eq",domain=[j])
z2_bound=Equation(container=m,name="z2_bound",domain=[i])

#Scenario 1:
#y1=x-A^T.z1
Y1_eq[j]=y1[j]==x[j]-Sum(i,A[i,j]*z1[i])
#0<=z1<=d1
z1_bound[i]=z1[i]<=d1[i]

#Scenario 2:
#y2=x-A^T.z2
Y2_eq[j]=y2[j]==x[j]-Sum(i,A[i,j]*z2[i])
#0<=z2<=d2
z2_bound[i]=z2[i]<=d2[i]

#obj=min(b^T*x + 0.5*[(l-q)^T*z1 - s^T*y1] + 0.5*[(l-q)^T*z2 - s^T*y2])
obj=Sum(j,b[j]*x[j]) + 0.5*(Sum(i,(l[i]-q[i])*z1[i])-Sum(j,s[j]*y1[j])) +
↪  0.5*(Sum(i,(l[i]-q[i])*z2[i])-Sum(j,s[j]*y2[j]))

model1 = Model(container=m,
               name="model1",
               equations=[Y1_eq,z1_bound,Y2_eq,z2_bound],
               sense=Sense.MIN,
               objective=obj,
               problem="LP")

model1.solve()

print("> Optimal Result: ",model1.objective_value)
print("> Parts Ordered")
print(x.records)

print ("---------Scenerio 1-----------------")
print ("- First demand")
print(d1.records)
```

```python
print ("- The number of units produced")
print(z1.records)


print ("---------Scenerio 2-----------------")
print ("- Second demand")
print(d2.records)
print ("- The number of units produced")
print(z2.records)
```

Now, we will explain the significance of the code snippet above and provide some examples of the model we used to address the problem presented in the exercise.

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

import numpy as np: This line imports the NumPy library and gives it the alias '**np**'. NumPy is a powerful library for numerical operations in Python, especially for working with arrays and matrices.

from gamspy import Container, Equation, Model, Parameter, Sense, Set, Sum, Variable: This line imports specific classes or modules from a library called "**gamspy**". These classes or modules include *Container, Equation, Model, Parameter, Sense, Set, Sum, and Variable*. These are likely components used for building and solving mathematical models, possibly related to optimization or mathematical programming.

```python
m=Container()
i=Set(container=m,name="i",records=["i1","i2","i3","i4","i5","i6","i7","i8"])
j=Set(container=m,name="j",records=["j1","j2","j3","j4","j5"])
```

m = Container(): This creates a container object named **m**. In mathematical modeling, a container is often a high-level structure that holds all the components of a model.

The second line creates a set named **i** inside the container **m**. The set represents different types of products or items, and it includes eight records: "i1" through "i8."

The third line creates another set named **j** inside the container **m**. This set represents different factories or manufacturing facilities, and it includes five records: "j1" through "j5."

The overall structure suggests that you are setting up a model where you have different types of products (set i) and different manufacturing facilities (set j).

```python
#Preorder cost per unit before demand
b_input=input("b=")
b_list=b_input.split()
b=Parameter(container=m,name="b",domain=j,records=np.array(b_list,dtype=int))
```

In mathematical modeling, parameters are constants or coefficients that affect the behavior of the model. In this case, 'b' represent the cost per unit of a component before demand is known, and the values are associated with different manufacturing facilities ('j').

```python
#Cost
l_input=input("l=")
l_list=l_input.split()
```

```python
l=Parameter(container=m,name="l",domain=i,records=np.array(l_list,dtype=int))

#Price
q_input=input("q=")
q_list=q_input.split()
q=Parameter(container=m,name="q",domain=i,records=np.array(q_list,dtype=int))
```

Cost Parameter (l): This line prompts the user to input values for the parameter 'l,' which presumably represents the cost of producing one unit of a product.

Price Parameter (q): This line prompts the user to input values for the parameter 'q,' which presumably represents the price at which one unit of a product is sold.

```python
#Preoder cost per unit after demand
s_input=input("s=")
s_list=s_input.split()
s=Parameter(container=m,name="s",domain=j,records=np.array(s_list,dtype=int))
#Requirement
A_matrix=[]
for k in range (8):
    row_values=input(f"A{k+1}j=")
    A_matrix.append(row_values.split())
A=Parameter(container=m,name="A",domain=[i,j],records=np.array(A_matrix,dtype=int))
```

Preorder Cost Parameter (s): This line prompts the user to input values for the parameter 's,' which presumably represents the cost per unit of a component after demand is known.

Requirement Parameter (A): This line creates a parameter 'A' within the container m. The parameter is associated with the sets i and j, representing a matrix of the number of components required to produce one unit of a product. The values are set based on the user input converted into a NumPy array of integers.

```python
#First demand
d1=Parameter(container=m,name='d1',domain=[i],records=np.random.binomial(10,0.5,8))

#Second demand
d2=Parameter(container=m,name='d2',domain=[i],records=np.random.binomial(10,0.5,8))
```

First Demand Parameter (d1): This line creates a parameter 'd1' within the container m. The parameter is associated with the set i (types of products), and its values are generated using a binomial distribution with parameters n=10 and p=0.5. The resulting array of random values (representing the first demand predictions from customers) is assigned to the 'records' attribute.

Second Demand Parameter (d2): This line creates another parameter 'd2' within the container m. Similar to 'd1', the values are generated using a binomial distribution with parameters n=10 and p=0.5.

In summary, you are generating random values for the first and second demand parameters (d1 and d2) based on a binomial distribution. These parameters likely represent the predicted demand for each type of product from customers in two different scenarios.

```python
#Variables
#The number of parts to be ordered before production
```

```
x=Variable(container=m,name="x",domain=[j],type="positive")
#The number of units produced
z1=Variable(container=m,name="z1",domain=[i],type="positive")
#Partleft
y1=Variable(container=m,name="y1",domain=[j],type="positive")
#The number of units produced
z2=Variable(container=m,name="z2",domain=[i],type="positive")
#Partleft
y2=Variable(container=m,name="y2",domain=[j],type="positive")

#Equation
Y1_eq=Equation(container=m,name="Y1_eq",domain=[j])
z1_bound=Equation(container=m,name="z1_bound",domain=[i])
Y2_eq=Equation(container=m,name="Y2_eq",domain=[j])
z2_bound=Equation(container=m,name="z2_bound",domain=[i])

#Scenario 1:
#y1=x-A^T.z1
Y1_eq[j]=y1[j]==x[j]-Sum(i,A[i,j]*z1[i])
#0<=z1<=d1
z1_bound[i]=z1[i]<=d1[i]

#Scenario 2:
#y2=x-A^T.z2
Y2_eq[j]=y2[j]==x[j]-Sum(i,A[i,j]*z2[i])
#0<=z2<=d2
z2_bound[i]=z2[i]<=d2[i]
```

Varriables

- x: Represents the number of parts to be ordered before production for each manufacturing facility (j).
- z1 and z2: Represent the number of units produced for each type of product (i) with demand scenarios d1 and d2.
- y1 and y2: Represent the remaining parts (inventory) for each manufacturing facility (j) after production with demand scenarios d1 and d2.

Equations and Constraints

- Y1_eq and Y2_eq: Represent equations for calculating the remaining parts (y1 and y2) after production with demand scenarios d1 and d2.
- z1_bound and z2_bound: Represent constraints on the production quantities (z1 and z2) to be within the limits of the predicted demands d1 and d2

We are setting up a mathematical model to optimize production and inventory decisions under different demand scenarios.

```
#obj=min(b^T*x + 0.5*[(l-q)^T*z1 - s^T*y1] + 0.5*[(l-q)^T*z2 - s^T*y2])
obj=Sum(j,b[j]*x[j]) + 0.5*(Sum(i,(l[i]-q[i])*z1[i])-Sum(j,s[j]*y1[j])) +
↪  0.5*(Sum(i,(l[i]-q[i])*z2[i])-Sum(j,s[j]*y2[j]))
```

```python

model1 = Model(container=m,
               name="model1",
               equations=[Y1_eq,z1_bound,Y2_eq,z2_bound],
               sense=Sense.MIN,
               objective=obj,
               problem="LP")

model1.solve()
```

Objective Function (obj): This objective function is a linear combination of terms, including the sum of b[j]*x[j] over all manufacturing facilities (j), and two terms involving the differences between (l-q) and s multiplied by the variables z1, y1, z2, and y2. The objective function represents a linear programming (LP) problem.

Model Definition (model1)

- **Model** is used to define a mathematical optimization model.
- **container=m:** The model is associated with the container m.
- **name="model1":** The model is given the name "model1."
- **equations=[Y1_eq, z1_bound, Y2_eq, z2_bound]:** The model includes the previously defined equations and constraints.
- **sense=Sense.MIN:** The optimization objective is to minimize the objective function.
- **objective=obj:** The objective function is specified.

Solve the Model (model1.solve()):

- The solve() method is called on the model to attempt to find the optimal solution.

It's important to note that the success of the optimization depends on the nature of the problem, the correctness of the model, and the solver's capabilities.

```python
print("> Optimal Result: ",model1.objective_value)
print("> Parts Ordered")
print(x.records)

print ("---------Scenerio 1-----------------")
print ("- First demand")
print(d1.records)
print ("- The number of units produced")
print(z1.records)


print ("---------Scenerio 2-----------------")
print ("- Second demand")
print(d2.records)
print ("- The number of units produced")
print(z2.records)
```

These print statements provide a summary of the key results from your optimization model, including the optimal objective value, the number of parts ordered, and the number of units produced under different demand scenarios.

In this code snippet, a mathematical model for production and inventory management is formulated and solved using the "gamspy" library. The model considers different scenarios with randomly generated demand values (d1 and d2) and optimizes the decision variables (x, z1, y1, z2, y2) based on various parameters such as preorder costs (b), production costs (l), selling prices (q), and post-demand preorder costs (s).

The objective function aims to minimize the total cost, taking into account the costs associated with ordering parts before production, producing units, and managing leftover parts after production in response to different demand scenarios. Constraints are defined for each scenario, ensuring that production quantities and leftover parts are within the bounds of the predicted demands.

The code successfully formulates and solves the linear programming problem, providing the optimal objective value and the values of the decision variables at the optimal solution. The results are printed for analysis, including the parts ordered (x), the demand and production quantities for each scenario (d1, z1, d2, z2).

In conclusion, this code demonstrates an optimization approach for production and inventory decisions under uncertainty, offering a structured and mathematical way to make informed decisions based on different demand scenarios. It highlights the versatility of the "gamspy" library in modeling and solving linear programming problems for real-world applications.

4.4.2 Some Example

We consider these cases to cover the problem.
+ Case1: If the firm is a thriving business:

```
Input:
    b = 1 2 3 4 5
    l = 10 20 30 40 50 60 70 80
    q = 100 200 300 400 500 600 700 800
    s = 0 1 2 3 4
    A1j = 4 5 4 6 3
    A2j = 6 3 5 6 3
    A3j = 5 7 8 3 5
    A4j = 3 5 7 8 2
    A5j = 6 7 8 3 4
    A6j = 5 7 3 6 8
    A7j = 3 4 7 2 6
    A8j = 7 4 3 7 3
  Output:
    > Optimal Result:  -13122.5
    > Parts Ordered
        j  level  marginal  lower  upper  scale
    0  j1  198.0       1.0    0.0    inf    1.0
    1  j2  216.0       2.0    0.0    inf    1.0
    2  j3  243.0       3.0    0.0    inf    1.0
    3  j4  210.0       4.0    0.0    inf    1.0
    4  j5  157.0       5.0    0.0    inf    1.0
    ---------Scenerio 1-----------------
    - First demand
        i  value
    0  i1    6.0
    1  i2    3.0
    2  i3    4.0
    3  i4    7.0
    4  i5    8.0
    5  i6    2.0
    6  i7    5.0
    7  i8    6.0
    - The number of units produced
        i  level  marginal  lower  upper  scale
    0  i1    6.0     -45.0    0.0    inf    1.0
    1  i2    3.0     -90.0    0.0    inf    1.0
    2  i3    4.0    -135.0    0.0    inf    1.0
    3  i4    7.0    -180.0    0.0    inf    1.0
    4  i5    8.0    -225.0    0.0    inf    1.0
    5  i6    2.0    -270.0    0.0    inf    1.0
    6  i7    5.0    -315.0    0.0    inf    1.0
    7  i8    6.0    -360.0    0.0    inf    1.0
    ---------Scenerio 2-----------------
    - Second demand
        i  value
    0  i1    2.0
    1  i2    6.0
    2  i3    3.0
    3  i4    5.0
    4  i5    5.0
    5  i6    4.0
    6  i7    4.0
    7  i8    6.0
    - The number of units produced
        i  level  marginal  lower  upper  scale
    0  i1    2.0     -45.0    0.0    inf    1.0
    1  i2    6.0     -90.0    0.0    inf    1.0
    2  i3    3.0    -135.0    0.0    inf    1.0
    3  i4    5.0    -180.0    0.0    inf    1.0
    4  i5    5.0    -225.0    0.0    inf    1.0
    5  i6    4.0    -270.0    0.0    inf    1.0
    6  i7    4.0    -315.0    0.0    inf    1.0
    7  i8    6.0    -360.0    0.0    inf    1.0
```

Esily we see that our Z is exactly same as demand vector D. This happend when q is much bigger than l, meaning that the selling price is higher than the cost making it. So to maximize the profit we should making at much at we can. That why Z is equally to D.

+ Case2

```
Input:
 b=9 8 11 6 7
 l=23 23 17 29 14 16 28 17
 q=270 235 340 200 110 293 174 230
 s=8 4 9 3 4
```

```
A1j=1 9 6 4 3
A2j=2 3 9 0 5
A3j=10 6 10 8 1
A4j=6 1 2 7 4
A5j=3 3 1 1 6
A6j=9 2 7 7 3
A7j=1 5 2 9 3
A8j=3 5 7 6 5
Output:
  > Optimal Result:  -914.7843137254902
  > Parts Ordered
      j       level  marginal  lower  upper  scale
  0  j1  170.000000       0.0    0.0    inf    1.0
  1  j2  117.000000       0.0    0.0    inf    1.0
  2  j3  195.039216       0.0    0.0    inf    1.0
  3  j4  159.000000       0.0    0.0    inf    1.0
  4  j5   98.000000       0.0    0.0    inf    1.0
  ---------Scenerio 1-----------------
  - First demand
      i  value
  0  i1    5.0
  1  i2    6.0
  2  i3    6.0
  3  i4    8.0
  4  i5    4.0
  5  i6    5.0
  6  i7    5.0
  7  i8    3.0
  - The number of units produced
      i  level      marginal  lower  upper  scale
  0  i1    5.0  0.000000e+00    0.0    inf    1.0
  1  i2    6.0  0.000000e+00    0.0    inf    1.0
  2  i3    6.0  0.000000e+00    0.0    inf    1.0
  3  i4    8.0  0.000000e+00    0.0    inf    1.0
  4  i5    0.0  1.400000e+01    0.0    inf    1.0
  5  i6    5.0  0.000000e+00    0.0    inf    1.0
  6  i7    0.0  1.776357e-15    0.0    inf    1.0
  7  i8    0.0  5.980392e-01    0.0    inf    1.0
  ---------Scenerio 2-----------------
  - Second demand
      i  value
  0  i1    4.0
  1  i2    6.0
  2  i3    6.0
  3  i4    6.0
  4  i5    4.0
  5  i6    6.0
  6  i7    3.0
  7  i8    3.0
  - The number of units produced
      i     level  marginal  lower  upper  scale
  0  i1  4.000000  0.000000    0.0    inf    1.0
```

```
   1  i2  6.000000  0.000000    0.0    inf    1.0
   2  i3  6.000000  0.000000    0.0    inf    1.0
   3  i4  6.000000  0.000000    0.0    inf    1.0
   4  i5  0.764706  0.000000    0.0    inf    1.0
   5  i6  6.000000  0.000000    0.0    inf    1.0
   6  i7  1.137255  0.000000    0.0    inf    1.0
   7  i8  0.000000  1.401961    0.0    inf    1.0
```

Contradict with above example. This example bring to us a more realistic scenario, when the profit is not to much like case 1 and in some case we loss. In such case we have to calulate carefully so that have the biggest profit. That we can't have the same value between Z and D.

5 Problem 2: the SLP-EPDR: Algorithmic Solutions

Extreme natural disasters (earthquakes, storms, fire, hurricanes, etc.) and unnatural ones (terrorist attacks, political issues, war, etc.) may strike a community with little warning and leave much damage and many casualties. The main goal of emergency response is to provide shelter and assistance to affected people as soon as possible.

The problem posed to solve the problem will be modeled by a travel network represented by nodes and paths by links. The objective is to move the people from dangerous areas to safe areas with the minimum evacuation time in a capacity-cost network $G(V, A, C, U, D)$, 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.

Since the evacuation process is divided into two stages: the first stage, it is assumed that the affected person cannot obtain information about the extent of the disaster and the path of the extent of the damage; after the time threshold T, they can get accurate road network information through several real-time monitoring devices. Therefore, solving the problem in stages: one is to predict the plan before a disaster occurs, the other is to adapt the plan when knowing the disaster information to random situations occurring at the specified time.

For ease of access, the variables in the problem are defined in Table 1

Table 1

Symbol	Definition
V	the set of nodes
A	the set of links
i, j	the index of nodes, $i, j \in A$
(i, j)	the index of directed links, $(i, j) \in A$
s	the index of scenario
S	the total number of scenarios
v	the supply value of source node
$\widetilde{T}$	the time threshold
T	the total number of time intervals
u_{ij}	the capacity on (i, j) physical link
$u^s_{ij}(t)$	the capacity of link (i, j) in scenario s at time t
$c^s_{i,j}(t)$	the travel time of link (i, j) in scenario s at time t
μs	the probability in scenario s

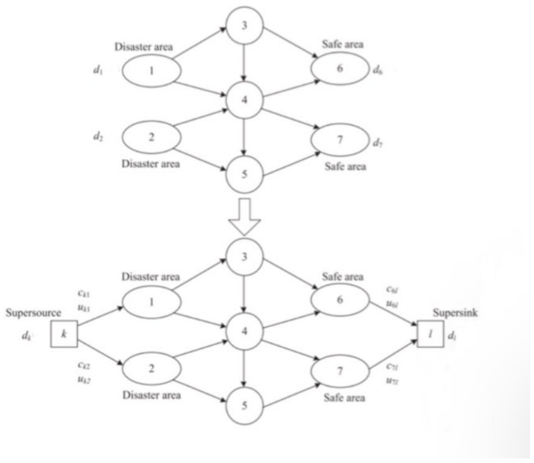
Two types of decision variables are introduced to construct the twostage stochastic evacuation model. In the first stage, a binary decision variable xij is used to represent the flow on link (i, j). The second decision variable $y^s_{ij}(t)$ is defined to represent the flow on link (i, j) in scenario s at time t. Meanwhile, Table 2 gives a straightforward overview for these two decision variables.

Table 2

Decision Variable	Definition
x_{ij}	the flow on link (i, j)
$y^s_{ij}(t)$	the flow on link (i, j) in scenario s at time t

In the first stage:

The problem is described as follows:



The purpose is to evacuate people from dangerous places to safe places. In reality, there are more than one sources and sinks in the evacuation network. Therefore, the physical network with multiple sources should be converted to an equivalent network with a single supersource, even to multiple sinks. It means a supersource k is added to the network, and meanwhile the dummy arcs (k, i) should be added. Where K is the set of source nodes, and let the capacity on the dummy arc be the value of the supply at node i,i.e., $u_{ki} = d_i$, $i \in K$. Hence the supply value at the supersource can be set as $d - k = \sum_{i \in K} d_i$. 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:

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

where d_i is a parameter with the following definition:

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

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

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

On a separate note, the flow balance constraint may generate a path with loops and subtours if there are potential loops in the evacuation network. To eliminate loops on the generated physical evacuation path, the link penalty p_{ij}, $(i, j) \in A$ is particularly introduced. With this consideration, the penalty function can be defined as
$f(\mathbf{X}) = \sum\limits_{(i,j)\in A} p_{ij}x_{ij}$
where $\mathbf{X} := \{x_{ij}\}_{(i,j)\in A}$

In the second stage:

Because the arc travel time and capacity will vary with the departure time. When adding a supersource, the travel times on the dummy arcs are $c_{ki} = d_i(t)$, $t \in \{0, 1, .., T\}$. Since the time of arriving at the sink for each flow may be different from each other, we first add a copy j for each original sink j, $j \in D$ and then a supersink l is added to the network. For each sink, we add the arc (j, j') to the network with infinite capacity, i.e., $u_{ij} = \infty$, and the travel time for each dummy arc is $c^s_{jj'}(t), \forall t \in \{0, 1, .., T\}$.

To ensure high accuracy in predicting pre-disaster planning to minimize loss of life and property when evacuating people, the problem must ensure an important time requirement.The affected people will be evacuated along the a priori evacuation plan in the first stage; in other words, the evacuation plan in different scenarios before the time threshold is the same as the a priori path. So the problem conditions are:
$\sum\limits_{t\leqslant\widetilde{T}} y^s_{ij}(t) = x_{ij}$, $(i, j) \in A$, $s = 1, 2, ..., S$

Therefore, the purpose of the second stage boils down to finding the minimum total time it takes for affected people to evacuate from the dangerous area to a safe place in each scenario s
$Q(Y, s) = \min \sum_{(i,j)\in A-s} c^s_{ij}(t).y^s_{ij}(t)$
$s.t.$
$\sum_{(i_t j_{t'})\in A_s} y^s_{ij}(t) - \sum_{(j_{t'},i_t)\in A_s} y^s_{ij}(t^{'}) = d^s_i(t)$, $\forall i \in V$, $t \in \{0, 1, ..., T\}s = 1, 2, ..., S$
$0 \leqslant y^s_{ij} \leqslant u^s_{ij}$, $\forall(i, j) \in A$, $t \in \{0, 1, ..., T\}$, $s = 1, 2, ..., S$
$\sum\limits_{t\leqslant\widetilde{T}} y^s_{ij}(t) = x_{ij}$, $(i, j) \in A$, $s = 1, 2, ..., S$

From the above two stages, we can come up with an adaptive plan model to minimize the penalty for the previous evacuation plan and the expected minimum time of the adaptive evacuation plan according to each scenario with a probability μs, $s = 1, 2, ..., S$ corresponds to each scenario as follows:

$$\begin{cases} \min \sum_{(i,j)\in A} p_{ij}x_{ij} + \sum_{s=1}^{S} \left(\mu_s. \sum_{(i,j)\in A_s} c^s_{ij}(t).y^s_{ij}(t)\right) \\ s.t. \\ \sum_{(i,j)\in A} x_{ij} - \sum_{(i,j)\in A} x_{ji} = d_i,\ \forall i \in V \\ 0 \leqslant x_{ij} \leqslant u_{ij},\ \forall(i, j) \in A \\ \sum_{(i_t j_{t'})\in A_s} y^s_{ij}(t) - \sum_{(j_{t'},i_t)\in A_s} y^s_{ij}(t^{'}) = d^s_i(t),\ \forall i \in V,\ t \in \{0, 1, ..., T\},\ s = 1, 2, ..., S \\ 0 \leqslant y^s_{ij} \leqslant u^s_{ij},\ \forall(i, j) \in A,\ t \in \{0, 1, ..., T\},\ s = 1, 2, ..., S \\ \sum\limits_{t\leqslant\widetilde{T}} y^s_{ij}(t) = x_{ij},\ (i, j) \in A,\ s = 1, 2, ..., S \quad (*) \end{cases}$$

$(*)$ is essentially an integer programming model which contains two types of decision variables, i.e., $\mathbf{X} := \{x_{ij}\}_{(i,j)\in A}$ and $\mathbf{Y} := \{y^s_{ij}\}_{(i,j)\in A\{0,1,...,T\},\, s=1,2,...,S}$ .. In this model, the coupling constraint is a complex constraint, which leads to the model cannot be solved in polynomial time.

Therefore, we use the Lagrangian multiplier $\alpha^s_{ij}(t), (i, j) \in A$, $s = 1,\ 2,\ ..., S$, $t \leqslant T$ for the coupling constraint, and then this constraint can be relaxed into the objective function in the

following form:
$\sum\limits_{s=1}^{S}\sum\limits_{(i,j)\in A}\alpha_{ij}^{s}\left(\left(\sum\limits_{t\leqslant\widetilde{T}}y_{ij}^{s}(t)\right)-x_{ij})\right)$

Therefore, after the relaxation of the formula,the model can be formulated as follows:

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

It is worthwhile to note that the variables X and Y can be separated from each other in above relaxed model

$\min\sum_{(i,j)\in A}p_{ij}x_{ij}+\sum_{s=1}^{S}\left(\mu_s.\sum_{(i,j)\in A_s}c_{ij}^{s}(t).y_{ij}^{s}(t)\right)+\sum\limits_{s=1}^{S}\sum\limits_{(i,j)\in A}\alpha_{ij}^{s}\left(\left(\sum\limits_{t\leqslant\widetilde{T}}y_{ij}^{s}(t)\right)-x_{ij})\right)$
$=\sum_{(i,j)\in A}p_{ij}x_{ij}+\sum_{s=1}^{S}\sum_{(i,j)\in A}\left(\sum_{t\in\{0,1,2,....T\}}\mu_s c_{ij}^{s}(t).y_{ij}^{s}(t)\right)+$
$\sum\limits_{s=1}^{S}\sum\limits_{(i,j)\in A}\sum\limits_{t\leqslant\widetilde{T}}\alpha_{ij}^{s}.y_{ij}^{s}(t)-\sum\limits_{s=1}^{S}\sum\limits_{(i,j)\in A}\alpha_{ij}^{s}.x_{ij}$
$=\sum_{(i,j)\in A}\left(p_{ij}-\sum\limits_{s=1}^{S}\alpha_{ij}^{s}(t)\right)x_{ij}+\sum\limits_{s=1}^{S}\sum\limits_{(i,j)\in A}\left(\sum_{t\in\{0,1,2,....T\}}\mu_s c_{ij}^{s}(t)+\sum\limits_{t\leqslant\widetilde{T}}\alpha_{ij}^{s}(t)\right)y_{ij}^{s}(t)$

By combining similar terms, the relaxed model is decomposed into two subproblems as follows:

5.1 SubProblem 1: Min-cost Flow Problem

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

$$\begin{cases}\min SP1(\alpha)=\sum_{(i,j)\in A}\left(p_{ij}-\sum\limits_{s=1}^{S}\alpha_{ij}^{s}(t)\right)x_{ij}\\ s.t.\\ \sum_{(i,j)\in A}x_{ij}-\sum_{(i,j)\in A}x_{ji}=d_i,\ \forall i\in V\\ 0\leqslant x_{ij}\leqslant u_{ij},\ \forall(i,j)\in A\end{cases}$$

The objective function of subProblem 1 can be defined as $g_{ij}:=p_{ij}-\sum\limits_{s=1}^{S}\sum\limits_{t\leqslant\widetilde{T}}\alpha_{ij}^{s}$ to represent the generalized cost of each link. Therefore, the sub-Problem 1 can be solved by the successive shortest path algorithm. For convenience, the optimal objective value of sub-Problem 1 is abbreviated as $Z_{SP1}(\alpha)^*$.

5.2 SubProblem 2: Time-Dependent Min-cost Flow Problem

The second subproblem of the relaxed model is associated with the decision variables Y , and its optimal objective value of the problem is abbreviated as $Z_{SP2}(\alpha)^*$, shown as follows:

$$\begin{cases} \min SP2(\alpha) = \sum\limits_{s=1}^{S} \sum\limits_{(i,j)\in A} \left(\sum_{t\in\{0,1,2,....T\}} \mu_s c_{ij}^s(t) + \sum\limits_{t\leqslant \widetilde{T}} \alpha_{ij}^s(t) \right) y_{ij}^s(t) \\ s.t. \\ \sum_{(i_t j_{t'})\in A_s} y_{ij}^s(t) - \sum_{(j_{t'},i_t)\in A_s} y_{ij}^s(t^{'}) = d_i^s(t),\ \forall i \in V,\ t\in\{0,1,...,T\},\ s=1,2,...,S \\ \sum\limits_{t\leqslant\widetilde{T}} y_{ij}^s(t) = x_{ij},\ (i,j)\in A,\ s=1,2,...,S \end{cases}$$

Both subproblem 1 and 2 can be solved using the successive path algorithm (algorithm 1)

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

Step 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.

Step 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:

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

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

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

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

SubProblem 2 can be further decomposed into a total of S subproblems, each of which can be referred to as the min-cost flow problem with time-dependent link travel times and capacities, namely

$$\begin{cases} minSP2(\alpha,s) = \Sigma_{(i,j)\in A}(\Sigma_{t\in\{0,1,...,T\}}\mu_s.c_{ij}^s(t) + \Sigma_{t\leqslant\tau}\alpha_{ij}^s(t))y_{ij}^s(t) \\ \text{s.t.}\Sigma_{(i_t j_{t'})\in A} y_{ij}^s(t) - \Sigma_{(j_{t'},i_t)\in A_s} y_{ij}^s(t') = d_t^s(t), \forall i\in V, t\in\{0,1,...,T\} \\ 0 \le y_{ij}^s(t) \le u_{ij}^s(t), \forall (i,j)\in A, t\in\{0,1,...,T\} \end{cases} \quad (16)$$

For each scenario $s \in 1,2,...,S$, subproblem (16) has a similar structure as subproblem (14) with time-dependent link cost $c_{ij}^s(t)$ and link capacity $u_{ij}^s(t)$ and the generalized cost $g_{ij}^s(t)$. Since the considered time period T is divided into two time stages, the generalized cost is defined as a piecewise function:

$$g_{ij}^s(t) = \begin{cases} \mu_s.c_y^s(t) + \alpha_i^s j(t), t \le \tau \\ \mu_s.c_y^s(t), \quad \tau \le t \le T \end{cases}$$

Since subproblem (16) is a time-dependent min-cost flow problem, and thus the algorithm 1 should be modified in Step 2. Firstly, parameters $A(y(t)), C(y(t))$, and $U(y(t))$ in the residual network $N(y(t))$ are defined as follows:

$$A_s(y(t)) = \{(i_t, j_{t'})|(i_t, j_{t'})\in A_s, y_i^s j < u_i^s j\} \cup \{(j_{t'}, i_t)|(j_{t'}, i_t)\in A_s, y_{ij}^s > 0\}, s=1,2,...,S$$

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

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

Secondly, the modified label-correcting algorithm (Ziliaskopoulos & Mahmassani, 1992) will be adopted to find the time-dependent mincost path in the residual network.

By solving the subproblem (14) and (15) with the relaxation solution X and Y, the optimal objective value ZLR for the relaxed model (13) with a set of given Lagrangian multiplier vector can be expressed as follows:

$$Z_{LR}^*(\alpha) = Z_{SP1}^*(\alpha) + Z_{SP2}^*(\alpha) \qquad (17)$$

Obviously, the optimal objective value of the relaxed model (13) is the lower bound of the optimal objective value of the original model (10). In order to obtain a high-quality solution, it is needed to obtain a lower bound which is close to the optimal objective value of the original model. That is, the greatest possible lower bound should be obtained, and the expression is given as follows:

$$Z_{LD}(\alpha^*) = max_{\alpha \geq 0} Z_L R(\alpha) \qquad (18)$$

Determination of potential optimal value

If the relaxed solutions obtained by solving the subproblems (14) and (15) are feasible to the original model (13), then they can be regarded as potential optimal values of the original model, each of which is also an upper bound to the optimal value of the original model. In the process of solving subproblem (14), $SP1(\alpha)$, an evacuate plan consisted of physical paths for each car is generated, denoted by UB. For solving the subproblem (16), $SP2(\alpha, S)s = 1, 2, ..., S$, a total of S evacuate plans consisted of time-dependent paths for each car are generated, denoted by UB, $s = 1, 2, ..., S$. According to the coupling constraint (4), the generated evacuated plans by the subproblems (14) and (16) are all feasible solutions to the original model (13). Therefore, the objective value for each feasible solution obtained by subproblems (14)

Solution procedure

If the lower bound (17) is just happened to be equal to the minimum upper bound obtained by subproblems (14) and (16), the returned feasible solution will be the exact optimal solution on the basis of the duality (Fisher, 1981). If not, we shall try to update the Lagrangian multipliers to reduce the gap between the lower and upper bounds of the optimal value to improve the quality of current feasible solution.This research applies the subgradient optimization algorithm to iteration process. The subgradient optimization algorithm is a general approach to reduce the relative gap between the upper and lower bounds by iteration process, and the basic principle and detailed course of treatment of this algorithm can be referred to papers such as Yang and Zhou (2014, 2017) and Wang, Yang, and Gao (2016); Wang, Yang, Gao, Li, et al. (2016). Therefore, the main procedure for solving the proposed model of this paper is briefly given in the following.

Firstly, the Lagrangian relaxation solution approach is used to solve the classic minimum cost flow problem and time-dependent minimum cost flow problem presented in above Section 4.1. Secondly, the optimal value of the Lagrangian dual problem provides a lower bound for the original problem. In the process of finding the optimal solution for the Lagrangian dual problem, it is necessary to calculate the minimum cost flow plan and time-dependent minimum cost plan in each scenario s by the successive shortest path algorithm. Thirdly, each calculated minimum cost evacuation plans of the Lagrangian dual problem is a feasible solution to the original problem. Hence, the objective value of each minimum cost evacuation plan for the original problem can be regarded as an upper bound for this original problem. Following is the detailed procedure of the Lagrangian relaxation approach based on subgradient optimization algorithm.

5.4 Successive path algorithm

The successive path algorithm is the algorithm use to find the min-cost path with the required flow between an OD pair

The successive path algorithm we use in this report is based on the algorithm 1 explained before, with some changes:

Step 1: Using the starting graph, create a residual network which will be used in finding the successive path. Set the current flow to 0.

Step 2: Using the Dijkistra's shortest path algorithm, finding the min-cost path correspond to the OD pair in the residual network. If no path is available, the algorithm terminate with no successive path available.

Step 3: With the min-cost path found, find the minimum capacity of all links within the path. The minimum capacity found is the maximum flow that can follow that path.

Step 4: The real flow we will use is the smaller value between the maximum flow and the difference between the required flow and the current flow. Increase the current flow by the real flow, as well as changing the residual network in change of the flow in the min-cost path.

Step 5: If the current flow is equal to the required flow, the residual network now represent the successive shortest path available. Else go back to **step 2**.

Advantage of successive path algorithm:

First, the time complexity is considerable. The Dijkistra's algorithm has a time complexity of $O(n^2 + V)$, with n is the number of nodes and V is the number of links. And assume that each path found has the maximum flow of only 1 (smallest possible), than we must found v paths, with v is the required flow, leads to the complexity of $O(v(n2 + V))$. Which is clearly less time-consuming than using the *MIP model*, which cannot even be solved in a polynominal time. Thus, with a big enough network, the number of n and V tends to be large, and with the time-dependent network, which we have to increase n and V by approximately T time, which make the difference in time-complexity become even more important.

Second, the successive shortest path algorithm presented here works by finding min-cost path, even if there is no path available (the algorithm terminate in **step 2**), by the time it reach the conclusion the current flow is exactly the maximum flow that can be transferred with the current network, which is still information we need to have another plan reserved. Compare to the *MIP model* which using all of linear functions to solve, if there is no path available than the model just terminate with nothing gained.

Disavantage of successive path algorithm:

It lies within the variety of case that the algorithm is available. Compare to the *MIP model*, which can solve the problem nearly in any circumstances, the algorithm has some limits.

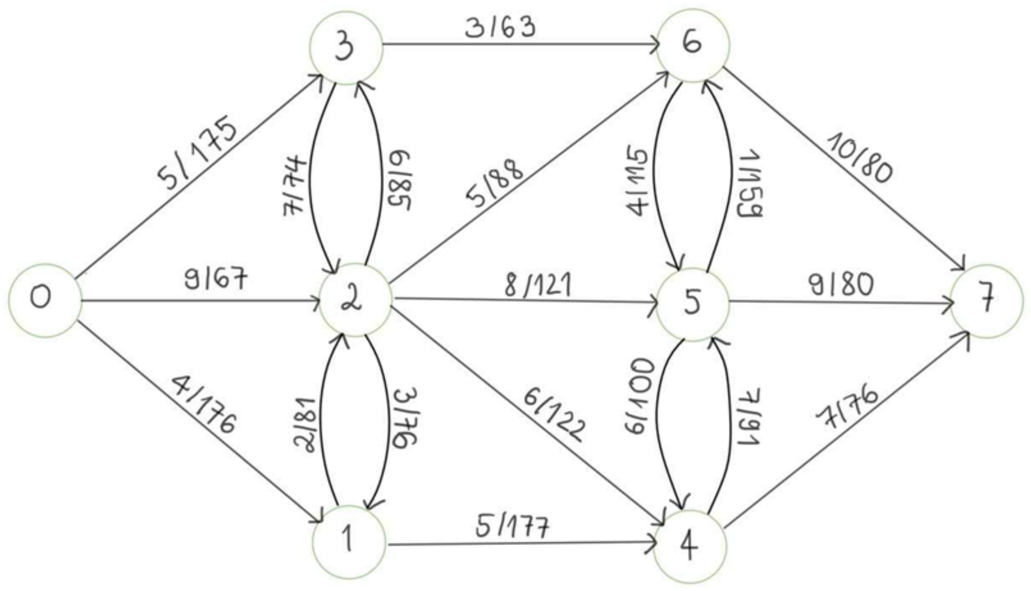
First, the residual network cannot contain any loop with the total cost is negative, which could leads to the Dijkistra's algorithm to be stuck in the loop, which may affects a lot because there are situations like that in reality. *Second*, the algorithm can only be solved in this specific situation, which there are only one pair of OD, and no other bounds like the time-dependent model, which contains a bound of the second decision variable y with the first variable x.

Best and worst cases for the algorithm: The best cases for the algorithm is when the required flow v is small enough compare to the maximum flow availabe in the network , or when the links' capacities is not to small compare to v. This leads to the number of times we have to search for the min-cost path is small.

The worst cases is the opposite. When the v is large compare to the maximum flow available, and the links' capacities is small, the number of times we have to search for the min-cost path is gradually increasing.

5.5 Example of Applying Algorithm 1

Assume that there is a small network which has **8 nodes** and **19 links**. The nodes are numbered **from 0 to 7** with **source node 0** and **sink node 7**. There are **20 people** located in **node 0**, which is assume to be disaster area. Our mission is to evacuate them as many as possible to **node 7**, supposed to be safe area. Thanks to **Algorithm 1**, we can decide how many people in maximum that can be evacuated by finding successive shortest path from **node 0** to **node 7**. We can also decide how many people that can move at a time as long as which path they can go in each step.
We denote $d_0 = 20$ to be the value of supply **node 0** (The number of people needed evacuated)
Here is initial residual network:



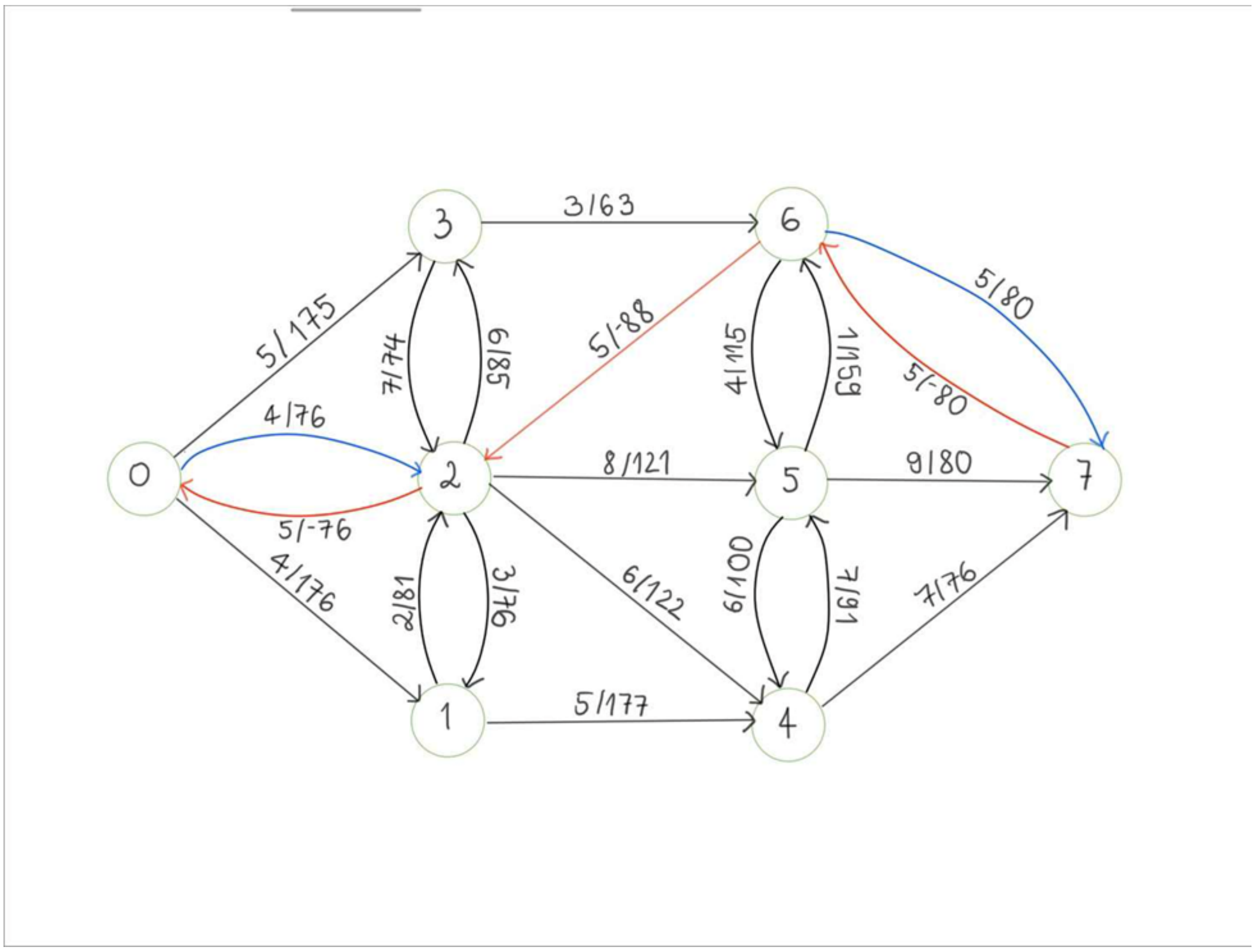
Iteration 1:

Shortest path found in this step: 0 – 2 – 6 – 7

Maximum flow in this step: 5

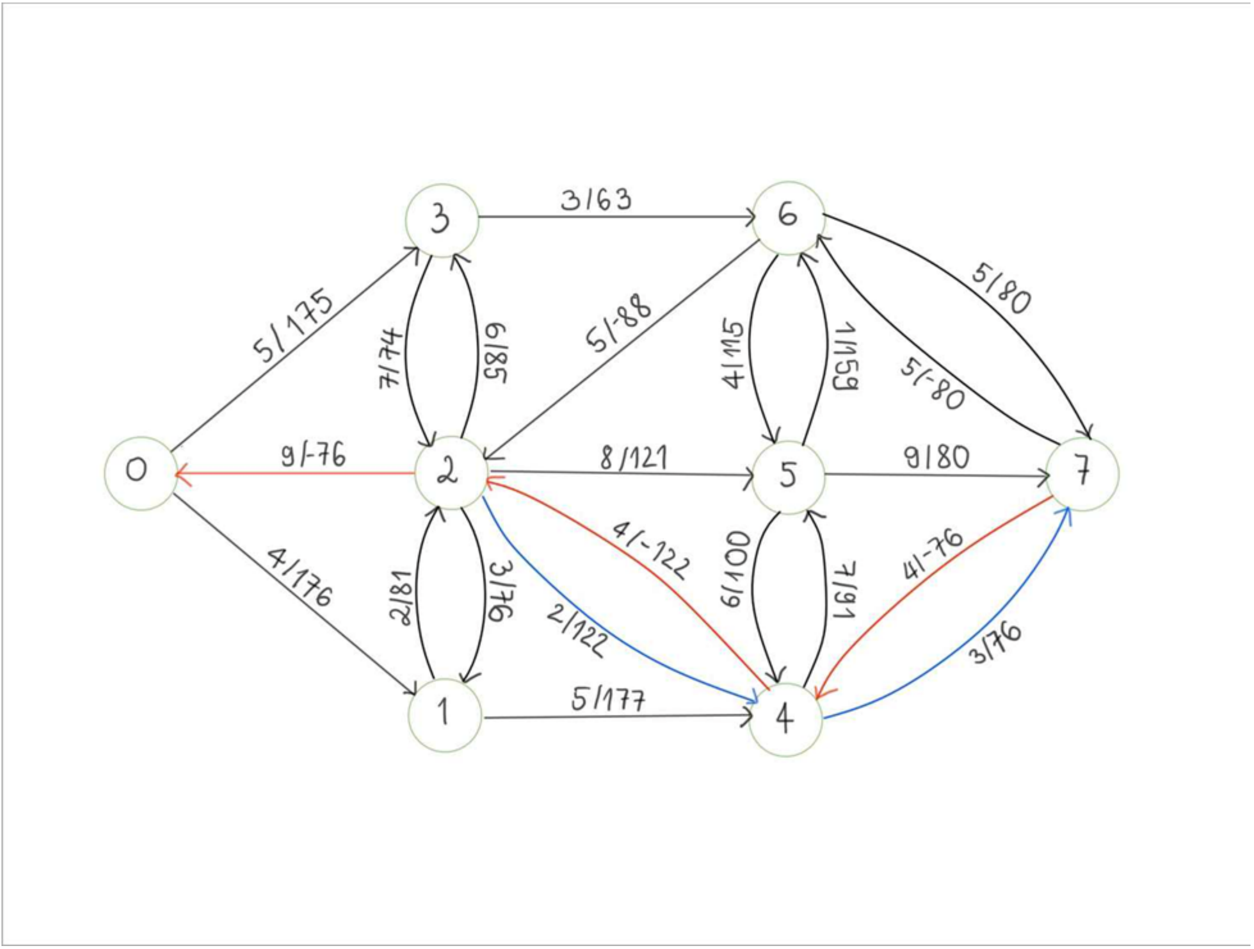
$d_0 = 15$

Recreate residual network:



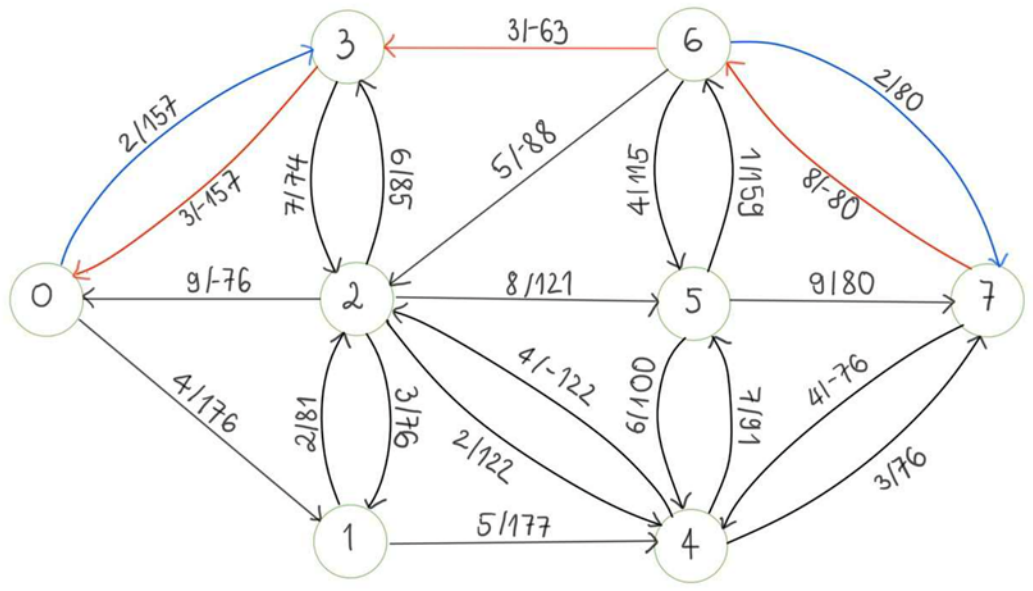
Iteration 2

Shortest path found in this step: $0-2-4-7$
Maximum flow in this step: 4
$d_0 = 11$
Recreate residual network:



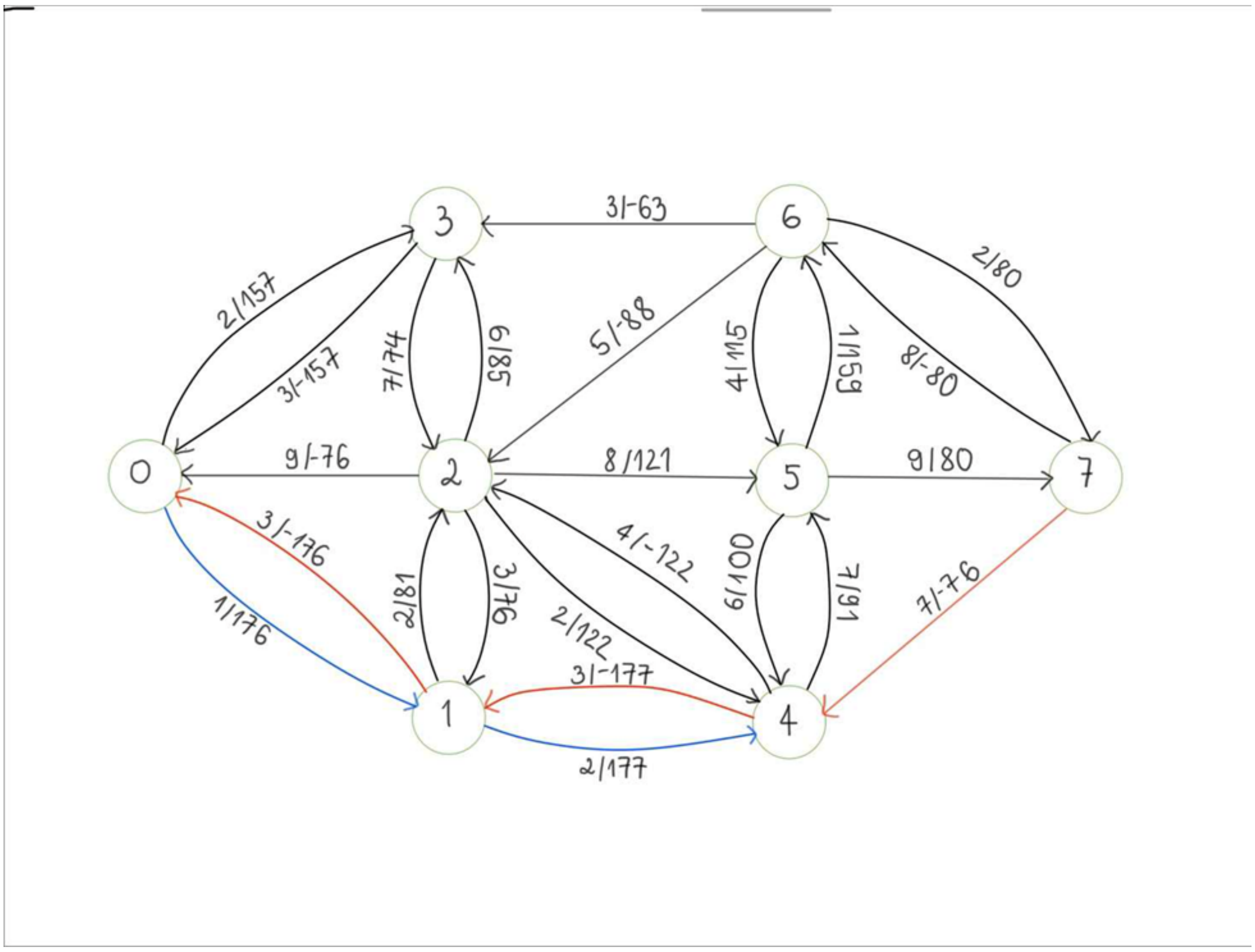
Iteration 3

Shortest path found in this step: 0 – 3 - 6 - 7
Max flow in this step: 3
$d_0 = 8$
Recreate residual network:



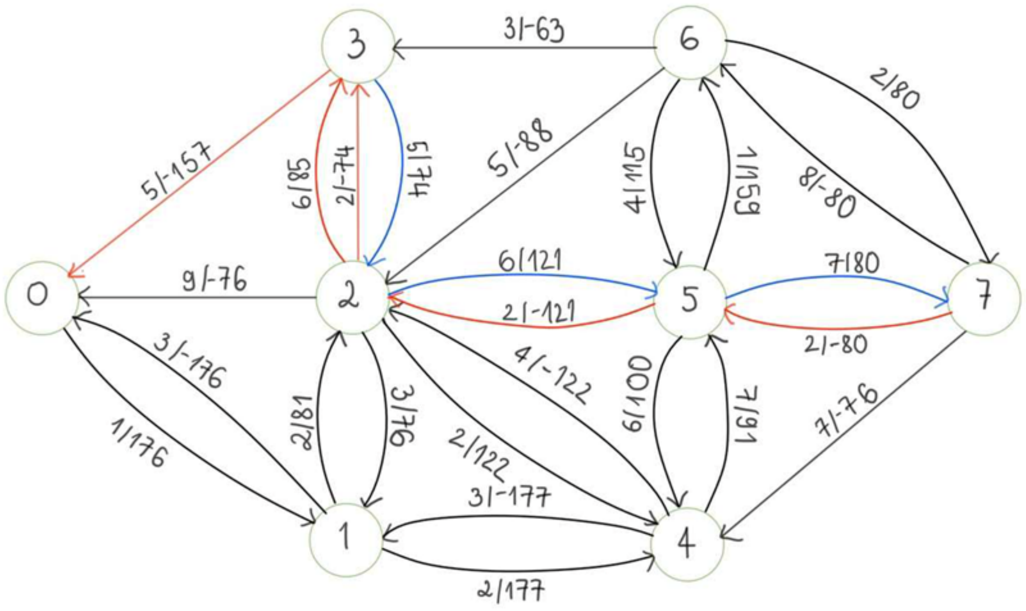
Iteration 4

Shortest path found in this step: 0 – 1 – 4 – 7
Maximum flow in this step: 3
$d_0 = 5$
Recreate residual network:



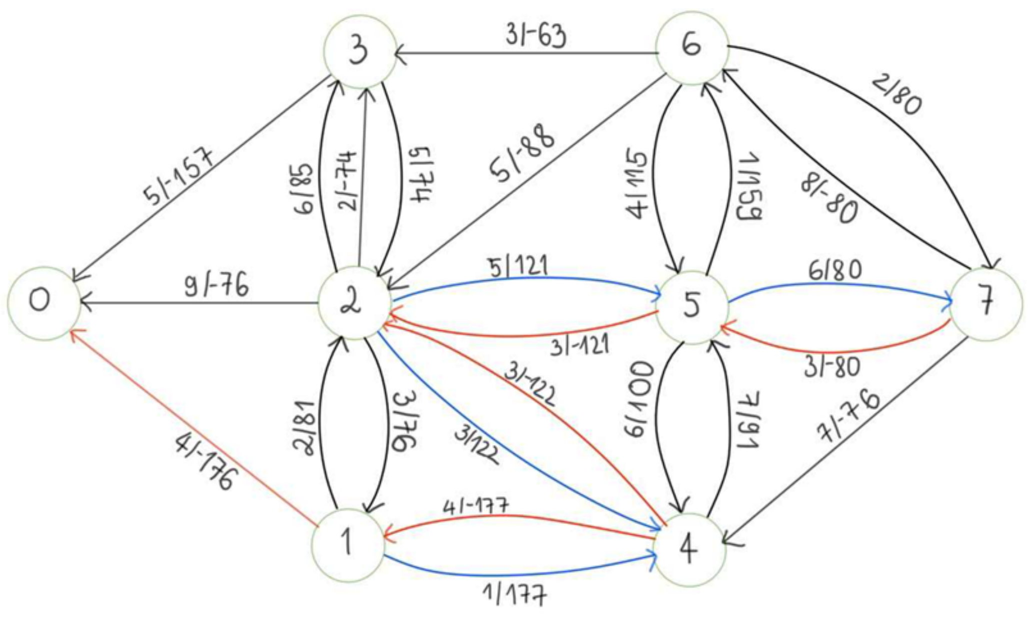
Iteration 5

Shortest path found in this step: $0 - 3 - 2 - 5 - 7$
Maximum flow in this step: 2
$d_0 = 3$
Recreate residual network:



Iteration 6

Shortest path found in this step: 0 – 1 – 4 – 2 – 5 – 7
Maximum flow in this step: 1
$d_0 = 2$
Recreate residual network:



Maximum flow in total: 18
Remark:
- As can be seen, the network has some links that are not used: (2;1), (1;2), (4;5), (5;4), (6;5), (5;6).
- There are 2 people who are not able to be evacuated because there is no path found in the residual network. In fact, the maximum available flow is just 18, so there is no way we can evacuate the 19th person in this network.

References

[1] (MPS-SIAM Series on Optimization) Alexander Shapiro, Darinka Dentcheva, Andrzej Ruszczynski (2009) - Lectures on Stochastic Programming_Modeling and Theory-SIAM-Society for Industrial and Applied Mathematic

[2] John R. Birge, François Louveaux - Introduction to Stochastic Programming-Springer (1997).

[3] Li Wang - A two-stage stochastic programming framework for evacuation planning in disaster responses - School of Modern Post, Beijing University of Posts and Telecommunications, China

[4] Kall P., Wallace S. - Stochastic Programming-John Wiley Sons (2003)

[5] Man VM. Nguyen and Khuong A. Nguyen - MATHEMATICAL MODELING Methods and Application (2023) - Faculty of Science - Mahidol University & Faculty of Computer Science & Engineering - HCMC University of Technology

[6] Material for MkDocs - Minimum-cost flow - Successive shortest path algorithm (2023) - [https://cp-algorithms.com/graph/min_cost_flow.html]

[7] _efer_ - MAXIMUM FLOW: PART ONE (2018) - [https://www.topcoder.com /thrive/articles/Maximum%20Flow%20Part%20one]

[8] Zealint - MINIMUM COST FLOW PART TWO: ALGORITHMS (2018) - [https://www.topcoder.com/thrive/articles/Minimum%20Cost%20Flow%20Part%20Two:%20Algorithms]

[9] geeksforgeeks - Python Program for Dijkstra's shortest path algorithm | Greedy Algo-7 - [https://www.geeksforgeeks.org/python-program-for-dijkstras-shortest-path-algorithm-greedy-algo-7/]