ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

MÔ HÌNH HÓA TOÁN HỌC (CO2011)

Bài tập lớn
Stochastic Programming and Application

Giáo viên hướng dẫn: Trần Hồng Tài
Sinh viên: Nguyễn Cao Cường - 2210429
Lưu Lê Gia Bảo - 2210220
Nguyễn Mạnh Hùng - 2211337
Hàng Nhựt Long - 2211874
Mã Trường Vũ - 2213995

Thành phố Hồ Chí Minh, tháng 11/2023

Mục lục

1 Danh sách thành viên và tiến độ công việc

Số thứ tự	Họ và tên	MSSV	Đóng góp công việc	Tỉ lệ
1	Nguyễn Cao Cường	2210429	- Phân công, triển khai công việc - Hiện thực các ràng buộc của bài toán - Triển khai code cho vấn đề 1	20%
2	Lưu Lê Gia Bảo	2210220	- Mô tả tóm tắt, thiết lập bài toán vấn đề 1 - Xây dựng mô hình, lựa chọn giải thuật cho vấn đề 2 - Viết báo cáo	20%
3	Nguyễn Mạnh Hùng	2211337	- Lựa chọn hướng tiếp cận vấn đề 1 - Chuyển vấn đề 2 thành bài toán min-cost flow - Viết báo cáo	20%
4	Hàng Nhựt Long	2211874	- Viết báo cáo phần mục tiêu bài toán vấn đề 1 - Xử lí bài toán vấn đề 2 - Viết báo cáo	20%
5	Mã Trường Vũ	2213995	- Hỗ trợ tìm hiểu GAMSPy cho code của vấn đề 1 - Viết báo cáo, mô tả code cho vấn đề 1 - Theo dõi tiến độ hoàn thành của nhóm	20%

2 Problem 1: Industry - Manufacturing

2.1 Tóm tắt bài toán

Một nhà máy cần nhập nguyên liệu và sản xuất các mặt hàng với số lượng từng loại sản phẩm phụ thuộc vào nhu cầu thực tế **chưa xác định trước** mà chỉ biết dưới dạng **biến ngẫu nhiên**. Các tham số và biến trong bài được lần lượt được giải thích như sau:

- Giả sử nhà máy sản xuất n loại sản phẩm.
- Nhà máy cần nhập m loại nguyên liệu.
- Với mỗi sản phẩm thứ i ($i = \overline{1,n}$), ta cần a_{ij} nguyên liệu thứ j để sản xuất ($j = \overline{1,m}$). Chú ý a_{ij} có thể bằng 0, vì có thể có loại sản phẩm không cần đến vật liệu đó để sản xuất.
- Nhu cầu sản phẩm từ thị trường là một **vectơ ngẫu nhiên** $D = (D_1, D_2, \ldots, D_n)$, trong đó D_i là số lượng sản phẩm loại i tương ứng.
- **Trước khi biết nhu cầu**, nhà máy mua mỗi nguyên liệu thứ j với giá c_j.
- **Sau khi biết nhu cầu**, phần nguyên liệu đã mua được chia thành hai loại:
  - Phần nguyên liệu được sử dụng để sản xuất, nhà máy sản xuất tốn l_i và bán với giá q_i cho một sản phẩm thứ i.
  - Phần nguyên liệu bị dư được bán với giá s_j ($s_j < c_j$) với mỗi nguyên liệu thứ j.

Vì nhà máy không mua thêm phần nguyên liệu còn thiếu, nhà máy chấp nhận không đáp ứng đủ nhu cầu. (the unsastified demand is lost)

Giả sử, nhà máy đã đặt mỗi nguyên liệu thứ j với số lượng x_j ($j = \overline{1,m}$). **Sau khi biết nhu cầu**, nhà máy cần xác định số lượng mỗi loại sản phẩm cần sản xuất. Ta gọi số lượng sản phẩm cần sản xuất thứ i là z_i ($i = \overline{1,n}$) và số lượng nguyên liệu thứ j **còn tồn trong kho** là y_j ($j = \overline{1,m}$).

Mục tiêu bài toán là xác định z, y sao cho lợi nhuận thu được **lớn nhất**. Tức là:

$$\max_{z,y} Z = \sum_{i=1}^{n} (q_i - l_i) \cdot z_i + \sum_{j=1}^{m} s_j \cdot y_j$$

Với một vectơ D ngẫu nhiên **được xác định cụ thể** $d = (d_1, d_2, \ldots, d_n)$, chúng ta có thể tìm được nghiệm tối ưu bằng cách giải bài toán quy hoạch tuyến tính nguyên

$$\min_{z,y} Z = \sum_{i=1}^{n}(l_i - q_i)\cdot z_i - \sum_{j=1}^{m} s_j \cdot y_j$$

với các ràng buộc như sau:

$$\begin{cases} y_j = x_j - \sum_{i=1}^{n} a_{ij}\cdot z_i, \quad j = \overline{1,m} & (1) \\ 0 \le z_i \le d_i, \quad i = \overline{1,n} & (2) \\ y_j \ge 0, \quad j = \overline{1,m} & (3) \end{cases}$$

Mỗi ràng buộc có ý nghĩa như sau:

- Phương trình (1) thể hiện số lượng còn lại của từng loại nguyên liệu bằng số lượng đã mua trừ cho số lượng mang đi sản xuất.
- Bất phương trình (2) thể hiện số lượng sản xuất không lớn hơn nhu cầu của mỗi loại sản phẩm.
- Bất Phương trình (3) thể hiện số lượng còn lại của từng loại nguyên liệu phải không âm.

Một cách tương đương, ta có mô hình **ở bước hai** (**second stage**) như sau:

$$\text{MODEL} = \begin{cases} \min_{z,y} Z = (l-q)^T \cdot z - s^T \cdot y \\ y = x - A^T z \\ 0 \le z \le d, \quad y \ge 0 \end{cases} \qquad (*)$$

với l, q, s, d là các vectơ **cột** và ma trận $A = [a_{ij}]$ có kích thước $n \times m$.

Ta thấy rằng, nghiệm của hệ $(*)$, tức là các vectơ z và y, phụ thuộc vào vectơ d **đã xác định** từ **vectơ ngẫu nhiên** D và nghiệm ở **bước một** (**first stage**) của bài toán $x = (x_1, x_2, \ldots, x_m)$. Gọi $Q(x,d)$ là **giá trị tối ưu** (optimal value) của $(*)$ và vectơ $c = (c_1, c_2, \ldots, c_m)$. Như vậy số lượng của mỗi loại nguyên liệu x_j được xác định bởi bài toán tối ưu:

$$\min_{x \ge 0} c^T \cdot x + \mathbb{E}[Q(x,D)] \qquad (**)$$

trong đó kỳ vọng được lấy tuân theo phân phối xác suất của vectơ D.

Phần đầu của hàm mục tiêu ($c^T \cdot x$) biểu diễn **tổng giá tiền mua nguyên liệu**. Phần còn lại của hàm mục tiêu ($\mathbb{E}[Q(x,D)]$) biểu diễn **kỳ vọng của kế hoạch sản xuất tối ưu**.

2.2 Mục tiêu bài toán

Bài toán $(*)$ và $(**)$ được gọi là **bài toán ngẫu nhiên hai bước** (**two-stage stochastic programming problem**), trong đó $(*)$ được gọi là **bài toán ở bước 2** (**second-stage problem** và $(**)$ được gọi là **bài toán ở bước 1** (**frist-stage problem**). Vì $(*)$ chứa vectơ ngẫu nhiên D, giá trị tối ưu của nó $Q(x,d)$ cũng là một giá trị ngẫu nhiên.

Trong trường hợp có **hữu hạn** các viễn cảnh $d^1, d^2, \ldots, d^K$ xảy ra với xác suất tương ứng $p_1, p_2, \ldots, p_K$, với $\sum_{k=1}^{K} = 1$, bài toán $(*) - (**)$ có thể được viết thành một **bài toán quy**

hoạch tuyến tính nguyên với quy mô lớn:

$$\begin{cases} \min c^T \cdot x + \sum_{k=1}^{K} p_k \left[(l-q)^T \cdot z^k - s^T \cdot y^k \right], \quad \text{subject to} \\ y^k = x - A^T z^k, \quad k = \overline{1,K} \\ 0 \le z^k \le d^k, \quad k = \overline{1,K} \\ y^k \ge 0, \quad k = \overline{1,K} \\ x \ge 0 \end{cases} \tag{***}$$

Mục tiêu bài toán là cần đi tìm các vectơ $x, y \in \mathbb{R}^m$ và $z \in \mathbb{R}^n$ thỏa mãn mô hình đã cho

2.3 Hiện thực

2.3.1 Sinh dữ liệu

Ta cần sinh ra tập dữ liệu cho **5 vector** là l, q, s, c, D và ma trận A, trong đó các vector $l, q, D \in \mathbb{R}^n$ và $s, c \in \mathbb{R}^m$, cuối cùng là ma trận $A \in \mathbb{R}^{n \times m}$.

```
# Set up the model
n = 8
m = 5
S = 2
ps = 1/2
MAX_VALUE = 10
```

Hình 1: Khởi tạo các giá trị để sinh dữ liệu

Khởi tạo các giá trị n, m là các kích thước của ma trận và vector, **S** là số **viễn cảnh**, **ps** là **xác suất** xảy ra của viễn cảnh, **MAX_VALUE** là biến mà chúng ta thiết lập để đặt khoảng giá trị cho mỗi phần tử trong ma trận ngẫu nhiên được sinh ra.

Sinh dữ liệu ngẫu nhiên bằng thư viện numpy:

```
# Random values for vectors and matrices
l_arr = np.random.randint(1, MAX_VALUE, n)
s_arr = np.random.randint(1, MAX_VALUE, m)
q_arr = l_arr + np.random.randint(1000* MAX_VALUE, 2000* MAX_VALUE, n)
c_arr = s_arr + np.random.randint(10* MAX_VALUE, 20* MAX_VALUE, m)
D_arr1 = np.random.binomial(10, 0.5, n)
D_arr2 = np.random.binomial(10, 0.5, n)
A_arr = np.random.randint(1, 10* MAX_VALUE, size=(n, m))
```

Hình 2: Tiến hành sinh dữ liệu ngẫu nhiên

Ta khai báo thư viện **numpy** và đặt tên là **np**, cú pháp như sau: **import numpy as np**. Tiếp theo sử dụng câu lệnh **np.random.randint(min, max, size)** để sinh ra tập dữ liệu **ngẫu nhiên và có giá trị nguyên** cho các vector/ma trận, giá trị các phần tử trong đoạn **(min,**

max) và có kích thước là **size**. Size này có thể mở rộng ra nhiều chiều. Ví dụ size = (n, m) sẽ tạo ma trận kích thước **nxm**.

Để đảm bảo thỏa mãn điều kiện (q > l) và (c > s) nên q = l + randomMatrix, c = s + randomMatrix.

Trong bài toán này ta có 2 viễn cảnh, nên ta sẽ tạo 2 vector **D1, D2** tương ứng. Hai vector này sẽ được phân phối ngẫu nhiên theo **phân phối nhị thức**. Ta dùng lệnh **np.random.binomial(time, prob, size)** sẽ thực hiện **time** lần thử, có xác suất thành công là **prob**, và ta sẽ thu được vector **size** phần tử.

Ta thu được tập dữ liệu như sau:

```
Values of vectors and matrices:
l:  [40 30 19 27 40 15  3  8]
q:  [86 64 55 66 46 29  9 49]
s:  [19 34 21 10 39]
c:  [49 42 52 55 79]
D1:  [3 5 3 5 7 8 4 7]
D2:  [6 7 7 6 5 6 5 8]
A:
 [[47 26 17 21 18]
 [18 46 29  3 16]
 [34 15 25  7 49]
 [ 4 30 46 13 49]
 [38 35  5  5 49]
 [12  5 45 38 46]
 [38 26  9  6  4]
 [ 4 31 48  6 14]]
```

Hình 3: Kết quả thu được khi sinh dữ liệu ngẫu nhiên

2.3.2 Giải bài toán

Để giải mô hình bài toán, ta cần sử dụng các đối tượng được hỗ trợ bởi GAMSPy và cần khai báo các đối tượng đó trong phần khai báo như sau : **from gamspy import Sum, Model, Container, Set, Parameter, Variable, Equation, Sense**

```python
# Create Container for GAMS model
container = Container(delayed_execution=True)

# Define sets
i = Set(container, "i", records= ["i" + str(i) for i in range(n)])
j = Set(container, "j", records= ["j" + str(j) for j in range(m)])
```

Hình 4: Tạo các set template i, j

Tạo một **Container** cho mô hình **GAMSPy**, đây là nơi chứa các đối tượng, tham số, biến và các ràng buộc mà ta cần thực hiện và xử lí cho bài toán.

Set(container, name, records): Ta thêm đối tượng vào **container**, với tên của đối tượng là **name**, và **records** là tập liệu cho set. Tạo 2 Set mẫu là i, j lần lượt đại diện cho tập hợp có n, m phần tử (với n, m là kích thước của các vector, ma trận). Với đoạn code trong hình sẽ tạo

ra set i = [i1, i2,.., in] và set j = [j1, j2,.., jm].

Tiếp theo, ta sẽ tạo các Parameter, nhưng cần phải tạo bộ dữ liệu record trước.

```
# Define records for parameters
l_record = [["i" + str(i), l_arr[i]] for i in range(n)]
q_record = [["i" + str(i), q_arr[i]] for i in range(n)]
s_record = [["j" + str(j), s_arr[j]] for j in range(m)]
c_record = [["j" + str(j), c_arr[j]] for j in range(m)]
D_record1 = [["i" + str(i), D_arr1[i]] for i in range(n)]
D_record2 = [["i" + str(i), D_arr2[i]] for i in range(n)]
A_record = [["i" + str(i), "j" + str(j), A_arr[i][j]] for i in range(n) for j in range(m)]
```

Hình 5: Tạo bộ dữ liệu record cho Parameter

Bằng cách tạo mảng thông thường, ở đây ta tạo mảng 2 chiều, giả sử l_arr = [10, 20, 30] thì kết quả thu được là :l_record = [[i1, 10], [i2, 20], [i3, 30]]. l_arr là ngẫu nhiên. Các bộ record khác tương tự.

```
# Define parameters
l = Parameter(container, "l", domain = [i], records = l_record)
q = Parameter(container, "q", domain = [i], records = q_record)
s = Parameter(container, "s", domain = [j], records = s_record)
c = Parameter(container, "c", domain = [j], records = c_record)
D1 = Parameter(container, "D1", domain = [i], records = D_record1)
D2 = Parameter(container, "D2", domain = [i], records = D_record2)
A = Parameter(container, "A", domain = [i, j], records = A_record)
```

Hình 6: Tạo các Parameter của bài toán

Parameter(container, name, domain, records) được thêm vào **container**, với tên đối tượng là **name**, miền giá trị **domain** của Parameter, và bộ giá trị **record** đã được tạo ở trên. Ta tạo cho các vector l, q, s, c, D1, D2 và ma trận A các parameter tương ứng.

```
# Define variables
x = Variable(container, name= "x", domain = [j], type = "Positive")
y1 = Variable(container, name= "y1", domain = [j], type = "Positive")
z1 = Variable(container, name= "z1", domain = [i], type = "Positive")
y2 = Variable(container, name= "y2", domain = [j], type = "Positive")
z2 = Variable(container, name= "z2", domain = [i], type = "Positive")
```

Hình 7: Tạo các Variable trong bài

Variable(container, name, domain, type) được thêm vào **container**, với tên đối tượng là **name**, miền giá trị **domain**, chọn **type Positive** để thiết lập giá trị nhận cho các biến này là số nguyên không âm.

Equation(container, name, type, domain) được thêm vào **container**, đặt **name**, chọn **type regular** để thực hiện các phương trình toán học thông thường cho các biến này như phương trình, bất phương trình,... và có miền giá trị **domain**.

```
# Define equations for variables
x_positive = Equation(container, name = "x_positive", type="regular", domain = [j])
# Define equations for scenario 1
y1_positive = Equation(container, name = "y1_positive", type="regular", domain = [j])
z1_positive = Equation(container, name = "z1_positive", type="regular", domain = [i])
y1_constraint = Equation(container, name = "y1_constraint", type="regular", domain = [j])
z1_constraint = Equation(container, name = "z1_constraint", type="regular", domain = [i])
# Define equations for scenario 2
y2_positive = Equation(container, name = "y2_positive", type="regular", domain = [j])
z2_positive = Equation(container, name = "z2_positive", type="regular", domain = [i])
y2_constraint = Equation(container, name = "y2_constraint", type="regular", domain = [j])
z2_constraint = Equation(container, name = "z2_constraint", type="regular", domain = [i])
```

Hình 8: Viết các phương trình cho các biến

```
# Build constraints for variables
x_positive[j] = x[j] >=0                                # x[j] >=0
# Constraints for scenario 1
y1_positive[j] = y1[j] >=0                              # y1[j] >=0
z1_positive[i] = z1[i] >=0                              # z1[i] >=0
z1_constraint[i] = z1[i] <= D1[i]                       # z1[i] <= D1[i]
y1_constraint[j] = y1[j] == Sum(i, x[j] - A[i, j]*z1[i])    # y1[j] = x[j] - A[i, j]*z1[i]
# Constraints for scenario 2
y2_positive[j] = y2[j] >=0                              # y2[j] >=0
z2_positive[i] = z2[i] >=0                              # z2[i] >=0
z2_constraint[i] = z2[i] <= D2[i]                       # z2[i] <= D2[i]
y2_constraint[j] = y2[j] == Sum(i, x[j] - A[i, j]*z2[i])    # y2[j] = x[j] - A[i, j]*z2[i]
```

Hình 9: Những ràng buộc của các biến x, y, z

Tạo các ràng buộc cho các biến x, y, z:

Để thực hiện tính xích ma, ta sử dụng hàm **Sum(index, expression)**. Khi thực hiện truyền vào một đối tượng Set tại vị trí **index**, hàm **Sum** sẽ lặp qua tất cả các phần tử trong Set và thực hiện tính toán biểu thức **expression** tại vị trí đó, sau đó tính tổng chúng và trả về kết quả.

```
# build module for objection function value
firstStage = Sum(j, c[j]*x[j])                          # calculate c[j]*x[j]
# Calculate module for scenario 1
secondStage_step1_sce1 = Sum(i, (l[i] - q[i])*z1[i])    # calculate (l[i] - q[i])*z1[i]
secondStage_step2_sce1 = Sum(j, s[j]*y1[j])             # calculate s[j]*y1[j]
# Calculate module for scenario 2
secondStage_step1_sce2 = Sum(i, (l[i] - q[i])*z2[i])    # calculate (l[i] - q[i])*z2[i]
secondStage_step2_sce2 = Sum(j, s[j]*y2[j])             # calculate s[j]*y2[j]

obj =  firstStage + ps*(secondStage_step1_sce1 - secondStage_step2_sce1) + ps*(secondStage_step1_sce2 - secondStage_step2_sce2)
```

Hình 10: Tiến hành tính bước 1 và bước 2 của bài toán

Dùng hàm **Sum** tính giá trị **one-Stage, two-Stage**, cuối cùng tổng hợp lại thành hàm mục tiêu (gồm one-Stage, giá trị của từng viễn cảnh cũng như xác suất xảy ra của chúng).

```python
# Solve the model with parameters
model = Model(
    container,
    name="myModel",
    equations=container.getEquations(),
    problem="LP",
    sense=Sense.MIN,
    objective=obj,
)
model.solve()
```

Hình 11: Xây dựng mô hình bài toán trong GAMSpy

Model(container, name, equations, problem, sense, objective): Thêm **model** vào **container** với tên là **name**, lấy các phương trình đã được thêm vào trong container trước đó bằng hàm **get()** của container, ta chọn **problem** là "**LP**", và **sense** là **MIN** để model xử lí bài toán **Linear Programing** làm minimum hàm mục tiêu, và truyền vào hàm mục tiêu biến **obj** đã được định nghĩa trước đó.

Sau khi chạy chương trình hoàn tất, ta thu được kết quả như sau:

```
Result:
x:  [229, 292, 272, 286, 228]
y1:  [353, 0, 101, 0, 0]
y2:  [0, 279, 0, 174, 128]
z1:  [8, 4, 3, 6, 8, 4, 3, 6]
z2:  [8, 4, 5, 4, 7, 7, 2, 4]
Objective Function Value:  -440104.25
```

Hình 12: Phương án tối ưu và giá trị hàm mục tiêu

3 To the SLP-EPDR: Algorithmic Solutions

3.1 Giới thiệu bài toán

Thảm họa thiên nhiên (động đất, bão lũ, hỏa hoạn,...) và những thảm họa phi tự nhiên khác (khủng bố, vấn đề chính trị, chiến tranh,...) có thể càn quét qua một cộng đồng người mà không cảnh báo trước, để lại nhiều thiệt hại và thương vong. Mục đích chính của **ứng phó khẩn cấp** (Emergency response) là để cung cấp phương án lánh nạn an toàn cho người dân nhanh nhất có thể. Chúng ta quan tâm đến việc nghiên cứu một kế hoạch sơ tán rõ ràng cho người dân từ vùng nguy hiểm cho đến những vùng an toàn. Đồng thời, tập trung giải quyết vấn đề trong việc mô hình hóa bài toán sơ tán. Ta biết rằng, rất khó để có thể ước lượng được chính xác những thiệt hại mà thảm họa gây ra, tác động lên những tuyến đường trong khu dân cư. Do đó, vấn đề này sẽ được quy về là một **bài toán ngẫu nhiên** (stochastic programing problem), và cái ngẫu nhiên là nằm ở thời gian di chuyển (travel time) và cả sức chứa (capacity) trên mỗi tuyến đường.

Như đã biết 2SLP-WR được dùng để tìm ra phương án tối ưu trước khi biết được những biến ngẫu nhiên. Trên cơ sở đó, bài toán sơ tán sẽ được mô hình hóa là một bài toán ngẫu nhiên

2 bước dựa trên những viễn cảnh (scenario-based 2-stage stochastic programing model) để biểu thị được sự ngẫu nhiên trong tác động của thảm họa.

3.2 Biểu diễn bài toán sơ tán dân cư dưới dạng bài toán hai bước

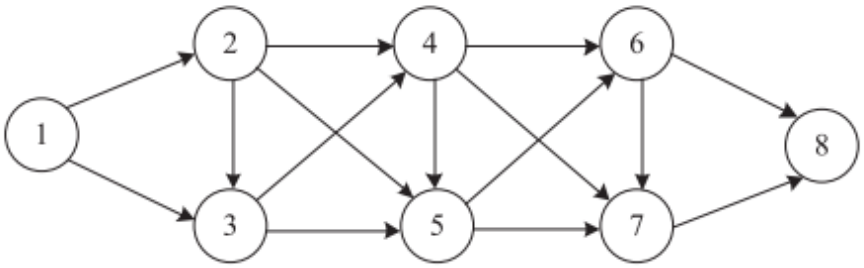
Trong phần này, chúng ta lấy sự xảy ra các trận động đất làm ví dụ để mô tả quá trình sơ tán dân cư từ vùng không an toàn tới vùng an toàn. Giả sử trong quá trình sơ tán, dân cư được nhận cảnh báo sớm và di chuyển ra khỏi vùng nguy hiểm bằng ô tô của họ.

Vào giai đoạn sau của việc sơ tán, thông tin của trận động đất được cập nhật chính xác. Tuy nhiên, ở giai đoạn đầu của việc sơ tán, chúng ta **chưa biết** được thông tin này. Trong trường hợp này, kế hoạch sơ tán ban đầu sẽ phụ thuộc vào số lượng các kịch bản có thể xảy ra. **Do đó, việc sơ tán nên được chia thành hai giai đoạn**:

- Trong giai đoạn đầu tiên, quá trình sơ tán cần được lên kế hoạch **trước khi biết về thông tin của trận động đất**.
- Trong giai đoạn thứ hai, kế hoạch sơ tán sẽ **được xác định một cách chính xác** sau khi **thông tin được biết**.

Mục tiêu của bài toán là tìm ra kế hoạch sơ tán tối ưu nhất **trong giai đoạn đầu** dựa trên những biến cố xảy ra ngẫu nhiên **trong giai đoạn hai**.

Lấy ví dụ minh họa với một mạng đơn giản gồm 8 nút (nodes) và 15 cạnh (links):

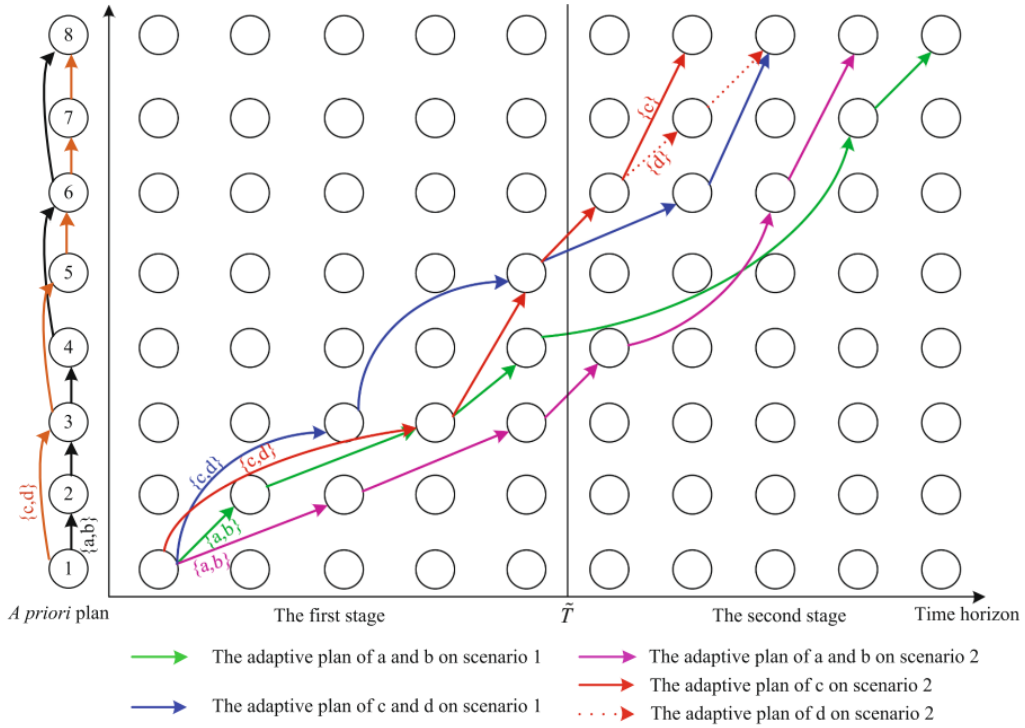


Hình 13: Mô phỏng một mạng sơ tán đơn giản

Trong mạng trên, ta giả sử node 1 và node 8 lần lượt là vùng có thiên tai và vùng an toàn. Bên cạnh đó, có bốn chiếc xe ký hiệu lần lượt là **a**, **b**, **c** và **d** được sơ tán tới vùng an toàn. **Trong giai đoạn đầu**, cả bốn chiếc xe được sơ tán theo **kế hoạch ban đầu** theo con đường:

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

Tuy nhiên, **trong giai đoạn sau**, tức là sau **ngưỡng thời gian** $\tilde{T}$, thông tin chính xác về trận động đất đã biết, ta cần cập nhật phần còn lại của kế hoạch cho phù hợp. Để đơn giản, ta xét hai kịch bản có thể xảy ra dưới đây:



Hình 14: Two-stage stochastic evacuation plan in time-dependent network

Dựa trên hình 14, ta thấy rằng:

- Trước ngưỡng thời gian $\tilde{T}$ (giai đoạn đầu - first-stage), việc sơ tán tuân theo kế hoạch dự định ban đầu (**a** và **b** di chuyển theo đường $1 \longrightarrow 2 \longrightarrow 3 \longrightarrow 4$, còn **c** và **d** di chuyển theo đường $1 \longrightarrow 3 \longrightarrow 5$)
- Sau ngưỡng thời gian $\tilde{T}$ (giai đoạn sau - second-stage), việc sơ tán được cập nhật. Ta xét hai viễn cảnh như hình 14:
  - viễn cảnh 1: **a**, **b** tiếp tục đi theo đường $7 \longrightarrow 8$ và **c**, **d** tiếp tục đi theo đường $6 \longrightarrow 8$.
  - viễn cảnh 2: **a**, **b** tiếp tục đi theo đường $6 \longrightarrow 8$ và **c**, **d** tiếp tục đi theo đường $7 \longrightarrow 8$.

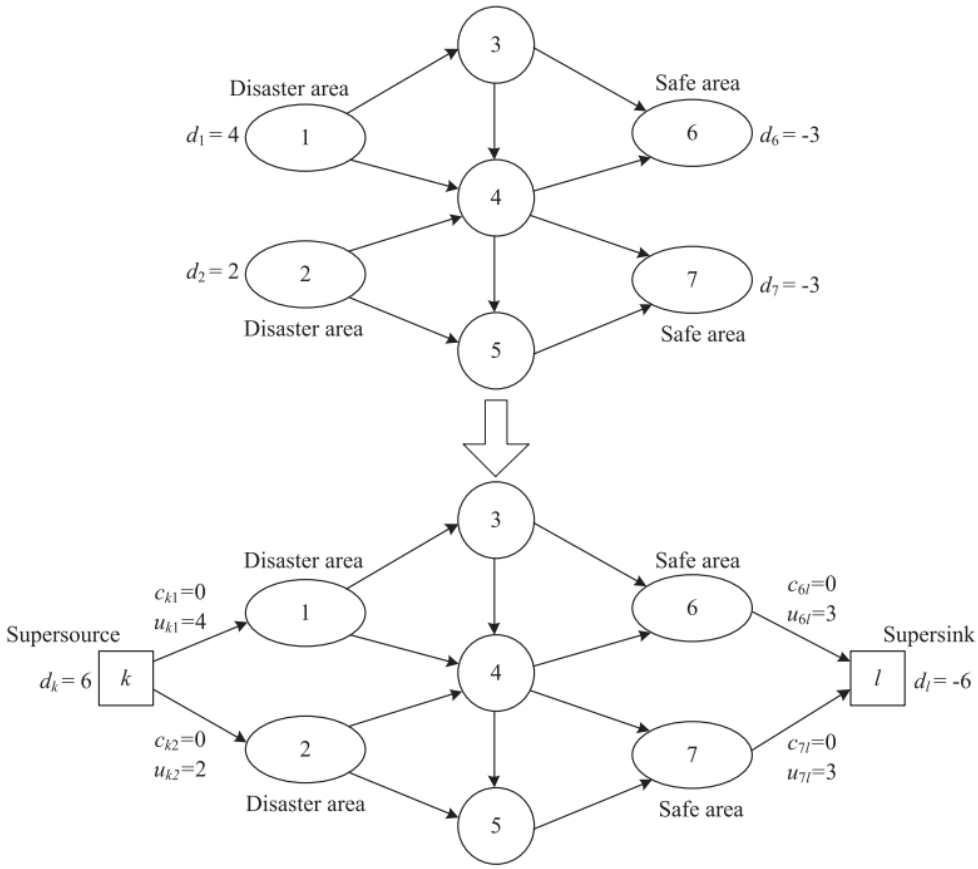
Tóm lại, ta cần xây dựng kế hoạch **trong giai đoạn đầu** (trước khi thông tin về động đất xảy ra)

3.3 Xử lý trường hợp nhiều nguồn và nhiều đích

Trên thực tế, một mạng có thể có nhiều nguồn (source) và nhiều đích (sink). Vì vậy, chúng ta cần chuyển về một mạng tương đương có **duy nhất một nguồn (supersource) và một đích (superlink)**.

- **Đối với Physical Network**

- Supersource: Supersource k được thêm vào mạng. Những **cạnh ảo** (k, i) được thêm vào với thời gian di chuyển $c_{ki} = 0, i \in K$ (K là tập hợp các node nguồn) và sức chứa trên mỗi cạnh ảo bằng với số người tại node nguồn thứ i, $u_{ki} = d_i, \forall i \in K$. Do đó sức chứa tại supersource bằng tổng sức chứa của các node source thành phần: $d_k = \sum_{i \in K} d_i$.
- Superlink: Tương tự với supersource

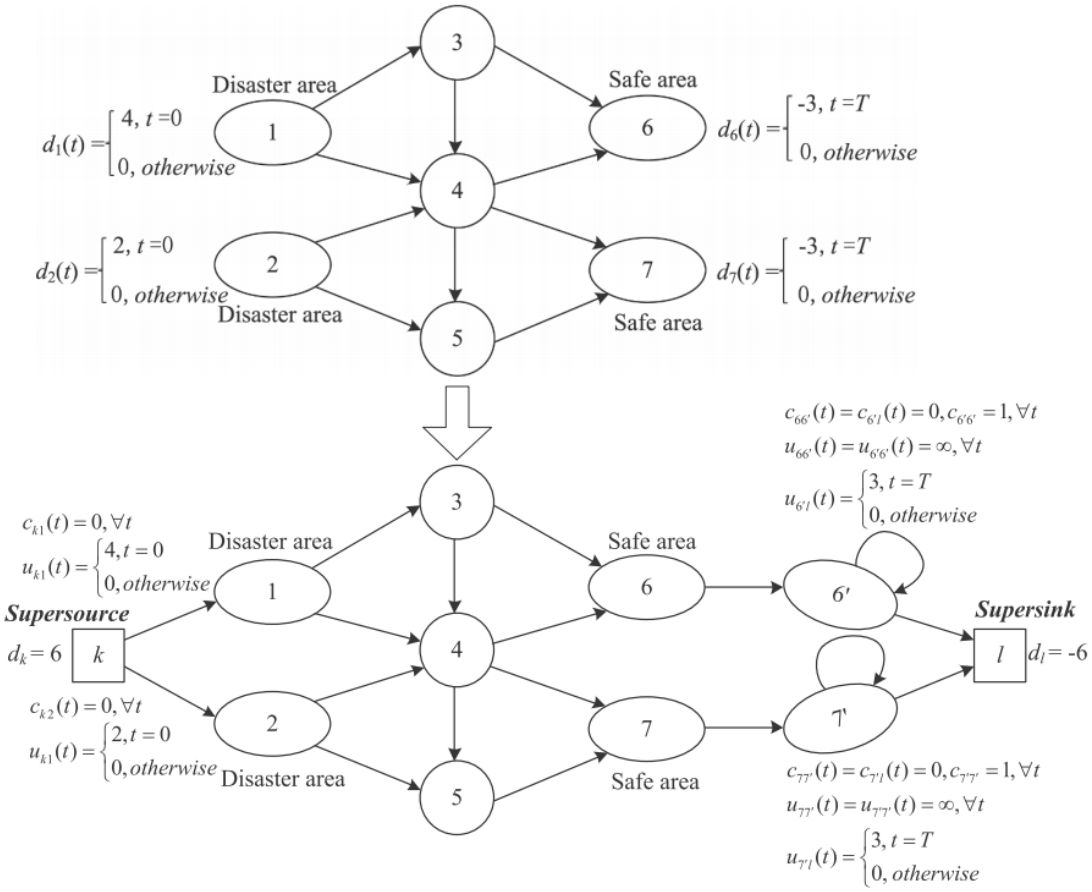


Hình 15: Supersource và supersink của Physical Network

- **Đối với Time-dependent Network** thời gian di chuyển và sức chứa trên mỗi tuyến đường tại mỗi thời điểm là khác nhau, tùy thuộc vào thời gian xuất phát (từ source) và thời gian đến (tới sink). Thế nên:
- cạnh ảo xuất phát từ supersource có:
  - thời gian di chuyển: $c_{ki}(t) = 0, i \in K$
  - sức chứa $u_{ki}(t) = d_i(t), t \in \{0, 1, \ldots, T\}$ (supply of node i at time t)
- Với mỗi sink $j, j \in D$, ta tạo 1 bản sao j', cạnh ảo jj' có:
  - thời gian di chuyển: $c^s_{jj'}(t) = 0, t \in \{0, 1, \ldots, T\}$

- sức chứa $u_{jj'}(t) = \infty$

- thêm vào mỗi node j' một khuyên (self-loop) có:
  - thời gian di chuyển: $c^s_{jj'}(t) = 1$
  - sức chứa $u_{jj'}(t) = \infty$
- với cạnh $j'l$ tới supersink:
  - thời gian di chuyển: $c^s_{j'l}(t) = 0$
  - sức chứa $u_{j'l}(T) = b_j(T)$ (demand of node j at time T)



Hình 16: Supersource và supersink của time-dependent Network

3.4 Chuyển về bài toán Min-cost flow problem

Bài toán sẽ mô tả chi tiết kế hoạch di tản dưới dạng **bài toán min-cost flow problem** với các giá trị về thời gian di chuyển (travel time) và sức chứa (capacity) của các cạnh hoàn toàn **ngẫu nhiên**. Mục tiêu là di tản người dân với vùng an toàn với **thời gian di chuyển nhỏ nhất** trên **capacity-cost network** $G(V, A, C, U, D)$:

- V là tập hợp các node.
- A là tập hợp các cạnh với giá trị *travel time* và *capacity* **ngẫu nhiên**.
- $C(i, j)$ là *travel time* trên cạnh $(i, j) \in A$, kí hiệu c_{ij}.
- $U(i, j)$ là *capacity* trên cạnh $(i, j) \in A$, kí hiệu u_{ij}.
- $D(i)$ là luồng (flow) đang có trên node $i \in V$, kí hiệu d_i.

Ta cũng giả sử rằng *travel time* trên cạnh $(i, j) \in A$ là **không đổi** theo thời gian.

3.5 Khởi tạo mô hình

Ta sẽ xây dựng cấc công thức cho bài toán này thông qua các biến, các ràng buộc, và các hàm mục tiêu. Để thuận tiện cho việc mô hình hóa, ta có các bảng ký hiệu như sau:

Kí hiệu	Ý nghĩa
V	Tập hợp các node (nút giao các tuyến đường)
A	Tập hợp các cạnh (các tuyến đường)
i, j	Chỉ số của các node, i, $j \in V$
(i, j)	Chỉ số của các cạnh có hướng (hướng i->j), $(i, j) \in A$
s	Chỉ số của các viễn cảnh
S	Tổng số viễn cảnh
v	Lượng người đi ra từ node nguồn
$\tilde{T}$	Ngưỡng thời gian (thời điểm nhận được thông tin chính xác)
T	Tổng số các mốc thời gian
u_{ij}	Sức chứa của một tuyến đường (i, j)
$u^s_{ij}(t)$	Sức chứa của một tuyến đường (i, j) trong viễn cảnh s ở thời điểm t
$c^s_{ij}(t)$	Thời gian di chuyển trên tuyến đường (i, j) trong viễn cảnh s ở thời điểm t
μ_s	Xác suất xảy ra sự kiện s
x_{ij}	Lượng người trên tuyến đường (i, j)
$y^s_{ij}(t)$	Lượng người trên tuyến đường (i, j) trong viễn cảnh s ở thời điểm t

Trong giai đoạn đầu, một biến x_{ij} được dùng để đại diện cho dòng người trên tuyến đường (i, j). Biến thứ 2 $y^s_{ij}(t)$ đại diện cho dòng người trên tuyến đường (i, j) trong viễn cảnh s ở thời điểm t.

Giai đoạn đầu tiên:

Trong giai đoạn đầu, một kế hoạch sơ tán khả thi sẽ được quyết định từ một siêu điểm nguồn đến một siêu điểm đích, lượng người trên các tuyến đường phải thõa mản **ràng buộc cân bằng**:

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

Trong đó d_i là tham số được định nghĩa như sau:

$$d_i = \begin{cases} v, & i = s \quad \text{(chênh lệch giữa lượng người ra và vào tại nguồn chính bằng v)} \\ -v, & i = t \quad \text{(tại điểm đích chỉ có luồng người vào)} \\ 0, & \text{otherwise} \quad \text{(tại các node khác, lượng người ra bằng lượng người vào)} \end{cases}$$

Đồng thời, lượng người trên các tuyến đường cũng phải thỏa **ràng buộc sức chứa**:

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

Để loại bỏ các vòng lặp và đường phụ, ta sẽ đặc biệt đưa ra link penalty $p_{ij} \in A$ .Hàm penalty có thể được định nghĩa:

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

Giai đoạn thứ hai:

Trong giai đoạn này, những người bị ảnh hưởng sẽ nhận được các chỉ dẫn có thời gian sơ tán tối thiểu ứng với thông tin vè thiên tai trong thời gian thực. Trước ngưỡng thời gian $\tilde{T}$, người dân sẽ sơ tán theo một kế hoạch đã được dự tính trước. Do đó, phương án sơ tán ban đầu là như nhau trong mọi viễn cảnh. Như vậy, các ràng buộc ghép nối của kế hoạch sơ tán trước ngưỡng thời điểm $\tilde{T}$ của mọi viễn cảnh có thể được biểu diễn như sau:

$$\sum_{t\leq \tilde{T}} y_{ij}^s(t) = x_{ij}, \quad (i,j) \in A, \quad s = \overline{1,S} \tag{4}$$

Tiếp theo, một mô hình của giai đoạn thứ 2 được tạo ra nhằm mục đích giảm thiểu tổng thời gian sơ tán của những người bị ảnh hưởng từ khu vực nguy hiểm đến khu vực an toàn trong mỗi viễn cảnh:

$$Q(Y,s) = \min \sum_{(i,j)\in A_s} c_{ij}^s(t) \cdot y_{ij}^s(t), \quad \text{subject to:} \tag{5}$$

$$\sum_{(i_t,j_{t'})\in A_s} y_{ij}^s(t) - \sum_{(j_{t'},i_t)\in A_s} y_{ji}^s(t') = d_i^s(t), \forall i \in V, t \in \{0,1,\ldots,T\}, s = \overline{1,S} \tag{6}$$

$$0 \leq y_{ij}^s(t) \leq u_{ij}^s(t), \forall (i,j) \in A, t \in \{0,1,\ldots,T\}, s = \overline{1,S} \tag{7}$$

$$\sum_{t\leq \tilde{T}} y_{ij}^s(t) = x_{ij}, (i,j) \in A, s = \overline{1,S} \tag{8}$$

Hàm mục tiêu (5) chính là tổng thời gian sơ tán của tất cả phương tiện giao thông trong viễn cảnh s. Ràng buộc (6) và (7) chính là ràng buộc cân bằng dòng người và ràng buộc sức chứa. Ràng buộc (8) là một ràng buộc ghép nối để đảm bảo rằng việc sơ tán trong các viễn cảnh trong giai đoạn 2 trước ngưỡng thời gian $\tilde{T}$ giống với kế hoạch sơ tán được dự tính trước trong giai đoạn đầu tiên

Diều ta quan tâm trong vấn đề sơ tán là có được một phương án tối ưu trong giai đoạn đầu bằng cách tính toán các viễn cảnh có thể xáy ra trong giai đoạn hai, với xác suất xảy ra mỗi viễn cảnh s là μ_s. Để có thể tối thiểu được penalty (của phương án sơ tán ban đầu) và giá trị kỳ vọng (của phương án thích ứng trong mỗi viễn cảnh), mô hình phương án cứu nạn 2 giai đoạn

phụ thuộc thời gian và biến ngẫu nhiên được mô tả như sau:

$$\begin{cases} \min \sum\limits_{(i,j)\in A} p_{ij} \cdot x_{ij} + \sum\limits_{s=1}^{S} \mu_s \cdot Q(Y,s), \quad \text{subject to} \\ \sum\limits_{(i,j)\in A} x_{ij} - \sum\limits_{(j,i)\in A} x_{ji} = d_i, \forall i \in V \\ 0 \le x_{ij} \le u_{ij}, \forall (i,j) \in A, \quad \text{in which} \\ Q(Y,s) = \min \sum\limits_{(i,j)\in A_s} c^s_{ij}(t) \cdot y^s_{ij}(t), \quad \text{subject to} \\ \sum\limits_{(i_t,j_{t'})\in A_s} y^s_{ij}(t) - \sum\limits_{(j_{t'},i_t)\in A_s} y^s_{ij}(t') = d^s_i(t), \forall i \in V, t \in \{0,1,\ldots,T\}, s = \overline{1,S} \\ 0 \le y^s_{ij}(t) \le u^s_{ij}(t), \forall (i,j) \in A, t \in \{0,1,\ldots,T\}, s = \overline{1,S} \\ \sum\limits_{t \le \bar{T}} y^s_{ij}(t) = x_{ij}, (i,j) \in A, s = \overline{1,S} \end{cases} \tag{9}$$

và bởi vì số lượng các viễn cảnh trong giai đoạn 2 là giới hạn, với $Q(Y,s)$ là phương án tối ưu của giai đoạn 2, mô hình trên tương đương với mô hình đơn giai đoạn sau:

$$\begin{cases} \min \sum\limits_{(i,j)\in A} p_{ij} \cdot x_{ij} + \sum\limits_{s=1}^{S} \left(\mu_s \cdot \sum\limits_{(i,j)\in A_s} c^s_{ij}(t) \cdot y^s_{ij}(t) \right), \quad \text{subject to} \\ \sum\limits_{(i,j)\in A} x_{ij} - \sum\limits_{(j,i)\in A} x_{ji} = d_i, \forall i \in V \\ 0 \le x_{ij} \le u_{ij}, \forall (i,j) \in A \\ \sum\limits_{(i_t,j_{t'})\in A_s} y^s_{ij}(t) - \sum\limits_{(j_{t'},i_t)\in A_s} y^s_{ij}(t') = d^s_i(t), \forall i \in V, t \in \{0,1,\ldots,T\}, s = \overline{1,S} \\ 0 \le y^s_{ij}(t) \le u^s_{ij}(t), \forall (i,j) \in A, t \in \{0,1,\ldots,T\}, s = \overline{1,S} \\ \sum\limits_{t \le \bar{T}} y^s_{ij}(t) = x_{ij}, (i,j) \in A, s = \overline{1,S} \end{cases} \tag{10}$$

Ngoài ra, nếu thông tin theo thời gian thực đã được cập nhật từ node gốc, thì ta không cần thiết phải xem xét kế hoạch sơ tán được dự tính trước (priori evacuation plan) nữa, sau đó mô hình sơ tán dựa trên Wait-and-see (WAS) có thể được xây dựng để giảm thiểu tổng thời gian di chuyển dự kiến trong nhiều viễn cảnh:

$$\begin{cases} \min \sum\limits_{s=1}^{S} \left(\mu_s \cdot \sum\limits_{(i,j)\in A_s} c^s_{ij}(t) \cdot y^s_{ij}(t) \right), \quad \text{subject to} \\ \sum (i_t, j_{t'}) \in A_s y^s_{ij}(t) - \sum\limits_{(j_{t'},i_t)\in A_s} y^s_{ij}(t') = d^s_i(t), \forall i \in V, t \in \{0,1,\ldots,T\}, s = \overline{1,S} \\ 0 \le y^s_{ij}(t) \le u^s_{ij}(t), \forall (i,j) \in A, t \in \{0,1,\ldots,T\}, s = \overline{1,S} \end{cases} \tag{11}$$

Mô hình (11) được ký hiệu là WAS. Về bản chất, miền khả thi của mô hình WAS rõ ràng có chứa miền khả thi của mô hình gốc, do đó mô hình WAS này chính là giới hạn dưới của mô hình gốc. Và do giữa các ràng buộc của các viễn cảnh khác nhau không có mối liên hệ nào, nên ta có thể phân tách mô hình thành S bài toán con dựa trên các viễn cảnh. Đối với mỗi viễn cảnh trong mô hình này, về cơ bản nó là một bài toán tìm dòng có chi phí tối thiểu (min-cost flow problem) phụ thuộc vào thời gian

3.6 Giải thuật cho mô hình

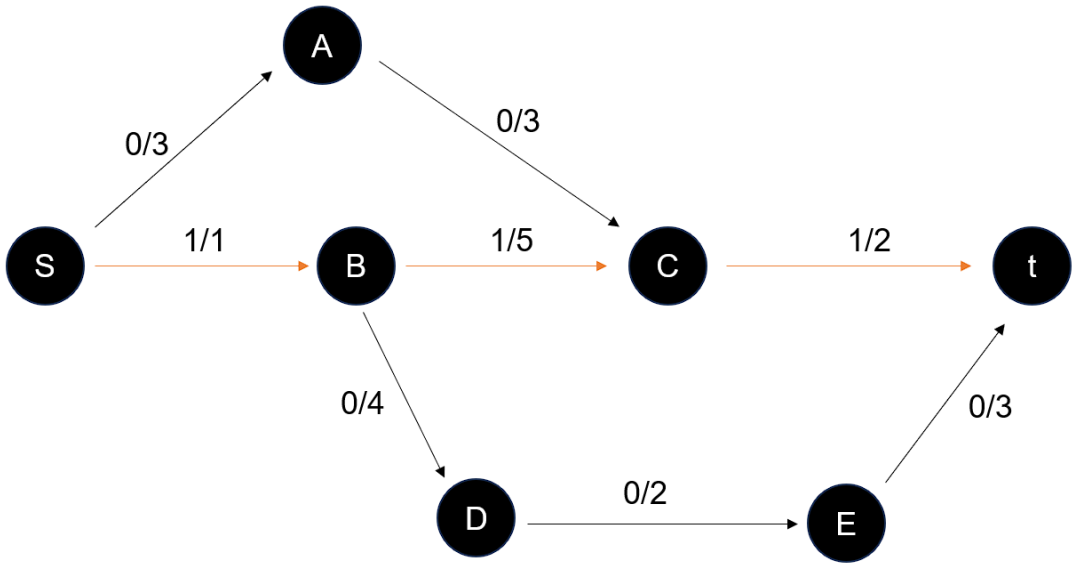
Trước khi vào chủ đề, ta trình bày các vấn đề liên quan

3.6.1 Mạng thặng dư - Residual Network

Mạng thặng dư $G'(E', V')$ của mạng $G(E, V)$ cho biết sức chứa còn lại trên mạng $G(E, V)$ khi đã gửi một số luồng f^* qua nó và được xây dựng như sau:

- Tập đỉnh $V' = V$
- Mỗi cạnh $e(u, v) \in E$ có giá trị luồng là $f[u, v]$ và sức chứa $c[u, v]$ tương ứng với hai cạnh trong E':
  - $e'(u, v)$ (cạnh xuôi) có $f'[u, v] = f[u, v]$, $c'[u, v] = c[u, v]$
  - $e'(v, u)$ (cạnh ngược) có $f'[v, u] = -f[u, v]$ và $c'[v, u] = 0$

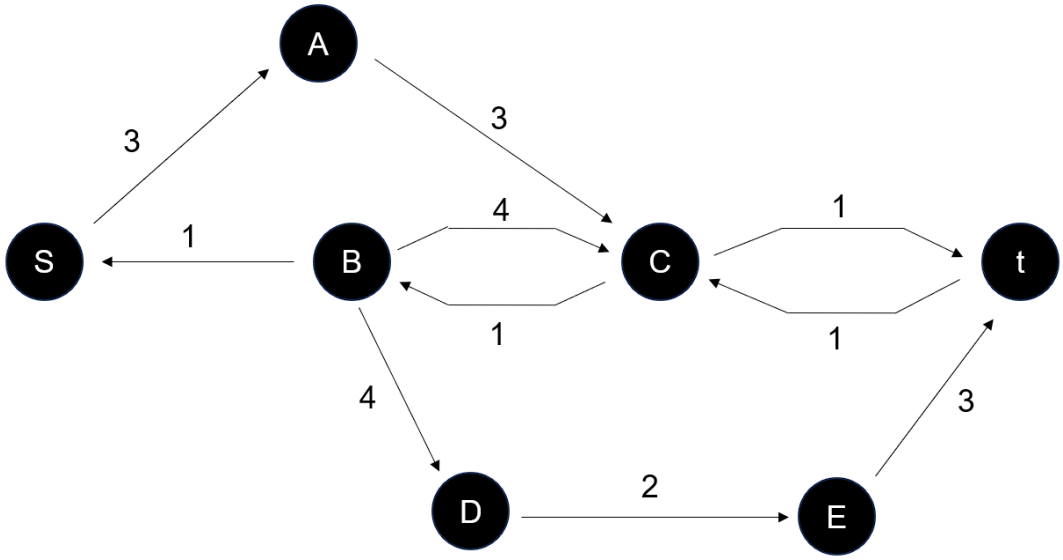
Dễ thấy tập cạnh xuôi trên G' chính là tập cạnh của G.



Hình 17: Ví dụ về một đồ thị G

3.6.2 Đường tăng luồng - Augmenting Path

Đường tăng luồng là một đường đi đơn từ đỉnh phát **s** (source) đến đỉnh thu **t** (sink) trong mạng thặng dư G' mà kênh trên đường đi chưa bị bão hòa ($f'[u, v] < c'[u, v]$. Một kênh $e'(u, v)$ được gọi là bão hòa nếu $f'(u, v) = c'(u, v)$)



Hình 18: Residual network của đồ thị G

Mô hình (10) về cơ bản là một mô hình quy hoạch nguyên có chứa 2 biến quyết định là $\mathbf{X} = \{x_{ij}\}_{(i,j)\in A}$ và $\mathbf{Y} = \{y_{ij}^s(t)\}_{(i,j)\in A, t\in\{0,1,\ldots,T\}, s=\overline{1,S}}$. Trong mô hình này, ràng buộc ghép nối là một ràng buộc phức tạp, dẫn đến việc mô hình không thể được giải trong thời gian đa thức. Do đó, ta sẽ nới lỏng ràng buộc ghép nối thành hàm mục tiêu thông qua phương pháp Lagrangian Relaxation. Sau đây, một cách tiếp cận kép dựa vào phương pháp trên sẽ được cung cấp nhằm phân tích bài toán ban đầu thành hai bài toán con dễ giải hơn, và giá trị mục tiêu là giới hạn dưới của giá trị tối ưu của mô hình (10) . Cụ thể, mô hình ban đầu sẽ được phân tách thành bài toán tìm dòng có chi phí tối thiểu (min-cost flow problem) tiêu chuẩn và bài toán phụ thuộc vào thời gian. Các bài toán này có thể được giải một cách hiệu quả bằng các thuật toán đúng đắn. Trong bài toán này, ta sẽ sử dụng thuật toán tìm con đường liên tục ngắn nhất với chi phí nhỏ nhất (Successive shortest path algorithm for min-cost flow problem). Các bước thực hiện gồm có:

Bước 1: Lấy biến x đại diện cho 1 luồng (flow) khả thi giữa bất kỳ điểm đầu-điểm cuối nào, và nó có chi phí nhỏ nhất trong số các luồng khả thi có cùng giá trị luồng (flow value).

Bước 2: Thuật toán sẽ kết thúc khi $x = v$, hoặc không còn con đường có chi phí tối thiểu nào trong mạng thặng dư (Residual Network), Ngược lại, con đường ngắn nhất có giá trị luồng lớn nhất sẽ được tính toán bằng thuật toán label correcting algorithm, sau đó chuyển đến bước 3. Trong đó $A(x), C(x), U(x)$ được định nghĩa:

$$A(x) = \{(i,j)|(i,j) \in A, x_{ij} < u_{ij}\} \cup \{(j,i)|(j,i) \in A, x_{ij} > 0\} \tag{12}$$

$$C(x) = \begin{cases} c_{ij}, (i,j) \in A, x_{ij} < u_{ij} \\ -c_{ji}, (j,i) \in A, x_{ji} > 0 \end{cases} \tag{13}$$

$$U_{ij}(x) = \begin{cases} u_{ij}, (i,j) \in A, x_{ij} < u_{ij} \\ x_{ji}, (j,i) \in A, x_{ji} > 0 \end{cases} \tag{14}$$

Bước 3: Tăng giá trị luồng dọc theo tuyến đường có chi phí tối thiểu, nếu giá trị của luồng được tăng không vượt quá v, quay lại bước 2.

3.6.3 Mô hình suy biến

Nhận thấy rằng ràng buộc ghép nối là một ràng buộc phức tạp, nên ta sẽ thêm vào nhân tử Lagrange $\alpha_{ij}^s(t), (i,j) \in A, s = 1,2,\ldots,, t \leq \tilde{T}$(!!!!!) cho ràng buộc này:

$$\sum_{s=1}^{S} \sum_{t \leq \tilde{T}} \sum_{(i,j) \in A} \alpha_{ij}^s(t)(y_{ij}^s(t) - x_{ij}) \tag{15}$$

Sau khi nới lỏng hàm mục tiêu (15), mô hình (10) được nới lỏng:

$$\begin{cases} \min \sum\limits_{(i,j)\in A} p_{ij} \cdot x_{ij} + \sum\limits_{s=1}^{S} \left(\mu_s \cdot \sum\limits_{(i,j)\in A_s} c_{ij}^s(t) \cdot y_{ij}^s(t) \right) + \\ \sum_{s=1}^{S} \sum_{t\leq\tilde{T}} \sum_{(i,j)\in A} \alpha_{ij}^s(t)(y_{ij}^s(t) - x_{ij}), \quad \text{subject to} \\ \sum\limits_{(i,j)\in A} x_{ij} - \sum\limits_{(j,i)\in A} x_{ji} = d_i, \forall i \in V \\ 0 \leq x_{ij} \leq u_{ij}, \forall (i,j) \in A \\ \sum\limits_{(i_t,j_{t'})\in A_s} y_{ij}^s(t) - \sum\limits_{(j_{t'},i_t)\in A_s} y_{ij}^s(t') = d_i^s(t), \forall i \in V, t \in \{0,1,\ldots,T\}, s = \overline{1,S} \\ 0 \leq y_{ij}^s(t) \leq u_{ij}^s(t), \forall (i,j) \in A, t \in \{0,1,\ldots,T\}, s = \overline{1,S} \end{cases} \tag{16}$$

Mô hình (16) được suy biến thành 2 bài toàn nhỏ hơn:
SubProblem 1: Min-cost Flow Problem

$$\begin{cases} \min SP1(\alpha) = \sum_{(i,j)\in A} \left(p_{ij} - \sum_{s=1}^{S} \sum_{t\leq\tilde{T}} \alpha_{ij}^s(t) \right) \cdot x_{ij}, \quad \text{subject to} \\ \sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ji} = d_i, \forall i \in V \\ 0 \leq x_{ij} \leq u_{ij}, \forall (i,j) \in A \end{cases} \tag{17}$$

SubProblem 2: Time-Dependent Min-cost Flow Problem
subProblem 2 của mô hình nới lỏng 13 có chứa biến quyết định $\mathbf{Y}$, giá trị tối ưu $Z_{SP2}^*(\alpha)$ như sau:

$$\begin{cases} \min SP2(\alpha) = \sum_{s=1}^{S} \sum_{(i,j)\in A} \left(\sum_{t\in[0,1,\ldots,T]} \mu_s \cdot c_{ij}^s(t) + \sum_{t\leq\tilde{T}} \alpha_{ij}^s(t) \right) y_{ij}^s(t), \quad \text{subject to} \\ \sum_{(i_t,j_{t'})\in A_s} y_{ij}^s(t) - \sum_{(j_{t'},i_t)\in A_s} y_{ij}^s(t') = d_i^s(t), \forall i \in V, t \in [0,1,\ldots,T], s = \overline{1,S} \\ 0 \leq y_{ij}^s(t) \leq u_{ij}^s(t), \forall (i,j) \in A, t \in [0,1,\ldots,T], s = \overline{1,S} \end{cases} \tag{18}$$

SubProblem 2 còn có thể suy biến thành tổng của S bài toán con, mỗi bài toán con là một min-cost flow problem phụ thuộc thời gian:

$$\begin{cases} \min SP2(\alpha, s) = \sum_{(i,j)\in A} \left(\sum_{t\in[0,1,\ldots,T]} \mu_s \cdot c_{ij}^s(t) + \sum_{t\leq\tilde{T}} \alpha_{ij}^s(t) \right) y_{ij}^s(t), \quad \text{subject to} \\ \sum_{(i_t,j_{t'})\in A_s} y_{ij}^s(t) - \sum_{(j_{t'},i_t)\in A_s} y_{ij}^s(t') = d_i^s(t), \forall i \in V, t \in [0,1,\ldots,T] \\ 0 \leq y_{ij}^s(t) \leq u_{ij}^s(t), \forall (i,j) \in A, t \in [0,1,\ldots,T] \end{cases} \tag{19}$$

Vì khoảng thời gian T được chia thành hai giai đoạn thời gian, chi phí tổng quát được định nghĩa là một hàm từng đoạn:

$$g_{ij}^s(t) = \begin{cases} \mu_s \cdot c_{ij}^s(t) + \alpha_{ij}^s(t), & t \leq \tilde{T} \\ \mu_s \cdot c_{ij}^s(t), & \tilde{T} < t \leq T \end{cases} \tag{20}$$

Bài toán con (20) là một bài toán Min-cost flow phụ thuộc vào thời gian và do đó thuật toán 1 nên được sửa đổi ở bước 2. Đầu tiên, tham số *A(y(t))*, *C(y(t))*, và *U(y(t))* trong mạng thặng dư *N(y(t))* được định nghĩa như sau:

$$\begin{aligned} A_s(y(t)) &= (i_t, j_{t'}) | (i_t, j_{t'}) \in A_s, y_{ij}^s < u_{ij}^s \vee (j_{t'}, i_t) | (j_{t'}, i_t) \in A_s, y_{ij}^s > 0, s = 1, 2, ..., S \\ c_{ij}^s(y(t)) &= \begin{cases} c_{ij}^s(t), (i_t, j_{t'}) \in A_s, y_{ij}^s(t) < u_{ij}^s(t), t \in 0, 1, ..., T \\ -c_{ji}^s(t'), (j_{t'}, i_t) \in A_s, \forall t' \in 0, 1, ..., T | y_{ji}^s(t') > 0, s = 1, 2, .., S \\ T, (j_{t'}, i_t) \in A_s, \forall t' \in 0, 1, ..., T | y_{ji}^s(t' = 0 \end{cases} \\ u_{ij}^s(y(t)) &= \begin{cases} u_{ij}^s(t) - y_{ij}^s(t), (i_t, j_{t'}) \in A_s, y_{ij}^s(t) < u_{ij}^s(t), t \in 1, 2, ..., T \\ y_{ji}^s(t), (j_{t'}, i_t) \in A_s, \forall t' \in 0, 1, ..., T | y_{ji}^s(t') > 0, s = 1, 2, ..., S \\ 0, (j_{t'}, i_t) \in A_s, \forall t' \in 0, 1, ..., T | y_{ji}^s(t') = 0 \end{cases} \end{aligned} \tag{21}$$

Tiếp theo, thuật toán label correcting (Ziliaskopoulos & Mahmassani, 1992) sẽ được áp dụng để tìm con đường phụ thuộc thời gian có chi phí nhỏ nhất trong mạng thặng dư.

Bằng cách giải bài toán con (17) và (18) với nghiệm nới lỏng $\mathbf{X}$ và $\mathbf{Y}$, giá trị mục tiêu tối ưu Z_{LR}^* của mô hình (16) với một tập các vector nhân Lagrangian có thể được biểu diễn như sau:

$$Z_{LR}^*(\alpha) = Z_{SP1}^*(\alpha) + Z_{SP2}^*(\alpha) \tag{22}$$

Hiển nhiên, giá trị mục tiêu tối ưu cho mô hình nới lỏng (16) là cận dưới của giá trị mục tiêu tối ưu của mô hình gốc (10). Để có được đáp án tốt, ta cần có cận dưới gần với giá trị mục tiêu tối ưu của mô hình gốc. Nghĩa là ta cần có cận dưới lớn nhất có thể có, và biểu thức được cho như sau:

$$Z_{LD}(\alpha^*) = max_{a \geq 0} Z_{LR}(\alpha) \tag{23}$$

Tài liệu

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

[2] GAMSPy documentation

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

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