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

MATHEMATICAL MODELING (CO2011)

Assignment

Stochastic Programming and Applications

Advisor: Nguyễn Tiến Thịnh
Students: Lê Võ Đăng Khoa - 2211606
Nguyễn Trần Bảo Khoa - 2211610
Nguyễn Đăng Khoa - 2211620
Nguyễn Tuấn Kiệt - 2211765
Nguyễn Văn Sơn - 2212949
Nguyễn Trần Minh Tâm - 2213035

HO CHI MINH CITY, NOVEMBER 2023

MỤC LỤC

1 Danh sách thành viên **2**

2 Kiến thức chuẩn bị **2**

2.1 Các khái niệm . . . 2

2.2 Lập trình tuyến tính ngẫu nhiên một giai đoạn - Không truy đòi (1-SLP) . . . 3

2.2.1 Phương pháp 1: sử dụng Ràng buộc cơ hội và Rủi ro chấp nhận được . . . 4

2.2.2 Phương pháp 2: dùng cho ràng buộc ngẫu nhiên $T(\alpha)\,x \leq h(\alpha)$ . . . 4

2.3 Lập trình ngẫu nhiên chung (GSP) với RECOURSE . . . 5

2.4 Lập trình tuyến tính ngẫu nhiên hai giai đoạn (2-SLP) . . . 6

2.4.1 Mô hình truy đòi SLP hai giai đoạn - (dạng đơn giản) . . . 6

2.4.2 Mô hình truy đòi SLP hai giai đoạn - (dạng chính tắc) . . . 7

3 To PROBLEM 1 [produce n products satisfying production $\geq$ demand] **8**

3.1 Phân tích, lời giải . . . 8

3.1.1 Phân tích . . . 8

3.1.2 Lời giải . . . 11

3.2 GAMSPY và Code minh họa . . . 13

4 To the SLP-EPDR: Algorithmic Solutions **16**

4.1 Phân tích . . . 16

4.1.1 Chương trình tuyến tính ngẫu nhiên để lập kế hoạch sơ tán trong ứng phó thiên tai (SLP-EPDR) . . . 16

4.1.2 Phân tích giải thuật 1 - Successive Shortest Path min cost flow(SSP) . . . 23

4.2 Code minh họa giải thuật Successive Shortest Path for min cost flow . . . 26

4.3 So sánh với giải thuật Edmonds-Karp min-cost và đánh giá độ hiệu quả của thuật toán . . . 28

4.3.1 Giải thuật Edmonds-Karp min-cost . . . 28

4.3.2 Đánh giá độ hiệu quả của thuật toán dựa vào thời gian . . . 31

1 Danh sách thành viên

No.	Fullname	Student ID	Problems	Percentage of work
1	Lê Võ Đăng Khoa	2211606	- Soạn báo cáo - Viết code Latex - Hỗ trợ code	16%
2	Nguyễn Trần Bảo Khoa	2211610	- Tìm hiểu lý thuyết - Phân tích câu hỏi 1 & 2 - Tìm ví dụ	16%
3	Nguyễn Đăng Khoa	2211620	- Tìm hiểu lý thuyết - Phân tích câu hỏi 1 & 2 - Tìm ví dụ	16%
4	Nguyễn Tuấn Kiệt	2211765	- Soạn báo cáo - Viết code Latex - Hỗ trợ code	16%
5	Nguyễn Văn Sơn	2212949	- Hỗ trợ phân tích đề - Hỗ trợ code - Tìm kiếm và thu thập tài liệu	16%
6	Nguyễn Trần Minh Tâm	2213035	- Code chính - Phân tích câu hỏi 1 & 2 - Tìm kiếm và thu thập tài liệu	20%

2 Kiến thức chuẩn bị

2.1 Các khái niệm

Chúng ta có một số vấn đề tối ưu hóa, chẳng hạn như quy hoạch tuyến tính , quy hoạch số nguyên (được học trong môn Mô hình hóa toán học của CSE). Trong đó tồn tại lý thuyết nâng cao cho mô hình xác định và các phương pháp số hiệu quả được tìm thấy. Và trong bài tập lớn này sẽ giới thiệu về các khái niệm toán học cho các vấn đề tối ưu hóa mô hình liên quan đến tính ngẫu nhiên - không chắc chắn và những vấn đề như vậy được gọi là Stochastic optimization problems hay được gọi ngắn gọn là **Stochastic Programming**(**SP**).

- **Tối ưu hóa toán học***(Mathematical Optimization)* là đưa ra quyết định, chủ yếu sử dụng cái phương pháp toán học.
- **Lập trình ngẫu nhiên** *(Stochastic Programming (SP))* là việc đưa ra quyết định trong điều kiện không chắc.

  Xem nó dưới dạng lập trình toán học tối ưu **(Mathematical Programming Optimization)** với các tham số ngẫy nhiên.

- **Chương trình tuyến tính ngẫu nhiên** *(Stochastic linear programs)* là chương trình tuyến tính, tức là hàm mục tiêu của nó là tuyến tính trong một số dữ liệu của bài toán có thể được coi là không chắc chắn.
- **Chương trình truy đòi** *(Recourse programs)* là những chương trình trong đó một số quyết định hoặc hành động truy đòi có thể được thực hiện sau khi phát hiện ra sự không chắc chắn.

Định nghĩa 1 (Chương trình tuyến tính (Linear program LP) với tham số ngẫu nhiên - SLP Stochastic linear program (SLP) là:

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

2.2 Lập trình tuyến tính ngẫu nhiên một giai đoạn - Không truy đòi (1-SLP)

Định nghĩa 2 (SLP một giai đoạn (Không truy đòi) 1-SLP). Xem xét chương trình LP(α) sau đât được tham số bởi một vector ngẫu nhiên α:

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

$$\text{s. t. } Ax = b, \text{ (ràng buộc chắc chắn)}$$

$$\text{and } Tx \geq h \text{ (ràng buộc ngẫu nhiên)}$$

với những giả định:

1. Ma trận $T = T(a)$ và vector $h = h(a)$ biểu diễn sự không chắc chắn thông qua các ràng buộc ngẫu nhiên
$$T(a)\, x \geq h(a) \Leftrightarrow a_1\, x_1 + \ldots + a_n\, x_n \geq h(a)$$
2. Giá trị (T, h) không xác định: Chúng không được biết cho đến khi một mô hình xuất hiện, $h(a)$ chỉ phụ thuộc a_j ngẫu nhiên.
3. Sự không chắc chắn được biểu diễn bằng phân bố xác suất của tham số ngẫu nhiên $(a_j) = a$ nên lập trình tuyến tính xác định (deterministic LP) là trường hợp suy biến của Lập trình tuyến tính ngẫu nhiên (Stochastic LP) khi a_j không đổi.

- Chúng ta xử lý các vấn đề quyết định khi vector $X = (x_1, x_2, \ldots, x_n) \in X$ của các biến quyết định phải được thực hiện trước khi biết được thực hiện của vector tham số $a \in \Omega$.
- Thường chúng ta đặt giới hạn dưới và trên cho x thông qua một miền $X = \{x \in \mathbb{R}^n | l \leq x \leq u\}$.

CÁCH TIẾP CẬN: Trong Lập trình Ngẫu nhiên (Stochastic Programming), chúng ta sử dụng một số giả định và sự thật.

Giả định cơ bản: Chúng ta biết phân phối xác suất (liên hợp) của dữ liệu. Do đó, phương pháp đầu tiên cung cấp Ràng buộc LP Xác suất (Cơ hội).

Phân tích Kịch bản: không hoàn hảo, nhưng hữu ích, là phương pháp thứ hai. Phương pháp kịch bản giả định rằng có một số hữu hạn các quyết định mà tự nhiên có thể thực hiện (kết quả của sự ngẫu nhiên). Mỗi quyết định có thể này được gọi là một kịch bản.

2.2.1 Phương pháp 1: sử dụng Ràng buộc cơ hội và Rủi ro chấp nhận được

- Chúng ta có thể thay thế $T\,x \geq h$ bởi những ràng buộc ngẫu nhiên $P[Tx \geq h] \geq p^2$ đối với một số mức độ tin cậy quy định $p \in (0.5, 1)$, (được xác định bởi chủ sở hữu vấn đề). Lập trình tuyến tính ngẫu nhiên (LP) trong định nghĩa (2) ở trên với các tham số ngẫu nhiên $\alpha = [\alpha_1, \alpha_2, ...]$ thì được gọi là ràng buộc xác xuất LP, hoặc 1-SLP.
- Rủi ro sau đó được xem xét một cách rõ ràng, được định nghĩa là một

$$\text{rủi ro chấp nhận được } r_a := P\left(\text{Không } [T\,x \geq h]\right) = P\left[T\,x \leq h_1\right] \leq 1 - p$$

  thì $(1 - p)$ là rủi ro chấp nhận được tối đa.
  Ràng buộc cơ hội $T\,x \leq h$ ngụ ý rằng
  rủi ro chấp nhận được r_a nhỏ hơn một số tối đa đã chỉ định $1 - p \in (0, 1)$.

Định nghĩa 3: LP ngẫu nhiên hoặc 1-SLP với Ràng buộc Xác suất được định nghĩa bởi một hàm mục tiêu hệ số ngẫu nhiên và ràng buộc cơ hội.

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

$$\begin{cases} A\mathbf{x} = \mathbf{b}\,(\mathbf{x} = (x_1, x_2, ..., x_n) \text{ là những biến quyết định}) \\ P[T\,\mathbf{x} = \alpha \cdot \mathbf{x} \leq h] \leq (1 - p) \quad (0 < p < 1) \end{cases}$$

NOTE: Chúng ta sử dụng vector tham số $\alpha = [\alpha_1, \alpha_2, \ldots]$ nói chung, và ký hiệu $\omega = [\omega_1, \omega_2, \ldots, \omega_S]$ cụ thể cho các trạng thái s được gọi là các kịch bản. Chúng ta xử lý mỗi kịch bản $\omega \in \omega$ có thể bằng một sự kết hợp có thể của nhiều tham số ngẫu nhiên α_i cùng một lúc trong một SP.

2.2.2 Phương pháp 2: dùng cho ràng buộc ngẫu nhiên $T(\alpha)\,x \leq h(\alpha)$

Sử dụng Phân tích Kịch bản của $T(\alpha)x \leq h(\alpha)$

Đối với mỗi kịch bản $(T^s; h^s)$, $s = 1, ..., S$, giải quyết

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

Loại chương trình này nhắm đến một mục tiêu tuyến tính cụ thể trong khi tính toán cho một hàm xác suất liên quan đến các kịch bản khác nhau. Do đó, chúng ta tìm ra một giải pháp tổng thể bằng cách xem xét các giải pháp cho từng kịch bản x_s $(s = 1, ..., S)$.

Ưu điểm: Mỗi vấn đề kịch bản là một LP. / so với / Nhược điểm: phân phối rời rạc $->$ mô hình LP số nguyên kết hợp. (Nói chung: có thể là một mô hình không lồi).

2.3 Lập trình ngẫu nhiên chung (GSP) với RECOURSE

Định nghĩa 4 (Chương trình ngẫu nhiên trong hai giai đoạn (vấn đề chung 2-SP)). Chương trình ngẫu nhiên hai giai đoạn (2-SP) mở rộng từ *Định nghĩa 2* có dạng:

$$\text{2-SP: } \min_x \text{ g(x) với g(x)} = g(x) = f(x) + E_\omega[v(x,\omega)]$$

khi đó,

$$x = (x_1, x_2, ..., x_n) \text{ là các biến quyết định giai đoạn đầu.}$$

f(x) có thể là tuyến tính hoặc không, là một phần của hàm mục tiêu g(x)
$*$*Giá trị trung bình* $Q(x) := E_\omega[v(x,\omega)]$ *của hàm số*

$$v : R^n \times R^s \to R$$

dựa trên ảnh hưởng của các kịch bản ω.*Q(x) là giá trị tối ưu của một bài toán bậc hai nào đó*

$$\min_{y \in R_p} \mid \textit{subject to } T.x + W.y = h$$

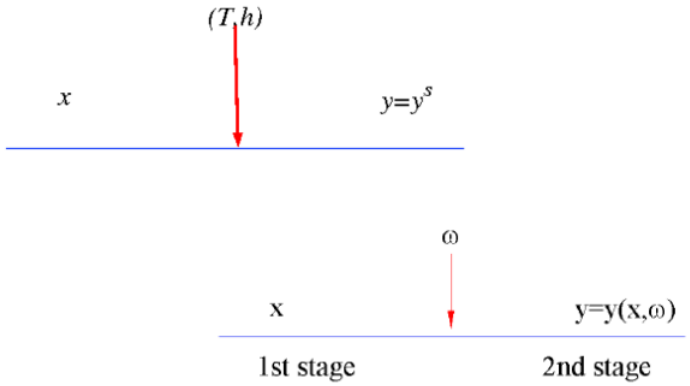
Các vectơ $\alpha(\omega)$ và $y = y(\omega)$ được đặt tên là các biến quyết định hiệu chỉnh, điều chỉnh hoặc truy đòi, chỉ được biết sau thí nghiệm e
Tóm lại, chúng ta Giảm thiểu tổng chi phí dự kiến $\mathbf{g(x) = f(x) + Q(x)}$ trong khi thỏa mãn:

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

Ở đây W được gọi là ma trận m x p và chúng ta bắt đầu với những trường hợp đơn giản m = 1, q là vecto chi phí đơn vị, có cùng thứ nguyên với y và y = y(ω) $\in R^p$

GIẢI THÍCH các vấn đề về mô hình truy đòi

- Mục tiêu lớn g(x) là xây dựng bởi f(x) và Q(x). Ở đây y là biến quyết định của giai đoạn LP, giá trị y phụ thuộc vào việc thực hiện (T,h) := (T(ω), h(ω)). Biến truy đòi y(ω) $\sim$ hành động khắc phục ví dụ sử dụng các nguồn lực sản xuất thay thế(làm thêm giờ,...)
- Đo lường rủi ro định lượng: độ lệch h(ω) - T(ω).x có liên quan.
- Ở đây RỦI RO được mô tả bằng chi phí truy đòi dự kiến Q(x) của quyết định x.
- Thực tế cần phải cải cách mô hình: q và W đến từ đâu?



Hình 1: Chế độ xem tiêu chuẩn của chương trình ngẫu nhiên hai giai đoạn Được phép của Maarten van der Vlerk, Univ. của Groningen, NL

2.4 Lập trình tuyến tính ngẫu nhiên hai giai đoạn (2-SLP)

2.4.1 Mô hình truy đòi SLP hai giai đoạn - (dạng đơn giản)

Định nghĩa 5 (Two-stage Stochastic LP với truy đòi: 2-SLPWR). Chương trình tuyến tính ngẫu nhiên hai giai đoạn có truy đòi (2-SLPWR) hoặc chính xác với việc trừng phạt hành động khắc phục thường được mô tả

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

hoặc tổng quát

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

với v(x, ω) := q. y

Ràng buộc:

$$\text{A.x} = \text{b} \qquad \text{ràng buộc giai đoạn 1(first-stage s.t)}$$
$$T(\omega).x + W.y(\omega) = h(\omega) \qquad \text{ràng buộc giai đoạn 2(second-stage s.t)}$$

hoặc rút gọn W.y = h(ω) - T(ω).x

♢ **Chương trình SLP chỉ định 2-SP ở trên cho mục tiêu. Hàm mục tiêu lớn g(x) có**
(1) xác định f(x)- là hàm tuyến tính, trong khi tính toán
(2) đối với hàm xác suất v(x, ω) liên quan đến các kịch bản khác nhau ω.

♢ **y = y(x, ω) $\in R_+^p$ được đặt tên là biến hành động truy đòi cho quyết định x và việc thực hiện ω.**

Hiệu chỉnh Phạt được thể hiện thông qua giá trị trung bình $Q(x) = E_\omega[v(x,\omega)]$
Để giải hệ bằng số, các phương pháp dựa trên vectơ ngẫu nhiên α có số hữu hạn các khả năng thực hiện, được gọi là kịch bản (scenarios).

Giá trị kỳ vọng của $Q(x)$ hiển nhiên với phân bố rời rạc của ω.
Giá trị kỳ vọng của $Q(x)$ hiển nhiên với phân bố rời rạc của ω. Vì vậy lấy $\Omega = \{\omega_k\}$ là một tập hợp hữu hạn có kích thước S (có một số hữu hạn các kịch bản $\omega_1, \ldots, \omega_S \in \Omega$, xác suất mỗi kịch bản là p_k.
Vì $y = y(x,\omega)$ nên kỳ vọng của $v(y) = v(x,\omega) := q \cdot y$ (một chi phí q cho tất cả y_k) là

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

Khi đó:

- p_k là mật độ của kịch bản ω_k, q là chi phí phạt một đơn vị
- chi phí phạt của việc sử dụng đơn vị y_k trong giai đoạn hiệu,phụ thuộc vào cả quyết định ở giai đoạn đầu và các kịch bản ngẫu nhiên ω

2.4.2 Mô hình truy đòi SLP hai giai đoạn - (dạng chính tắc)

Bây giờ chúng ta mô tả hoàn toàn hệ thống trên trong trường hợp tuyến tính.
Định nghĩa 6 (Chương trình tuyến tính ngẫu nhiên có Hành động Đền bù (2-SLPWR)). Chương trình tuyến tính ngẫu nhiên 2 giai đoạn với Hành động Đền bù có thể được sắp xếp như sau:

$$\begin{gathered}
\text{2-SLP: } \min_x \text{ g(x) với} \\
\text{g(x)} := c^T \cdot x + v(y) \\
\text{Dưới ràng buộc (s.t.) } Ax = b \text{ với } x \in X \subset \mathbb{R}^n,\ x \geq 0 \\
\text{v(z)} := \min_{y \in \mathbb{R}^p_+} q^T y \text{ subject to } W \cdot y = h(\omega) - T(\omega) \cdot x =: z
\end{gathered}$$

Trong đó $v(y) := v(x,\omega)$ là hàm giá trị giai đoạn thứ hai, và
$y = y(x,\omega) \in \mathbb{R}^p_+$ là một hành động đền bù cho quyết định x và hiện thực của ω.

1. Chi phí đền bù kỳ vọng của quyết định x là $Q(x) := \mathbb{E}_\omega[v(x,\omega)]$ theo Phương trình (5). [chính xác là chi phí kỳ vọng của đường lối đền bù $y(\alpha)$, cho mọi chính sách $x \in \mathbb{R}^n$.] Do đó, tổng quát chúng ta tối thiểu hóa tổng chi phí kỳ vọng $\min_{x \in \mathbb{R}^n, y \in \mathbb{R}^p_+} c^T \cdot x + Q(x)$.

2. Chúng ta thiết kế các biến quyết định thứ hai $y(\omega)$ để chúng ta có thể (điều chỉnh, sửa đổi hoặc) phản ứng với các ràng buộc gốc (4.2) một cách thông minh (hoặc tối ưu): chúng ta gọi nó là hành động đền bù!

$$x \longrightarrow T, h, \omega \longrightarrow y.$$

3. Giá trị tối ưu của Bài toán lập kế hoạch tuyến tính giai đoạn thứ hai là $v^* = v(y^*)$, với $y^* = y^*(x, \omega)$ là giải pháp tối ưu của nó, ở đây $y^* \in \mathbb{R}^p_+$. Tổng giá trị tối ưu là $c^T \cdot x^* + v(y^*)$.

3 To PROBLEM 1 [produce n products satisfying production $\geq$ demand]

Sử dụng mô hình 2-SLPWR được cung cấp trong Phương trình 7 và 8 khi $n = 8$ sản phẩm, số lượng kịch bản $S = 2$ với mật độ $p_s = \frac{1}{2}$, số lượng linh kiện cần đặt hàng trước khi sản xuất $m = 5$, chúng ta ngẫu nhiên mô phỏng vector dữ liệu $\mathbf{b}$, $\mathbf{l}$, $\mathbf{q}$, $\mathbf{s}$ và ma trận $\mathbf{A}$ kích thước $n \times m$.

Chúng ta cũng giả định rằng vector yêu cầu ngẫu nhiên $\omega = \mathbf{D} = (D_1, D_2, \ldots, D_n)$ trong đó mỗi ω_i với mật độ p_i tuân theo phân phối nhị thức $\text{Bin}(10, \frac{1}{2})$.

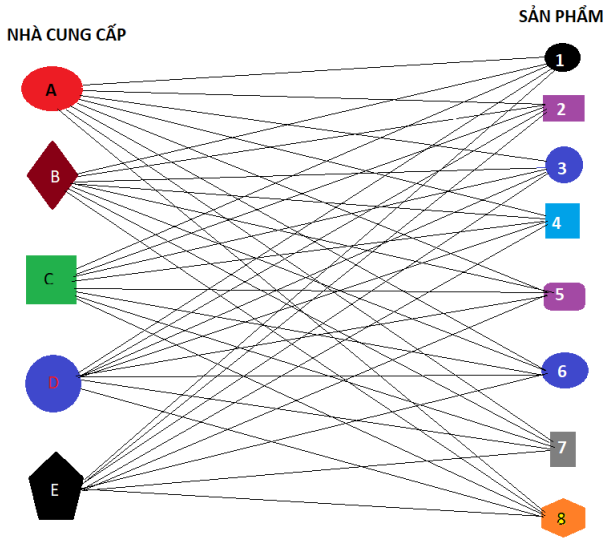
YÊU CẦU: Xây dựng các mô hình số học của Phương trình 7 và 8 trong yêu cầu bài tập lớn 2023 với dữ liệu mô phỏng. Tìm giải pháp tối ưu $\mathbf{x}$, $\mathbf{y} \in \mathbb{R}^m$ và $\mathbf{z} \in \mathbb{R}^n$ thông qua phần mềm thích hợp (như GAMSPy).

3.1 Phân tích, lời giải

3.1.1 Phân tích

a) Mô hình 2-SLPWR

Xét một nhà máy sản xuất n = 8 sản phẩm, số lượng nguyên liệu cần đặt hàng trước khi sản xuất là m = 5. Hình (2) thể hiện mỗi quan hệ của nhà máy với m nhà cung cấp



Hình 2: Mối quan hệ giữa nhà cung cấp và nhà máy sản xuất

Một đơn vị sản phẩm i cần $a_{ij} \geq 0$ đơn vị của nhà cung cấp j, với $i = 1,..., 8$ và $j = 1,..., 5$. Nhu cầu về sản phẩm được mô hình hóa dưới dạng vectơ $\omega = D = (D_1, D_2,..., D_8)$.

– **Giai đoạn 2 (second stage)**

Đối với giá trị quan sát (a hiện thực) $d = (d_1, d_2,..., d_8)$ của vectơ nhu cầu ngẫu nhiên D ở trên, chúng ta có thể tìm kiếm kế hoạch sản xuất tốt nhất bằng cách giải chương trình tuyến tính ngẫu nhiên (SLP).

- Biến quyết định $z = (z_1, z_2,..., z_8)$ - số lượng đơn vị sản phẩm được sản xuất.
- Biến quyết định $y = (y_1, y_2,..., y_5)$ - số lượng nguyên liệu còn lại trong kho.

Vì vậy ta có:

$$\textbf{\textit{LSP:}} \min_{z,y} Z = \sum_{i=1}^{n}(l_i - q_i) - \sum_{j=1}^{m} s_j.y_j \tag{1}$$

Hay với n = 8, m = 5, ta có

$$\textbf{\textit{LSP:}} \min_{z,y} Z = \sum_{i=1}^{8}(l_i - q_i) - \sum_{j=1}^{5} s_j.y_j$$

Trong đó:

- s_j là giá bán nguyên liệu còn thừa, điều kiệu $s_j < c_j$, với c_j là giá nhập nguyên liệu j ban đầu.
- x_j, $j = 1, ..., 5$ là số lượng nguyên liệu j cần đặt trước lúc sản xuất.
- l_i, $i = 1,..., 8$ là chi phí cần để sản xuất 1 đơn vị sản phẩm i.
- s_i, $i = 1,..., 8$ là đơn giá bán 1 đơn vị sản phẩm i và $q_i > l_i$.

$$\textit{Ràng buộc:} \begin{cases} y_j = x_j - \sum_{i=1}^{n} a_{ij}.\, z_i;\ \ j = 1,..., m \\ \\ 0 \leq z \leq d,\ y_j \geq 0;\ \ j = 1, ..., m \end{cases}$$

Toàn bộ mô hình của giai đoạn 2 có thể được biểu diễn với biểu diễn tương đương như sau

$$\textbf{MODEL 2-S} = \begin{cases} \min\limits_{x,y} Z = b^T.\, z - s^T\, y \\ \text{với } b = (b_i := l_i - q_i) \text{ là hệ số chi phí} \\ y = x - A^T.\, z;\ \text{ với } A = [a_{ij}] \text{ là ma trận cỡ n} \times \text{m} \\ 0 \leq z \leq d,\quad y_j \geq 0; \end{cases} \tag{2}$$

Với $n = 8$, $m = 5$ và từ mô hình (2) và những nội dung trên, ta thấy các vectơ x, y phụ thuộc vào việc thực hiện d của nhu cầu ngẫu nhiên $\omega = D$ cũng như quyết định x ở giai đoạn 1 (first-stage) $x = (x_1, x_2,..., x_5)$.

– **Giai đoạn 1 (first stage)**

Toàn bộ mô hình **2-SLPWR** dựa trên một qui tắc sản xuất phổ biến đó là sản xuất ≥ nhu

cầu.

Theo cách tiếp cận dựa trên phân phối, ta đặt $Q(x) := E[Z(z,y) = E_\omega[x,\omega$ biểu thị giá trị tối ưu của biểu thức (1)

Kí hiệu $c = (c_1, c_2, ..., c_5)$ là chi phí đặt hàng của từng nguyên liệu thứ j (trước khi biết nhu cầu)

Các đại lượng x_j được xác định từ bài toán tối ưu sau:

$$min\ g(x,y,z) = c^T.x\ +\ E[Z(z,y)] \tag{3}$$

Phần đầu tiên của hàm mục tiêu biểu thị chi phí đặt hàng trước và x. Ngược lại, phần thứ hai thể hiện chi phí dự kiến của kế hoạch sản xuất tối ưu (2), được tính bằng số lượng đặt hàng z cập nhật, đã sử dụng nhu cầu ngẫu nhiên $D = d$ với mật độ của chúng.

Dựa vào yêu cầu bài là số lượng kịch bản S = 2 với mật độ ps $=\frac{1}{2}$ nên có 2 kịch bản s_1 và s_2 xảy ra với xác suất $p_1 = \frac{1}{2}$ và $p_2 = \frac{1}{2}$, nên ta có

$$Q(x) = E_\omega[Z] = \sum_{k=1}^{2} p_k[(l_i - q_i)^T.z - s^T.y]$$

bài toán hai giai đoạn (2) - (3) có thể được viết dưới dạng một bài toán quy hoạch tuyến tính dưới dạng mô hình sau:

$$\begin{aligned}
\text{Min} \quad & c^T x + \sum_{k=1}^{K} p_k[(l-q)^T z^k - s^T y^k] \\
\text{s.t.} \quad & y^k = x - A^T z^k, \quad k = 1, ..., K, \\
& 0 \le z^k \le d^k, \quad y^k \ge 0, \quad k = 1, ..., K, \\
& x \ge 0,
\end{aligned}$$

Thay số với K=2 , n = 8 sản phẩm và m= 5 nhà cung cấp

b) Vectơ yêu cầu ngẫu nhiên $\omega = D$

Chúng ta cũng giả định rằng vector yêu cầu ngẫu nhiên $\omega = D = (D_1, D_2, \ldots, D_n)$ $(n = 8)$ trong đó mỗi ω với mật độ p_i tuân theo phân phối nhị thức Bin$(10, \frac{1}{2})$

- Phân phối nhị thức ***Bin(n, p)***
  - Phân phối nhị thức là một phân phối xác suất mô tả số lần thành công trong n thử nghiệm Bernoulli độc lập và giống nhau, với xác suất thành công p trong mỗi lần thử nghiệm. Điều này thường được ký hiệu là $X \sim Bin(n,p)$
  - Trong mô hình này, n là số lần thử nghiệm và p là xác suất thành công trong mỗi lần thử nghiệm. Công thức tổng quát cho phân phối nhị thức là:

$$P(X = k) = \binom{n}{k}.p^k.(1-p)^{n-k}$$

Ở đây :

◇ X là biến ngẫu nhiên biểu diễn số lần thành công

◇ k là số lần thành công đạt được

◇ $\binom{n}{k}$ là số cách chọn k thành công từ n lần thử nghiệm.

- Phân phối nhị thức thường được sử dụng để mô hình hóa các tình huống trong đó chúng ta quan sát số lần thành công của một sự kiện trong một chuỗi các thử nghiệm Bernoulli độc lập, như việc đếm số lần tung đồng xu đến mặt sấp trong n lần tung. Đây là một công cụ quan trọng trong thống kê và xác suất, đặc biệt khi nói đến mô hình hóa và dự đoán kết quả của các loạt thử nghiệm độc lập.

3.1.2 Lời giải

Dựa vào cơ sở lí thuyết của mô hình 2-SLPWR phân tích bên trên, ta có mô hình để giải bài toán lúc đầu như sau.

Hàm mục tiêu:

$$\begin{aligned} \text{Min} \quad & c^T x + \sum_{k=1}^{K} p_k[(l-q)^T z^k - s^T y^k] \\ \text{s.t.} \quad & y^k = x - A^T z^k, \quad k = 1, ..., K, \\ & 0 \le z^k \le d^k, \quad y^k \ge 0, \quad k = 1, ..., K, \\ & x \ge 0, \end{aligned}$$

trong đó $\begin{cases} c \text{ là giá mua} \\ l \text{ là giá sản xuất sản phẩm} \\ q \text{ là giá bán sản phẩm} \\ s \text{ là giá bán nguyên liệu tồn kho} \\ k \text{ là kịch bản} \\ A \text{ là ma trận quan hệ giữa nguyên liệu và sản phẩm} \end{cases}$,các biến quyết định $\begin{cases} x \text{ là số nguyên liệu mua ban đầu} \\ y \text{ là số sản nguyên liệu tồn kho} \\ z \text{ là số sản phẩm bán được} \end{cases}$

Giả sử ta có các vecto chứa các thông tin sau:

c =

$$\begin{bmatrix} 10 & 20 & 30 & 40 & 50 \end{bmatrix}$$

l =

$$\begin{bmatrix} 10 & 20 & 30 & 40 & 50 & 60 & 70 & 80 \end{bmatrix}$$

q =

$$\begin{bmatrix} 20 & 30 & 40 & 50 & 60 & 70 & 80 & 90 \end{bmatrix}$$

s =

$$\begin{bmatrix} 5 & 10 & 15 & 20 & 25 & 30 & 35 & 40 \end{bmatrix}$$

D =

$$\begin{bmatrix} 5 & 6 & 5 & 5 & 5 & 6 & 5 & 7 \\ 8 & 5 & 6 & 5 & 5 & 7 & 3 & 3 \end{bmatrix}$$

A =

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \end{bmatrix}$$

Từ các thông tin trên, ta có:

Hàm mục tiêu

$$\begin{aligned} 10x_1 + 20x_2 + 30x_3 + 40x_4 + 50x_5 + \frac{1}{2}\left(-5y_1^1 - 10y_2^1 - 15y_3^1 - 20y_4^1 - 25y_5^1 - 440\right) \\ + \frac{1}{2}\left(-5y_1^2 - 10y_2^2 - 15y_3^2 - 20y_4^2 - 25y_5^2 - 420\right) \end{aligned}$$

Ràng buộc

$$\begin{aligned} y_1^1 &= x_1^1 + 0x_2^1 + 0x_3^1 + 0x_4^1 + 0x_5^1 - 164 \\ y_2^1 &= 0x_1^1 + x_2^1 + 0x_3^1 + 0x_4^1 + 0x_5^1 - 208 \\ y_3^1 &= 0x_1^1 + 0x_2^1 + x_3^1 + 0x_4^1 + 0x_5^1 - 252 \\ y_4^1 &= 0x_1^1 + 0x_2^1 + 0x_3^1 + x_4^1 + 0x_5^1 - 296 \\ y_5^1 &= 0x_1^1 + 0x_2^1 + 0x_3^1 + 0x_4^1 + x_5^1 - 340 \\ y_1^2 &= x_1^2 + 0x_2^2 + 0x_3^2 + 0x_4^2 + 0x_5^2 - 142 \\ y_2^2 &= 0x_1^2 + x_2^2 + 0x_3^2 + 0x_4^2 + 0x_5^2 - 184 \\ y_3^2 &= 0x_1^2 + 0x_2^2 + x_3^2 + 0x_4^2 + 0x_5^2 - 226 \\ y_4^2 &= 0x_1^2 + 0x_2^2 + 0x_3^2 + x_4^2 + 0x_5^2 - 268 \\ y_5^2 &= 0x_1^2 + 0x_2^2 + 0x_3^2 + 0x_4^2 + x_5^2 - 310 \\ x, y &\geq 0 \end{aligned}$$

Ta tối ưu bài toán trên bằng phương pháp Simplex

• Step 1

B	Cb	P	x1	x2	x3	x4	x5	y11	y21	y31	y41	y51	y12	y22	y32	y42	y52	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	Q
			10	20	30	40	50	-2.5	-5	-7.5	-10	-12.5	-2.5	-5	-7.5	-10	-12.5	M	M	M	M	M	M	M	M	M	M	
x6	M	164	1	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	164
x7	M	208	0	1	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	∞
x8	M	252	0	0	1	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	∞
x9	M	296	0	0	0	1	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	∞
x10	M	340	0	0	0	0	1	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	∞
x11 ←	M	142	1	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	142
x12	M	184	0	1	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	0	0	0	∞
x13	M	226	0	0	1	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	0	0	∞
x14	M	268	0	0	0	1	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	0	∞
x15	M	310	0	0	0	0	1	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	∞
min		2390M	2M-10	2M-20	2M-30	2M-40	2M-50	-M+2.5	-M+5	-M+7.5	-M+10	-M+12.5	-M+2.5	-M+5	-M+7.5	-M+10	-M+12.5	0	0	0	0	0	0	0	0	0	0	

Hình 3: Simplex step 1

• Step 2

B	Cb	P	x1	x22	x3	x4	x5	y11	y21	y31	y41	y51	y12	y22	y32	y42	y52	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	Q
			10	20	30	40	50	-2.5	-5	-7.5	-10	-12.5	-2.5	-5	-7.5	-10	-12.5	M	M	M	M	M	M	M	M	M	M	
x6	M	22	0	0	0	0	0	-1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	-1	0	0	0	0	∞
x7	M	208	0	1	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	208
x8	M	252	0	0	1	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	∞
x9	M	296	0	0	0	1	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	∞
x10	M	340	0	0	0	0	1	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	∞
x_1	10	142	1	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	∞
x12 ←	M	184	0	1	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	0	0	0	184
x13	M	226	0	0	1	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	0	0	∞
x14	M	268	0	0	0	1	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	0	∞
x15	M	310	0	0	0	0	1	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	∞
min		2106M+1420	0	2M-20	2M-30	2M-40	2M-50	-M+2.5	-M+5	-M+7.5	-M+10	-M+12.5	M-7.5	-M+5	-M+7.5	-M+10	-M+12.5	0	0	0	0	0	-2M+10	0	0	0	0	

Hình 4: Simplex step 2

Và sau đó ta cứ tiếp tục tìm biến ra và biến vào và đến bước cuối cùng khi tìm được nghiệm tối ưu thì dừng (hoặc không tìm được) thì dừng. Dưới đây là bảng kết quả của bước cuối.

• Step 11

B	Cb	P	x1	x2	x3	x4	x5	y11	y21	y31	y41	y51	y12	y22	y32	y42	y52	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	Q
			10	20	30	40	50	-2.5	-5	-7.5	-10	-12.5	-2.5	-5	-7.5	-10	-12.5	M	M	M	M	M	M	M	M	M	M	
x12	-2.5	22	0	0	0	0	0	-1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	-1	0	0	0	0	
x22	-5	24	0	0	0	0	0	0	-1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	-1	0	0	0	
x32	-7.5	26	0	0	0	0	0	0	0	-1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	-1	0	0	
x42	-10	28	0	0	0	0	0	0	0	0	-1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	-1	0	
x52	-12.5	30	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	-1	
x_1	10	164	1	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	
x_2	20	208	0	1	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	
x_3	30	252	0	0	1	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	
x_4	40	296	0	0	0	1	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	
x_5	50	340	0	0	0	0	1	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	
min		41175	0	0	0	0	0	-5	-10	-15	-20	-25	0	0	0	0	0	-M+7.5	-M+15	-M+22.5	-M+30	-M+37.5	-M+2.5	-M+5	-M+7.5	-M+10	-M+12.5	

Hình 5: Simplex step 11

KẾT QUẢ TỐI ƯU CỦA BÀI TOÁN TRÊN

- Giá trị tối ưu chi phí nhỏ nhất $F = 40745$
- Số lượng nguyên liệu cần nhập hàng là: $X = (164, 208, 252, 296, 340)$
- Số lượng nguyên liệu tồn kho $Y = (0, 0, 0, 0, 0, 22, 24, 26, 28, 30)$

Vậy theo kịch bản 2, ta sẽ có giá trị tối ưu nhất.

3.2 GAMSPY và Code minh họa

GAMS là viết tắt của General Algebraic Modeling Language (Ngôn ngữ mô hình đại số tổng quát) và là một trong những ngôn ngữ mô hình hóa được sử dụng rộng rãi nhất., Ta có thể sử dụng GAMS + Python (GAMSPY) để giải quyết bài toán 2-SLPWR.

Dưới đây là ứng dụng Gamspy và code

1. **Khởi tạo dữ liệu**

- Tạo DataFrame df với các cột như "Nguyên liệu," "Giá dư," "Giá ban đầu," "Sản phẩm," "Chi phí sản xuất," "Giá bán," và "Nhu cầu."

```
# tạo dữ liệu cho tính toán va ghi vào file Excel
data = {"Nguyên liệu": ["nl1","nl2","nl3","nl4", "nl5"] + [None]*3}
df = pd.DataFrame(data)

a = np.random.randint(1, 100, size=5)
nl_du = [["nl1", int(a[0])], ["nl2", int(a[1])], ["nl3", int(a[2])], ["nl4", int(a[3])], ["nl5", int(a[4])]]
df["Giá dư"] = list(a) + [None]*3

a = a + np.random.randint(0, 100, size=5)
nl_bd = [["nl1", int(a[0])], ["nl2", int(a[1])], ["nl3", int(a[2])], ["nl4", int(a[3])], ["nl5", int(a[4])]]
df["Giá ban đầu"] = list(a) + [None]*3

df[" "] = ""
df["Sản phẩm"] = ["sp1","sp2","sp3","sp4", "sp5", "sp6", "sp7", "sp8"]
```

Hình 6

- Sinh dữ liệu ngẫu nhiên cho giá dư, giá ban đầu, chi phí sản xuất, giá bán, và nhu cầu.

```
a = np.random.randint(1, 100, size=8)
sp_cpxs = [["sp1",int(a[0])],["sp2", int(a[1])],["sp3", int(a[2])],["sp4", int(a[3])],["sp5", int(a[4])],["sp6", int(a[5])],["sp7", int(a[6])],["sp8", int(a[7])]]
df["Chi phí sản xuất"] = list(a)

a = a + np.random.randint(1, 100, size=8)
sp_gb = [["sp1",int(a[0])],["sp2", int(a[1])],["sp3", int(a[2])],["sp4", int(a[3])],["sp5", int(a[4])],["sp6", int(a[5])],["sp7", int(a[6])],["sp8", int(a[7])]]
df["Giá bán"] = list(a)

df[" "] = ""
df["Nhu cầu"] = ["sp1","sp2","sp3","sp4", "sp5", "sp6", "sp7", "sp8"]

a = np.random.binomial(10, 0.5, 8)
nhu_cau_1 = [["sp1",int(a[0])],["sp2", int(a[1])],["sp3", int(a[2])],["sp4", int(a[3])],["sp5", int(a[4])],["sp6", int(a[5])],["sp7", int(a[6])],["sp8", int(a[7])]]
df["Nhu cầu 1"] = list(a)
zz1 = list(a)

a = np.random.binomial(10, 0.5, 8)
nhu_cau_2 = [["sp1",int(a[0])],["sp2", int(a[1])],["sp3", int(a[2])],["sp4", int(a[3])],["sp5", int(a[4])],["sp6", int(a[5])],["sp7", int(a[6])],["sp8", int(a[7])]]
df["Nhu cầu 2"] = list(a)
zz2 = list(a)

nl_da_dung_1 = [["nl1", -1], ["nl2", -1], ["nl3", -1], ["nl4", -1], ["nl5", -1]]
nl_da_dung_2 = [["nl1", -1], ["nl2", -1], ["nl3", -1], ["nl4", -1], ["nl5", -1]]

A = np.random.randint(1, 100, size=(8, 5))
A = A.tolist()
```

Hình 7

2. **Ghi Dữ Liệu Vào File Excel**

Ghi DataFrame df vào file Excel với tên là "du_lieu.xlsx" và sheet_name là "Dữ liệu."

```
# Ghi dữ liệu vào file Excel
with pd.ExcelWriter('du_lieu.xlsx') as writer:
    df.to_excel(writer, sheet_name='Dữ liệu', index=False)
```

Hình 8

3. **Tạo mô hình tối ưu hóa**

Sử dụng thư viện Gamspy để tạo một container m để chứa mô hình tối ưu hóa.

```
#Container
m = Container()

#Set
i = Set(m, "nl", records=["nl1","nl2","nl3","nl4", "nl5"], description="factory") # nhà máy
j = Set(m, "sp", records=["sp1","sp2","sp3","sp4", "sp5", "sp6", "sp7", "sp8"], description="product") # sản phẩm

#Parameter
c = Parameter(m, "c", domain=[i], description="c", records=nl_bd) # giá nhập 1 nguyên liệu
l = Parameter(m, "l", domain=[j], description="l", records=sp_cpxs) # chi phí sản xuất
q = Parameter(m, "q", domain=[j], description="q", records=sp_gb) # giá bán
s = Parameter(m, "s", domain=[i], description="s", records=nl_du) # giá bán ế
rhs = Parameter(m, "rhs", domain=[i], description="rhs", records=nl_da_dung_1) # ...
rhs2 = Parameter(m, "rhs2", domain=[i], description="rhs2", records=nl_da_dung_2) # ...
demand = Parameter(m, "demand", domain=[j], description="demand", records=nhu_cau_1) # nhu cầu
demand2 = Parameter(m, "demand2", domain=[j], description="demand2", records=nhu_cau_2) # nhu cầu

#Variable
x = Variable(m, "x", domain=[i], description="x", type="Positive") #
y = Variable(m, "y", domain=[i], description="y", type="Positive") #
y2 = Variable(m, "y2", domain=[i], description="y2", type="Positive") #


#Equation
st = Equation(m, "st", domain=[i], description="rang buoc") #
st2 = Equation(m, "st2", domain=[i], description="rang buoc2")

st[i] = (x[i]-y[i]) == rhs[i]
st2[i] = (x[i]-y2[i]) == rhs2[i]

```

Hình 9

4. **Giải bài toán**

- Tạo một mô hình tối ưu hóa với gamspy.
- Định nghĩa mục tiêu tối ưu hóa và ràng buộc.
- Sử dụng solver để giải bài toán tối ưu hóa và in kết quả.

```
#Model
transport = Model(
    m,
    name="transport",
    equations=m.getEquations(),
    problem="LP",
    sense=Sense.MIN,
    objective=nl_bd[0][1]*x["nl1"]+nl_bd[1][1]*x["nl2"]+nl_bd[2][1]*x["nl3"]+nl_bd[3][1]*x["nl4"]+nl_bd[4][1]*x["nl5"]- ......
)
star = time.time()
transport.solve(output=sys.stdout)
assert math.isclose(transport.objective_value, transport.objective_value , rel_tol=0.001)

```

Hình 10

5. **Ghi kết quả vào file Excel**

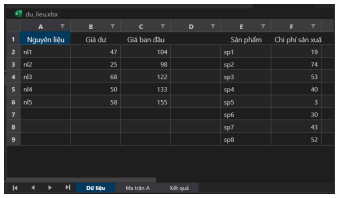
Ghi kết quả của ma trận A và kết quả của biến x, y vào file Excel.

Kết quả sau khi chạy đoạn code trên là một file Excel có 3 sheet bao gồm dữ liệu, ma trận A và kết quả.

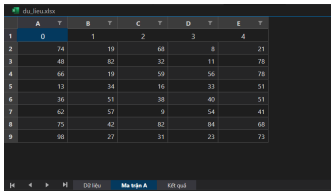
```
# Ghi ma trận A vào file Excel
df_A = pd.DataFrame(A)
with pd.ExcelWriter('du_lieu.xlsx', mode='a') as writer:
    df_A.to_excel(writer, sheet_name='Ma trận A', index=False)

# Ghi kết quả vào file Excel
data = {"x": list(x.records["level"]) + [None]*3}
df_kq = pd.DataFrame(data)
df_kq["y1"] = list(y.records["level"]) + [None]*3
df_kq["y2"] = list(y2.records["level"]) + [None]*3
df_kq["z1"] = zz1
df_kq["z2"] = zz2
df_kq["Objective"] = [transport.objective_value] + [None]*7
with pd.ExcelWriter('du_lieu.xlsx', mode='a') as writer:
    df_kq.to_excel(writer, sheet_name='Kết quả', index=False)
end = time.time()
print(end - star)
print("...............................................................................")
```

Hình 11

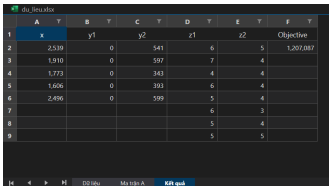
Sau mỗi lần chạy, các dữ liệu sẽ được random mới hoàn toàn, và kết quả cũng đổi. Dưới đây là một trường dữ liệu nào đó.



Hình 12: Dữ liệu ban đầu



Hình 13: Ma trận A



Hình 14: Kết quả

4 To the SLP-EPDR: Algorithmic Solutions

Một số thuật toán Giải pháp được đưa ra trong Phần 4 của Tài liệu tham khảo. 1, và năm nay 2023 Sinh viên CSE - HCMUT chỉ có thể thử cách tiếp cận đầu tiên (Thuật toán 1) dựa trên Bài toán dòng chi phí tối thiểu

Tìm hiểu, triển khai và xác minh tính hiệu quả của Thuật toán đã nghiên cứu

4.1 Phân tích

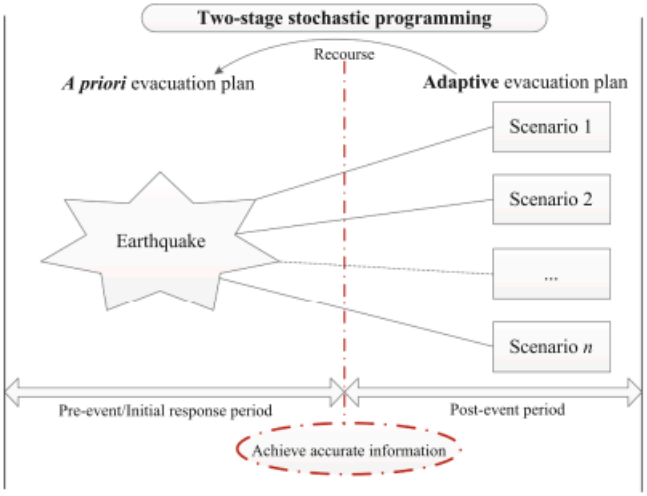
4.1.1 Chương trình tuyến tính ngẫu nhiên để lập kế hoạch sơ tán trong ứng phó thiên tai (SLP-EPDR)

- *Vấn đề bài báo*

  Bài này thuộc về nghiên cứu và nhằm giải quyết vấn đề lập kế hoạch đường sơ tán cho những người bị ảnh hưởng bằng cách xây dựng một khung mô hình chung khi xảy ra thảm họa. Như mọi người đã biết, rất khó để ước tính tác động và thiệt hại trên đường đô thị do thảm họa (chẳng hạn như động đất, bão, v.v.) vì hầu như không thể biết trước cường độ. Do đó, vấn đề sơ tán này thường được coi là một vấn đề lập trình ngẫu nhiên trong đó tính ngẫu nhiên không chỉ phát sinh từ thời gian di chuyển mà còn cả năng lực. Nói cách khác, thiệt hại có thể xảy ra trên một con đường nhất định có thể ngăn cản người dân thoát khỏi khu vực thảm họa dẫn đến thời gian và sức chứa di

chuyển của liên kết ngẫu nhiên.

Bài báo tập trung vào chiến lược kết hợp lựa chọn lộ trình tiên nghiệm (trước thảm họa) và thích ứng (sau thảm họa), có thể đạt được bằng lập trình ngẫu nhiên hai giai đoạn, để xác định kế hoạch sơ tán cho những người bị ảnh hưởng xem **Hình 15**.



Hình 15: Minh họa về sự xuất hiện của các sự kiện động đất hoặc chiến tranh

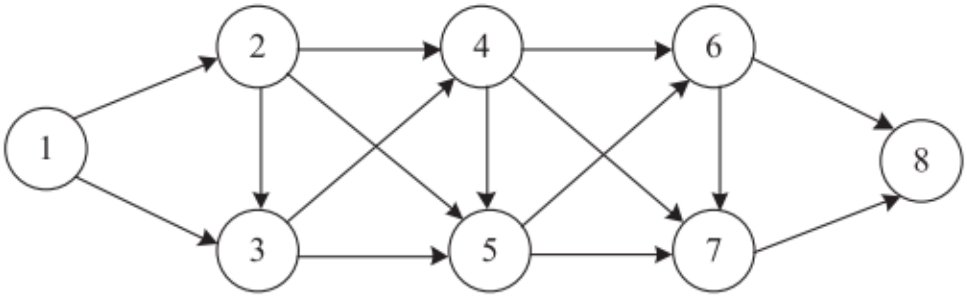
Tính ngẫu nhiên có thể được phản ánh bởi một số lượng hữu hạn các kịch bản, mỗi kịch bản đều gắn liền với một xác suất đã biết. Phương pháp giải quyết tính ngẫu nhiên này thường được áp dụng trong nhiều vấn đề quản lý cứu trợ thiên tai khác nhau. Ngoài ra, tối ưu hóa mạnh mẽ, ví dụ: tiêu chí tối thiểu-tối đa, là một phương pháp hiệu quả khác để giải quyết vấn đề không chắc chắn, yêu cầu các giá trị có khả năng xảy ra cao nhất cùng với giới hạn trên và giới hạn dưới của các tham số ngẫu nhiên. Cụ thể, một phần đường giao thông có thể bị phá hủy trong sự kiện, gây ra thời gian và năng lực di chuyển ngẫu nhiên khi di chuyển trên đường. Nói cách khác, các quyết định ở giai đoạn đầu không có tính dự đoán được đưa ra trước khi nhận ra sự không chắc chắn. Các quyết định ở giai đoạn thứ hai (truy đòi), có điều kiện dựa trên các quyết định ở giai đoạn đầu, được đưa ra sau khi nhận ra thời gian và năng lực di chuyển ngẫu nhiên. Do đó, mục tiêu là lập kế hoạch sơ tán trước sự kiện tối ưu trong giai đoạn đầu tiên, trong điều kiện không chắc chắn sẽ phải đối mặt trong giai đoạn thứ hai. Vấn đề bài báo đề cập là lên kế hoạch sơ tán người dân trong vùng bị ảnh hưởng khi gặp tai họa, xuyên suốt bài báo tác giả đã từng bước một phát triển một thuật toán tối ưu giúp tìm ra tuyến đường sơ tán hiệu quả nhất cho người dân gặp nạn.

Với các tiêu chí bao gồm thời gian di tản ngắn nhất, số lượng người có thể di tản nhiều nhất trong điều kiện cho phép, đồng thời phải có thể thích nghi, đưa ra những thay đổi lộ trình dựa vào các dữ liệu thực tế. Như vậy đây là bài toán hai giai đoạn với giai đoạn một là kế hoạch ban đầu dựa trên các điều kiện thực tế khi chưa xảy ra tai họa và giai đoạn hai là kế hoạch thích nghi với các điều kiện thực tế khi tai họa đã xảy ra sao cho phù hợp với các tiêu chí trên.

- *Mô hình hóa bài toán SLP-EPDR*

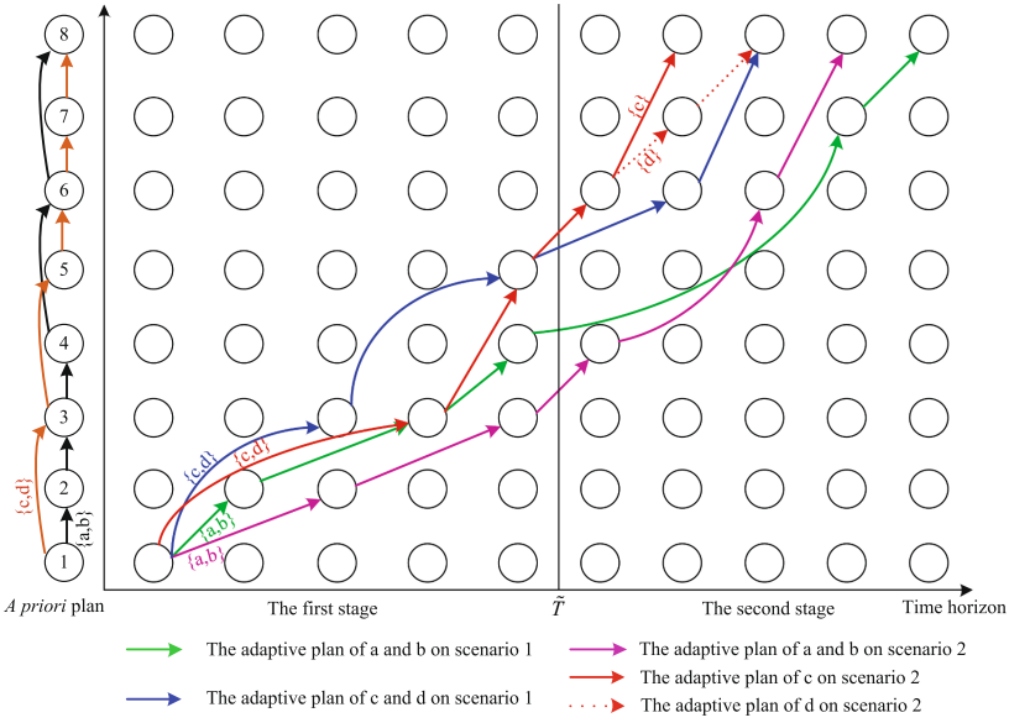
  Để giúp người đọc hình dung rõ hơn các tuyến đường di tản, tác giả đã đơn giản hóa bằng cách sử

dụng các node (hình tròn có chỉ số) đại diện cho các vị trí trên bản đồ và đường đi được thể hiện bằng các mũi tên.



Hình 16: Minh họa về mạng lưới đường sơ tán

Lưu ý: kế hoạch giai đoạn một sẽ được sử dụng cho đến khi có được các thông tin thực tế về tình hình tai họa, kế hoạch hai sẽ đề xuất một tuyến đường di tản phù hợp với điều kiện lúc đó.

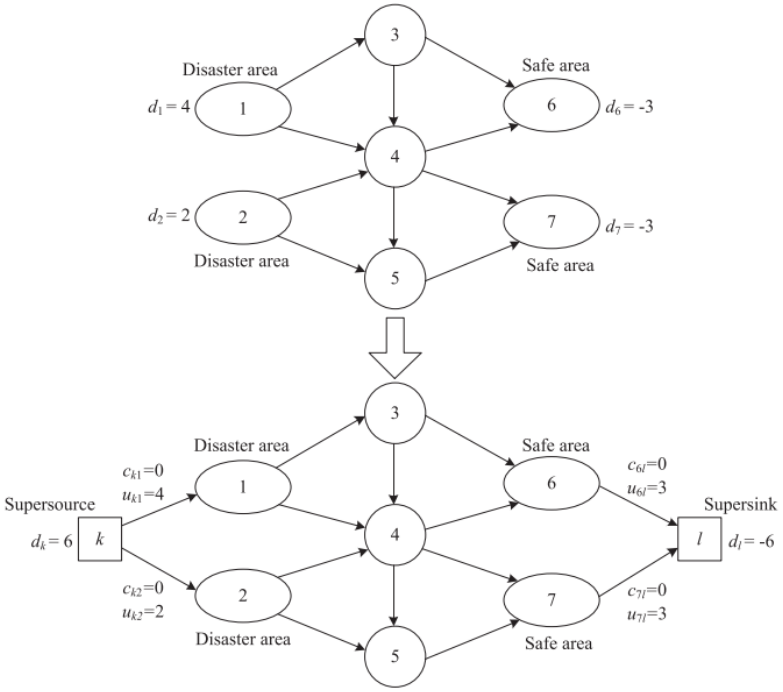


Hình 17: Kế hoạch sơ tán ngẫu nhiên hai giai đoạn trong mạng phụ thuộc thời gian

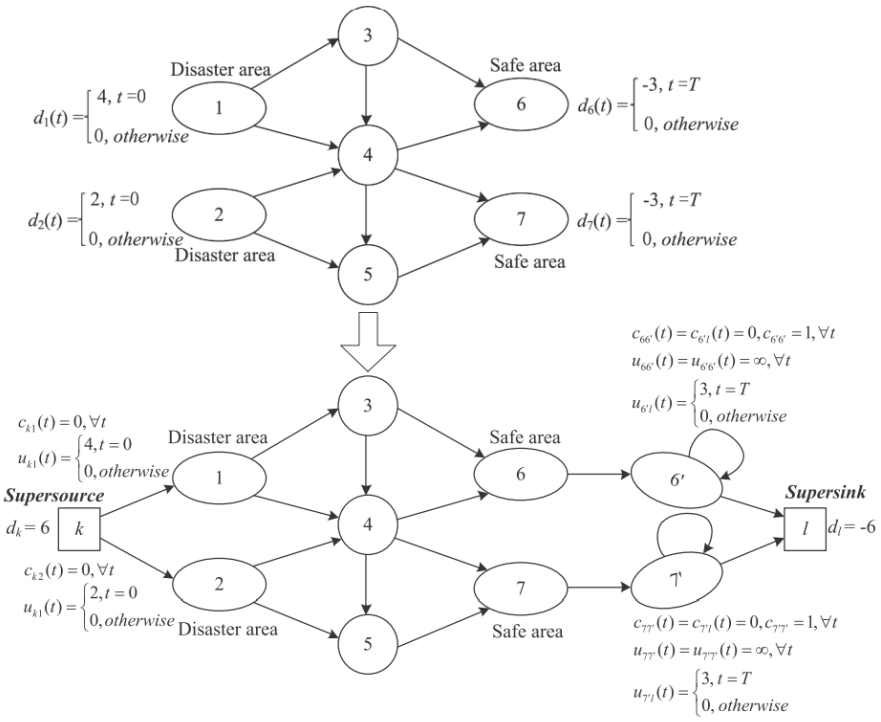
Ngưỡng $\tilde{T}$ trong hình 17 là thời điểm nhận được thông tin thực tế

Trong vấn đề sơ tán ngẫu nhiên hai giai đoạn, có nhiều hơn một nguồn và đích trong mạng lưới sơ tán thực tế. Do đó, mạng di tản có nhiều nguồn nên được chuyển đổi thành một mạng tương đương với một supersource duy nhất, thậm chí thành nhiều đích.Trên thực tế, một Supersource k được thêm vào mạng và trong khi đó các cung giả (k, i) nên được thêm vào với thời gian di chuyển với giá trị 0, tức là $c_{ki} = 0, i \in K$,trong đó K là tập hợp các nút nguồn, và để công suất trên cung giả là giá trị của nguồn cung cấp tại nút i, tức là $u_{ki} = d_i, i \in$ Do đó, giá trị cung cấp tại supersource (nguồn) có thể được đặt là $\sum_{i \in K} d_i$. Tương tự, Supersink (đích) có thể được chuyển

đổi theo cùng một cách. Sau đây là một ví dụ để minh họa cách chuyển đổi nhiều nguồn và đích thành Supersource (nguồn) và Supersink(đích) (xem Hình 18).



Hình 18: Ví dụ chuyển đổi Supersource và Supersink trong mạng di tản



Hình 19: Ví dụ về chuyển đổi supersource và supersink trong mạng phụ thuộc thời gian.

Đối với một mạng phụ thuộc vào thời gian, thời gian và khả năng di chuyển sẽ thay đổi theo thời gian khởi hành. Khi thêm một supersource, thời gian di chuyển trên các vòng cung giả là $c_{ki}(t) = 0, tm \in 0, 1, 2, ..., T, i \in K)$ và khả năng di chuyển.Hơn nữa, nhiều đích được chuyển đổi thành Supersink. Vì thời gian đến đích cho mỗi luồng có thể khác nhau, trước tiên chúng tôi thêm một bản sao j' cho mỗi đích đến gốc $j, j \in D$, và sau đó Supersink l được thêm vào mạng. Đối với mỗi đích, chúng ta thêm cung (j, j') vào mạng với công suất vô hạn, tức là, $u_{jj'} = \infty$ và thời gian di chuyển cho mỗi cung giả được coi là bằng không, $c^s_{jj'}(t) = 0, T \in 0, 1, T$. Ngoài ra, một vòng lặp tự được thêm vào cho mỗi nút j', trong đó $u_{j'j'}(t) = \infty$ và $c_{j'j'}(t) = 1$. Hơn nữa, công suất của cung (j', l) bằng với nhu cầu của nút j tại thời điểm t, tức là $u_{j'v}(T) = b_j(T)$, và công suất tại thời điểm khác được đặt là 0, tức là $u_{j'l}(t) = 0, t \in 0, 1, ..., T$. Ngoài ra, thời gian di chuyển được giả định là bằng không trong thời gian được coi là $C_{j'l}(t) = 0, t \in 1, 2, ..., T$. (xem Hình 19)

- Một số ký hiệu tác giả sử dụng trong thuật toán:

Kí hiệu	**Định nghĩa**
V	tập hợp các node
A	tập hợp các đường đi
i, j	chỉ số các node
(i, j)	đường đi từ node i đến node j
s	chỉ số của trường hợp
S	số lượng các trường hợp
v	số người cần di tản
T	tổng thời gian di tản
u_{ij}	số người tối đa di chuyển trên đường từ iđến j
$c_{i,j}$	thời gian di chuyển từ i đến j
$u^s_{ij}(t)$	số người tối đa di chuyển trên đường từ i đến j tại thời điểm t trong trường hợp s
$c^s_{ij}(t)$	thời gian di chuyển từi đến jtại thời điểm i trong trường hợp s
μ_s	xác suất ở ngữ cảnh s

Bảng 1: Chỉ số và tham số được sử dụng trong công thức toán học

Kí hiệu	**Định nghĩa**
x_{ij}	số người di chuyển từi đến j
$y_{ij}(t)$	số người di chuyển từ i đến j tại thời điểm t trong trường hợp s.

Bảng 2: Các biến quyết định được sử dụng trong công thức toán học.

– Giai đoạn 1 (first-stage) - Kế hoạch di tản ban đầu trước ngưỡng $\widetilde{T}$

Trong giai đoạn đầu tiên, cần xác định phương án sơ tán khả thi từ nguồn đến đích đến. Luồng trên mỗi liên kết phải đáp ứng ràng buộc cân bằng luồng dưới đây:

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

Trong đó:

$$d_i = \begin{cases} v & \text{nếu } i = s \\ -v & \text{nếu } i = t \\ 0 & \text{còn lại} \end{cases}$$

Trong khi đó, luồng trên mỗi liên kết cũng phải thỏa mãn ràng buộc về dung lượng:

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

Ràng buộc cân bằng luồng có thể tạo ra một đường dẫn có các vòng lặp. Để loại bỏ các vòng lặp trên đường sơ tán vật lý được tạo ra, hình phạt liên kết $p_{ij}, (i,j) \in A$ được giới thiệu đặc biệt. Với sự xem xét này, hàm phạt có thể được định nghĩa là

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

– Giai đoạn 2 (second-stage) - Kế hoạch di tản ban đầu trước ngưỡng $\widetilde{T}$

Giai đoạn thứ hai là đánh giá giai đoạn đầu tiên để có được kế hoạch sơ tán tối ưu và hiệu quả. Trong giai đoạn này, những người bị ảnh hưởng sẽ nhận được các lối đi thích ứng với thời gian sơ tán tối thiểu theo thông tin thảm họa thời gian thực. Trước ngưỡng thời gian, những người bị ảnh hưởng sẽ được sơ tán theo kế hoạch sơ tán ưu tiên trong giai đoạn một; nói cách khác, kế hoạch sơ tán trong các tình huống khác nhau trước ngưỡng thời gian đều giống với lộ trình tiên nghiệm. Theo thảo luận ở trên, các ràng buộc ghép nối cho kế hoạch sơ tán của từng kịch bản trước ngưỡng thời gian $\widetilde{T}$ có thể được xây dựng như sau:

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

trước ngưỡng thời gian $\widetilde{T}$, phương án sơ tán trong từng kịch bản của giai đoạn 2 cũng giống như phương án sơ tán sơ tán ở giai đoạn 1.

Sau đây, mô hình giai đoạn hai được thiết lập với mục tiêu giảm thiểu tổng thời gian của những người bị ảnh hưởng được sơ tán từ khu vực nguy hiểm đến khu vực an toàn trong từng kịch bản:

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

s.t

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

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

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

Như vậy chúng ta sẽ có mô hình 2 giai đoạn:

$$\begin{cases} min \sum_{ij \in A} p_{ij} x_{ij} + \sum_{s=1}^{S} (\mu_s \sum_{ij \in A_s} c_{ij}^s(t).y_{ij}^s(t)) \\ st \\ \sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = d_i, \forall i \in V \\ 0 \leq x_{ij} \leq u_{ij}, \forall (i,j) \in A \\ \sum_{(i_t, j_{t'}) \in A_s} y_{ij}^s(t) - \sum_{(j_{t'}, i_t) \in A_s} y_{ji}^s(t') = d_i^s(t), \forall i \in V, t \in 0, 1, ..., T, s = 1, 2, ..., S \\ 0 \leq y_{ij}^s(t) \leq u_{ij}^s(t), \forall (i,j) \in A, t \in 0, 1, ..., T, s = 1, 2, ..., S \\ \sum_{t \leq \widetilde{T}} y_{ij}^s(t) = x_{ij}, (i,j) \in A \; s = 1, 2, ..., S \end{cases} \tag{4}$$

- Phương pháp giải nhân tử Lagrange

  Mô hình (4) về cơ bản là một mô hình lập trình số nguyên chứa hai loại biến quyết định. Trong mô hình này, ràng buộc ghép là một ràng buộc phức tạp, dẫn đến mô hình không thể giải được trong thời gian đa thức. Vì vậy, phần này dự định nới lỏng ràng buộc ghép vào hàm mục tiêu bằng phương pháp nới lỏng Lagrange, cách tiếp cận kép dựa trên phương pháp thư giãn Lagrange để phân tách bài toán ban đầu thành hai bài toán con dễ giải, có giá trị mục tiêu là giới hạn dưới của giá trị tối ưu của mô hình (4). Cụ thể, mô hình ban đầu được phân tách thành bài toán luồng chi phí tối thiểu tiêu chuẩn và bài toán phụ thuộc vào thời gian, có thể được giải quyết một cách hiệu quả bằng các thuật toán chính xác. Vấn đề then chốt của phương pháp thư giãn Lagrangge là tìm ra giới hạn trên của bài toán quan tâm. Tình cờ thay, các giải pháp nới lỏng lại khả thi đối với vấn đề ban đầu.

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

Dưới đây là mô hình bài toán với nhân tử Lagrange

$$\begin{cases} \min \sum ij \in A p_{ij} x_{ij} + \sum_{s=1}^{S} (\mu_s \sum_{ij \in A_s} c_{ij}^s(t).y_{ij}^s(t)) + \sum_{s=1}^{S} \sum_{t \leq \widetilde{T}} \sum_{(i,j) \in A} \alpha_{ij}^s(t)(y_{ij}^s(t) - x_{ij}) \\ st \\ \sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = d_i, \forall i \in V \\ 0 \leq x_{ij} \leq u_{ij}, \forall (i,j) \in A \\ \sum_{(i_t, j_{t'}) \in A_s} y_{ij}^s(t) - \sum_{(j_{t'}, i_t) \in A_s} y_{ji}^s(t') = d_i^s(t), \forall i \in V, t \in 0, 1, ..., T, s = 1, 2, ..., S \\ 0 \leq y_{ij}^s(t) \leq u_{ij}^s(t), \forall (i,j) \in A, t \in 0, 1, ..., T, s = 1, 2, ..., S \\ \sum_{t \leq \widetilde{T}} y_{ij}^s(t) = x_{ij}, (i,j) \in A \; s = 1, 2, ..., S \end{cases} \tag{5}$$

Từ đây chúng ta có thể chia bài toán ra thành 2 bài toán phụ

∗ **Bài toán Min-cost flow (Subproblem 1)**

$$\begin{cases} \min \text{SP1}(\alpha) = \sum_{ij \in A}(p_{ij} - \sum_{s=1}^{S}\sum_{t \leq \widetilde{T}} \alpha_{ij}(t)^s)x_{ij} \\ st \\ \sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ji} = d_i, \forall i \in V \\ 0 \leq x_{ij} \leq u_{ij}, \forall (i,j) \in A \end{cases} \tag{6}$$

∗ **Bài toán Min-cost flow phụ thuộc thời gian (Subproblem 2)**

$$\begin{cases} \min \text{SP2}(\alpha) = \sum_{s=1}^{S}\sum_{(ij)\in A}(\sum_{t\in[0,1,...,T]} \mu_s.c_{ij}(t) + \sum_{t\leq\widetilde{T}} \alpha_{ij}^s(t))y_{ij}^s(t) \\ st \\ \sum_{(i_t,j_{t'})\in A_s} y_{ij}^s(t) - \sum_{(j_{t'},i_t)\in A_s} y_{ji}^s(t') = d_i^s(t), \forall i \in V, t \in 0,1,...,T, s = 1,2,...,S \\ 0 \leq y_{ij}^s(t) \leq u_{ij}^s(t), \forall (i,j) \in A, t \in 0,1,...,T, s = 1,2,...,S \\ \sum_{t\leq\widetilde{T}} y_{ij}^s(t) = x_{ij}, (i,j) \in A\ s = 1,2,...,S \end{cases} \tag{7}$$

Hai bài toán phụ (6) và (7) có thể giải bằng giải thuật 1 (Susesive Shortest Path), nghiệm tối ưu là tổng của 2 bài toán con. Ở đây ta chỉ xét bài toán 1 cho đơn giản hóa quá trình phân tích cũng như đánh giá độ hiệu quả, tối ưu của thuật toán SSP.

4.1.2 Phân tích giải thuật 1 - Successive Shortest Path min cost flow(SSP)

Bước 1: Lấy biến x là dòng chấp nhận được giữa bất kỳ điểm OD nào và nó có thời gian tối thiểu trong các dòng chấp nhận được có cùng giá trị dòng.

Bước 2: Thuật toán sẽ dừng nếu giá trị dòng của x đạt đến v hoặc không có đường đi có chi phí (c) tối thiểu trong dòng còn dư $(V, A(x), C(x), U(x), D)$. Mặt khác, đường đi ngắn nhất sẽ được xác định bằng cách sử dụng thuật toán ***Label-correcting***, sau đó đến với *Bước 3*. Các hàm $A(x), C(x), U(x)$ ở các dòng còn dư được định nghĩa như sau:

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

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

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

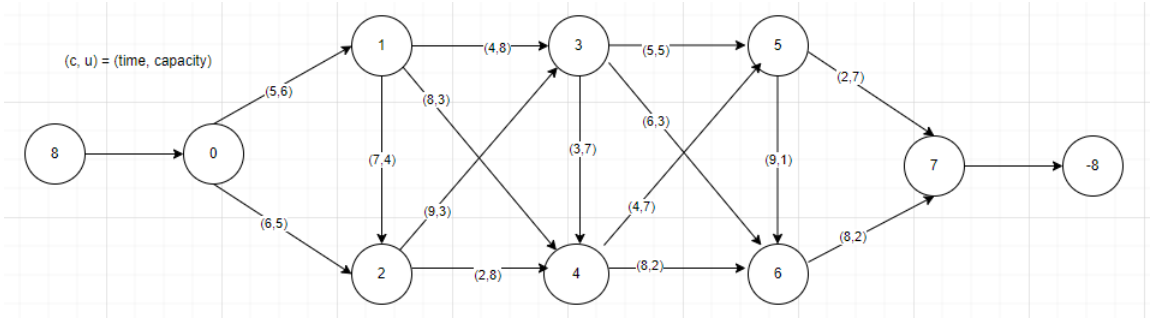
Bước 3: Tăng dòng dựa theo đường đi ngắn nhất vừa tìm được, nếu chưa đạt đến v, quay về *Bước 2*.

Thuật toán label-correcting:

1. Khởi tạo: Gán nhãn cho tất cả các đỉnh trong đồ thị. Nhãn của đỉnh xuất phát được đặt là 0, còn nhãn của các đỉnh khác được đặt là vô cùng lớn.

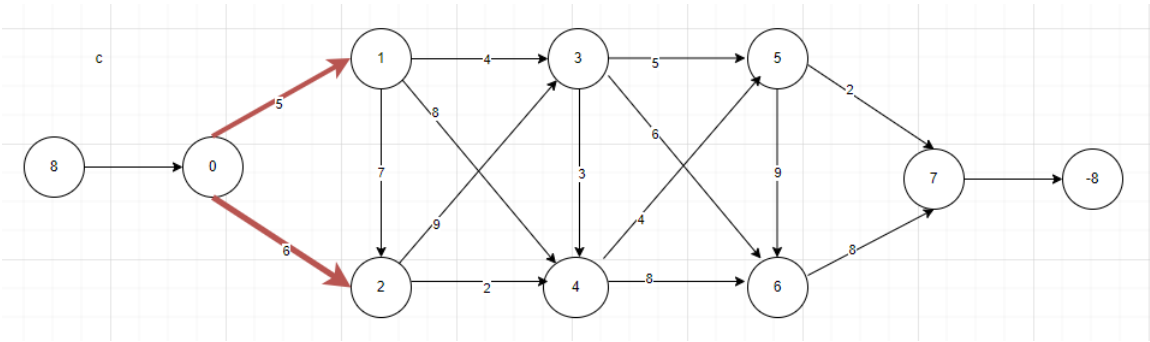
2. Lặp qua các đỉnh: Bắt đầu vòng lặp, duyệt qua tất cả các đỉnh trong đồ thị.
3. Cập nhật nhãn: Với mỗi đỉnh, kiểm tra các cạnh nối từ đỉnh hiện tại đến các đỉnh kề. Nếu có thể cập nhật nhãn của đỉnh kề thông qua đỉnh hiện tại sao cho khoảng cách từ đỉnh xuất phát đến đỉnh kề thông qua đỉnh hiện tại là tốt hơn (nhỏ hơn) so với nhãn hiện tại của đỉnh kề, thì cập nhật nhãn của đỉnh kề bằng giá trị mới này.
4. Kiểm tra điều kiện dừng: Sau khi duyệt qua tất cả các đỉnh và cập nhật nhãn, kiểm tra xem có sự cải thiện nào không. Nếu không có sự cải thiện nào (tức là không có đỉnh nào có thể cập nhật nhãn được nữa) thuật toán kết thúc, nếu có thì quay lại *bước 2*.

Ví dụ về giải thuật 1



Hình 20: Đồ thị đầu vào

- ***Bước 1: Chọn dòng bắt đầu từ 0 làm dòng bắt đầu***



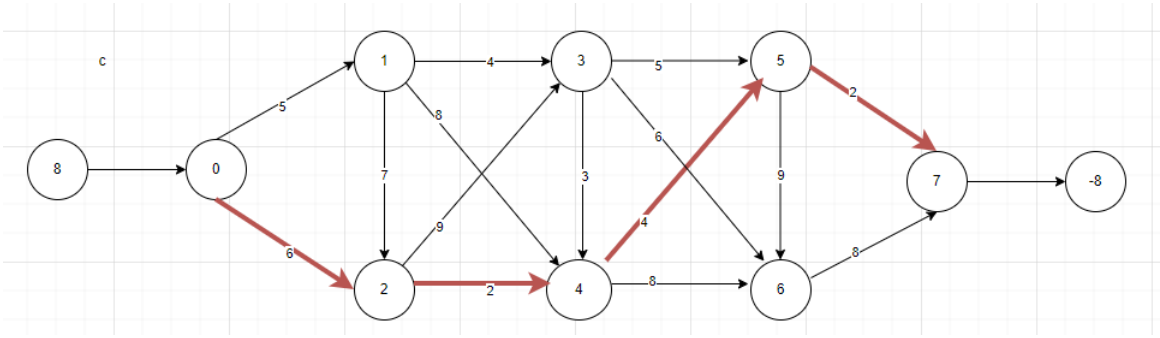
Hình 21: Chọn đỉnh

- ***Bước 2: Tìm được đi ngắn nhất bằng thuật toán Labbel_ correcting***

  Áp dụng thuật toán ***Label_ correcting:***

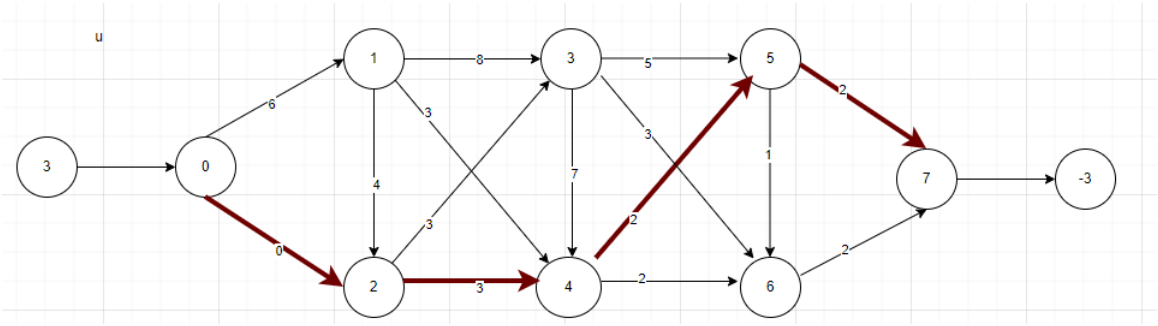
Step	Vertice	Adjacent	Update
1	0	0->1 = 5, 0->2 = 6	d1 = 5, d2 = 6
2	1	1->2 = 7, 1->3 = 4, 1->4 = 8	d3 = 9, d4 = 13
3	2	2->3 = 9, 2->4 = 2	d4 = 8
4	3	3->4 = 3, 3->5 = 5, 3->6 =6	d5 = 14, d6 = 15
5	4	4->5 = 4, 4->6 = 8	d5 = 13
6	5	5->6 = 9, 5->7 = 2	d7 = 15
7	6	6->7 = 8	
8	7		

Bảng 3: Thuật toán label_correcting



Hình 22: Đường đi ngắn nhất

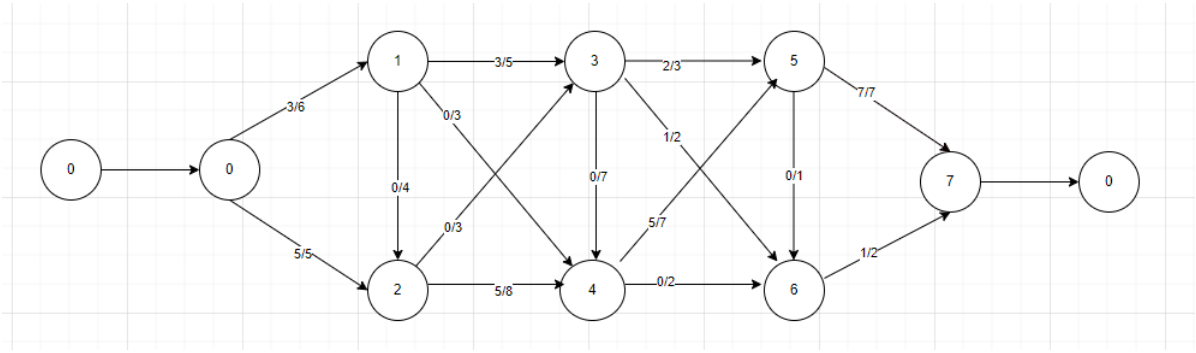
- ***Bước 3: Giảm flow của từng dòng xuống theo dòng thấp nhất***



Hình 23: Giảm flow của các dòng

Tiếp tục thực hiện các bước trên cho đến khi x đạt được v hoặc không tìm được đường đi ngắn nhất nữa thì thuật toán kết thúc.

⁎ **Kết quả:**



Hình 24: Kết quả

⁎ **Tổng thời gian tối ưu (Min cost flow)**

$$T_{min} = 3 \times 5 + 5 \times 6 + 0 \times 7 + 3 \times 4 + 0 \times 8 + 0 \times 9 + 5 \times 2 + 0 \times 3 + 2 \times 5 + 1 \times 6 + 5 \times 4 + 0 \times 8 + 0 \times 9 + 7 \times 2 + 1 \times 8 = 125$$

4.2 Code minh họa giải thuật Successive Shortest Path for min cost flow

- **Định nghĩa lớp Edge gồm có: node đầu, node kết thúc, sức chứa, thời gian**

```
8   # Định nghĩa lớp Edge (cạnh)
9   class Edge:
10      def __init__(self, from_node, to, capacity, cost):
11          self.from_node = from_node  # Nút bắt đầu
12          self.to = to  # Nút kết thúc
13          self.capacity = capacity  # Dung lượng tối đa
14          self.cost = cost  # Chi phí
```

Hình 25: Lớp Edge

- **Successive Shortest Path**

  Khai báo các cạnh và thông số của bài toán

```
89  # List các cạnh và các trọng số
90  edges = [
91      Edge(0, 1, 6, 5),
92      Edge(0, 2, 5, 6),
93      Edge(1, 2, 4, 7),
94      Edge(1, 3, 8, 4),
95      Edge(1, 4, 3, 8),
96      Edge(2, 3, 3, 9),
97      Edge(2, 4, 8, 2),
98      Edge(3, 4, 7, 3),
99      Edge(3, 5, 5, 5),
100     Edge(3, 6, 3, 6),
101     Edge(4, 5, 7, 4),
102     Edge(4, 6, 2, 8),
103     Edge(5, 6, 1, 9),
104     Edge(5, 7, 7, 2),
105     Edge(6, 7, 2, 8)
106 ]
107
108 Node = 8
109 demand = 8
110 source = 0
111 sink = 7
```

Hình 26: Khai báo giá trị

Tìm đường đi ngắn nhất

```python
# Tìm đường đi ngắn nhất từ v0 đến các nút khác
def shortest_paths(n, v0, d, p, adj, capacity, cost):
    d = [INF]*n
    d[v0] = 0
    inq = [False]*n
    q = deque([v0])
    p = [-1]*n

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

Hình 27: Hàm tìm đường đi ngắn nhất

Tìm luồng có chi phí nhỏ nhất

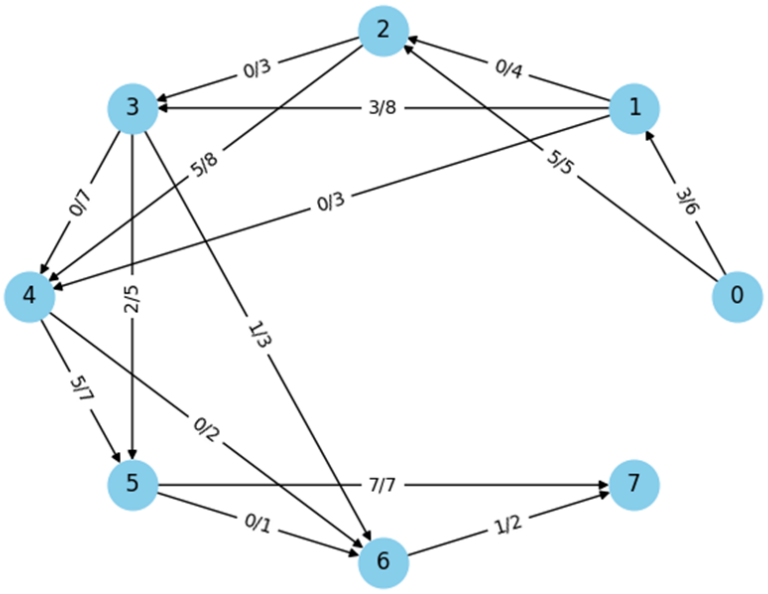
```python
    # Vẽ đồ thị
    G = nx.DiGraph()  # Khởi tạo đồ thị có hướng
    for e in edges:  # Duyệt qua các cạnh
        G.add_edge(e.from_node, e.to, capacity=e.capacity, flow=flow[e.from_node][e.to])  # Thêm cạnh vào đồ thị

    pos = nx.circular_layout(G)  # Xác định vị trí các nút
    nx.draw(G, pos, with_labels=True, node_size=700, node_color='skyblue')  # Vẽ đồ thị
    edge_labels = nx.get_edge_attributes(G, 'flow')  # Lấy nhãn của các cạnh
    for u, v, d in G.edges(data=True):
        edge_labels[(u, v)] = f"{d['flow']}/{d['capacity']}"  # Thêm giá trị luồng/dung lượng vào nhãn cạnh
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)  # Vẽ nhãn của các cạnh
    plt.show()  # Hiển thị đồ thị
```

Hình 28: Hàm tìm luồng có chi phí nhỏ nhất

Vẽ đồ thị

```python
    # Vẽ đồ thị
    G = nx.DiGraph()  # Khởi tạo đồ thị có hướng
    for e in edges:  # Duyệt qua các cạnh
        G.add_edge(e.from_node, e.to, capacity=e.capacity, flow=flow[e.from_node][e.to])  # Thêm cạnh vào đồ thị

    pos = nx.circular_layout(G)  # Xác định vị trí các nút
    nx.draw(G, pos, with_labels=True, node_size=700, node_color='skyblue')  # Vẽ đồ thị
    edge_labels = nx.get_edge_attributes(G, 'flow')  # Lấy nhãn của các cạnh
    for u, v, d in G.edges(data=True):
        edge_labels[(u, v)] = f"{d['flow']}/{d['capacity']}"  # Thêm giá trị luồng/dung lượng vào nhãn cạnh
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)  # Vẽ nhãn của các cạnh
    plt.show()  # Hiển thị đồ thị
```

Hình 29: Hàm vẽ đồ thị



Hình 30: Dồ thị

- **Kết quả**

```
ubuntu@ubuntu:~/Desktop/Python$ time python3 ssp.py
The minimum cost maximum flow is 125

real    0m0.423s
user    0m0.703s
sys     0m0.445s
ubuntu@ubuntu:~/Desktop/Python$
```

Hình 31: Thời gian chạy

4.3 So sánh với giải thuật Edmonds-Karp min-cost và đánh giá độ hiệu quả của thuật toán

Thuật toán Edmonds–Karp là một trường hợp đặc biệt của thuật toán Ford–Fulkerson cho việc tìm luồng cực đại trong mạng.

4.3.1 Giải thuật Edmonds-Karp min-cost

Tìm đường đi ngắn nhất bằng thuật toán Bellman-Ford

```python
# Thuật toán Bellman-Ford để tìm đường đi ngắn nhất trong đồ thị có trọng số
def bellman_ford(graph, source, sink):
    # Khởi tạo các khoảng cách từ nguồn tới các đỉnh khác là vô cùng
    distances = {node: float('inf') for node in graph.nodes}
    distances[source] = 0  # Khoảng cách từ nguồn đến chính nó là 0

    # Lặp để cập nhật khoảng cách từ nguồn tới các đỉnh khác
    for _ in range(graph.number_of_nodes() - 1):
        for u, v in graph.edges:
            capacity = graph[u][v].get('capacity', 0)
            cost = graph[u][v]['cost']
            # Nếu khoảng cách mới nhỏ hơn khoảng cách hiện tại và cạnh có dung lượng dương
            if distances[u] + cost < distances[v] and capacity > 0:
                distances[v] = distances[u] + cost

    return distances
```

Hình 32: Thuật toán Bellman-Ford

Tìm min-cost flow bằng thuật toán Edmonds-Karp

```python
# Thuật toán Edmonds-Karp để tìm Tìm luồng có chi phí nhỏ nhất
def edmonds_karp_min_cost(graph, source, sink, demand):
    # Tạo bản sao của đồ thị để thao tác
    residual_graph = graph.copy()
    min_cost = 0  # Khởi tạo chi phí tối thiểu

    while True:
        distances = bellman_ford(residual_graph, source, sink)

        # Nếu không thể đến được đích, thoát khỏi vòng lặp
        if distances[sink] == float('inf'):
            break

        # Tìm đường đi ngắn nhất từ nguồn tới đích
        path = nx.shortest_path(residual_graph, source=source, target=sink, weight='cost')

        # Tìm đoạn có sức chứa nhỏ nhất (sức chứa nhỏ nhất trên đường đi)
        bottleneck_capacity = min(residual_graph[u][v].get('capacity', 0) for u, v in zip(path, path[1:]))
        for u, v in zip(path, path[1:]):
            # Nếu đường đi đã đầy, gán chi phí lớn để loại bỏ đường này
            if residual_graph[u][v].get('capacity', 0) == 0:
                residual_graph[u][v]['cost'] = float('inf')

        if bottleneck_capacity != 0:
            for u, v in zip(path, path[1:]):
                if bottleneck_capacity < demand:
                    # Cập nhật dung lượng còn lại của cạnh và chi phí tối thiểu
                    if 'capacity' in residual_graph[u][v]:
                        residual_graph[u][v]['capacity'] -= bottleneck_capacity
                        min_cost += residual_graph[u][v].get('cost', 0) * bottleneck_capacity
                        # print("from", u, "to", v, "with cost", residual_graph[u][v].get('cost', 0), "and capacity", bottleneck_capacity)
                    else:
                        residual_graph[v][u]['capacity'] += bottleneck_capacity

```

Hình 33

```
54
55                  # Nếu có cạnh ngược, cập nhật dung lượng của cạnh đó
56                  if residual_graph.has_edge(v, u):
57                      if 'capacity' in residual_graph[v][u]:
58                          residual_graph[v][u]['capacity'] += bottleneck_capacity
59                      else:
60                          residual_graph[u][v]['capacity'] -= bottleneck_capacity
61              else:
62                  # Cập nhật dung lượng còn lại của cạnh và chi phí tối thiểu
63                  if 'capacity' in residual_graph[u][v]:
64                      residual_graph[u][v]['capacity'] -= demand
65                      min_cost += residual_graph[u][v].get('cost', 0) * demand
66                      # print("from", u, "to", v, "with cost", residual_graph[u][v].get('cost', 0), "and capacity", demand)
67                  else:
68                      residual_graph[v][u]['capacity'] += demand
69
70                  # Nếu có cạnh ngược, cập nhật dung lượng của cạnh đó
71                  if residual_graph.has_edge(v, u):
72                      if 'capacity' in residual_graph[v][u]:
73                          residual_graph[v][u]['capacity'] += demand
74                      else:
75                          residual_graph[u][v]['capacity'] -= demand
76
77          # Cập nhật dung lượng còn lại của nhu cầu
78          if bottleneck_capacity < demand:
79              demand -= bottleneck_capacity
80          else:
81              return min_cost
82
```

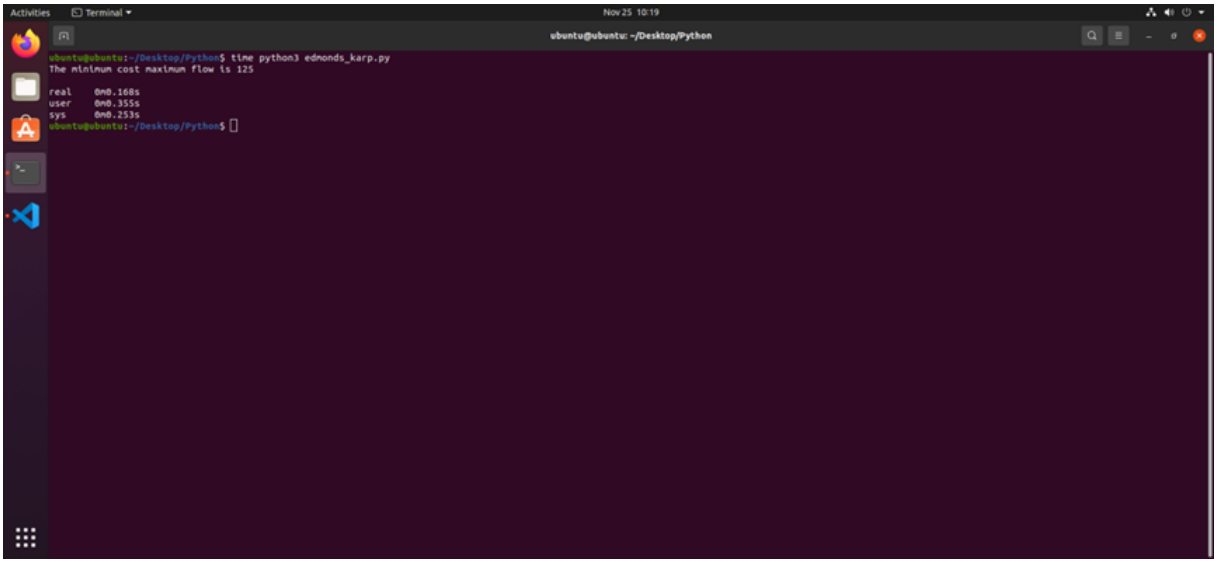
Hình 34

Nạp thông tin và chạy thuật toán

```
84  # Dữ liệu cung cấp
85  start_nodes = np.array([0, 0, 1, 1, 1, 2, 2, 3, 3, 3, 4, 4, 5, 5, 6])
86  end_nodes   = np.array([1, 2, 2, 3, 4, 3, 4, 4, 5, 6, 5, 6, 6, 7, 7])
87  capacities  = np.array([6, 5, 4, 8, 3, 3, 8, 7, 5, 3, 7, 2, 1, 7, 2])
88  unit_costs  = np.array([5, 6, 7, 4, 8, 9, 2, 3, 5, 6, 4, 8, 9, 2, 8])
89  supplies = [8, 0, 0, 0, 0, 0, 0, -8]
90  demand = 8
91
92  graph = nx.DiGraph()
93
94  # Thêm cạnh với dung lượng và chi phí đơn vị
95  for i in range(len(start_nodes)):
96      graph.add_edge(start_nodes[i], end_nodes[i], capacity=capacities[i], cost=unit_costs[i])
97
98  # Thêm nguồn cung và rò rỉ cho từng đỉnh
99  for i, supply in enumerate(supplies):
100     graph.nodes[i]['supply'] = supply
101
102 # Xác định đỉnh nguồn và đỉnh đích
103 source_node = 0
104 sink_node = 7
105
106 # Tính luồng có chi phí tối thiểu sử dụng Edmonds-Karp và Bellman-Ford
107 min_cost_flow = edmonds_karp_min_cost(graph, source_node, sink_node, demand)
108
109 print("The minimum cost maximum flow is", min_cost_flow)
110
```

Hình 35

Kết quả và thời gian chạy



Hình 36: Kết quả

4.3.2 Đánh giá độ hiệu quả của thuật toán dựa vào thời gian

- **Edmonds-Karp**: Sử dụng Bellman-Ford để tìm đường đi ngắn nhất. Trong trường hợp tốt nhất, độ phức tạp thời gian của Edmonds-Karp là $O(VE^2)$, trong đó V là số lượng đỉnh và E là số lượng cạnh trong mạng.
- **Successive Shortest Path (SSP)**: Dựa trên thuật toán Label-correcting để cập nhật luồng và chi phí nhỏ nhất. Độ phức tạp thời gian của SSP là $O(VE^2)$ trong trường hợp xấu nhất. Thuật toán này cải thiện thời gian chạy so với Bellman-Ford khi chỉ cập nhật những đỉnh có sự thay đổi trong quá trình tìm đường đi ngắn nhất.

	Thuật toán Successive Shortest Path (SSP)	**Thuật toán Edmonds-Karp**
Ưu điểm	• Dựa trên thuật toán Label-Correcting để cập nhật luồng và chi phí nhỏ nhất. • Có thể hiệu quả trên đồ thị lớn với cấu trúc đặc biệt. • Linh hoạt và có thể sử dụng để giải cả bài toán luồng tối ưu và bài toán luồng tối ưu với chi phí tối thiểu.	• Dựa trên luồng cực đại và chi phí tối thiểu.. • Cập nhật cả đồ thị còn lại để tìm kiếm đường đi ngắn nhất, giảm thiểu sự phức tạp. • Sử dụng thuật toán Bellman-Ford để tìm đường đi ngắn nhất, có thể xử lý trọng số âm.
Nhược điểm	Không hiệu quả trên đồ thị có trọng số âm, có thể không hiệu quả trên đồ thị lớn.	Độ phức tạp thời gian trong trường hợp xấu nhất là $O(VE^2)$,do phải lặp qua nhiều lần tìm đường đi ngắn nhất.
Độ phức tạp	Xấu nhất $O(V^2E)$	Xấu nhất $O(VE^2)$

Bảng 4: So sánh giữa thuật toán SSP và Edmonds-Karp.

• Chọn lựa giữa Edmonds-Karp và Successive Shortest Path phụ thuộc vào yêu cầu cụ thể của vấn đề và tính chất của đồ thị đầu vào. Nếu trọng số âm là một yếu tố quan trọng, có thể ưu tiên Edmonds-Karp.

Ngược lại, nếu cần giải quyết một loạt các bài toán khác nhau, Successive Shortest Path có thể là lựa chọn linh hoạt hơn.

References

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

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

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