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

MÔ HÌNH HÓA TOÁN HỌC
-CO2011-

Bài tập lớn

"Stochastic Programming"

GVHD: Mai Xuân Toàn
SV: Lê Hoàng Khánh Vinh- 2213963
Trần Đại Việt - 2213951
Nguyễn Duy Vũ - 2213997
Trần Tiến Lực - 2011592
Phạm Hồng Quân - 2114554

TP. HỒ CHÍ MINH, THÁNG 12/2023

Contents

1 Lời mở đầu

Trong hành trình không ngừng mở rộng biên giới của tri thức, chúng tôi, những sinh viên của trường Đại học Bách khoa Thành phố Hồ Chí Minh, khoa Khoa học và Kỹ thuật Máy tính, đã được trao cơ hội quý báu để khám phá và nghiên cứu một lĩnh vực đầy thách thức nhưng cũng không kém phần hấp dẫn: Stochastic Programming. Đề tài này không chỉ mang lại cho chúng tôi những kiến thức chuyên sâu mà còn giúp chúng tôi hiểu rõ hơn về ứng dụng thực tiễn của những lý thuyết toán học trong việc giải quyết các vấn đề phức tạp của thế giới thực. Bài báo cáo này được chia làm 8 chương. Chương 2, 3, 4 và 5 sẽ giới thiệu và trình bày về các khái niệm cơ bản trong Stochastic Programming. Trong chương 6, 7 và 8, bài báo cáo sẽ giới thiệu vấn đề trong thực tế và các cách giải quyết tương ứng sử dụng Stochastic Programming, từ đó chúng ta có thể thấy được tính ứng dụng cao của Stochastic Programming vào thực tiễn.

Ngoài ra, chúng tôi xin bày tỏ lòng biết ơn sâu sắc đến giảng viên hướng dẫn của mình, thầy Mai Xuân Toàn, người đã không chỉ truyền đạt kiến thức mà còn truyền cảm hứng và đam mê nghiên cứu khoa học cho chúng tôi. Sự hướng dẫn tận tâm và kiên nhẫn của thầy đã là nguồn động viên lớn lao, giúp chúng tôi vượt qua mọi khó khăn và thách thức trong suốt quá trình thực hiện đề tài này. Những giờ phút làm việc cật lực trong phòng thí nghiệm, những buổi thảo luận sôi nổi và những đêm trắng trải qua cùng sách vở và dữ liệu đã trở thành một phần không thể thiếu trong cuộc sống đại học của chúng tôi. Chúng tôi tin rằng mỗi giọt mồ hôi, mỗi nỗ lực không mệt mỏi đều là những bước đệm vững chắc cho tương lai của mỗi người và cho sự phát triển của ngành khoa học máy tính nói riêng và ngành kỹ thuật nói chung. Với tất cả lòng kính trọng và biết ơn, chúng tôi kính mong rằng bài báo cáo này sẽ thể hiện được sự cống hiến và đam mê mà chúng tôi đã dành cho đề tài. Chúng tôi cũng hy vọng rằng những kết quả nghiên cứu của mình sẽ góp phần nhỏ vào kho tàng tri thức chung của ngành và mở ra những hướng đi mới cho các nghiên cứu tiếp theo.

2 Danh sách thành viên và công việc

No.	Fullname	Student ID	Problems	% of work
1	Lê Hoàng Khánh Vinh	2213963	- Ứng dụng 1: Stochastic Programming for evacuation planning in disaster response.	25%
2	Trần Đại Việt	2213951	- Soạn thảo LaTex. - Lý thuyết về Stochastic Programming	25%
3	Nguyễn Duy Vũ	2213997	- Tìm hiểu GAMSPy. - Làm phần Industry Manufactoring.	25%
4	Trần Tiến Lực	2011592	- Ứng dụng 2: Stochastic Program in Telecommunication.	12.5%
5	Phạm Hồng Quân	2114554	- Ứng dụng 2: Stochastic Program in Telecommunication.	12.5%

3 Giới thiệu về Stochastic Programming và vấn đề tối ưu

Trong một câu ngạn ngữ của Pháp nói rằng: *"G´erer, c'est pr´evoir "* có nghĩa là *"Nghệ thuật của việc quản lý là sự nhìn thấy trước"*. Đúng như vậy, trong cuộc sống, mỗi chúng ta luôn phải đối mặt với nhiều vấn đề dưới dạng những bài toán và yêu cầu trước mắt đặt ra là phải giải quyết những vấn đề đó một cách tối ưu nhất. Vậy chúng ta tìm ra giải pháp tối ưu đó bằng cách nào?

Đối với những người từ lâu đã quen thuộc với lĩnh vực tối ưu hóa (*Optimization Problem*), họ có thể dễ dàng nêu tên những phương pháp hiệu quả có thể sử dụng như quy hoạch tuyến tính (*Linear Programming*), quy hoạch bậc 2 (*Quadratic Programming*), tối ưu lồi (*Convex Optimization*), tối ưu phi tuyến tính (*Nonlinear Optimization*),. . .

Stochastic Programming thoạt đầu nhìn có thể giống những phương pháp trên nhưng nó không có công thức tổng quát nào thể hiện được vấn đề này. Và bởi lẽ qua các môn như Xác Xuất Thống Kê, chúng ta biết được rằng tương lai là thứ không thể dự đoán một cách chính xác mà nên coi chúng dưới sự ngẫu nhiên và không chắc chắn, từ đó *Stochastic Programming* được phát triển nhờ mục đích tìm ra quyết định tối ưu trong các vấn đề liên quan đến dữ liệu đầu vào không chắc chắn như những việc chuẩn bị xảy ra ở tương lai. Trong thuật ngữ này, ngẫu nhiên (*Stochastic*) trái ngược với thuật ngữ xác định (*Deterministic*), có nghĩa là một số dữ liệu đầu vào là ngẫu nhiên, trong đó các phần khác nhau của vấn đề có thể được mô hình hóa thành các chương trình toán học tuyến tính hoặc phi tuyến tính. Lĩnh vực này, còn được gọi là tối ưu hóa trong điều kiện không chắc chắn (*Optimization Under Uncertainty*), đang phát triển nhanh chóng với sự đóng góp của nhiều ngành nghề như các hoạt động nghiên cứu, kinh tế, toán học, xác suất và thống kê,...

3.1 Giới thiệu về Stochastic Programming và sự không chắc chắn

- Mathematical Optimization

  Tối ưu hóa toán học là việc tập hợp các nguyên tắc và phương pháp toán học được sử dụng để giải các bài toán định lượng trong nhiều ngành, bao gồm vật lý, sinh học, kỹ thuật, kinh tế và kinh doanh. Tập trung chủ yếu vào việc đưa ra quyết định (*Decision Making*) để có thể tối ưu hóa tối đa (*Maximize*) hoặc tối thiểu (*Minimize*) phần định lượng (*Objective Function*) đó. Chủ đề này phát triển từ việc nhận thức rằng các vấn đề định lượng trong các ngành tuy khác nhau nhưng đều có những yếu tố toán học quan trọng chung. Vì điểm chung này, nhiều vấn đề có thể được xây dựng và giải quyết bằng cách sử dụng tập hợp các ý tưởng và phương pháp thống nhất tạo nên lĩnh vực tối ưu hóa (*Optimization*).

- Uncertainty

  Trong một số bài toán tối ưu, có những trường hợp mà tham số đầu vào của bài toán đó không được xác định rõ ràng mà được xét dưới dạng không chắc chắn, những tham số đó có nhiều đầu ra với phân phối xác suất khác nhau và được xác định từ trước. Thấy được việc đó, chúng ta có thể biểu diễn chúng dưới dạng các biến ngẫu nhiên.

  Có thể lấy một số trường hợp như: Chi phí sản xuất và phân phối các mặt hàng thường sẽ bị ảnh hưởng bởi chi phí nguyên liệu (phần chi phí dễ bị thay đổi theo giá cả thị trường và rất khó để có thể dự đoán trước), nhu cầu thị trường trong tương lai của từng loại hàng hóa hay giá thu mua của các loại cây trồng dựa trên điều kiện thời tiết,...

  Uncertainty sẽ được biểu diễn dưới dạng các biến ngẫu nhiên với kết quả được biểu hiện bằng ω. Tập hợp các kết quả sẽ được biểu thị bằng Ω.

- Stochastic Programming (SP)

  Stochastic Programing là một bộ khung để mô hình hóa các vấn đề tối ưu hóa liên quan

đến sự không chắc chắn. Trong một chương trình ngẫu nhiên, một số hoặc tất cả các tham số bài toán là không chắc chắn (*Random Parameters*) nhưng tuân theo phân bố xác suất đã biết. Điều này trái ngược với tối ưu hóa tất định (*Deterministic Optimization*), trong đó tất cả các tham số bài toán được giả định là đã biết chính xác.

Mục tiêu của *Stochastic Programing* là tìm ra quyết định tối ưu hóa các tiêu chí được lựa chọn trong khi xét đến đến sự không chắc chắn của các tham số bài toán một cách thích hợp. Bởi vì nhiều quyết định trong cuộc sống xung quanh chúng ta liên quan đến sự không chắc chắn, Stochastic Programing đã được ứng dụng trong nhiều lĩnh vực khác nhau từ tài chính, vận tải đến tối ưu hóa năng lượng,...

- Decision Variable

  Biến quyết định (*Decision Variable*) là một ẩn số trong bài toán tối ưu hóa. Nó đại diện cho số lượng hoặc biểu tượng và có thể đảm nhận bất kỳ tập hợp giá trị nào, liên quan đến vấn đề quyết định hoặc cơ hội. Các biến này còn được gọi là biến thiết kế hoặc biến thao tác. Chúng thường được dùng để đo lường nguồn lực được sử dụng hoặc mức độ của các hoạt động khác nhau sẽ được thực hiện, chẳng hạn như số lượng hàng hóa được sản xuất. Giá trị của các biến quyết định xác định giá trị của hàm mục tiêu của bài toán cần giải quyết. Với mục tiêu là việc tối ưu hóa, việc tìm giá trị của các biến quyết định là mục tiêu để giải quyết vấn đề của bài toán.

- Deterministic Constraints

  Các ràng buộc xác định (*Deterministic Constraints*) là các điều kiện mà giải pháp cho các vấn đề tối ưu hóa phải được thỏa mãn, trong đó mối quan hệ giữa các biến của bài toán phải được xác định và không được mang tính ngẫu nhiên. Những ràng buộc này có thể có nhiều loại, chủ yếu là ràng buộc đẳng thức, ràng buộc bất đẳng thức và ràng buộc số nguyên. Tập hợp các lời giải thỏa mãn (*Candidate Solution*) tất cả các ràng buộc xác định này được gọi là tập khả thi (*Feasible Set*).

  Ví dụ: Trong một bài toán tối ưu hóa đơn giản, bạn muốn cực tiểu hóa một hàm mục tiêu xác định, tuân theo một số ràng buộc nhất định. Những ràng buộc này có thể là "$x1 + x2 \leq 10$" (ràng buộc bất đẳng thức) và "$x1 - x2 = 5$" (ràng buộc đẳng thức). Giải pháp cho vấn đề tối ưu hóa phải thỏa mãn các ràng buộc xác định này.

- Probabilistic Constraints

  Ràng buộc xác suất (*Probabilistic Constraints*) là một loại ràng buộc trong bài toán tối ưu hóa có liên quan đến sự không chắc chắn trong một số hoặc tất cả các tham số của bài toán. Mục tiêu là tìm ra quyết định tối ưu hóa hàm mục tiêu với điều kiện là phải thỏa mãn một số ràng buộc với xác suất cao. Ví dụ: ràng buộc xác suất có thể được sử dụng để mô hình hóa nhu cầu về một sản phẩm, chi phí sản xuất, độ tin cậy của hệ thống hoặc rủi ro của danh mục đầu tư.

- Stochastic Linear Programs

  Quy hoạch tuyến tính (*Stochastic Linear Programs*) với hàm mục tiêu cần được tối ưu hóa của chương trình này là một hàm tuyến tính và một số tham số đầu vào của nó được xét dưới dạng không chắc chắn với nhiều giá trị đầu ra với những xác suất khác nhau.

- Recourse Programs

Hành động truy đòi (*Recourse Programs*) là một quyết định có thể được đưa ra sau khi chúng ta giải quyết được phần nào các tham số được xét dưới sự không chắc chắn.

Ví dụ: trong bài toán quy hoạch ngẫu nhiên hai giai đoạn (*Two-Stage Stochastic Programming*), quyết định ở giai đoạn đầu tiên được đưa ra trước khi sự không chắc chắn xảy ra và quyết định ở giai đoạn thứ hai là hành động truy đòi phụ thuộc vào kết quả của sự kiện. Hành động truy đòi có thể được coi là một cách điều chỉnh hoặc sửa chữa quyết định ở giai đoạn đầu để ứng phó với sự không chắc chắn. Mục tiêu của quy hoạch ngẫu nhiên có truy đòi (*Stochastic Programming with Recourse*) là tìm ra quyết định tối ưu ở giai đoạn đầu nhằm giảm thiểu tổng chi phí dự kiến, bao gồm chi phí của hành động truy đòi.

3.2 Các khái niệm cơ bản, giả định của Stochastic Programming

Quy hoạch tuyến tính với các tham số ngẫu nhiên là một loại vấn đề tối ưu hóa trong đó một số tham số không được xác định chính xác mà được xét dưới dạng ngẫu nhiên. Điều này tạo ra sự không chắc chắn cho vấn đề, từ đó khiến cho việc giải quyết vấn đề trở nên phức tạp hơn.

- **Biến ngẫu nhiên tuân theo phân phối đồng nhất**
  Biến ngẫu nhiên tuân theo phân phối đồng nhất (Uniform/Random Variables Following Distributions) là một loại biến ngẫu nhiên có thể nhận bất kỳ giá trị nào trong một phạm vi xác định với xác suất bằng nhau. Phạm vi được xác định bởi giá trị tối thiểu và tối đa và mọi giá trị trong phạm vi này đều có khả năng là kết quả của bài toán. Kiểu phân phối này thường được sử dụng trong mô phỏng và mô hình hóa để biểu thị các tham số hoặc biến không chắc chắn có cơ hội nhận bất kỳ giá trị nào trong một phạm vi nhất định như nhau. Trong quy hoạch tuyến tính, các biến ngẫu nhiên tuân theo phân phối đồng nhất thường được dùng để biểu diễn sự không chắc chắn của các tham số trong bài toán. Ngoài sự phân bố cụ thể của các biến, tham số này sẽ phụ thộc vào bản chất bài toán và bản chất của sự không chắc chắn .

Định nghĩa 1: Quy hoạch tuyến tính với tham số ngẫu nhiên (*Linear program (LP) with random parameters - SLP*)

$$\text{Minimize} \quad Z = g(x) = f(x) = c^T \cdot x, \quad \text{subject to} \quad A \cdot x = b, and \quad T \cdot x \geq h \tag{3.1}$$

Với $x = x_1, x_2, \ldots, x_n$ (Biến quyết định - Decision Variable)

Ma trận thực A và vector B dùng để biểu diễn các ràng buộc xác định (*Deterministic Constraints*)

Các tham số ngẫu nhiên T, h trong ràng buộc $T \cdot x \geq h$ thể hiện các ràng buộc xác xuất của bài toán.

Ví Dụ 1

$$\text{Minimize} \quad z = x_1 + x_2, \quad s.t. \quad x_1 \geq 0 \quad x_2 \geq 0 \tag{3.2}$$

$$\begin{cases} \omega_1 \cdot x_1 + x_2 \geq 7 \\ \omega_2 \cdot x_1 + x_2 \geq 4 \end{cases} \tag{3.3}$$

Trong đó những tham số ω_1, ω_2 là biến ngẫu nhiên tuân theo phân phối đồng nhất.

- Trong bài này, ta sẽ cho các tham số ω_1, ω_2 được phân phối đồng nhất (Uniform). Cụ thể, ω_1 tuân theo phân phối đồng nhất trong khoảng từ 1 đến 4 ($\omega_1 \sim Uniform(1, 4)$), và ω_2 tuân theo phân phối đồng nhất trong khoảng từ 1/3 đến 1 ($\omega_2 \sim Uniform(1/3, 1)$).

Vậy chúng ta sẽ xử lý bài toán này như thế nào?

Sẽ có những cách cách tiếp cận chính sau đây:

1. **Wait-And-See**

   Cách tiếp cận "Wait-and-see" là một phương pháp chiến lược thường được sử dụng trong quá trình ra quyết định. Cách tiếp cận này giống như một ván cờ, trong đó người ta không thể di chuyển cho đến khi hành động của đối thủ rõ ràng. Bằng cách đó, người ta có thể đưa ra quyết định sáng suốt hơn dựa trên nước đi của đối thủ.

   Cách tiếp cận này thường được sử dụng khi có sự không chắc chắn về kết quả thu được khi chúng ta có nhiều lựa chọn khác nhau và người ta thích trì hoãn hành động hơn là mạo hiểm đưa ra một quyết định sai lầm. Nó giống như nói: "Hãy chờ xem điều gì sẽ xảy ra trước khi chúng ta quyết định phải làm gì tiếp theo". Cách tiếp cận này có thể được áp dụng trong nhiều bối cảnh khác nhau, chẳng hạn như kinh doanh, chính trị, quyết định cá nhân,... Tuy nhiên, điều quan trọng cần lưu ý là mặc dù cách tiếp cận này đôi khi có thể có ích trong việc tránh đưa ra các quyết định vội vàng nhưng nó cũng có thể dẫn đến việc bỏ lỡ các cơ hội nếu lạm dụng hoặc sử dụng không phù hợp.

   - Khi áp dúng chiến lược "Wait-and-see" trong bài toán này, chúng ta phải đợi cho cho đến khi có thêm điều kiện để xác định chính xác giá trị ω_1, ω_2 hoặc xét từng trường các giá trị mà ω_1, ω_2 có thể nhận sau đó đưa ra để đưa ra quyết định đúng đắn nhất.
   - Giả sử trong trường hợp ta xác định được $\omega_1 = 1$ và $\omega_2 = 1$
     - Lúc này tất cả ràng buộc của bài toán sẽ là ràng buộc xác định và ta có thể dễ dàng tìm ra được quyết định tối ưu nhất cho bài toán

$$\text{Minimize} \quad z = x_1 + x_2, \quad s.t. \quad x_1 \geq 0 \quad x_2 \geq 0 \tag{3.4}$$

$$\begin{cases} x_1 + x_2 \geq 7 \\ x_1 + x_2 \geq 4 \end{cases} \tag{3.5}$$

     - Vậy quyết định hay nghiệm tối ưu của bài toán lúc này sẽ là:

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 7 \\ 0 \end{pmatrix} \tag{3.6}$$

2. **Guess At Uncertainty**

   "Đoán trước sự không chắc chắn" là một chiến lược được sử dụng trong quy hoạch ngẫu nhiên, đặc biệt khi xử lý các tham số không chắc chắn hoặc ngẫu nhiên. Cách tiếp cận này liên quan đến việc đưa ra những phỏng đoán có căn cứ về các tham số không chắc chắn trong một bài toán theo những chiến lược và mục đích phù hợp, từ đó chúng ta sẽ xác định được sơ bộ các tham số ngẫu nhiên và lúc này các ràng buộc ngẫu nhiên sẽ được xác định và trở thành các ràng buộc xác định từ đó việc tìm ra các quyết định tối ưu của các biến quyết định sẽ trở nên dễ dàng hơn.

   Trong SLP, một số tham số của bài toán có thể được coi là không chắc chắn. Để xử lý sự không chắc chắn này, người ta có thể lựa chon cách "đoán" các giá trị hợp lý cho các tham số này. Những dự đoán này có thể dựa trên nhiều chiến lược khác nhau như:

- **Unbiased**

  Chọn giá trị trung bình (Mean Value) cho từng tham số ngẫu nhiên (Random Parameters).
  Cách lựa chọn này sẽ giả định rằng giá trị kỳ vọng của các tham số ngẫu nhiên sẽ là giá trị có khả năng xảy ra cao nhất.

  – Nếu bạn đang phải đối mặt với những vấn đề liên quan đến chuỗi cung ứng và các tham số ngẫu nhiên hay không chắc chắn trong bài toán như nhu cầu về sản phẩm nào đó. Thì lúc này chiến lược thích hợp nhất để lựa chọn giá trị cho tham số ngẫu nhiên đó sẽ là chọn giá trị trung bình của tham số đó.

  – Tuy nhiên, điều quan trọng cần lưu ý là chiến lược này giả định rằng tương lai sẽ giống như quá khứ, nghĩa là những sự kiện sắp xảy ra phải không ảnh hưởng nhiều đến những tham số không chắc chắn trong bài toán, điều này có thể không phải lúc nào cũng đúng. Do đó, nó thường được sử dụng kết hợp với các phương pháp khác (như phương pháp bi quan (Pessimistic) hoặc lạc quan (Optimistic) để có được nhiều giải pháp khả thi và đưa ra quyết định sáng suốt hơn.

  – *Ví dụ*

  Thử áp dụng chiến lược Unabised cho ví dụ 1, lúc này ta sẽ có được:

$$\omega \sim uniform(a,b) => \widehat{\omega} = \frac{a+b}{2} \tag{3.7}$$

  từ đó ta có được $\widehat{\omega} = (\frac{5}{2}, \frac{2}{3})$ => Bài toán của chúng ta sẽ trở thành

$$\begin{cases} Minimize \quad z = x_1 + x_2 \\ x_1 \geq 0 \quad x_2 \geq 0 \\ \\ \frac{5}{2}x_1 + x_2 \geq 7 \\ \\ \frac{2}{3}x_1 + x_2 \geq 4 \end{cases} \tag{3.8}$$

  Kết quả tối ưu của ví dụ khi áp dụng Unbiased là:

$$\begin{cases} z = \frac{50}{11} \\ x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \frac{18}{11} \\ \frac{32}{11} \end{pmatrix} \end{cases} \tag{3.9}$$

- **Pessimistic**

  Trong lập trình ngẫu nhiên, thuật ngữ "Pesimictic" đề cập đến một chiến lược, trong đó, bạn sẽ chọn các giá trị trong trường hợp xấu nhất cho các tham số ngẫu nhiên ω.
  Cách tiếp cận này thường được sử dụng khi xử lý sự không chắc chắn trong quá trình ra quyết định. Đó là một chiến lược thận trọng nhằm chuẩn bị cho tình huống xấu nhất, khó khăn nhất có thể xảy ra. Bằng cách này, bất kể giá trị thực tế của ω là bao nhiêu, bạn vẫn chuẩn bị cho trường hợp xấu nhất. Điều quan trọng cần lưu ý là những gì tạo thành giá trị của tham số ngẫu nhiên trong "trường hợp xấu nhất", giá trị này phụ thuộc vào vấn đề cụ thể của bài toán đang giải quyết hay tác động của tham số đó vào bài toán.
  Sau đây là cách chúng ta giải quyết bài toán bằng cách dùng giá trị tệ nhất (Worst Case value)

- Xác định phạm vi của sự không chắc chắn: Xác định giới hạn trên và dưới của tham số không chắc chắn. Ví dụ: nếu bạn đang giải quyết vấn đề về nhu cầu trong một vấn đề về chuỗi cung ứng, bạn có thể biết rằng nhu cầu có thể thay đổi giữa giới hạn dưới và giới hạn trên nhất định.
- Xác định hàm mục tiêu: Hàm mục tiêu phải được xác định theo cách phản ánh được tình huống xấu nhất. Điều này thường liên quan đến việc tối đa hóa chi phí hoặc giảm thiểu lợi ích.
- Giải quyết vấn đề tối ưu hóa: Giải quyết vấn đề tối ưu hóa với hàm mục tiêu được xác định ở bước 2. Giải pháp cho vấn đề này sẽ cho bạn quyết định thực hiện tốt nhất trong trường hợp xấu nhất.
- Hãy nhớ rằng, giá trị trong "trường hợp xấu nhất" có thể phụ thuộc vào vấn đề cụ thể đang cần được giải quyết và tác động của các tham số trong bài toán đó. Điều quan trọng cần lưu ý là cách tiếp cận này mang tính bảo thủ và không phải lúc nào cũng mang lại giải pháp hiệu quả nhất nếu trường hợp xấu nhất khó xảy ra.

- **Optimistic**

  Thuật ngữ "Optimistic" là một trong những chiến lược phổ biến để xử lý với các tham số ngẫu nhiên.

  Ngược lại với chiến lược "Pessimistic" lúc này ta sẽ chọn giá trị của các tham số ngẫu nhiên là giá trị sẽ giúp cho hàm mục tiêu của bài toán đang xử lý cho ra kết quả tốt nhất theo yêu cầu hoặc cho ra những điều kiện tốt nhất phù hợp cho bài toán đó. Nói cách khác, chiến lược "Optimistic" sẽ chọn giá trị cho các tham số ngẫu nhiên sao cho giá trị của tham số ngẫu nhiên sẽ mang lại kết quả tốt nhất mà ta có thể thu được.

  Tuy nhiên cũng giống như chiến lược "Pessimistic" chiến lược này sẽ mang tính bảo thủ nhiều hơn và kết quả cuối cùng sẽ có thể là một giải pháp không hiệu quả cho bài toán nếu bài toán bị rơi vào các trường hợp có xác xuất xảy ra ít hơn.

3. **Probabilistic Constraints** Giống những chiến lược trên, chiến lược này sẽ giúp chúng ta giải quyết được các vấn đề về các tham số ngẫu nhiên. Chúng ta sẽ tìm hiểu kỹ hơn về chiến lược này ở phần tiếp theo.

4 One-stage Stochastic linear programming - No recourse

4.1 Mô hình hóa lại vấn đề

Trong phần này chúng ta sẽ đưa ra các ví dụ trực quan để tìm hiểu về mô hình quy hoạch tuyến tính một giai đoạn không có sự truy đòi

Định nghĩa 2: Quy hoạch tuyến tính 1 giai đoạn và không có sự truy đòi (*SLP with one-stage (No recourse) : 1-SLP*)

Cho rằng mô hình LP (α) sau đây sẽ được tham số hóa bởi một vector ngẫu nhiên là vector α

$$\begin{cases} Minimize \quad Z = g(x) \quad = f(x) = c^T \cdot x = \sum_{j=1}^{n} c_j \cdot x_j \\ \qquad s.t. \quad A \cdot x = b, \quad \text{(Certain Constraints)} \\ \qquad và \quad T \cdot x \geq h \quad \text{(Stochastic Constraints)} \end{cases} \tag{4.1}$$

Trong mô hình trên, ta sẽ có được những nhận định sau:

1. Ma trận T và vector h sẽ mang tính ngẫu nhiên trong mình và chính nó sẽ thể hiện sự không chắc chắn của bài toán.

$$T(\alpha) \cdot x \geq h(\alpha) \quad \Longleftrightarrow \quad a_1 \cdot x_1 + a_2 \cdot x_2 + \cdots + a_n \cdot x_n \quad \geq \quad h(\alpha) \tag{4.2}$$

2. Giá trị của T và h sẽ không được xác định trước khi mô hình bài toán được thiết lập và h(α) sẽ chỉ phụ thuộc vào vector α ngẫu nhiên.

3. **Uncertainty**
Sự không chắc chắn sẽ được thể hiện ở sự phân bố xác suất của tham số ngẫu nhiên là vector α, từ đó ta thấy được bài toàn sẽ trở thành một bài toán quy hoạch tuyến tính với các ràng buộc xác định nếu vector α được xác định.

   - Trong dạng bài này, chúng ta phải giải quyết và xác định các biến quyết định $x = (x_1, x_2, \ldots, x_n)$ trước khi xác định được vector $\alpha \in \Omega$
   - Ngoài ta chúng ta còn thường xác định giới hạn trên và giới hạn dưới cho biến quyết định X, thường là biến quyết định sẽ được xác định trong một miền như sau $\chi = \{x \in R^n : l \leq x \leq u\}$

4.2 Các cách tiếp cận để giải quyết bài toán

Fundemental assumption Bởi vì chúng ta đã biết về phân phối xác xuất của dữ liệu hay là của các tham số ngẫu nhiên. Từ đó việc xây dựng các ràng buộc xác suất sẽ là một trong những cách tiếp cận phổ biến trong bài toán quy hoạch tuyến tính một giai đoạn không truy đòi (No Recourse).
The Scenario Analysis Đây là một cách tiếp cận khác để giải quyết sự không chắc chắn, trong cách tiếp cận này ta sẽ nhận định rằng bài toán sẽ có một số lượng kết quả tối ưu có hạn tùy thuộc vào giá trị đầu ra của các tham số ngẫu nhiên. Từ đó ta sẽ phân tích và đánh giá các quyết định đó và đưa ra quyết định cuối cùng. Mỗi một quyết định tối ưu khả thi sẽ được gọi là "Scenario". Cách tiếp cận này tuy không có độ chính xác cao nhưng nó lại rất hữu dụng trong các bài toán trong cuộc sống.

4.2.1 Cách tiếp cận 1: Sử dụng ràng buộc về xác suất và rủi ro chấp nhận được (Chance Constraints and Acceptable Risk)

Trong phần này chúng ta sẽ đề cập về cách tiếp cận để giải quyết vấn đề tối ưu hóa của Stochastic Program với ràng buộc về xác suất.

Định nghĩa 3: Quy hoạch tuyến tính 1 giai đoạn với ràng buộc về xác suất (*Stochastic LP or 1-SLP with Probabilistic Constraints*)
Trong đó vector α sẽ là vector hệ số ngẫu nhiên của bài toán và sẽ là hệ số biểu diễn ràng buộc xác suất $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$

$$\begin{cases} SP: & \min_x Z, \quad Z = f(x) = c^T \cdot x = \sum_{j=1}^{n} c_j \cdot x_j, \quad c_j \in \mathbb{R} \\ s.t. & A \cdot x = b \quad (x = (x_1, x_2, \ldots, x_n) \quad Decision\ variable\ của\ mô\ hình) \\ and & P[T \cdot x = a \cdot x \leq h] \leq (1 - p) \quad (0 < p < 1) \end{cases} \tag{4.3}$$

Trong mô hình trên, chúng ta sẽ thay điều kiện "Stochastic Constraints" cơ bản thành "Probabilistic Constraints" $P[T \cdot x \geq h] \geq p$ với p là mức độ chính xác do chính người giải quyết

bài toán quy định.

Ngoài ra hệ số $1-p$ trong mô hình sẽ là mức độ rủi ro chấp nhận được của bài toán, nói một cách khác $1-p$ là rủi ro cao nhất có thể chấp nhận được của quyết định được đưa ra

- Mức độ rủi ro này sẽ được định nghĩa như sau

$$acceptable\ risk \quad r_x := P[Not(T \cdot x \geq h)] = P[T \cdot x \leq h] \leq 1 - p \tag{4.4}$$

  Như vậy rủi ro chấp nhận được r_x là phần xác suất xuất hiện sau khi ta quyết định được biến quyết định của bài toán, từ đó đánh giá quyết định đó thỏa mãn được bao nhiêu ràng buộc xác suất của mô hình đã biết phân phối xác suất của các biến ngẫu nhiên.

Ví dụ

Chúng ta xem xét n cơ hội đầu tư với tỷ suất lợi nhuận ngẫu nhiên $R_1, R_2, \ldots, R_n$ trong năm tới. Chúng ta có số vốn ban đầu nhất định và mục tiêu của chúng ta là sẽ tối đa hóa số lợi nhuận thu về được sau khi đầu tư, với điều kiện trong tình huống việc đầu tư bị thu lỗ, phần thu lỗ sẽ không quá p số tiền vốn ban đầu, trong đó $p \in (0, 1)$. Yêu cầu như vậy được gọi là ràng buộc rủi ro đã giới thiệu ở trên

Gọi $(x_1, x_2, \cdot, x_n)$ là phần vốn đầu tư của chúng ta vào n tài sản. Sau một năm, khoản đầu tư của chúng ta thay đổi giá trị theo tỷ lệ có thể được biểu thị bằng

$$g(x, R) = \sum_{i=1}^{n} R_i \cdot x_i \tag{4.5}$$

Chúng ta sẽ xây dựng được mô hình của bài toán trên như sau

$$\begin{cases} Max \quad \sum_{i=1}^{n} E[R_i] \cdot x_i \\ s.t.\ P[\sum_{i=1}^{n} R_i \cdot x_i \geq r_x] \geq p \\ \sum_{i=1}^{n} x_i = 1 \\ x \geq 0 \end{cases} \tag{4.6}$$

- Nếu chọn r_x là -0.1 thì trong trường hợp kết quả đầu tư không được như mong muốn, chúng ta thua lỗ nhiều nhất là 10% so với số vốn ban đầu bỏ ra

4.2.2 Cách tiếp cận 2: Sử dụng Stochastic Constraints ($T(\alpha) \cdot x \leq h(\alpha)$)

Cách tiếp cận này sẽ chú trọng hơn về việc phân tích từng kịch bản sẽ xảy ra dựa vào các giá trị của các tham số ngẫu nhiên từ đó thống kê và phân tích để đưa ra được quyết định tối ưu đúng nhất.

Xét từng kịch bản hay là từng giá trị tiềm năng của tham số ngẫu nhiên $(T^s; h^s), s = 1, 2, \ldots, S$, chúng ta sẽ tìm ra quyết định tối ưu nhất của mô hình trong từng kịch bản đó

$$Minimize \quad \{f(x) = c^T \cdot x; \quad s.t. \quad A \cdot x = b, T^s \cdot x \leq h^s\} \tag{4.7}$$

- Điểm mạnh: Mỗi kịch bản chúng ta cần giải quyết đều là LP cho nên việc tìm quyết định tối ưu cho từng kịch bản sẽ trở nên dễ dàng hơn
- Điểm hạn chế: Nếu số lượng kịch bản tăng lên, ta sẽ khó mô hình hóa lại được cách giải bài toán này.

5 Generic Stochastic Programming (GSP) with RECOURSE

Chúng ta đã thấy được một mô hình cơ bản của quy hoạch tuyến tính ngẫu nhiên (Stochastic Programming) là một mô hình của bài toán quy hoạch tuyến tính với tất cả hay một vài hệ số bị thay thế bởi các tham số ngẫu nhiên, từ đó mô hình sẽ trở nên phức tạp hơn và khó được giải quyết, mặt khác chúng ta có thể biết được phân phối xác suất của các biến ngẫu nhiên đó. Lúc này chúng ta sẽ tạo nên một cách tiếp cận cho mô hình này. Cho rằng những biến quyết định sẽ phải được quyết định trước khi biết chính xác được giá trị của các tham số ngẫu nhiên hay là người ra quyết định biết được chính xác những giá trị của các tham số ngẫu nhiên đó, vậy lúc này mô hình sẽ trở nên khó để có thể được xác định rõ ràng. Trong hoàn cảnh chứa nhiều sự không chắc chắn như vậy, để có thể hiểu được ý nghĩa của sự tối ưu hay tính khả thi của quyết định và thiết lập mô hình của bài toán, chúng ta cần phải bổ sung thêm nhiều khái niệm sau đây.

Trước hết chúng ta sẽ tạo nên mô hình đơn giản của bài toán:

$$\begin{cases} \textit{Minimize} \quad c^T \cdot x \\ s.t. \quad A \cdot x = b \\ T(\omega) \cdot x \geq h(\omega) \\ x \in \mathbb{R}^{n_1} \end{cases} \tag{5.1}$$

Trong đó, ma trận A là một ma trận xác định và b là một vector xác định. Bên cạnh đó, $T(\omega)$ và p(ω) là ma trận và vector ngẫu nhiên của bài toán.

Như đã đề cập từ trước, mô hình này chưa xác định rõ ràng và nếu chúng ta xét nó dưới dạng một mô hình tối ưu hóa thì tối thiểu rằng quyết định của bài toán hay các Decision Variable phải được quyết định trước khi ω được xác định chính xác. Tuy nhiên, nếu cho biến quyết định x phụ thuộc vào giá trị xác định $\bar{\omega}$ của vector ngẫu nhiên ω thì quyết định tối ưu của bài toán sẽ mang tính ngẫu nhiên, chúng ta sẽ ký hiệu nó là $V(\omega)$. Cách tiếp cận để đưa ra giả định này là mô hình Wait-and-see, nói cách khác, biến quyết đinh của bài toán này sẽ được xác định theo quy tắc $\omega \mapsto x^*(\omega)$ và $x^*(\bar{\omega})$ sẽ là quyết định tối ưu trong trường hợp $\omega = \bar{\omega}$ xảy ra. Từ đó ta thấy rằng sử dùng các kiến thức phân phối về xác suất sẽ là cách tiếp cận hợp lí để mô hình lại bài toán và đó cũng chính là hướng tiếp cận để giải quyết bài toán quy hoạch ngẫu nhiên sử dụng truy đòi.

Ta sẽ thiết lập lại mô hình như sau:

Định nghĩa 4: Quy hoạch tuyến tính 2 giai đoạn (*Stochastic program in two stages (generic 2-SP problem)*)

$$2 \text{ - } SP: \quad \min_x g(x) \quad với \quad g(x) = f(x) + E_\omega[v(x,\omega)] \tag{5.2}$$

Trong đó

- $x = x_1, x_2, \ldots, x_{n_1}$ là biến quyết định của giai đoạn đầu tiên
- f(x) có thể là hàm tuyến tính hoặc không và là một phần của hàm mục tiêu g(x)
- Giá trị trung bình (The Mean) của Q(x) là $E_\omega[v(x,\omega)]$ với $v : \mathbb{R}^n \times \mathbb{R}^s \mapsto \mathbb{R}$ và nó sẽ phụ thuộc vào kịch bản ω. Q(x) sẽ là giá trị tối ưu của từng kịch bản ở giai đoạn 2

$$\min_{y \in \mathbb{R}^{n_2}} \quad | \quad T \cdot x + W \cdot y = h \tag{5.3}$$

- Vector $y = y(\omega)$ trong mô hình này sẽ được gọi là biến quyết định truy đòi (Recourse Decision Variable)

Tóm lại, chúng ta cần phải đạt được mục đích là tối ưu hóa hàm mục tiêu g(x) trong khi phải thỏa mãn $W \cdot y(\omega) = h(\omega) - T(\omega) \cdot x$

- W sẽ được gọi là ma trận truy đòi $m \times n_2$ (Recourse Matrix) và vector q sẽ là vector đại diện cho chi phí truy đòi (Recourse Cost) và sẽ có cùng kích thước với y, $y = y(\omega) \in \mathbb{R}^{n_2}$ và ma trận W sẽ được gọi là ma trận truy đòi và E_ω sẽ đại diện cho giá trị kỳ vọng ứng với sự ngẫu nhiên của vector ω

Tiếp theo, chúng ta sẽ làm rõ về mô hình vấn đề truy đòi (Recourse Modeling Issues)

- Sử dụng hành động truy đòi (Recourse Action) bản chất là khi quyết định hay biến quyết định của mô hình được quyết định giá trị xác định trước khi chúng ta hay người đưa ra quyết định biết được chính xác giá trị của các tham số ngẫu nhiên, hành động này sẽ khiến cho các biến quyết định trở nên Infeasible và chúng có khả năng sẽ vi phạm các ràng buộc xác suất của mô hình. Và để cho việc quyết định trước không bị vi phạm những ràng buộc đó, chi phí bồi thường hay bổ sung sẽ được thêm vào mô hình. Và để trực quan hơn, ta sẽ định nghĩa hàm chi phí bổ sung Q(x) như sau

$$\begin{cases} Q(x) & = & E_\omega \bar{v}(x,\omega), & x \in \mathbb{R}^{n_1}, \\ \bar{v}(x,\omega) & = & \textit{Minimize} \quad q \cdot y & : \quad W \cdot y \geq h(\omega) - T(\omega) \cdot x, \quad y \in \mathbb{R}^{n_2} \end{cases} \tag{5.4}$$

- Tùy yêu cầu của mô hình ma trận truy đòi W mà vector chi phí bổ sung q sẽ có giá trị khác nhau. Tóm lại, truy đòi sẽ là phần chi phí bổ sung mà chúng ta cần phải tối thiểu hóa sao cho trong bất cứ kịch bản nào, quyết định tối ưu sẽ thỏa mãn được các ràng buộc xác suất của mô hình

6 Two-Stage Stochastic linear programming (2-SLP)

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

Mô hình truy đòi ở phần trên có thể khái quát hóa lại bằng cách chấp nhận ma trận truy đòi W và vector chi phí bổ sung q sẽ phụ thuộc vào ω. Trong trường hợp đó, chúng ta thu được công thức chung của mô hình truy đòi tuyến tính hai giai đoạn với các biến truy đòi liên tục

$$\begin{cases} \textit{Optimize:} \quad c^T \cdot x + \bar{Q}(x) & : \quad A \cdot x = b, x \in \mathbb{R}^{n_1} \\ \bar{Q}(x) = E_\omega \bar{v}(x,\omega) \\ \bar{v}(x,\omega) = \textit{Optimize} \quad q(\omega) \cdot y & : \quad W(\omega) \cdot y = h(\omega) - T(\omega) \cdot x, y \in \mathbb{R}^{n_2} \end{cases} \tag{6.1}$$

- Ta dùng n_1 để biểu diễn kích thước của vector của giai đoạn 1 là biến x, m_1 là số lượng ràng buộc của giai đoạn thứ nhất, n_2 là kích thước của vector ứng với hành đồng truy đòi của biến y và m_2 sẽ là số lượng ràng buộc của giai đoạn 2. Từ đó, c sẽ là một vector có kích thước n_1, A sẽ là một ma trận có kích thước $m_1 \times n_1$, b là một vector có kích thước m_1, $q(\omega)$ là một n_2-vector ngẫu nhiên, $W(\omega)$ là một ma trận $m_2 \times n_2$ ngẫu nhiên, $h(\omega)$ là m_2-vector ngẫu nhiên và ma trận ngẫu nhiên $T(\omega)$ có kích thước $m_2 \times n_1$

Từ đây chúng ta sẽ đưa ra được mô hình **two-stage** Stochastic LP có truy đòi

Định nghĩa 5: Two-stage Stochastic LP With Recourse : 2 - SLPWR

$$\begin{cases} \textit{2 - SLP:} \quad \min_{x\in X} c^T \cdot x + \min_{y(\omega)\in Y} E_\omega[q \cdot y] \\ \textit{hay trong trường hợp tổng quát hơn} \\ \textit{2 - SLP:} \quad \min_{x\in X, y(\omega)\in Y} E_\omega[c^T \cdot x + v(x,\omega)] \\ \textit{với} \quad v(x,\omega) := q \cdot y \end{cases} \tag{6.2}$$

$$\textit{S.t.}$$

$$\begin{cases} A \cdot x = b \quad \textit{First stage Constraints} \\ T(\omega) \cdot x + W \cdot y(\omega) = h(\omega) \quad \textit{Second stage Constraints} \end{cases}$$

- Hàm mục tiêu của mô hình này là một hàm lớn và được chia ra thành các hàm nhỏ hơn
  1. Hàm tuyến tính $f(x) = c^T \cdot x$
  2. Hàm xác suất (có chứa các tham số ngẫu nhiên) $v(x,\omega)$ liến quan đến từng kịch bản ω khác nhau.
- $y = y(x,\omega) \quad \in \mathbb{R}^{n_2}$ là biến được sinh ra từ hành động truy đòi của mô hình với quyết định x và giá trị được xác định sau đó của ω
- Hành động truy đòi có thể xem là việc xuất hiện thêm chi phí phạt do vi phạm các ràng buộc xác suất khi quyết định được đưa ra trước khi biết được ràng buộc chính xác, và phần chi phí đó sẽ được dùng để sửa lại và bổ sung vào để thỏa mãn các ràng buộc tại kịch bản đang xảy ra.
- Tổng phí phạt sẽ được tính bằng giá trị kỳ vọng $Q(x) = E_\omega[v(x,\omega]$
  - Để tìm được giá trị kỳ vọng của $Q(x) = E_\omega[v(x,\omega]$, chúng ta sẽ dùng cách tiếp cận theo phương pháp **Scenarios analysis** dựa trên sự phân phối xác suất của các tham số ngẫu nhiên cho trước, nghĩa là những tham số ngẫu nhiên đó có hữu hạn giá trị đầu ra và chúng ta sẽ gọi chúng là các kịch bản (Scenarios)
  - **Giá trị kỳ vọng** của Q(x) sẽ được xác định khi ta đã có được sự phân phối xác suất chính xác của ω

    Ta sẽ chọn $\Omega = \{\omega_k\}$ là tập hợp hữu hạn có độ lớn là S (số lượng kịch bản là hữu hạn $\omega_1, \omega_2, \ldots, \omega_S \in \Omega$ với sự phân phối xác suất tương ứng là p_k)
    Từ đó $y = y(x,\omega)$ nên giá trị kỳ vọng của $v(y) = v(x,\omega) := q \cdot y$ là

$$Q(x) = E_\omega[v(x,\omega)] = \sum_{k=1}^{S} p_k \cdot q \cdot y_k = \sum_{k=1}^{S} p_k \cdot v(x,\omega_k) \tag{6.3}$$

    * p_k là mật độ xác suất của kịch bản ω_k, q là chí phí phạt mỗi đơn vị
    * $q \cdot y_k = v(x,\omega_k)$ là phần chi phí phạt do sử dụng y_k đơn vị để bổ sung và hiệu chỉnh để thỏa các ràng buộc và sẽ phụ thuộc vào cả biến quyết định x của giai đoạn thứ nhất và kịch bản ngẫu nhiên ω

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

Trong phần này, chúng ta sẽ thiết lập mô hình Two-stage SLP Recourse model - (canonical form) trong trường hợp tuyến tính

Định nghĩa 6: Quy hoạch tuyến tính 2 giai đoạn sử dụng hành động truy đòi (*Stochastic linear program With Recourse action (2-SLPWR)*)

Dạng canonical 2-stage stochastic linear program sử dụng truy đòi sẽ được thiết lập như sau

$$\begin{cases} 2\text{ - }SLP: \quad \min_x g(x) \\ g(x) := c^T \cdot x + v(y) \\ s.t. \quad A \cdot x = b \quad x \in X \subset \mathbb{R}^n, x \geq 0 \\ v(z) := \min_{y \in \mathbb{R}^{n_2}} \quad q \cdot y \quad s.t. \quad W \cdot y = h(\omega) - T(\omega) \cdot x =: z \end{cases} \tag{6.4}$$

Trong đó

1. $v(y) := v(x, \omega)$ sẽ là giá trị của hàm tại giai đoạn 2 của phương trình và $y = y(x, \omega) \in \mathbb{R}^{n_2}$ sẽ là vector đại diện cho các đơn vị cần bổ sung bồi thường ứng với hành động truy đòi của mô hình cho quyết định x và sự xác định của kịch bản ω.

2. Giá trị kỳ vọng của phần chi phí cho hành động truy đòi của quyết định x của mô hình sẽ là $Q(x) := E_\omega[v(x, \omega)]$ (giá trị kỳ vọng của chi phí cho hành động truy đòi $y(\omega)$ cho bất kỳ giá trị thỏa mãn của $x \in \mathbb{R}^{n_1}$). Nói chung từ đó ta sẽ tối thiểu hóa được tổng giá trị kỳ vọng của $\min_{x \in \mathbb{R}^{n_1}, y \in \mathbb{R}^{n_2}} \quad c^T \cdot x + Q(x)$.

3. Chúng ta sẽ thiết kế sao cho biến quyết định của giai đoạn 2 là $y(\omega)$ có thể tùy ý được điều chỉnh để có thể thỏa được những ràng buộc ban đầu theo một cách thông minh hay còn gọi là tối ưu nhất, nói cách khác chúng ta sẽ gọi nó là hành động truy đòi (Recourse Action).
$x - T, h, \omega - > y.$

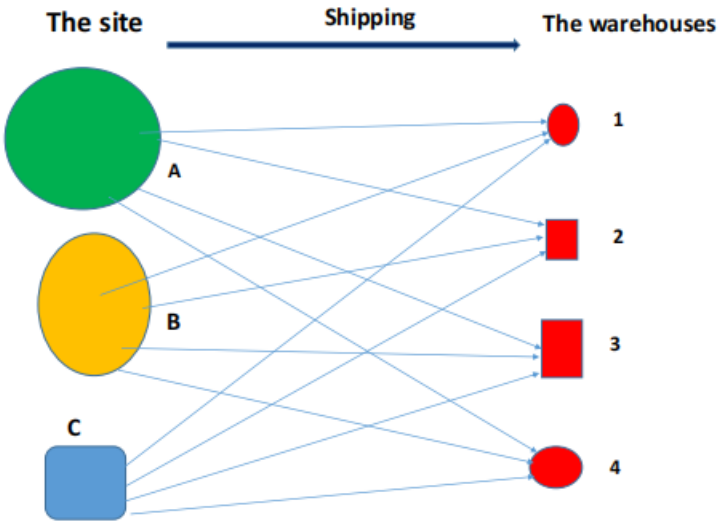
4. Giá trị tối ưu của giai đoạn 2 của mô hình $(2nd - stageLP)$ là $v^* = v(y^*)$, với $y^* = y^*(x, \omega)$ sẽ là quyết định tối ưu của giai đoạn đó. Và cuối cùng giá trị tối ưu của mô hình sẽ là $c^T \cdot x^* + v(y^*)$.

7 PROBLEM 1. [Industry- Manufacturing]

7.1 Giới thiệu vấn đề

Giả sử một công ty F sản xuất được n sản phẩm. Có nhiều bộ phận (sub-assemblies) khác nhau được đặt hàng từ tổng cộng m nhà cung cấp bên thứ ba (the sites). Một đơn vị sản phẩm i yêu cầu $a_{ij} \geq 0$ đơn vị của bộ phận j, với $i = 1, \ldots, n$ và $j = 1, \ldots, m$. Nhu cầu cho từng loại sản phẩm được mô hình hóa thành vector ngẫu nhiên $\omega = D = (D_1, D_2, \ldots, D_n)$. Trước khi biết được nhu cầu của khách hàng, công ty có thể đặt trước các bộ phận từ nhà cung cấp với chi phí là b_j cho mỗi đơn vị bộ phận j và số lượng bộ phận được đặt là x_j, $j = 1, \ldots m$. Sau khi có được thông tin về nhu cầu của khách hàng, công ty có thể quyết định số lượng sản phẩm được sản xuất để đáp ứng một phần nhu cầu sao cho không sử dụng vượt quá số lượng bộ phận hiện có. Lượng nhu cầu không được thực hiện sẽ bị mất. Công ty cần xác định số lượng đơn vị sản phẩm được thực hiện z_i, $i = 1, \ldots, n$, và số lượng bộ phận tồn kho $y_j, j = 1, \ldots, m$.
Sơ đồ sau đây cho thấy một kế hoạch vận chuyển của công ty F từ $m = 3$ các nhà cung cấp và

$n = 4$ vị trí sản phẩm (products, hoặc warehouses).



Bài toán giai đoạn thứ hai:
Với mỗi giá trị xác định $d = (d_1, d_2, \ldots, d_n)$ của vector nhu cầu ngẫu nhiên D, chúng ta có thể tìm ra phương án sản xuất tốt nhất bằng cách giải bài toán quy hoạch tuyến tính ngẫu nhiên với các biến quyết định $z = (z_1, z_2, \ldots, z_n)$ biểu diễn số lượng đơn vị sản phẩm được sản xuất, và biến quyết định $y = (y_1, y_2, \ldots, y_m)$ biểu diễn số lượng bộ phận tồn kho

$$\text{LSP} : \min_{z,y} Z = \sum_{i=1}^{n} (l_i - q_i) z_i - \sum_{j=1}^{m} s_j y_j, \tag{7.1}$$

$$\text{subject to} \begin{cases} y_j = x_j - \sum_{i=1}^{n} a_{ij} z_i, \ j = 1, \ldots, m \\ 0 \le z_i \le d_i, \ i = 1, \ldots, n; \ y_j \ge 0, \ j = 1, \ldots, m \end{cases} \tag{7.2}$$

Toàn bộ mô hình (của giai đoạn thứ hai) có thể được biểu diễn tương đương như sau:

$$\text{MODEL} = \begin{cases} \min_{z,y} Z = c^T \cdot z - s^T \cdot y \\ \text{với } c = (c_i := l_i - q_i) \text{ là hệ số chi phí} \\ y = x - A^T z, \text{ với } A = [a_{ij}] \text{ là ma trận } n \times m \text{ chiều} \\ 0 \le z \le d, \ y \ge 0 \end{cases} \tag{7.3}$$

Bài toán giai đoạn thứ nhất:
Toàn bộ mô hình 2-SLPWR dựa trên một quy luật phổ biến là **production** $\le$ **demand**.
Chúng ta kí hiệu $Q(x) := E[Z(z, y)] = E_\omega[x, \omega]$ biểu diễn giá trị tối ưu của bài toán (7.3). Đặt $b = (b_1, b_2, \ldots, b_m)$ là chi phí đặt trước b_j cho mỗi đơn vị của bộ phận j (trước khi biết được nhu cầu). Giá trị của x_j được xác định từ bài toán tối ưu sau:

$$\min g(x, y, z) = b^T \cdot x + Q(x) = b^T \cdot x + E[Z(z)] \tag{7.4}$$

Trong đó, $Q(x) = E_\omega[Z] = \sum_{i=1}^{n} p_i c_i z_i$ tuân theo phân phối xác suất của $\omega = D$.
Phần thứ nhất của hàm mục tiêu biểu diễn cho chi phí đặt trước và x. Ngược lại, phần thứ hai biểu diễn giá trị kì vọng của kế hoạch sản xuất tối ưu (7.3), cho bởi số lượng sản phẩm đã được cập nhật sau khi biết được nhu cầu ngẫu nhiên $D = d$ với mật độ của chúng.

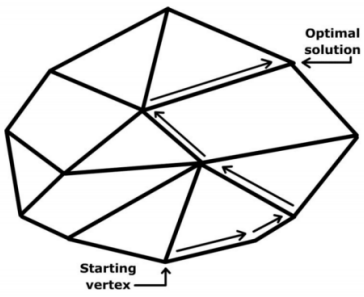
7.2 Thuật toán giải pháp

Một trong những cách phổ biến để giải quyết bài toán quy hoạch tuyến tính ngẫu nhiên là xây dựng và giải bài toán dạng đơn định tương đương (deterministic equivalent). Giả sử các tham số tuân theo một phân phối rời rạc hữu hạn với S khả năng và mỗi khả năng ω_k xảy ra với xác suất $P(\omega_k) = p_k,\ k = 1, \dots, S$ và $\sum_{k=1}^{S} p_k = 1$.
Khi này, bài toán ban đầu trở thành:

$$\begin{cases} \min \sum_{j=1}^{m} b_j x_j + \sum_{k=1}^{S} p_k (\sum_{i=1}^{n} (l_i - q_i) z_{ki} - \sum_{j=1}^{m} s_j y_{kj}) \\ \text{s.t.} \\ y_{kj} = x_j - \sum_{i=1}^{n} a_{ij} z_{ki},\ j = 1, \dots, m \\ 0 \le z_{ki} \le d_{ki},\ i = 1, \dots, n;\ x_j \ge 0,\ y_{kj} \ge 0,\ j = 1, \dots, m,\ k = 1, \dots, S \end{cases}$$

Trong đó, z_{ki}, d_{ki} và y_{kj} lần lượt là số lượng sản phẩm i được sản xuất, nhu cầu cho sản phẩm i và số lượng bộ phận j tồn kho trong trường hợp $k,\ k = 1, \dots, S$.
Chúng ta có thể giải quyết bài toán trên bằng phương pháp đơn hình được trình bày như sau:

1. *Ý tưởng*
   Phương pháp đơn hình bắt đầu từ một phương án cực biên nào đó (tùy ý) của bài toán (tức là một đỉnh của miền ràng buộc). Tiếp đó kiểm tra xem phương án hiện có đã là phương án tối ưu hay chưa, bằng cách so sánh giá trị hàm mục tiêu tại đỉnh đó với giá trị hàm mục tiêu tại các đỉnh kề với nó. Nếu đúng thì dừng quá trình tính toán. Trái lại, phương pháp sẽ cho cách tìm một phương án cực biên mới tốt hơn (với giá trị hàm mục tiêu nhỏ hơn) mà nó là một đỉnh kề với đỉnh trước đó. Quá trình này tiến hành cho tới khi tìm được phương án tối ưu hoặc phát hiện bài toán đã cho không có lời giải. Mặc dù số đỉnh của bài toán nói chung rất lớn, nhưng trên thực tế phương pháp này chỉ đòi hỏi kiểm tra một phần tương đối nhỏ các đỉnh. Chính điều đó đã thể hiện hiệu quả thực tế của phương pháp đơn hình.



Hình 7.1: Mô tả hình học cho phương pháp đơn hình

2. *Định nghĩa*

   - Xét bài toán quy hoạch tuyến tính dạng chính tắc chuẩn:

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & \\ & Ax = b \\ & x \geq 0 \end{aligned}$$

   - Đặt $S = \{x \in \mathbb{R} | Ax = b,\ x \geq 0\}$ là tập phương án khả dĩ của bài toán quy hoạch tuyến tính với A là một ma trận $m \times n$ chiều độc lập tuyến tính
   - Cho $A = (B, N)$, với B là một ma trận vuông cấp m độc lập tuyến tính. Khi đó, B được gọi là một cơ sở.
   - Cho $x = \begin{pmatrix} x_B \\ x_N \end{pmatrix}$. Chúng ta có $Bx_B + Nx_N = b$. Cho $x_N = 0$ được $x_B = B^{-1}b$. $x = \begin{pmatrix} B^{-1}b \\ 0 \end{pmatrix}$ được gọi là một nghiệm cơ sở. x_B được gọi là các biến cơ sở (basic variable), x_N được gọi là các biến không cơ sở (nonbasic variable).
   - Nếu nghiệm cơ sở cũng khả dĩ, khi này, $B^{-1}b \geq 0$, và $x = \begin{pmatrix} B^{-1}b \\ 0 \end{pmatrix}$ được gọi là nghiệm cơ sở khả dĩ (basic feasible solution).

3. *Thuật toán*

   - **Bước 0:** Đưa bài toán quy hoạch tuyến tính về dạng chính tắc chuẩn (nếu cần).
   - **Bước 1:** Tính một cơ sở ban đầu B và nghiệm cơ sở khả dĩ $x = \begin{pmatrix} B^{-1}b \\ 0 \end{pmatrix}$.
   - **Bước 2:** Nếu $r_N = c_N^T - c_B^T B^{-1} N \geq 0$, **dừng**, x là nghiệm tối ưu. Ngược lại, thực hiện bước 3.
   - **Bước 3:** Chọn j thỏa mãn $c_i^T - c_B^T B^{-1} a_j < 0$, nếu $\overline{a_j} = B^{-1}a_j \leq 0$, **dừng**, bài toán không bị giới hạn (unbounded). Ngược lại, thực hiện bước 4.
   - **Bước 4:** Tính:

$$\lambda = \min \left\{ \frac{\overline{b}_i}{\overline{a}_{ij}} | \overline{a}_{ij} > 0 \right\} = \frac{\overline{b}_r}{\overline{a}_{rj}}$$

     Cho $x := x + \lambda d_j$, với $d_j = \begin{pmatrix} B^{-1}a_j \\ e_j \end{pmatrix}$. Đi đến bước 2.

Dựa vào cơ sở lý thuyết của phương pháp đơn hình, chúng ta có thể xây dựng chương trình chạy bằng ngôn ngữ Python cùng với thư viện GAMSpy để giải quyết bài toán quy hoạch tuyến tính ngẫu nhiên nêu trên và tìm ra kế hoạch sản xuất tối ưu cho công ty.
Thư viện được sử dụng:

- GAMSpy
- NumPy
- pandas
- Matplotlib

Câu lệnh được sử dụng:

Câu lệnh	Ý nghĩa
pd.read_excel()	Đọc file excel vào pandas DataFrame
df.to_excel()	Viết một vật thể vào tập tin excel
pd.ExcelWriter()	Lớp cho việc viết các vật thể vào bảng tính excel
ax.bar()	Dùng để vẽ biểu đồ thanh
mpl.savefig()	Lưu biểu đồ thành tập tin hình ảnh
Container()	Lớp Container trong GAMSPy đóng vai trò là trung để quản lý các cấu trúc dữ liệu thiết yếu
Set()	Set tương ứng với các chỉ số trong biểu diễn đại số của các mô hình
Parameter()	Parameter được sử dụng để nhập dữ liệu từ danh sách có thể được gắn nhãn theo tập hợp
Variable()	Khởi tạo biến trong mô hình
Equation()	Tạo ra ràng buộc mới cho mô hình
Model()	Lớp mô hình được sử dụng để thu thập các phương trình thành các nhóm và gắn nhãn chúng để chúng có thể được giải.
solve()	Dùng để giải quyết mô hình được chỉ định

```
from gamspy import Model, Sense, Container, Equation, Variable, Parameter, Set,
    Sum
import matplotlib.pyplot as mpl
import pandas as pd
import numpy as np

#import data from input.xlsx
input = pd.read_excel("input.xlsx", sheet_name=["Product", "Subassembly", "
    Scenario"])

#check for the data errors
for _ in input["Product"].to_numpy():
    if (_ < 0).any():
        exit("Invalid data in Product")
for _ in input["Subassembly"].to_numpy():
    if (_ < 0).any():
        exit("Invalid data in Subassembly")
for _ in input["Scenario"].to_numpy():
    if (_ < 0).any():
        exit("Invalid data in Scenario")
if np.sum(input["Scenario"]["P"].to_numpy()) != 1:
    exit("Invalid data in Scenario")

#Build model from data input
m = Container()

i = Set(container=m, name="i", description="list of products", records=input["
    Product"]["Name"].to_numpy())
j = Set(container=m, name="j", description="list of subassemblies", records=input[
    "Subassembly"]["Name"].to_numpy())
k = Set(container=m, name="k", description="list of scenarios", records=input["
    Scenario"]["S"].to_numpy())

b = Parameter(container=m, name="b", domain=j, description="cost of subassembly",
    records=input["Subassembly"]["b"].to_numpy())
```

```
s = Parameter(container=m, name="s", domain=j, description="salvage assess",
    records=input["Subassembly"]["s"].to_numpy())
l = Parameter(container=m, name="l", domain=i, description="cost of product",
    records=input["Product"]["l"].to_numpy())
q = Parameter(container=m, name="q", domain=i, description="price of product",
    records=input["Product"]["q"].to_numpy())
A = Parameter(container=m, name="A", domain=[i, j], description="product i
    requires a_(ij) unit of part j",records=input["Product"].iloc[:,3:].to_numpy()
    )
D = Parameter(container=m, name="D", domain=[k, i], description="demand of product
     i in scenario s", records=input["Scenario"].iloc[:, 2:].to_numpy())
P = Parameter(container=m, name="P", domain=k, description="probability of
    scenario s", records=input["Scenario"]["P"].to_numpy())

x = Variable(container=m, name="x", domain=j, description="number of ordered parts
    ", type="positive")
y = Variable(container=m, name="y", domain=[k, j], description="number of left
    parts", type="positive")
z = Variable(container=m, name="z", domain=[k, i], description="number of products
    ", type="positive")

eq1 = Equation(container=m, name="eq1", domain=[k, j])
eq1[k, j] = y[k, j] == x[j] - Sum(i, A[i, j] * z[k, i])

eq2  = Equation(container=m, name="eq2", domain=[k, i])
eq2[k, i] = z[k, i] <= D[k, i]

ans = Model(container=m, name="ans", equations=[eq1, eq2], problem="LP" , sense=
    Sense.MIN, objective=Sum(j, b[j] * x[j]) + Sum(k, P[k] * (Sum(i, (l[i] - q[i])
     * z[k, i]) - Sum(j, s[j] * y[k, j]))))
#Solve problem
ans.solve()

#plot graphs from the output
fig, ax = mpl.subplots()
k_list = k.records["uni"].to_list()
i_list = i.records["uni"].to_list()
j_list = j.records["uni"].to_list()
size = len(k_list)
ax.bar(j_list, x.records["level"].to_list(), width=0.6)
mpl.title("Number of preordered subassemblies")
mpl.xlabel("Subassembly")
mpl.ylabel("Number")
mpl.savefig("assembly.png")
arr_xy = (1 - y.records["level"].to_numpy().reshape(size, len(j)) / x.records["
    level"].to_numpy()) * 100
arr_z = z.records["level"].to_numpy() / D.records["value"].to_numpy() * 100
arr_z = arr_z.reshape(size, len(i_list))
for n in range(size):
    fig, ax1 = mpl.subplots()
    ax1.bar(j_list, arr_xy[n], width=0.6)
    mpl.title("Percentage of used subassembly in scenario " + k_list[n])
    mpl.xlabel("Subassembly")
    mpl.ylabel("Percentage")
    mpl.savefig("xy_in_scenario" + k_list[n] + ".png")
    fig, ax2 = mpl.subplots()
    colors = np.where(arr_z[n] > 80, 'green', 'red')
    ax2.bar(i_list, arr_z[n], width=0.6, color=colors)
    mpl.title("Percentage of demand satisfied in scenario " + k_list[n])
    mpl.xlabel("Product")
    mpl.ylabel("Percentage")
    mpl.savefig("z_in_scenario" + k_list[n] + ".png")
```

```
79
80 #write output data to output.xlsx
81 df1 = pd.DataFrame(x.records)
82 df2 = pd.DataFrame(y.records)
83 df3 = pd.DataFrame(z.records)
84 df4 = pd.DataFrame([ans.objective_value], index=["Objective Value"])
85 with pd.ExcelWriter("output.xlsx") as writer:
86     df1.to_excel(writer, sheet_name="order")
87     df2.to_excel(writer, sheet_name="inventory")
88     df3.to_excel(writer, sheet_name="produce")
89     df4.to_excel(writer, sheet_name="objective", header=False)
```

7.3 Ví dụ

Công ty F sản xuất $n = 8$ sản phẩm, số khả năng có thể xảy ra là $S = 2$ với mật độ $p_s = 1/2$, số lượng bộ phận được đặt mua trước khi sản xuất là $m = 5$. Dữ liệu về các vector b, l, q, s, ma trận A có kích thước $m \times n$ và vector nhu cầu ngẫu nhiên $\omega = D = (D_1, D_2, \ldots, D_n)$ được mô phỏng trong tập tin *input.xlsx* như sau:

	A	B	C	D	E	F	G	H	I
1	**Name**	**l**	**q**	**1**	**2**	**3**	**4**	**5**	
2	1	2	121	5	6	0	6	4	
3	2	3	61	4	1	0	6	3	
4	3	2	38	2	3	2	0	0	
5	4	1	80	0	2	4	2	10	
6	5	2	79	1	6	7	0	0	
7	6	3	96	3	3	3	3	0	
8	7	4	56	4	0	2	2	1	
9	8	2	31	5	0	0	1	0	
10									

Hình 7.2: Product sheet

	A	B	C	D
1	**Name**	**b**	**s**	
2	1	5	2	
3	2	4	1	
4	3	4	2	
5	4	3	1	
6	5	3	2	
7				

Hình 7.3: Subassembly sheet

	A	B	C	D	E	F	G	H	I	J	K
1	**S**	**P**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	
2	1	0.5	451	310	401	351	387	409	309	310	
3	2	0.5	399	317	357	338	447	361	354	364	
4											

Hình 7.4: Scenario sheet

Sau khi chạy chương trình, chúng ta thu được kết quả trong tập tin *output.xlsx* như sau:

	A	B	C	D	E	F	G	H
1		**j**	**level**	**marginal**	**lower**	**upper**	**scale**	
2	**0**	1	8462.231	0	0	inf	1	
3	**1**	2	8411.308	0	0	inf	1	
4	**2**	3	6986	0	0	inf	1	
5	**3**	4	7070.846	0	0	inf	1	
6	**4**	5	6376.923	0	0	inf	1	
7								

Hình 7.5: Number of ordered parts

	A	B	C	D	E	F	G	H	I
1		**k**	**j**	**level**	**marginal**	**lower**	**upper**	**scale**	
2	**0**	1	1	0	1.365385	0	inf	1	
3	**1**	1	2	0	3	0	inf	1	
4	**2**	1	3	226	0	0	inf	1	
5	**3**	1	4	0	1.173077	0	inf	1	
6	**4**	1	5	0	1	0	inf	1	
7	**5**	2	1	0	1.634615	0	inf	1	
8	**6**	2	2	188.3077	0	0	inf	1	
9	**7**	2	3	0	2	0	inf	1	
10	**8**	2	4	0	0.826923	0	inf	1	
11	**9**	2	5	95.92308	0	0	inf	1	
12									

Hình 7.6: Number of left parts

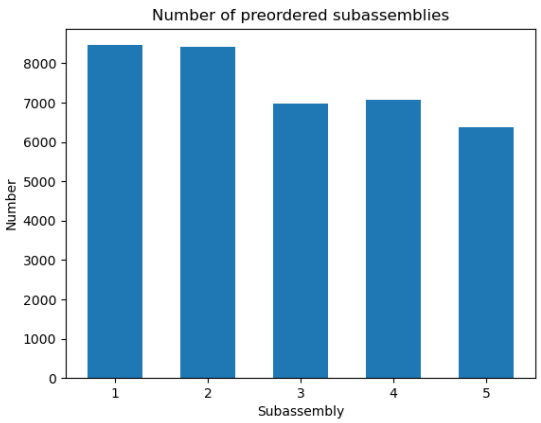
	A	B	C	D	E	F	G	H	I
1		k	i	level	marginal	lower	upper	scale	
2	0	1	1	451	0	0	inf	1	
3	1	1	2	251.3077	0	0	inf	1	
4	2	1	3	401	0	0	inf	1	
5	3	1	4	351	0	0	inf	1	
6	4	1	5	387	0	0	inf	1	
7	5	1	6	409	0	0	inf	1	
8	6	1	7	309	0	0	inf	1	
9	7	1	8	310	0	0	inf	1	
10	8	2	1	399	0	0	inf	1	
11	9	2	2	317	0	0	inf	1	
12	10	2	3	357	0	0	inf	1	
13	11	2	4	338	0	0	inf	1	
14	12	2	5	447	0	0	inf	1	
15	13	2	6	361	0	0	inf	1	
16	14	2	7	354	0	0	inf	1	
17	15	2	8	307.8462	0	0	inf	1	
18									

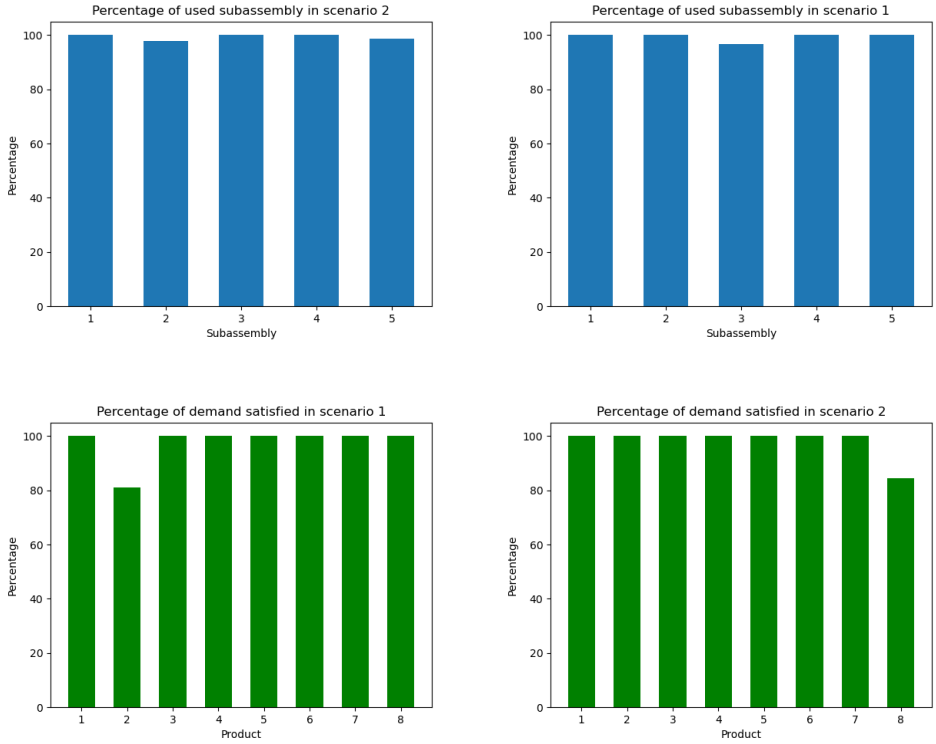
Hình 7.7: Number of products

	A	B	C
1	Objective Value	-58198.57692	
2			

Hình 7.8: Objective value

Từ các dữ liệu trên, chúng ta có được các đồ thị sau:





8 Application I: Stochastic Linear Program for Evacuation Planning In Disaster Responses (SLP-EPDR)

8.1 Giới thiệu

Thảm họa tự nhiên (động đất, bão, hỏa hoạn, sóng thần, v.v...) và thảm họa do con người (khủng bố, chiến tranh, v.v...) có thể xảy ra một cách bất ngờ trong cộng đồng của chúng ta và gây ra nhiều hậu quả khó lường, để lại nhiều thương vong. Mục tiêu chính của chúng ta trong những tình huống khẩn cấp này là cung cấp chỗ trú và hỗ trợ những người đang bị ảnh hưởng càng sớm càng tốt.

Có nhiều cách để giải quyết vấn đề này, một số nhà nghiên cứu tập trung vào việc cung cấp cho những người ở nơi an toàn các nhu yếu phẩm và vật dụng cần thiết và đặt những trung tâm trú ẩn an toàn tại những địa điểm thích hợp để kịp thời cung cấp những thứ đó cho người dân. Cách thứ hai, cũng là cách chúng ta sẽ áp dụng trong bài báo cáo này, chính là tạo ra những kế hoạch sơ tán để các người dân ở vùng ảnh hưởng bởi thảm họa có thể nhanh chóng di tản đến những nơi an toàn.

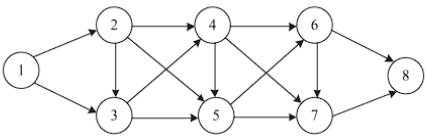
Như chúng ta đã biết, việc ước lượng và dự đoán trước mức độ và thiệt hại mà một thảm họa mang lại là cực kì khó khăn vì chúng ta không thể biết trước được cường độ của nó. Vậy nên cách tiếp cận vấn đề của chúng ta sẽ là sử dụng mô hình quy hoạch tuyến tính ngẫu nhiên hai giai đoạn với yếu tố ngẫu nhiên là thời gian di chuyển cũng như sức chứa của mỗi tuyến đường tại mỗi thời điểm cụ thể trong suốt thời gian diễn ra thảm họa. Thêm vào đó, hai giai đoạn của mô hình quy hoạch tuyến tính trên sẽ được chúng ta giải quyết bằng việc áp dụng bài toán luồng chi phí tối thiểu (Minimum-cost Flow), cụ thể việc hiện thực mô hình sẽ được ghi rõ ở phần sau.

8.2 Mô hình quy hoạch tuyến tính hai giai đoạn

8.2.1 Phân tích bài toán sơ tán hai giai đoạn

Giả sử chúng ta đang phải đối mặt với thảm họa là động đất. Trong quá trình sơ tán, mọi người đều có điện thoại di động để có thể cập nhật thông tin mới nhất về tình hình của trận động đất và có thể tự di chuyển đến nơi an toàn bằng phương tiện riêng của mình. Sau khi nhận được tín hiệu nguy hiểm, thông tin về trận động đất sẽ được thông báo tới người dân thông qua các thiết bị giao tiếp tiên tiến. Tuy nhiên, trước khi có được những thông tin về tình hình hiện tại, những phản ứng khẩn cấp và việc sơ tán sẽ phải được thực hiện mà không biết chính xác về mức độ nguy hiểm và phạm vi ảnh hưởng của trận động đất. Trong trường hợp này, người dân sẽ bắt đầu di tản theo kế hoạch dựa trên những tình huống có thể xảy ra trong tương lai. Do đó, kế hoạch sơ tán sẽ được chia thành hai giai đoạn, trong giai đoạn một (giai đoạn không dự kiến được), các quyết định di tản phải được đưa ra trước những điều không chắc chắn có thể xảy ra trong tương lai và kế hoạch sơ tán giai đoạn hai sẽ được xác định sau khi các thông tin chính xác đã được nắm bắt. Dựa vào phân tích ở trên, mục tiêu của chúng ta chính là tạo ra kế hoạch sơ tán giai đoạn một tối ưu nhất trước khi có thông tin về sự không chắc chắn ở giai đoạn hai.

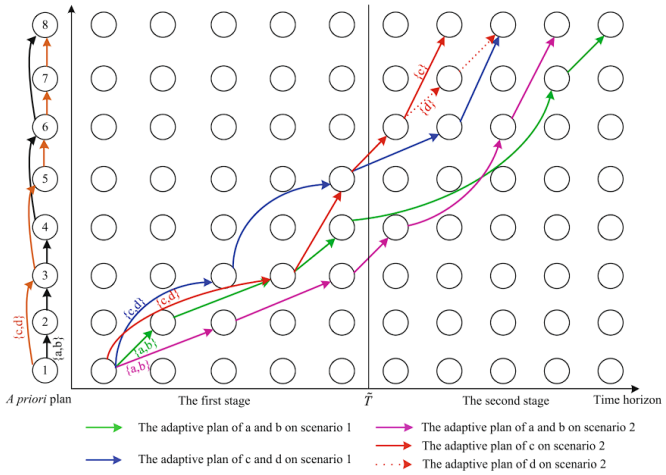
Ví dụ, mô hình hai giai đoạn cho việc sơ tán người dân trong thảm họa sẽ được minh họa bằng một đồ thị đơn giản gồm 8 đỉnh và 15 cạnh. Ở đây đỉnh 1 và 8 được xem lần lượt là nơi xảy ra động đất và nơi an toàn. Trước khi động đất diễn ra hay trong giai đoạn đầu của trận động đất, chúng ta có bốn xe cần được di chuyển đến nơi an toàn dựa trên kế hoạch tiên nghiệm (a priori plan) là xe a, b, c và d. Sau đó khi chúng ta đã có thông tin chính xác về tình hình của các tuyến đường, hay sau ngưỡng thời gian $\tilde{T}$, người dân có thể tự thích nghi với những tình huống đang diễn ra mà thay đổi kế hoạch cho hợp lí.



Hình 8.1: Minh họa về mạng lưới đường sơ tán

Để đơn giản hóa vấn đề, ta sẽ xét mô hình gồm hai trường hợp ngẫu nhiên để minh họa cho bài toán. Hình 6.2 cho ta thấy đồ thị về thời gian và không gian cho đồ thị ở Hình 6.1 bằng cách chia thời gian thành các khoảng rời rạc. Trong Hình 6.2, trục nằm ngang thể hiện cho thời gian và trục đứng thể hiện các đỉnh trong đồ thị. Trục thời gian được chia ra thành hai phần tương ứng với hai giai đoạn, kế hoạch tiên nghiệm được mô tả ở phần bên trái và kế hoạch thích nghi được mô tả ở phần bên phải. Chúng ta có thể thấy rằng, trước ngưỡng thời gian $\tilde{T}$, đường đi của các xe trong trường hợp 1 và 2 giống như đường đi trong kế hoạch tiên nghiệm. Nhưng sau khi có thông tin cần thiết, tức là sau $\tilde{T}$, đường đi của các xe bắt đầu có sự thay đổi so với kế hoạch tiên nghiệm.

Ví dụ đơn giản ở trên đã giúp chúng ta phân tích về mô hình quy hoạch ngẫu nhiên hai giai đoạn dựa trên các trường hợp ngẫu nhiên. Kế hoạch sơ tán tiên nghiệm trong giai đoạn một phải được thực hiện trước khi những yếu tố ngẫu nhiên được xác định ở giai đoạn hai. Sau đó, giai đoạn hai sẽ giúp chúng ta có được kế hoạch sơ tán với từng kịch bản cụ thể. Trong bài báo cáo này, chúng ta sẽ tập trung vào việc tạo ra kế hoạch sơ tán tiên nghiệm bằng cách xem xét sự truy đòi cho từng trường hợp có thể xảy ra ở giai đoạn hai.



Hình 8.2: Kế hoạch sơ tán dựa trên mô hình Two-stage stochastic trên mạng lưới đường phụ thuộc vào thời gian

8.2.2 Mô tả bài toán sơ tán hai giai đoạn bằng bài toán luồng chi phí tối thiểu (Minimum-cost Flow)

Chúng ta xây dựng nên kế hoạch di tản chi tiết cho người dân bằng bài toàn luồng chi phí tối thiểu với thời gian di chuyển và sức chứa của mỗi tuyến đường (cạnh) là ngẫu nhiên. Mục tiêu của chúng ta là đưa người dân từ nơi nguy hiểm đến nơi an toàn với thời gian sơ tán ngắn nhất trong mạng lưới gồm sức chứa và chi phí $G(V, A, C, U, D)$, với V là tập hợp các đỉnh, A là tập hợp các cạnh với sức chứa và thời gian di chuyển ngẫu nhiên, $C(i, j)$ biểu diễn thời gian di chuyển trên cạnh $(i, j) \in A$, kí hiệu là c_{ij}, $U(i, j)$ là sức chứa của cạnh (i, j), kí hiệu là u_{ij} và $D(i)$ là luồng tại đỉnh $i \in V$, kí hiệu là d_i. Chúng ta giả sử là thời gian di chuyển trên cạnh $(i, j) \in A$ là như nhau nếu số lượng người di chuyển trên cạnh đó không vượt quá sức chứa. Nghĩa là, việc nhiều người, nhưng không vượt quá sức chứa, cùng đi trên một tuyến đường trong trường hợp này sẽ không ảnh hưởng đến thời gian di chuyển trên tuyến đường đó. Tóm lại, trong vấn đề mà chúng ta đang xét, nguồn của mạng lưới chính là nơi xảy ra thảm họa, đích đại diện cho những nơi an toàn, những đỉnh khác là giao lộ và các cạnh trên mạng lưới đó đại diện cho những tuyến đường để mọi người di chuyển. Trên thực tế có thể có nhiều nguồn và đích, trong trường hợp này chúng ta sẽ gộp các nguồn và đích lại thành một đỉnh lớn để đại diện cho một nguồn và một đích.

Mô hình mà chúng ta đang xét khác với bài toán luồng chi phí tối thiểu thông thường. Cụ thể, mô hình quy hoạch ngẫu nhiên có truy đòi đại diện cho tình huống cả hai giai đoạn một và hai xảy ra trong các khoảng thời gian khác nhau trong cùng một mạng lưới sơ tán. Lưu ý rằng, kế hoạch của giai đoạn một có thể không khả thi khi những thông tin trong giai đoạn hai được tiết lộ, nhưng điều này sẽ được giải quyết bằng cách cho phép làm giảm thời gian sơ tán ở giai đoạn hai. Trong trường hợp này, hàm mục tiêu của mô hình chúng ta sẽ bao gồm thêm những mức phạt ở giai đoạn một và giá trị kỳ vọng của sự truy đòi có thể xảy ra ở giai đoạn hai.

8.2.3 Mô hình hóa bài toán

Một số ký hiệu của bài toán:

Table 1
Subscripts and Parameters Used in Mathematical Formulation.

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

Table 2
Decision Variables Used in Mathematical Formulation.

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

Chúng ta sẽ sử dụng hai biến quyết định để mô hình hóa bài toán sơ tán hai giai đoạn. Trong giai đoạn một, x_{ij} được dùng để chỉ luồng trên cạnh (i, j). Trong giai đoạn hai, $y_{ij}^s(t)$ được dùng để chỉ luồng trên cạnh (i, j) trong trường hợp s tại thời điểm t.

Trong giai đoạn một, một kế hoạch sơ tán khả thi phải được xác định từ đỉnh nguồn tới đích. Các luồng trên mỗi cạnh phải thỏa mãn ràng buộc về sự cân bằng như sau:

$$\sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ji} = d_i \tag{8.1}$$

Với d_i là một tham số có giá trị tương ứng là:

$$d_i = \begin{cases} v, & i = s \\ -v & i = t \\ 0, & otherwise \end{cases} \tag{8.2}$$

Thêm vào đó, luồng trên mỗi cạnh cũng phải thỏa mãn ràng buộc về sức chứa:

$$0 \leq x_{ij} \leq u_{ij}, \quad \forall (i, j) \in A \tag{8.3}$$

Một điều chúng ta nên lưu ý là ràng buộc về sự cân bằng có thể khiến cho giải thuật của chúng ta tạo ra những đường đi với các khuyên hoặc chu trình nếu như mạng lưới của chúng ta có bao gồm nó. Để giải quyết vấn đề này, một tham số biểu diễn mức phạt trên mỗi cạnh, kí hiệu là p_{ij},$(i,j) \in A$, đã được thêm vào. Và hàm phạt của chúng ta được định nghĩa như sau:

$$f(X) = \sum_{(i,j)\in A} p_{ij} \cdot x_{ij} \tag{8.4}$$

Với $X := \{x_{ij}\}_{(ij)\in A}$

Trong giai đoạn thứ hai, những người dân đang bị ảnh hưởng sẽ nhận được lộ trình di chuyển với thời gian sơ tán ngắn nhất dựa vào tình hình thực tế của thảm họa. Trước ngưỡng thời gian $\tilde{T}$, những người bị ảnh hưởng sẽ di tản theo kế hoạch tiên nghiệm như trong giai đoạn một. Nói cách khác, mọi kế hoạch sơ tán trong mọi kịch bản sẽ là như nhau khi chưa đến ngưỡng thời gian $\tilde{T}$. Ví dụ, nếu chúng ta có dòng chảy trên cạnh (i,j) là $x_{ij} = 2$, thì dòng chảy trên cạnh (i,j) trước ngưỡng thời gian $\tilde{T}$ cũng là 2 hay $y^s_{ij}(t) = 2$. Dựa vào thảo luận trên, chúng ta có thể thêm vào mô hình một ràng buộc ghép nối cho các kế hoạch sơ tán trong những kịch bản khác nhau như sau:

$$y^s_{ij} = x_{ij}, \quad (i,j) \in A, s = 1,2,\dots,S. \tag{8.5}$$

Sau đây là mô hình giai đoạn hai của bài toán với tiêu chí là tối thiểu hóa thời gian sơ tán của những người đang trong vùng ảnh hưởng bởi thảm họa đến nơi an toàn trong từng kịch bản:

$$Q(Y,s) = \min \sum_{(i,j)\in A_s} c^s_{ij}(t) \cdot y^s_{ij}(t) \tag{8.6}$$

s.t.

$$\sum_{(i_t,j_{t'})\in A_s} y^s_{ij}(t) - \sum_{(j_{t'},i_t)\in A_s} y^s_{ji}(t') = d^s_i(t), \quad \forall i \in V, t \in \{0,1,\dots,T\} \\ s = 1,2,\dots,S \tag{8.7}$$

$$0 \le y^s_{ij}(t) \le u^s_{ij}(t), \quad \forall (i,j) \in A, t \in \{0,1,\dots,T\}, s = 1,2,\dots,S \tag{8.8}$$

$$y^s_{ij}(t) = x_{ij}, (i,j) \in A, s = 1,2,\dots,S \tag{8.9}$$

Hàm mục tiêu (8.6) là tổng thời gian sơ tán của mọi phương tiện (người dân) trong từng kịch bản s. Các ràng buộc (8.7) và (8.8) lần lượt là ràng buộc cân bằng và ràng buộc sức chứa. Ràng buộc (8.9) là ràng buộc ghép nối để chắc chắn rằng lộ trình di tản của người dân trong từng kịch bản ở giai đoạn hai trước ngưỡng thời gian $\tilde{T}$ giống với kế hoạch sơ tán tiên nghiệm ở giai đoạn một.

Để mô hình hóa giai đoạn một của bài toán, chúng ta sẽ tính toán giá trị kỳ vọng của của kế hoạch thích ứng cho mỗi kịch bản trong giai đoạn hai, xác suất xuất hiện của mỗi kịch bản được kí hiệu là μ_s, $s = 1,2,\dots,S$. Mô hình sơ tán hai giai đoạn trong môi trường ngẫu nhiên và phụ thuộc vào thời gian được biểu diễn như sau:

$$\begin{cases} \min \sum\limits_{(i,j)\in A} p_{ij}\cdot x_{ij} + \sum\limits_{s=1}^{S} \mu_s \cdot Q(Y,s) \\ s.t. \\ \sum\limits_{(i,j)\in A} x_{ij} - \sum\limits_{(j,i)\in A} x_{ji} = d_i, \quad \forall i \in V \\ 0 \le x_{ij} \le u_{ij}, \quad \forall (i,j) \in A \\ \textit{Trong đó:} \\ Q(Y,s) = \min \sum_{(i,j)\in A_s} c_{ij}^s(t)\cdot y_{ij}^s(t) \\ s.t. \\ \sum\limits_{(i_t,j_{t'})\in A_s} y_{ij}^s(t) - \sum\limits_{(j_{t'},i_t)\in A_s} y_{ji}^s(t') = d_i^s(t), \quad \forall i \in V \\ t \in \{0,1,\ldots,T\}, \quad s = 1,2,\ldots,S \\ 0 \le y_{ij}^s(t) \le u_{ij}^s(t), \forall (i,j)\in A, t \in \{0,1,\ldots,T\}, s = 1,2,\ldots,S \\ y_{ij}^s(t) = x_{ij}, (i,j) \in A, s = 1,2,\ldots,S \end{cases} \tag{8.10}$$

Vì giai đoạn hai của mô hình trên có số kịch bản hữu hạn, chúng ta có thể biểu diễn lại bằng một mô hình một giai đoạn tương đương như sau:

$$\begin{cases} \min \sum\limits_{(i,j)\in A} p_{ij}\cdot x_{ij} + \sum\limits_{s=1}^{S} (\mu_s \cdot \sum\limits_{(i,j)\in A_s} c_{ij}^s(t)\cdot y_{ij}^s(t)) \\ s.t. \\ \sum\limits_{(i,j)\in A} x_{ij} - \sum\limits_{(j,i)\in A} x_{ji} = d_i, \quad \forall i \in V \\ 0 \le x_{ij} \le u_{ij}, \quad \forall (i,j) \in A \\ \sum\limits_{(i_t,j_{t'})\in A_s} y_{ij}^s(t) - \sum\limits_{(j_{t'},i_t)\in A_s} y_{ji}^s(t') = d_i^s(t), \quad \forall i \in V \\ t \in \{0,1,\ldots,T\}, \quad s = 1,2,\ldots,S \\ 0 \le y_{ij}^s(t) \le u_{ij}^s(t), \forall (i,j)\in A, t \in \{0,1,\ldots,T\}, s = 1,2,\ldots,S \\ y_{ij}^s(t) = x_{ij}, (i,j) \in A, s = 1,2,\ldots,S \end{cases} \tag{8.11}$$

8.3 Phương pháp giải quyết bài toán

Từ mô hình gốc của bài toàn, ta có thể thấy rằng ràng buộc ghép nối (8.5) là một ràng buộc khó. Vì vậy, chúng ta có thể sử dụng nhân tử Lagrange $\alpha_{ij}^s(t), (i,j) \in A, s = 1,2,\ldots,S, \quad t \le \tilde{T}$ để thay thế cho ràng buộc này, sau đó ràng buộc này sẽ được thư giãn thành một phần của hàm mục tiêu dưới dạng:

$$\sum_{s=1}^{S} \sum_{t\le\tilde{T}} \sum_{(i,j)\in A} \alpha_{ij}^s(t)\cdot(y_{ij}^s(t) - x_{ij}) \tag{8.12}$$

Mô hình (8.11) theo đó mà chúng ta có thể viết lại thành:

$$\begin{cases} \min \sum\limits_{(i,j)\in A} p_{ij}\cdot x_{ij} + \sum\limits_{s=1}^{S}(\mu_s \cdot \sum\limits_{(i,j)\in A_s} c_{ij}^s(t)\cdot y_{ij}^s(t)) + \sum\limits_{s=1}^{S}\sum\limits_{t\le\tilde{T}}\sum\limits_{(i,j)\in A}\alpha_{ij}^s(t)\cdot(y_{ij}^s(t)-x_{ij}) \\ s.t. \\ \sum\limits_{(i,j)\in A} x_{ij} - \sum\limits_{(j,i)\in A} x_{ji} = d_i, \quad \forall i\in V \\ 0\le x_{ij}\le u_{ij}, \quad \forall (i,j)\in A \\ \sum\limits_{(i_t,j_{t'})\in A_s} y_{ij}^s(t) - \sum\limits_{(j_{t'},i_t)\in A_s} y_{ji}^s(t') = d_i^s(t), \quad \forall i\in V \\ t\in\{0,1,\dots,T\}, \quad s=1,2,\dots,S \\ 0\le y_{ij}^s(t)\le u_{ij}^s(t), \forall (i,j)\in A, t\in\{0,1,\dots,T\}, s=1,2,\dots,S \end{cases} \tag{8.13}$$

Từ mô hình (8.13), ta có thể tách hai biến X và Y ra thành hai bài toán, theo đó mô hình của chúng ta cũng được thư giãn bằng cách phân rã thành hai bài toán con như sau:

Bài toán con 1: Bài toán min-cost flow.

Bài toán con một có thể được xem như là một bài toán Min-cost Flow và mô hình của bài toán được cho như sau:

$$\begin{cases} \min SP1(\alpha) = \sum_{i,j}\in A(p_{ij} - \sum_{s=1}^{S}\sum_{t\le\tilde{T}}\alpha_{ij}^s(t))\cdot x_{ij} \\ s.t. \\ \sum\limits_{(i,j)\in A} x_{ij} - \sum\limits_{(j,i)\in A} x_{ji} = d_i, \quad \forall i\in V \\ 0\le x_{ij}\le u_{ij}, \quad \forall (i,j)\in A \end{cases} \tag{8.14}$$

Ta có thể viết lại hàm chi phí trong mô hình trên thành $g_{ij} := p_{ij} - \sum\limits_{s=1}^{S}\sum\limits_{t\le\tilde{T}}\alpha_{ij}^s(t)$. Theo đó, bài toán con 1 có thể được giải quyết bằng thuật toán successive shortest path và ta sẽ gọi kết quả thu được của bài toán con 1 là $Z_{SP1}^*(\alpha)$.

Bài toán con 2: Bài toán min-cost flow phụ thuộc vào thời gian.

Bài toán con thứ 2 của mô hình (8.13) được dùng để tìm giá trị của biến quyết định Y, kết quả của bài toán ta sẽ kí hiệu là $Z_{SP2}^*(\alpha)$.

$$\begin{cases} \min SP2(\alpha) = \sum\limits_{s=1}^{S}\sum\limits_{(i,j)\in A}\Big(\sum\limits_{t\in\{0,1,\dots,T\}}\mu_s\cdot c_{ij}^s(t) + \sum\limits_{t\le\tilde{T}}\alpha_{ij}^s(t)\Big)\cdot y_{ij}^s(t) \\ s.t. \\ \sum\limits_{(i_t,j_{t'})\in A_s} y_{ij}^s(t) - \sum\limits_{(j_{t'},i_t)\in A_s} y_{ji}^s(t') = d_i^s(t), \quad \forall i\in V \\ t\in\{0,1,\dots,T\}, \quad s=1,2,\dots,S \\ 0\le y_{ij}^s(t)\le u_{ij}^s(t), \forall (i,j)\in A, t\in\{0,1,\dots,T\}, s=1,2,\dots,S \end{cases} \tag{8.15}$$

Bài toán con 2 có thể được phân rã thêm thành tổng cộng là S bài toán con của nó mà mỗi bài toán ấy tương đương với một bài toán min-cost flow với thời gian di chuyển và sức chứa trên mỗi cạnh là phụ thuộc vào thời gian, biểu diễn như sau:

$$\begin{cases} \min SP2(\alpha, s) = \sum\limits_{(i,j)\in A} \Big(\sum\limits_{t\in\{0,1,\ldots,T\}} \mu_s \cdot c^s_{ij}(t) + \sum\limits_{t\le\tilde{T}} \alpha^s_{ij}(t) \Big) \cdot y^s_{ij}(t) \\ s.t. \\ \sum\limits_{(i_t,j_{t'})\in A_s} y^s_{ij}(t) - \sum\limits_{(j_{t'},i_t)\in A_s} y^s_{ji}(t') = d^s_i(t), \quad \forall i \in V, t \in \{0,1,\ldots,T\} \\ 0 \le y^s_{ij}(t) \le u^s_{ij}(t), \forall (i,j) \in A, t \in \{0,1,\ldots,T\}, s = 1,2,\ldots,S \end{cases} \tag{8.16}$$

Cho từng kịch bản $s \in 1,2,\ldots,S$, bài toán con (8.16) có cùng cấu trúc với bài toán con (8.14) với chí phí $c^s_{ij(t)}$ và sức chứa của các liên kết $u^s_{ij}(t)$ có giá trị phụ thuộc vào thời gian. Bởi vì khoảng thời gian T được chia thành 2 giai đoạn, hàm chi phí khái quát $g^s_{ij}(t)$ được định nghĩa như sau:

$$g^s_{ij}(t) = \begin{cases} \mu_s \cdot c^s_{ij}(t) + \alpha^s_{ij}(t), & t \le \tilde{T} \\ \mu_s \cdot c^s_{ij}(t) & \tilde{T} < t \le T \end{cases} \tag{8.17}$$

Để giải bài toán con (8.16), đầu tiên chúng ta cần định nghĩa lại các tham số $A(y(t))$, $C(y(t))$ và $U(y(t))$ trong mạng dư $N(y(t))$ như sau:

$$A_s(y(t)) = \{(i_t, j_{t'}) | (i_t, j_{t'}) \in A_s, y^s_{ij} < u^s_{ij}\} \cup \{(j_{t'}, i_t) | (j_{t'}, i_t) \in A_s, y^s_{ij} < 0\}, s = 1,2,\ldots,S$$

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

$$u^s_{ij}(y(t)) = \begin{cases} u^s_{ij} t - y^s_{ij}(t), (i_t, j_{t'}) \in A_s, \quad y^s_{ij}(t) < u^s_{ij}(t), \quad t \in \{0,1,\ldots,T\} \\ y^s_{ji}(t), (j_{t'}, i_t) \in A_s, \quad \forall \{t' \in \{0,1,\ldots,T\} | y^s_{ji}(t') > 0\} \\ s = 1,2,\ldots,S \\ T, (j_{t'}, i_t) \in A_s, \quad \forall \{t' \in \{0,1,\ldots,T\} | y^s_{ji}(t') = 0\} \end{cases} \tag{8.18}$$

Sau đó, thuật toán label-correcting cải tiến sẽ được sử dụng để tìm đường đi với chi phí tối thiểu và không phụ thuộc vào thời gian trong mạng dư trên.
Bằng cách giải bài toán con (8.14) và (8.15), nghiệm tối ưu Z^*_{LR} của mô hình (8.13) với tập hợp các nhân tử Lagrange là vector α có thể được viết như sau:

$$Z^*_{LR}(\alpha) = Z^*_{SP1}(\alpha) + Z^*_{SP2}(\alpha) \tag{8.19}$$

Ta có thể thấy rằng nghiệm tối ưu này chính là cận dưới của nghiệm tối ưu của mô hình gốc (8.13). Để có được kết quả chính xác nhất, chúng ta phải tìm được cận dưới gần với nghiệm tối ưu của mô hình gốc nhất, nghiệm đó được kí hiệu là:

$$Z^*_{LD}(\alpha^*) = \max_{\alpha \ge 0} Z_{LR}(\alpha) \tag{8.20}$$

Và ta có thể xem nghiệm của bài toán gốc thu được khi sử dụng kết quả của X và Y thu được khi giải bài toán con 1 và 2 là cận trên của nghiệm tối ưu của bài toán gốc.

Vậy quá trình giải và tìm nghiệm cho mô hình của bài toán quy hoạch tuyến tính hai giai đoạn cho việc sơ tán người dân khỏi thảm họa sẽ diễn ra như sau:

1. Khởi tạo biến lặp $k = 1$. Khởi tạo giá trị cho nhân tử Lagrange $\alpha_{ij}^{s}(t)$. Khởi tạo giá trị của cận trên $UB^o = +\infty$
2. Giải bài toán con 1 và 2 để tìm giá trị của biến quyết định X và Y. Tính cận trên, cận dưới của bài toán (8.11) và khoảng cách tương đối giữa chúng.
3. Cập nhật giá trị cho nhân tử Lagrange bằng phương pháp subgradient. Sau đó lặp lại bước 2 với số lần lặp $k = K$ tùy ý.
4. Trả về nghiệm khả thi tốt nhất trong khi thực hiện quá trình trên.

Thế nhưng, dựa trên độ phức tạp cũng như phạm vi của bài tập lớn và kiến thức của môn Mô hình hóa toán học, chúng ta sẽ chỉ giải quyết bài toán con 1 (8.14) bằng giải thuật successive shortest path.

8.4 Thuật toán successive shortest path cho bài toán min-cost flow

Cho một mạng G gồm n đỉnh và m cạnh, mỗi cạnh sẽ bao gồm sức chứa và chi phí để gửi mỗi đơn vị của luồng đi qua nó. Đỉnh nguồn s và đích t cũng được chỉ rõ trong G. Với một giá trị K cho trước, chúng ta sẽ phải tìm những luồng thỏa mãn giá trị K và trong số những luồng đó, chọn luồng tốn ít chi phí nhất. Đây được gọi là bài toán min-cost flow và có thể được giải quyết bằng giải thuật successive shortest path. Giải thuật này khá giống với giải thuật Edmonds-Karp được dùng để tính toán số luồng tối đa.

8.4.1 Trường hợp đơn giản

Đầu tiên, chúng ta xét trường hợp đơn giản nhất của bài toán, đồ thị đang xét là đồ thị có hướng, và chỉ có tối đa một cạnh giữa bất kì cặp đỉnh nào trong đồ thị. Ví dụ, nếu (i, j) là một cạnh thì sẽ không được tồn tại thêm cạnh (j, i).

Gọi U_{ij} là sức chứa của cạnh (i, j) nếu cạnh đó tồn tại. Gọi C_{ij} là chi phí cho từng đơn vị của luồng di chuyển trên cạnh (i, j). Cuối cùng, gọi F_{ij} là luồng trên cạnh (i, j). Ban đầu, mọi luồng sẽ được khởi tạo là 0 cho mọi cạnh.

Chúng ta chỉnh sửa mạng G như sau: cho mỗi cạnh (i, j) chúng ta thêm một cạnh đảo ngược (j, i) vào trong G với $U_{ji} = 0$ và $C_{ji} = -C_{ij}$. Vì cạnh (j, i) chưa hề tồn tại trong mạng gốc nên G vẫn không phải là một đa đồ thị trong trường hợp này. Thêm vào đó, ta sẽ thêm một điều kiện $F_{ji} = -F_{ij}$ trong khi thực hiện vòng lặp của giải thuật.

Chúng ta định nghĩa một mạng dư (Residual Network) với các giá trị luồng là F tương tự như trong giải thuật Edmonds-Karp: mạng dư của chúng ta sẽ chỉ bao gồm những cạnh chưa bão hòa (những cạnh có $F_{ij} < U_{ij}$) và sức chứa của cạnh đó được tính lại bằng $R_{ij} = U_{ij} - F_{ij}$.

Sau khi đã định nghĩa xong các yếu tố của đồ thị, chúng ta sẽ bắt đầu thực hiện giải thuật. Tại mỗi bước lặp, chúng ta sẽ tìm đường đi ngắn nhất trong đồ thị dư (Residual Graph) từ s đến t bằng một thuật toán label-correcting. Khác với thuật toán Edmonds-Karp, chúng ta tìm đường đi ngắn nhất dựa trên chi phí của mỗi cạnh, không phải số lượng cạnh ngắn nhất. Nếu không còn tồn tại đường đi ngắn nhất cần tìm, giải thuật sẽ kết thúc. Trong trường hợp chúng ta tìm được đường đi ngắn nhất, ta sẽ tăng lượng luồng di chuyển trên đường đi đó nhiều nhất

có thể, nói cách khác, chúng ta sẽ tìm sức chứa dư (Residual Capacity) nhỏ nhất trên đường đi, tăng lượng luồng trên những cạnh thuộc đường đi đó bằng giá trị vừa tìm được và giảm lượng luồng trên những cạnh đảo ngược bằng giá trị tương tự. Nếu tại một thời điểm nào đó, lượng luồng của chúng ta đạt đến giá trị K, thuật toán sẽ dừng lại và trả về chi phí thấp nhất tương ứng.

8.4.2 Trường hợp đồ thị vô hướng / đa đồ thị.

Trường hợp mạng của chúng ta là một đồ thị vô hướng hay một đa đồ thị thì điều đó cũng không làm xảy ra mâu thuẫn về mặt định nghĩa so với thuật toán đã trình bày ở trên. Thuật toán successive shortest path vẫn sẽ hoạt động tốt trên những đồ thị này nhưng việc triển khai sẽ khó khăn hơn.
Một cạnh vô hướng (i, j) tương đương với hai cạnh có hướng (i, j) và (j, i) với sức chứa và chi phí như nhau. Vì thuật toán ở trên tạo thêm cạnh đảo ngược cho mỗi cạnh có hướng, một cạnh vô hướng của chúng ta sẽ biến thành bốn cạnh có hướng trong đồ thị dư G, và đồ thị của chúng ta trở thành một đa đồ thị.

Để giải quyết vấn đề đa đồ thị, đầu tiên luồng trên mỗi cạnh thuộc đa cạnh Multiple edges phải được phân biệt rõ ràng. Thứ hai, khi tìm đường đi ngắn nhất, ta phải xem xét nên chọn cạnh nào trong số đa cạnh đó. Theo đó, thay vì sử dụng mảng thông thường để lưu trữ những đỉnh thuộc đường đi ngắn nhất, chúng ta phải lưu trữ thêm cạnh mà chúng ta đã sử dụng trên đường đi. Thứ ba, chúng ta thực hiện việc điều chỉnh lượng luồng trên đường đi tương tự như đối với đồ thị có hướng, lưu ý về việc chọn cạnh và cạnh đảo ngược của nó trong số đa cạnh. Vậy là ta đã giải quyết xong các rắc rối khi đối mặt với đa đồ thị.

8.4.3 Độ phức tạp của thuật toán

Thuật toán này có độ phức tạp tỉ lệ với lũy thừa của kích thước đầu vào. Chi tiết hơn, trong trường hợp tệ nhất, mỗi lần lặp chúng ta chỉ có thể tăng thêm 1 đơn vị của luồng, và mất tổng cộng $O(F)$ lần lặp để tìm được chi phí tối thiểu cho bài toán, khiến cho tổng thời gian cần thiết để chạy giải thuật là $O(F \cdot T)$ với T là thời gian để tìm đường đi ngắn nhất từ s đến t.

Nếu thuật toán label-correcting chúng ta chọn là Bellman-Ford, tổng thời gian để chạy thuật toán là $O(Fmn)$. Chúng ta cũng có thể sử dụng giải thuật Dijkstra cải tiến để tìm đường đi ngắn nhất, như vậy thuật toán của chúng ta sẽ cần $O(mn)$ cho bước tiền xử lý và $O(m \log(n))$ cho mỗi lần lặp. Như vậy tổng thời gian cần thiết cho thuật toán được giảm xuống còn $O(mn + Fm \log(n))$. Ngoài ra chúng ta có thể kết hợp thuật toán Dijkstra cải tiến ở trên với thuật toán Dinic để giới hạn lại số lần lăp tối đa từ F thành $\min(F, nC)$, với C là chi phí tối đa của mỗi đường đi.

8.5 Phần code cho thuật toán successive shortest path

Quay trở lại với bài toán con 8.14, ta có đồ thị trong trường hợp của bài toán quy hoạch ngẫu nhiên hai giai đoạn là đồ thị có hướng như đã đề cập trong trường hợp đơn giản. Cụ thể hơn, đồ thị mà chúng ta sẽ xét là một grid network gồm 50 đỉnh được đánh số từ 0 đến 49 với nguồn là đỉnh 0 và đích là đỉnh 49. Và để đơn giản hóa vấn đề, vì chúng ta chỉ xét Sub-problem 1 nên hàm chi phí của chúng ta là $p_{ij} - \sum_{s=1}^{S} \sum_{t \leq \bar{T}} \alpha_{ij}^{s}(t)$ sẽ được định nghĩa lại thành c_{ij}.

Đầu tiên, chúng ta cần định nghĩa một class để đại diện cho mỗi cạnh trong đồ thị đang xét, chúng ta sẽ dùng nó như một tập hợp của các cạnh $(i, j) \in A$.

```
# Edge: Set of edge on our network
class Edge:
```

```
    def __init__(self, from_, to, capacity, cost):
        self.from_ = from_
        self.to = to
        self.capacity = capacity
        self.cost = cost
```

Tiếp theo, ta phải chọn một thuật toán label-correcting phù hợp để tìm đường đi ngắn nhất từ nguồn đến đích cho mỗi lần lặp. Và thuật toán được sử dụng trong bài báo cáo này sẽ là thuật toán SPFA (Shortest Path Faster Algorithm), một thuật toán được cải tiến từ thuật toán Bellman-Ford thông thường. Thuật toán này còn có khả năng phát hiện Negative Cycle nếu nó tồn tại trong đồ thị của chúng ta. Nhưng vì chúng ta đang xét grid network nên có thể bỏ qua phần này.

```
# SPFA Algorithm (Improved Bellman-Ford) to get the shortest path
def shortest_paths(n, v0, adj, cost, capacity):
    # d: distance from source to other nodes
    d = [INF]*n
    d[v0] = 0

    # inqueue: To keep track of which nodes are currently in the queue
    inq = [False]*n

    # q: queue for shortest path algorithm
    q = deque([v0])

    # p: Keep track of the path by store the previous node of the current node
    p = [-1]*n

    # count: Counter to check for Negative Cycle
    count = [0]*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]:
                    q.append(v)
                    inq[v] = True
                    count[v] += 1
                    if count[v] > n:
                        return None  # Negative cycle

    return d, p
```

Sau khi đã định nghĩa xong những gì cần thiết, chúng ta sẽ dựa vào những gì đã trình bày về các bước của thuật toán ở mục trên để hiện thực hàm min-cost-flow như sau:

```
def min_cost_flow(N, edges, K, s, t):
    print("Total Flow: ", K)

    # Create adjacency list
    adj = [[] for _ in range(N)]
    cost = [[0]*N for _ in range(N)]
    capacity = [[0]*N for _ in range(N)]

    # Create residual network
    for e in edges:
        adj[e.from_].append(e.to)
        adj[e.to].append(e.from_)
```

```
        cost[e.from_][e.to] = e.cost
        cost[e.to][e.from_] = -e.cost
        capacity[e.from_][e.to] = e.capacity

    # Initialize the result variables
    flow = 0
    cost_ = 0


    while flow < K:
        result = shortest_paths(N, s, adj, cost, capacity)

        # Case 1: Negative Cycle
        if result is None:
            raise ValueError("Negative cycle detected")
        d, p = result

        # Case 2: There is no shortest path from source to sink
        if d[t] == INF:
            break;

        # Case 3: Shortest path exists
        f = K - flow
        cur = t
        path = []
        while cur != s:
            f = min(f, capacity[p[cur]][cur])
            path.append(cur)
            cur = p[cur]
        path.append(s)
        path = path[::-1]

        # Print out the shortest path
        print('Path:', ' -> '.join(map(str, path)))
        print('Flow sent on this path: ', f)
        print('Cost per flow on this path: ', d[t])
        print("------------------------------------------------------------------")

        # Update the residual network
        flow += f
        cost_ += f * d[t]
        cur = t
        while cur != s:
            capacity[p[cur]][cur] -= f
            capacity[cur][p[cur]] += f
            cur = p[cur]

    # Return the answer
    if flow < K:
        return -1
    else:
        return cost_
```

Vậy là chúng ta đã có hàm min-cost-flow để có thể lên kế hoạch phân bố lượng flow trên mỗi cạnh sao cho thời gian sơ tán của người dân là ngắn nhất. Việc còn lại là sinh dữ liệu cho grid network để kiểm tra việc thực thi của chương trình.

Để sinh dữ liệu cho grid network, chúng ta sẽ cần đến hai thư viện của Python là *networkx* để tạo grid network và *matplotlib* để vẽ grid network. Chúng ta sẽ tạo một grid network với kích thước 5x10, mỗi cạnh của network sẽ chứa thông tin bao gồm chi phí cũng như sức chứa của nó dưới định dạng là "cost | capacity". Các giá trị của mỗi cạnh sẽ được thiết lập ngẫu nhiên sử

dụng thư viện *random*. Sau khi đã tạo xong network, chúng ta đánh số lại các đỉnh từ 0 đến 49, tô màu cho mỗi đỉnh, cuối cùng là vẽ đồ thị của grid network hiện tại và in ra màn hình. Cụ thể đoạn code sau sẽ mô tả lại hành động sinh dữ liệu cho mạng của chúng ta:

```
import networkx as nx
import matplotlib.pyplot as plt
import random

# Create grid network 5x10 (50 nodes)
G = nx.grid_2d_graph(5, 10)

# Add cost and capacity for each edge
for (u, v) in G.edges():
    G.edges[u, v]['cost'] = random.randint(1, 10)  # Cost from 1 to 10
    G.edges[u, v]['capacity'] = random.randint(200, 700)  # Capacity fromm 200 to
    700

# Change the node number to 0-49
mapping = {node: i for i, node in enumerate(G.nodes())}
G = nx.relabel_nodes(G, mapping)

# Fix the position of the grid network on 2-D plane
pos = {i: (i % 10, 4 - i // 10) for i in range(50)}

# Set the color the each node
source_node = 0  # Source is node 0
target_node = 49  # Sink is node 49
node_colors = ['salmon' if node == source_node else 'lightgreen' if node ==
    target_node else 'skyblue' for node in G.nodes()]
# Draw the network
nx.draw(G, pos, with_labels=True, node_size=500, node_color=node_colors, font_size
    =8)

# Create label for each edge with the format "cost | capacity"
edge_labels = {(u, v): f"{G.edges[u, v]['cost']} | {G.edges[u, v]['capacity']}"
    for u, v in G.edges()}
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_color='green',
    rotate=False)

# Plot the network
plt.show()
```

Vậy là việc hiện thực hàm min-cost-flow và việc sinh dữ liệu đã được hoàn tất. Chúng ta chỉ cần ghép những thứ rời rạc ở trên thành một đoạn code hoàn chỉnh thì đã có thể có một chương trình để giải quyết bài toán luồng chi phí tối thiểu trên một grid network gồm 50 đỉnh. Đoạn code hoàn chỉnh cho vấn đề 2 như sau:

```
import networkx as nx
import matplotlib.pyplot as plt
import random
from collections import deque


INF = 1e9

# Edge: Set of edge on our network
class Edge:
    def __init__(self, from_, to, capacity, cost):
        self.from_ = from_
        self.to = to
        self.capacity = capacity
        self.cost = cost
```

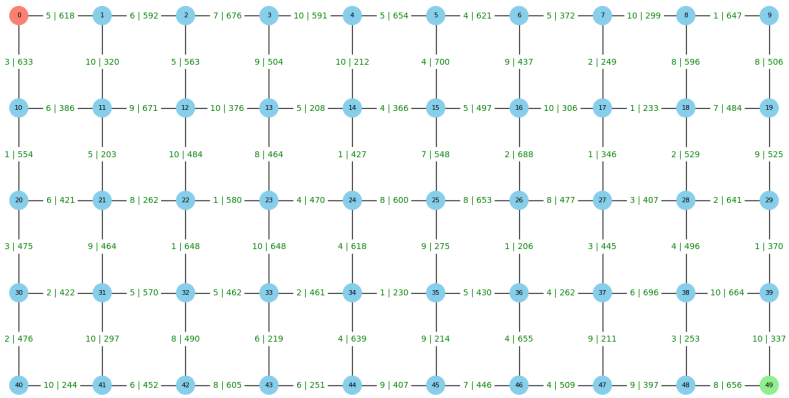
```python

# SPFA Algorithm (Improved Bellman-Ford) to get the shortest path
def shortest_paths(n, v0, adj, cost, capacity):
    # d: distance from source to other nodes
    d = [INF]*n
    d[v0] = 0

    # inqueue: To keep track of which nodes are currently in the queue
    inq = [False]*n

    # q: queue for shortest path algorithm
    q = deque([v0])

    # p: Keep track of the path by store the previous node of the current node
    p = [-1]*n

    # count: Counter to check for Negative Cycle
    count = [0]*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]:
                    q.append(v)
                    inq[v] = True
                    count[v] += 1
                    if count[v] > n:
                        return None  # Negative cycle

    return d, p

def min_cost_flow(N, edges, K, s, t):
    print("Total Flow: ", K)

    # Create adjacency list
    adj = [[] for _ in range(N)]
    cost = [[0]*N for _ in range(N)]
    capacity = [[0]*N for _ in range(N)]

    # Create residual network
    for e in edges:
        adj[e.from_].append(e.to)
        adj[e.to].append(e.from_)
        cost[e.from_][e.to] = e.cost
        cost[e.to][e.from_] = -e.cost
        capacity[e.from_][e.to] = e.capacity

    # Initialize the result variables
    flow = 0
    cost_ = 0


    while flow < K:
        result = shortest_paths(N, s, adj, cost, capacity)

        # Case 1: Negative Cycle
        if result is None:
            raise ValueError("Negative cycle detected")
```

```
        d, p = result

        # Case 2: There is no shortest path from source to sink
        if d[t] == INF:
            break;

        # Case 3: Shortest path exists
        f = K - flow
        cur = t
        path = []
        while cur != s:
            f = min(f, capacity[p[cur]][cur])
            path.append(cur)
            cur = p[cur]
        path.append(s)
        path = path[::-1]

        # Print out the shortest path
        print('Path:', ' -> '.join(map(str, path)))
        print('Flow sent on this path: ', f)
        print('Cost per flow on this path: ', d[t])
        print("_____________________________________________________________________")

        # Update the residual network
        flow += f
        cost_ += f * d[t]
        cur = t
        while cur != s:
            capacity[p[cur]][cur] -= f
            capacity[cur][p[cur]] += f
            cur = p[cur]

    # Return the answer
    if flow < K:
        return -1
    else:
        return cost_



# Create grid network 5x10 (50 nodes)
G = nx.grid_2d_graph(5, 10)

# Add cost and capacity for each edge
for (u, v) in G.edges():
    G.edges[u, v]['cost'] = random.randint(1, 10)  # Cost from 1 to 10
    G.edges[u, v]['capacity'] = random.randint(200, 700)  # Capacity from 200 to
    700

# Change the node number to 0-49
mapping = {node: i for i, node in enumerate(G.nodes())}
G = nx.relabel_nodes(G, mapping)

# Fix the position of the grid network on 2-D plane
pos = {i: (i % 10, 4 - i // 10) for i in range(50)}

# Set the color the each node
source_node = 0  # Source is node 0
target_node = 49  # Sink is node 49
node_colors = ['salmon' if node == source_node else 'lightgreen' if node ==
    target_node else 'skyblue' for node in G.nodes()]
# Draw the network
```

```
nx.draw(G, pos, with_labels=True, node_size=500, node_color=node_colors, font_size=8)

# Create label for each edge with the format "cost | capacity"
edge_labels = {(u, v): f"{G.edges[u, v]['cost']} | {G.edges[u, v]['capacity']}" for u, v in G.edges()}
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_color='green', rotate=False)


# Create the graph based on the grid network
edges = []
for u, v, attr in G.edges(data=True):
    edges.append(Edge(u, v, attr['capacity'], attr['cost']))
    edges.append(Edge(v, u, 0, -attr['cost']))  # Reverse edge in residual network

# Determine the source and sink
s = 0  # Source is node 0
t = 49  # Sink is node 49

# The amount of flow to be sent from source to sink
K = 893  # For example: Flow is 893
N = len(G.nodes())

# Call the min-cost flow function
cost = min_cost_flow(N, edges, K, s, t)
print("Minimum cost to transfer all flow: ", cost)


# Plot the network
plt.show()
```

8.6 Một số ví dụ minh họa

Trường hợp 1: Có thể gửi toàn bộ Flow từ nơi nguy hiểm đến nơi an toàn.



Hình 8.3: Grid network trong trường hợp 1.

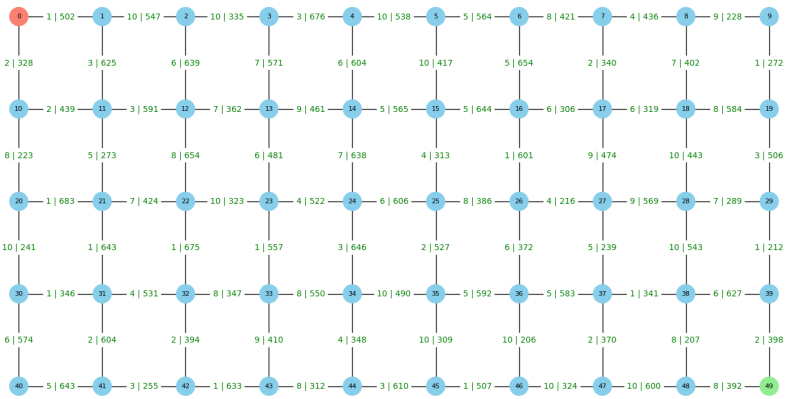
Cho grid network với dữ liệu từng cạnh như trên, sau khi thực thi hàm min-cost-flow, ta có được những đường đi ngắn nhất cùng với số flow và chi phí trên mỗi đường đi như sau:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SQL CONSOLE

PS D:\Coding Project\Python\GAMSPy> & D:/Python/python.exe "d:/Coding Project/Python/GAMSPy/MM_231_Q2.py"
Total Flow:  893
Path: 0 -> 10 -> 20 -> 30 -> 31 -> 32 -> 33 -> 34 -> 35 -> 36 -> 37 -> 38 -> 48 -> 49
Flow sent on this path:  230
Cost per flow on this path:  48
-----------------------------------------------------------------------------
Path: 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 17 -> 18 -> 28 -> 29 -> 39 -> 49
Flow sent on this path:  233
Cost per flow on this path:  60
-----------------------------------------------------------------------------
Path: 0 -> 10 -> 20 -> 21 -> 22 -> 23 -> 24 -> 25 -> 26 -> 36 -> 37 -> 38 -> 48 -> 49
Flow sent on this path:  23
Cost per flow on this path:  61
-----------------------------------------------------------------------------
Path: 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 17 -> 27 -> 28 -> 29 -> 39 -> 49
Flow sent on this path:  16
Cost per flow on this path:  61
-----------------------------------------------------------------------------
Path: 0 -> 10 -> 20 -> 30 -> 31 -> 32 -> 33 -> 34 -> 44 -> 45 -> 46 -> 47 -> 48 -> 49
Flow sent on this path:  192
Cost per flow on this path:  62
-----------------------------------------------------------------------------
Path: 0 -> 10 -> 20 -> 21 -> 22 -> 23 -> 24 -> 25 -> 26 -> 27 -> 28 -> 29 -> 39 -> 49
Flow sent on this path:  88
Cost per flow on this path:  63
-----------------------------------------------------------------------------
Path: 0 -> 10 -> 20 -> 21 -> 22 -> 23 -> 24 -> 25 -> 26 -> 36 -> 46 -> 47 -> 48 -> 49
Flow sent on this path:  21
Cost per flow on this path:  65
-----------------------------------------------------------------------------
Path: 0 -> 1 -> 2 -> 12 -> 13 -> 14 -> 15 -> 16 -> 26 -> 36 -> 46 -> 47 -> 48 -> 49
Flow sent on this path:  90
Cost per flow on this path:  68
-----------------------------------------------------------------------------
Chi phí tối thiểu để chuyển flow là: 52332
```

Hình 8.4: Kết quả của grid network trong hình 8.3.

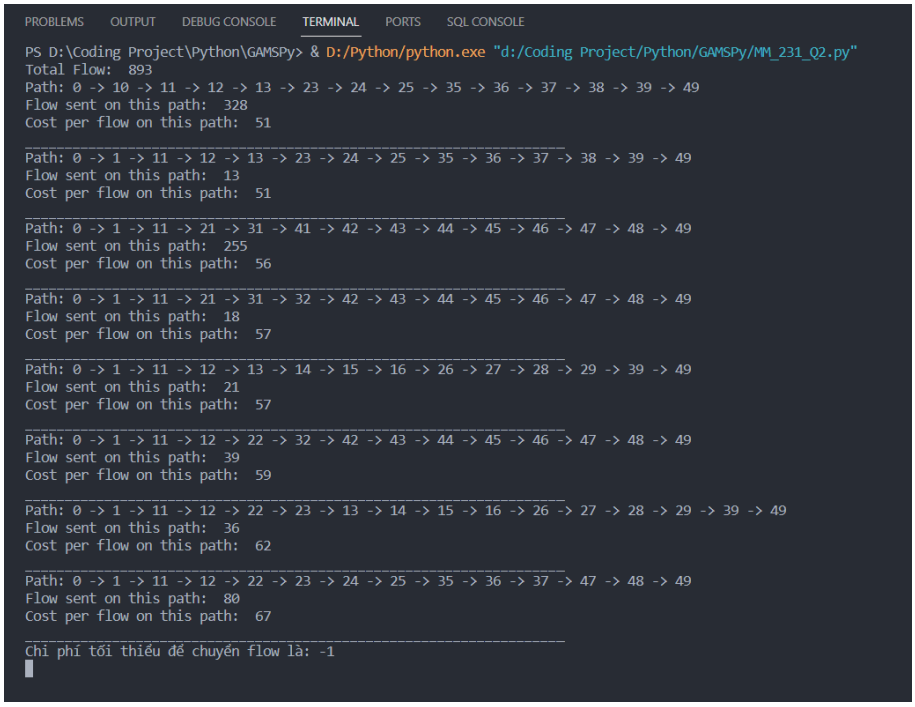
Có thể thấy chương trình của chúng ta liên tục tìm những đường đi ngắn nhất và gửi lượng flow từ đỉnh 0 đến 49 cho đến khi số lượng flow cần phải chuyển đã đạt số lượng yêu cầu là 893. Mỗi đường đi sẽ bao gồm chi phí của cũng như số lượng flow tối đa có thể gửi trên đường đi đó. Cuối cùng tổng chi phí để gửi toàn bộ flow sẽ được in ra ở dòng cuối cùng của terminal.

Trường hợp 2: Không thể gửi toàn bộ Flow từ nơi nguy hiểm đến nơi an toàn.



Hình 8.5: Grid network trong trường hợp 2.

Tương tự như trường hợp 1, ta cũng có kết quả cho grid network ở hình 8.5 như sau:



Hình 8.6: Kết quả của grid network trong hình 8.5.

Trong trường hợp này, sau khi xét đến đường đi cuối cùng với chi phí là 67, chương trình của chúng ta không thể tìm được một đường đi khác từ đỉnh 0 đến đỉnh 49 nữa. Vì vậy kết quả trả về sẽ là -1 để thể hiện rằng chúng ta không thể sơ tán toàn bộ người dân đến nơi an toàn dựa trên hệ thống đường xá hiện tại.

Trường hợp 3: Có chu trình âm trong network đang xét.

Chúng ta tạm thời bỏ qua trường hợp này vì network ta đang xét là một grid network, tức là không hề có chu trình trong đồ thị. Nhưng ta phải lưu ý rằng nếu đồ thị của chúng ta là một đồ thị thông thường có chu trình và chi phí trên mỗi cạnh có thể âm thì ta phải có phương pháp phù hợp để giải quyết trường hợp xuất hiện chu trình âm trong đồ thị.

9 APPLICATION II: SP in Telecommunications

9.1 Giới thiệu vấn đề

Trong bài toán này, chúng ta giả định rằng cơ sở hạ tầng của **network** đã được xây dựng. Một nửa vấn đề của nhà cung cấp dịch vụ Internet có thể được giải quyết bởi chủ sở hữu mạng nhưng cũng có thể là nhà cung cấp dịch vụ ảo không sở hữu mạng riêng và thuê mạng từ chủ sở hữu mạng nào đó. Các quyết định cần xem xét bao gồm việc giới thiệu dịch vụ theo từng giai đoạn, có thể hình thành triển khai giai đoạn 1, sau đó là giai đoạn 2 tùy theo phản ứng thị trường. Ngoài ra, các quyết định bao gồm giá cả dịch vụ. Trong số các khía cạnh khác nhau của vấn đề, ta xem xét khía cạnh địa lý, sự không ổn định của nhu cầu và chi phí, cơ cấu chi phí bao gồm chi phí cố định và biến đổi, cạnh tranh và thay thế giữa các dịch vụ, mối quan hệ

giữa các tác nhân thị trường. Quyết định tiếp tục dự án phụ thuộc vào lợi nhuận dự án và vào **options** được đưa vào dự án. Ví dụ như lựa chọn mở rộng, lựa chọn từ bỏ, lựa chọn nâng cấp công nghệ,... Ta nên bắt đầu phát triển mô hình từ trường hợp đơn giản, chỉ bao gồm một số tính năng có liên quan và mở rộng mô hình theo từng bước. Mô hình sẽ được trình bày thông qua ba bước phát triển.

9.2 Mô hình giảm thiểu chi phí xác định trong một thời kỳ

Chúng ta bắt đầu bằng việc chỉ xem xét một giai đoạn quyết định và có đầy đủ thông tin về nhu cầu cũng như những điều không chắc chắn khác. Mặc dù những giả định này rất khó xảy ra ngoài thực tế, nhưng mô hình kết quả sẽ tạo tiền đề cho những mô hình thực tế hơn. Trong bước này, chúng ta giả định rằng chương trình triển khai thỏa mãn đầy đủ các yêu cầu đã biết. Vì giá dịch vụ cố định nên doanh thu cố định, do đó cách duy nhất nhà cung cấp dịch vụ có thể làm để hiệu quả lợi nhuận đó chính là giảm thiểu chi phí.
Ký hiệu:
$i = 1 : n$ - chỉ số cho các khu vực cấu thành một lãnh thổ. Dân số người dùng tồn tại ở mỗi khu vực tạo ra nhu cầu.
$i = 1 : m$ - chỉ mục cho các vị trí máy chủ có thể.
y_j - là biến nhị phân, có giá trị bằng 1 nếu quyết định đặt **server** ở vị trí j và giá trị 0 là ngược lại.
x_{ij} - số lượng nhu cầu ở khu i được phục vụ bởi **server** đặt tại j.
f_j - chi phí cố định để thiết lập máy chủ ở vị trí j.
c_{ij} - chi phí phục vụ một đơn vị nhu cầu từ khu vực i bởi máy chủ tại vị trí j.
d_i - nhu cầu sinh ra tại khi vực i.
g_j - dung lượng máy chủ đặt tại vị trí j.

Mô hình:

Tìm chương trình triển khai máy chủ $y = (y_1, ..., y_m)$ và phân công nhóm người dùng tới máy chủ $x = x_{ij}$, $x = 1 : n$, $j = 1 : m$ là nghiệm của:

$$\min_{x,y} f_j y_j + \sum_{j=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{9.1}$$

$$\sum_{j=1}^{m} x_{ij} \geq d_i, i = 1 : n \tag{9.2}$$

$$\sum_{i=1}^{n} x_{ij} \leq g_j y_j, i = 0 : n \tag{9.3}$$

với $y_j \in 0, 1$ và $x_{ij} \geq 0$. Trong đó, số hạng đầu tiên của biểu thức đầu tiên thể hiện chi phí triển khai cố định, số hạng thứ hai thể hiện chi phí biến đổi để phục vụ nhu cầu. Các ràng buộc (9.2) được áp dụng để đạt được sự thỏa mãn về nhu cầu, trong khi các điều kiện (9.3) là những giới hạn. Đây là mô hình cơ sở khởi đầu để phát triển mô hình quy hoạch ngẫu nhiên với các kịch bản khác nhau về nhu cầu tương đối và số lượng lớn hơn trong các giai đoạn triển khái.

9.3 Mô hình giảm thiểu chi phí ngẫu nhiên hai giai đoạn

Hai giai đoạn được xem xét là: giai đoạn 1 hiện tại với nhu cầu đã biết, giai đoạn 2 trong tương lai với nhu cầu không chắc chắn được mô tả bằng một số kịch bản hữu hạn. Mỗi kịch bản được mô tả bằng giá trị nhu cầu ở các khu vực khác nhau trong giai đoạn 2 và xác suất xảy ra kịch bản này. Các quyết định của giai đoạn 2 bao gồm việc triển khai bổ sung các máy chủ và phân bổ lại nhu cầu cho các máy chủ để đáp ứng nhu cầu đã được biết đến. Mô hình tuân theo khuôn khổ quy hoạch ngẫu nhiên.

Ký hiệu:

$r = q : R$ - chỉ số cho các kịch bản nhu cầu.
d_i^r - nhu cầu tạo ra bởi vùng i theo kịch bản r
p^r - xác suất của kịch bản r
z_j^r - biến nhị phân nhận giá trị 1 nếu trong kịch bản r, quyết định đặt máy chủ ở vị trí j được thực hiện và 0 nếu ngược lại
x_{ij}^r - lượng nhu cầu từ khu vực i được phục vụ bởi máy chủ đặt ở vị trí j trong kịch bản r.
α - hệ số chiết khấu chi phí giai đoạn 2 về hiện tại
Mỗi kịch bản được đặc trưng bởi một (d^r, p^r) với $d^r = (d_1^r, ..., d_n^r)$

Mô hình:

Tìm chương trình triển khai máy chủ giai đoạn 1 $y = (y_1, ..., y_m)$ và phân công nhóm người dùng tới máy chủ $x = x_{ij}$, $x = 1 : n$, $j = 1 : m$ là nghiệm của:

$$\min_{x,y} f_j y_j + \sum_{j=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \alpha \sum_{r=1}^{R} p_r Q(r, y) \tag{9.4}$$

tuân theo điều kiện (9.2)-(9.3) đã đề cập ở trên. Ở biểu thức (9.4) xuất hiện thêm hạng tử thứ ba thể hiện chi phí chiết khấu khi triển khai giai đoạn 2 tính trung bình theo các kịch bản. Chi phí liên quan đến kịch bản r là Q(r, y) và nó phụ thuộc vào quyết định triển khai giai đoạn 1 y. Những chi phí này có được từ việc giải bài toán cho từng kịch bản r:

$$Q(r, y) = \min_{x^r, z^r} \sum_{j=1}^{m} f_j z_j^r + \sum_{j=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij}^r \tag{9.5}$$

$$\sum_{j=1}^{m} x_{i,j}^r \geq d_i^r, i = 1 : n \tag{9.6}$$

$$\sum_{i=1}^{n} x_{ij}^r \leq g_j(y_j + z_j^r), j = 1 : m \tag{9.7}$$

tương tự như giai đoạn 1 (từ các biểu thức (9.1), (9.2), (9.3)) và chọn triển khai giai đoạn 2 $z^r = (z_1^r, ..., z_n^r)$ và phân chia lại nhóm người được phục vụ bởi máy chủ $x^r = x_{ij}^r, i = 1 : n, j = 1 : m$ từ việc giảm thiểu chi phí triển khai cố định và chi phí dịch vụ thay đổi cho một kịch bản nhất định r.

9.4 Mô hình tối đa hóa lợi nhuận ngẫu nhiên hai giai đoạn với việc định giá.

Trong môi trường cạnh tranh được bãi bỏ quy định, tối đa hóa lợi nhuận là mục tiêu phù hợp hơn so với việc giảm thiểu chi phí mạng lưới. Về cơ bản, nó trở nên khác biệt so với việc giảm thiểu chi phí đơn giản khi các quyết định về giá được xem xét đồng thời với các quyết định triển khai. Các mô hình trở nên phức tạp hơn vì giá cả ảnh hưởng đến nhu cầu và sự phụ thuộc này tạo ra sự phi tuyến tính. Tuy nhiên, phân tích có ý nghĩa cũng khả thi trong trường hợp này.

Ký hiệu:

h_0 - giá tham chiếu trong giai đoạn 1
d_{i0} - cầu tham chiếu tại vùng i trong giai đoạn 1 tương ứng với giá tham chiếu h_0
w_i - độ co giãn của cầu tại vùng i trong giai đoạn 1
h - mức tăng giá so với giá tham chiếu trong giai đoạn 1
h_0^r - giá tham chiếu dịch vụ giai đoạn 2 theo kịch bản r
d_{i0}^r - cầu tham chiếu tại khu vực i trong giai đoạn 2 tương ứng với giá tham chiếu h_0^r theo kịch bản r
w_i^r - Độ co giãn của cầu tại vùng i trong giai đoạn 2 theo kịch bản r
h^r - mức tăng giá so với giá tham chiếu trong Giai đoạn 2 theo kịch bản r

1. *Mô hình nhu cầu:*
   Đây là phần quan trọng của mô hình lợi nhuận. Xét triển khai trong giai đoạn 1, Giá dịch vụ bằng $h_0 + h$ và quyết định giá bao gồm việc lựa chọn mức tăng giá h có thể dương hoặc âm. Giả sử rằng nhu cầu d_i ở vùng i trong giai đoạn này phụ thuộc vào giá của dịch vụ theo hàm $d_i = f_i(h_0 + h)$ và trong vùng lân cận của điểm h_0 sự phụ thuộc này có thể được tuyến tính hóa thông qua:
   $$d_i = d_{i0} - w_i h \tag{9.8}$$
   Các mối quan hệ tương tự mô tả hành vi nhu cầu trong Giai đoạn 2 cho từng kịch bản r = 1 : R. Mỗi kịch bản được xác định trong trường hợp này bởi một bộ dữ liệu $(d_{i0}^r, h_0^r, w_i^r, p^r)$ xác định sự phụ thuộc của nhu cầu vào giá dựa trên công thức trên trong 1 kịch bản r nào đó.

2. *Mô hình quyết định*
   Tìm lượng gia tăng ở Giai đoạn 1 cho giá dịch vụ h , chương trình triển khai máy chủ $y = (y_1, ..., y_m)$, và phân chia nhóm người dùng cho máy chủ $x = \{x_{ij}\}, i = 1 : n, j = 1 : m$ bằng cách tìm nghiệm của bài toán:
   $$\max_{h,x,y} W(h) - C(y, x) + \alpha \sum_{r=1}^{R} p_r P(r, y) \tag{9.9}$$
   thỏa mãn
   $$w_i + \sum_{j=1}^{m} x_{ij} \geq d_{i0}, \quad i = 1 : n \tag{9.10}$$
   và ràng buộc (9.3). Ở đây, $W(h)$ là doanh thu trong Giai đoạn 1:
   $$W(h) = \sum_{i=1}^{n} (h + h_i)(d_0 - w_i h) \tag{9.11}$$

và $C(y, x)$ là chi phí trong Giai đoạn 1 được định nghĩa ở (9.1). Toán hạng thứ ba trong (9.9) đại diện cho lợi nhuận trong Giai đoạn 2 được lấy trung bình qua các kịch bản và đã chiết khấu đến hiện tại, trong đó $P(r, y)$ là lợi nhuận trong Giai đoạn 2 dưới kịch bản r. Nó được lấy làm giá trị tối ưu của bài toán truy đòi sau:

$$P(r, y) = \max_{h^r, x^r, z^r} W(r, h^r) - C(r, z^r, x^r) \tag{9.12}$$

sao cho:

$$w_i^r h^r + \sum_{j=1}^{m} x_{ij}^r \geq d_{i0}^r, \quad i = 1 : n \tag{9.13}$$

và thỏa mãn các ràng buộc bổ sung (9.7). Ở đây, $W(r, h^r)$ là doanh thu trong Giai đoạn 2 dưới kịch bản r được tính tương tự như (9.11) và $C(r, z^r, x^r)$ là chi phí trong Giai đoạn 2 dưới kịch bản r được lấy từ (9.5).

Có một đặc điểm quan trọng của mô hình này mà không có trong các mô hình (9.1)-(9.3) và (9.4)-(9.7). Trong khi (9.4)-(9.7) có thể được chuyển đổi thành một bài toán LP nguyên hỗn hợp bằng cách xem xét các bài toán xác định tương đương, không có phép chuyển đổi tương tự nào có thể áp dụng cho mô hình (9.9)-(9.13). Đó là bởi vì doanh thu $W(r, h)$ và $W(r, h^r)$ phụ thuộc phi tuyến tính vào các biến quyết định h và h^r. Ngay cả trong trường hợp đơn giản nhất của mô hình nhu cầu tuyến tính (9.8), sự phụ thuộc này cũng là bậc hai. Do đó, các kỹ thuật số học chuyên biệt nên được áp dụng trong trường hợp này, với hướng phân rã là cách tiếp cận hứa hẹn nhất.

3. *Đánh giá cơ hội đầu tư, lựa chọn thực tế*
Một trong những ứng dụng quan trọng nhất của mô hình (9.9)-(9.13) là đánh giá tính lợi nhuận của dự án đầu tư, bao gồm triển khai dịch vụ mới. Những phát triển gần đây trong lĩnh vực tài chính doanh nghiệp cho thấy tầm quan trọng của việc xem xét các lựa chọn thực tế để đánh giá chính xác các dự án công nghiệp Trigeorgis (1996).. Trong khi đối với các ngành công nghiệp truyền thống khác, các kỹ thuật đánh giá trực tiếp có thể tương tự cho việc đánh giá các lựa chọn tài chính, thì đối với các ngành công nghiệp sáng tạo với các dự án độc đáo, các phương pháp như vậy là khó áp dụng. Khi đó các mô hình lập trình ngẫu nhiên là một giải pháp thay thế hợp lệ để đánh giá lựa chọn thực tế. Hãy sử dụng mô hình (9.9)-(9.13) cho mục đích này. Cụ thể, hãy đánh giá các lựa chọn mở rộng, nâng cấp công nghệ, dở bỏ hoặc chuyển đổi một phần cơ sở hạ tầng.

4. *Lựa chọn mở rộng (lựa chọn chờ và xem)*
Lựa chọn này đã được tích hợp trong mô hình (9.9)-(9.13), trong đó có khả năng thêm các máy chủ bổ sung trong Giai đoạn 2 dựa trên phản ứng của thị trường. Giá trị của lựa chọn này có thể được tính như sau. Ký hiệu P^* cho giá trị tối ưu của mô hình (9.9)-(9.13). Đây là giá trị của dự án với lựa chọn mở rộng. Giá trị $\hat{P}$ của cùng dự án mà không có lựa chọn mở rộng thu được bằng cách giải quyết cùng một mô hình với các biến nhị phân z^r được giữ cố định bằng 0 cho tất cả các kịch bản. Rõ ràng, $\hat{P} \leq P^*$. Giá trị của lựa chọn là hiệu $\hat{} - P^*$.

5. *Lựa chọn nâng cấp công nghệ*
Đây là một lựa chọn có giá trị vì nó có thể thay đổi đáng kể việc đánh giá dự án. Ví dụ nổi tiếng nhất là mạng di động GSM, được phát triển khi mà công nghệ làm điện thoại di động đủ nhỏ chưa cho phép. Để đánh giá lựa chọn này, cần xem xét cách mà sự phát triển công nghệ có thể ảnh hưởng đến các yếu tố khác nhau của mô hình (9.9)-(9.13). Ví dụ, sự phát triển công nghệ có thể dẫn đến giảm chi phí cố định cho việc cài đặt máy chủ

và/hoặc tăng dung lượng máy chủ có thể có trong Giai đoạn 2. Trong trường hợp này, cần đưa các tính năng này vào định nghĩa của các kịch bản. Các ký hiệu là:

f_j^r - chi phí cố định để thiết lập máy chủ tại vị trí j trong kịch bản r.

g_i^r - dung lượng của máy chủ được đặt tại vị trí i trong kịch bản r.

Mô hình thay đổi như sau. Phần (9.9)-(9.10) vẫn giữ nguyên vì nó mô tả Giai đoạn 1 được triển khai với công nghệ đã biết. Phần (22)-(23) có ràng buộc về khả năng được sửa đổi thay thế (9.7):

$$\sum_{i=1}^{n} x_{ij}^r \leq g_j y_j + g_j^r z_j^r, \quad j = 1:m \tag{9.14}$$

Ngoài ra, chi phí $C(r, z^r, x^r)$ ở (9.12) được tính

$$C(r, z^r, x^r) = \min_{x^r, z^r} \sum_{j=1}^{m} f_j^r z_j^r + \sum_{j=1}^{m} \sum_{i=1}^{n} c_{ij} x_{ij}^r \tag{9.15}$$

Giải mô hình (9.9)-(9.13) với điều chỉnh (24)-(25) sẽ cho giá trị P^{**} của dự án với lựa chọn nâng cấp công nghệ. Giá trị này được so sánh với giá trị của dự án P^* mà không nâng cấp công nghệ, và hiệu $P^{**} - P^*$ sẽ cho ra giá trị của lựa chọn.

6. *Lựa chọn dỡ bỏ*
Đây là một lựa chọn có giá trị khi phản ứng của thị trường là không chắc chắn. Nếu nhu cầu không đáp ứng, giải pháp hợp lý là cắt giảm chi phí bảo trì ở các khu vực có nhu cầu ít và có thể rút ra một phần chi phí cố định bằng cách bán hoặc cho thuê cơ sở hạ tầng máy chủ. Các ký hiệu bổ sung là:

b_j^r - chi phí bảo trì cho máy chủ tại vị trí j trong Giai đoạn 2 dưới kịch bản r.

β_j^r - tỷ lệ phần trăm chi phí cố định có thể thu hồi bằng cách dỡ bỏ máy chủ tại vị trí j trong kịch bản r.

u_j^r - biến nhị phân có giá trị 1 nếu máy chủ tại vị trí j được dỡ bỏ trong kịch bản r.

Mô hình thay đổi như sau. Phần (9.9)-(9.10) liên quan đến Giai đoạn 1 vẫn được giữ nguyên. Phần ràng buộc về dung lượng (9.12)-(9.13) được sửa đổi thay thế (9.7):

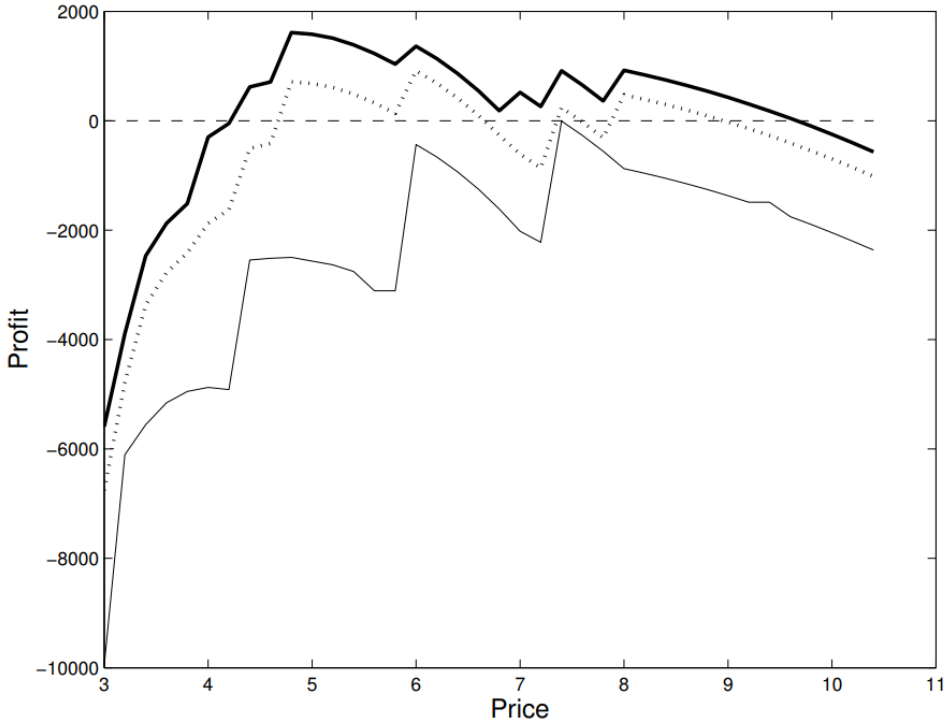
$$\sum_{i=1}^{n} x_{ij}^r \leq g_j(y_j + z_{rj} - u_{rj}), j = 1:m, r = 1:R \tag{9.16}$$

Và các ràng buộc dỡ bỏ bổ sung:

$$u_{rj} \leq y_j, \quad j = 1:m, \quad r = 1:R \tag{9.17}$$

Doanh thu $W(r, h^r)$ và chi phí $C(r, z^r, x^r)$ từ (9.9) được tính bởi

$$W(r, h^r) = \sum_{i=1}^{n} (h_0^r + h^r)(d_{i0}^r - w_i^r h^r) + \sum_{j=1}^{m} \beta_j^r f_j u_j^r. \tag{9.18}$$



Hình 9.1: Đánh giá các phương án thực tế trong trường hợp giới thiệu dịch vụ

$$C(r, z^r, x^r) = \sum_{j=1}^{m} f_j z_j^r + b_j^r. \tag{9.19}$$

Giải mô hình (9.9)-(9.13) với điều chỉnh (9.16)-(9.19) sẽ cho giá trị P^{++} của dự án với lựa chọn dỡ bỏ cơ sở hạ tầng. Giá trị này được so sánh với giá trị của dự án không có lựa chọn dỡ bỏ P^+ được tính bằng cách giải cùng mô hình với các biến u_j^r được đặt thành 0. Hiệu $P^{++} - P^+$ là giá trị của lựa chọn.

Dưới đây là ví dụ về việc đánh giá lựa chọn được mô tả trong Hình 1. Hình này cho thấy sự phụ thuộc của giá trị dự án vào giá dịch vụ $h_0 + h$ trong trường hợp giá dịch vụ Giai đoạn 2 được giữ nguyên theo giá dịch vụ Giai đoạn 1, tức là $h_{r0} \equiv h_0$ và $h_r \equiv h$. Ba lựa chọn thay thế được biểu diễn trong hình. Lựa chọn đầu tiên được biểu thị bằng đường cong mảnh và mô tả sự phụ thuộc của giá trị dự án vào giá cho trường hợp không có lựa chọn mở rộng và không có lựa chọn nâng cấp công nghệ trong Giai đoạn 2. Lựa chọn thứ hai cho phép lựa chọn mở rộng, nhưng không cho phép lựa chọn nâng cấp công nghệ và được biểu thị bằng đường cong chấm. Lựa chọn thứ ba được biểu thị bằng đường đậm và cho phép cả hai lựa chọn trong Giai đoạn 2.

Lợi nhuận của dự án mà không có lựa chọn thêm nào là không dương ngay cả khi chọn giá dịch vụ tốt nhất. Dự án sẽ có lãi khi cho phép các lựa chọn thêm. Có hai vùng lợi nhuận liên quan đến giá dịch vụ. Vùng thứ nhất tương ứng với mức giá dịch vụ thấp được thiết

kế để kích thích nhu cầu tăng cao và vùng thứ hai tương ứng với hành vi ít tích cực hơn với giá cao và nhu cầu nhỏ hơn. Các vùng lợi nhuận này sẽ mở rộng khi có thêm lựa chọn bổ sung để nâng cấp công nghệ. Trong trường hợp không cân nhắc các lựa chọn, mô hình khuyến khích dùng mức giá cao, trong khi các lựa chọn làm cho mô hình linh hoạt hơn khi cho phép kích thích nhu cầu mạnh mẽ hơn với mức giá thấp hơn.

10 Kết luận

Trong bài báo cáo này, chúng ta đã khám phá sâu rộng về lý thuyết và các ứng dụng của Stochastic Programming. Chúng ta đã thấy rằng, thông qua việc sử dụng các mô hình toán học chính xác, Stochastic Programming cho phép chúng ta giải quyết các vấn đề phức tạp trong thực tế một cách hiệu quả, đặc biệt là trong lĩnh vực Evacuation Planning và Telecommunication. Trong lĩnh vực Evacuation Planning, Stochastic Programming đã chứng minh được giá trị của mình trong việc tối ưu hóa các kế hoạch sơ tán, giảm thiểu rủi ro và đảm bảo an toàn cho cộng đồng. Các mô hình đã được áp dụng để dự đoán và chuẩn bị cho các tình huống khẩn cấp, từ đó giúp cứu sống nhiều sinh mạng. Trong lĩnh vực Telecommunication, việc áp dụng Stochastic Programming đã giúp tối ưu hóa việc phân bổ tài nguyên và quản lý băng thông, đảm bảo sự ổn định và hiệu quả của các mạng lưới truyền thông. Điều này không chỉ cải thiện chất lượng dịch vụ mà còn góp phần vào sự phát triển bền vững của ngành công nghiệp này.

Nhìn chung, Stochastic Programming là một công cụ mạnh mẽ, cung cấp khả năng dự đoán và quản lý rủi ro trong nhiều tình huống không chắc chắn. Sự linh hoạt và khả năng thích ứng của nó với các tình huống thực tế đã và sẽ tiếp tục là chìa khóa cho sự thành công trong nhiều lĩnh vực khác nhau. Chúng ta có thể mong đợi rằng, với sự tiến bộ không ngừng của công nghệ và toán học, Stochastic Programming sẽ tiếp tục mở rộng phạm vi ứng dụng của mình và đóng góp nhiều hơn nữa vào việc giải quyết các thách thức toàn cầu.

References

[1] Minimum-cost flow - Successive shortest path algorithm "<**https://cp-algorithms.com/graph/min_cost_flow.html**>"

[2] Maximum flow - Ford-Fulkerson and Edmonds-Karp "<**https://cp-algorithms.com/graph/edmonds_karp.html**>"

[3] Bellman-Ford Algorithm "<**https://cp-algorithms.com/graph/bellman_ford.html**>"

[4] Alexei A. Gaivoronski (2002). Stochastic Optimization Problems in Telecommunications

[5] Alexander Shapiro, Darinka Dentcheva, Andrzej Ruszczynski (2009). Lectures on stochastic programming modeling and theory.

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

[7] Li Wang (2020). A two-stage stochastic programming framework for evacuation planning in disaster responses.

[8] GAMSPy document "<**https://gamspy.readthedocs.io/en/latest/index.html**>".

[9] Robert J. Vanderbei (2000). Linear Programming: Foundations and Extensions.

[10] Athanasios Ziliaskopoulos, Dimitrios Kotzinos, Hani S. Mahmassani (1997). Design and implementation of parallel time-dependent least time path algorithms for intelligent transportation systems applications.