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

MATHEMATICAL MODELLING (CO2011)

Assignment

Stochastic Programming and Applications

Advisor: Nguyễn Tiến Thịnh
Students: Võ Nguyễn Đức Phát - 2212540.
Nguyễn Đăng Cường - 2210432.
Nguyễn Bá Việt Quang - 2212741.
Lý Thanh Nhật Quang - 2212737.
Trần Uy - 2213897.

HO CHI MINH CITY, DECEMBER 2023

Contents

1 Member list & Workload

No.	Fullname	Student ID	Problems	Percentage of work
1	Võ Nguyễn Đức Phát	2212540	- Problem 1: Cơ sở lý thuyết - Problem 2: Mô tả Bài toán, giải quyết Bài toán	100%
2	Nguyễn Đăng Cường	2210432	- Problem 1: Giải quyết bài toán Coding - Problem 2: Mô tả Bài toán	100%
3	Nguyễn Bá Việt Quang	2212741	- Problem 1: Giải quyết bài toán - Problem 2: Độ phức tạp & So sánh thuật toán	100%
4	Lý Thanh Nhật Quang	2212737	- Problem 1: Cơ sở lý thuyết - Problem 2: Độ phức tạp & So sánh thuật toán	100%
5	Trần Uy	2213897	- Problem 1: Cơ sở lý thuyết - Problem 2: Coding, giải quyết bài toán	100%

2 Problem 1. Industry - Manufacturing

2.1 Cơ sở lý thuyết

Bài toán 2-SLPWR là bài toán cố gắng đưa các vấn đề thực tiễn trong cuộc sống để về một cái chúng ta có thể nhận xét, phân tích được, đó là đưa về Mô hình toán học. 2-SLPWR là viết tắt của 2 Stage Linear Programming With Recourse. Tức là bài toán của chúng ta sẽ chia làm 2 giai đoạn: gọi đó là First Stage và Second Stage. Trong cả 2 Stage sẽ có những Biến, gọi là biến quyết định (Decision Variables). Ta sẽ thay đổi các Biến này trong quá trình giải bài toán để có thể tìm ra kết quả tối ưu nhất có thể.
First Stage: Ta sẽ chọn biến quyết định của First Stage, và nó sẽ ảnh hưởng đến cả bài toán và những Stage sau. Để có thể chọn chính xác sao cho giá trị đạt tối ưu nhất thì ta cần nhìn đến Second Stage.

2.2 Giải quyết bài toán

Problem 1 sẽ được giải quyết sử dụng Stochastic linear program (SLP) bao gồm 2 Stage. Mô tả đề bài như sau: đề bài ban đầu có **n** sản phẩm, mỗi sản phẩm cần có **m** linh kiện, mục tiêu của chúng ta là phải sử dụng 2 Stage Linear Program (2-SLPWR) để tìm ra số lượng sản phẩm cần sản xuất và số lượng linh kiện ban đầu sao cho chi phí sản xuất là tiết kiệm nhất.

- Với đề bài được cho: số sản phẩm $\mathbf{n} = 8$, và số linh kiện $\mathbf{m} = 5$. Ta được phép cho các giá trị đầu vào là các Ma trận ngẫu nhiên. Lần lượt là

$$\mathbf{b} = \begin{bmatrix}2\\4\\3\\2\\2\end{bmatrix} \quad \mathbf{s} = \begin{bmatrix}1\\1\\1\\2\\1\end{bmatrix} \quad \mathbf{a} = \begin{bmatrix}1&2&1&0&6\\2&3&5&2&1\\4&1&4&5&1\\1&5&3&1&5\\2&1&2&1&0\\1&4&5&5&4\\3&5&2&1&2\\6&7&5&1&8\end{bmatrix} \quad \mathbf{l} = \begin{bmatrix}60\\60\\10\\50\\20\\40\\70\\50\end{bmatrix} \quad \mathbf{q} = \begin{bmatrix}90\\70\\95\\75\\85\\85\\80\\100\end{bmatrix}$$

2.2.0.1 Chú thích 1 Các ma trận cho trước

- b: Giá tiền đặt trước của 5 linh kiện (b_j với j = 1, 2,... m) trước khi biết demand
- s: Giá tiền sao cho $s_j \leqslant b_j$ (với j = 1, 2,... m)
- a: Là sản phẩm thứ i cần j linh kiện (i = 1, 2,... n và j = 1, 2,... m. Ví dụ: $a_{15} = 6$, tức là sản phẩm 1 cần 6 linh kiện 5 để sản xuất)
- l: Giá tiền bỏ ra thêm để phù hợp với nhu cầu của sản phẩm i (i = 1, 2,... n)
- q: Ma trận gồm giá bán của từng sản phẩm i (j = 1, 2,... n)

Để thuận tiện cho việc giải quyết bài toán, ta sẽ gộp 2 ma trận về giá cả **l** và **q** lại thành 1, sử dụng phép trừ ma trận (**l - q**). Ta gọi đó là ma trận lq:

$$\mathbf{lq} = \begin{bmatrix}-30\\-10\\-85\\-25\\-65\\-45\\-10\\-50\end{bmatrix}$$

Bắt đầu với bài toán, đây là bài toán Lập trình ứng dụng ngẫu nhiên. Tức là ta chưa biết được cái nhu cầu của từng sản phẩm là bao nhiêu, và sẽ có bao nhiêu viễn cảnh (scenerio) có thể xảy ra. Đề bài đã đưa ra, số scenerio **S** = 2 và xác suất xảy ra mỗi scenerio $\rho = 1/2$. Đi sâu vào bài toán, ta sẽ đưa bài toán này về dạng 2-SLPWR trong đó tuân theo luật **production** $\geq$ **demand**.

The first-stage problem:

Ta tuân theo luật và bắt đầu tìm ra giá trị tối ưu của First-Stage, với bài toán này biến quyết định của First Stage là biến **x** với giá trị preorder cost là ma trận b = (b_1, b_2,... b_m) đã được nêu trên. Số lượng của x_j được xác định dựa vào biểu thức sau:

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

Tuy nhiên, vì đây là bài toán 2-SLPWR cho nên, để có thể giải ra được biểu thức trên, ta cần phải xem xét và nhìn về Stage 2 của bài toán.

The second-stage problem:

Ở giai đoạn 2, ta sẽ có 2 biến quyết định là y và z tương ứng với cả 2 scenerio. Mỗi scenerio có xác suất xảy ra là 50%. Đề bài mô tả $\omega = D = (D_1, D_2, ..., D_n)$ trong đó w_i với xác suất p_i

tuân theo quy tắc Binomial Distribution Bin(10, 1/2). Ta tìm ra được quy tắc là Demand phải bé hơn 10 và từ đó đưa ra được các số ngẫu nhiên tạo nên 2 ma trận ứng với mỗi scenerio.

$$\omega_1 = \begin{bmatrix} 9 \\ 8 \\ 6 \\ 4 \\ 8 \\ 4 \\ 3 \\ 2 \end{bmatrix} \qquad \omega_2 = \begin{bmatrix} 5 \\ 4 \\ 7 \\ 5 \\ 6 \\ 3 \\ 8 \\ 7 \end{bmatrix}$$

2.2.0.2 Chú thích 2 Các ma trận demand

- ω_1: là ma trận chứa các Demand của scenerio 1
- ω_2: là ma trận chứa các Demand của scenerio 2

Ta tiến hành chuyển toàn bộ Bài toán về Model toán học:

$$\begin{cases} \min \text{ g(x, y, z)} = b^T \text{ . x} + lq^T \text{ . z - } s^T \text{ . y} \\ \text{y = x - az} \\ 0 \leqslant \text{z} \leqslant \text{d, y} \geqslant 0. \end{cases}$$

với b, lq, a là các ma trận đã được chú thích ở 2.2.0.1

- x là ma trận (5x1): số linh kiện cần đặt trước sản xuất
- y là ma trận (5x1): số linh kiện còn xót lại ở trong Kho
- y có 2 ma trận là: y_1 và y_2 ứng với scenerio 1 và scenerio 2
- z là ma trận (8x1): số sản phẩm cần đặt trước sản xuất (Điều kiện: z $\leqslant$ demand)

Để tìm ra được chi phí tối ưu, ta viết lại:

$$\text{z} = b^T \text{ . } x + 0.5 \left[lq^T \text{ . } z_1 - s^T \text{ . } y_1\right] + 0.5 \left[lq^T \text{ . } z_2 - s^T \text{ . } y_2\right]$$

$$z = \begin{bmatrix} b_1 & b_2 & b_3 & b_4 & b_5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} + 0.5(\begin{bmatrix} lq_1 & lq_2 & ... & lq_8 \end{bmatrix} \begin{bmatrix} z1_{11} \\ z1_{21} \\ ... \\ z1_{81} \end{bmatrix} - \begin{bmatrix} s_{11} & s_{21}... & s_{51} \end{bmatrix} \begin{bmatrix} y1_{11} \\ y1_{21} \\ ... \\ y1_{51} \end{bmatrix})$$

$$+ 0.5(\begin{bmatrix} lq_1 & lq_2 & ... & lq_8 \end{bmatrix} \begin{bmatrix} z2_{11} \\ z2_{21} \\ ... \\ z2_{81} \end{bmatrix} - \begin{bmatrix} s_{11} & s_{21}... & s_{51} \end{bmatrix} \begin{bmatrix} y2_{11} \\ y2_{21} \\ ... \\ y2_{51} \end{bmatrix})$$

Ta sẽ sử dụng giải thuật Simplex và tìm ra được các nghiệm tối ưu cũng như kết quả.

2.3 Giải thuật Simplex - Coding

Để có thể giải quyết các bài toán 2-SLPWR như trên, ta có thể dùng giải thuật Simplex để có thể giải quyết nó, ở đây chúng em sẽ giải bài toán này trên ngôn ngữ lập trình Python. Thư viện bọn em sử dụng là GAMsPY:

```
print Run file Assignment.py
```

Sau khi chạy File Assignment.py kèm với file báo cáo, kết quả chi phí tối ưu ta ra được sẽ là **-613.0**

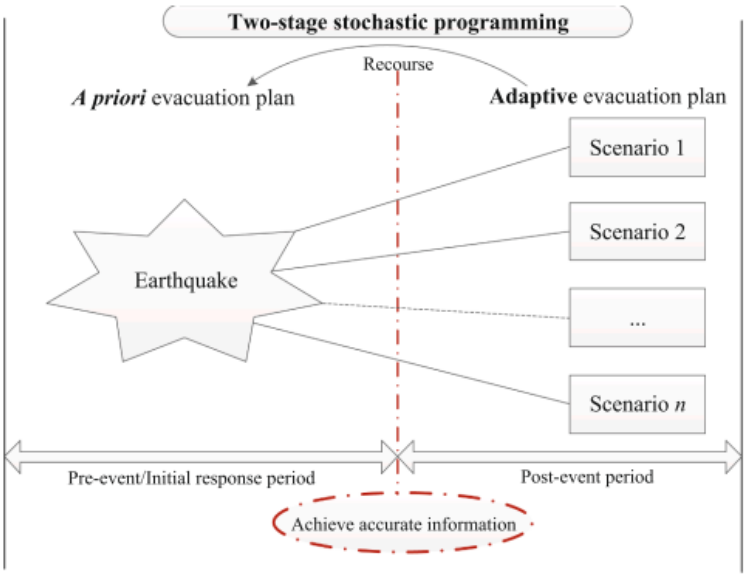
Kết luận: Ta thấy chi phí tối ưu là 1 số âm, điều đó chứng tỏ là chúng ta đang sinh lợi nhuận, việc tính toán đặt trước các linh kiện sản phẩm mang lại cho ta 1 kế hoạch để sản xuất đạt lợi nhuận tốt nhất. Từ đó ta thấy rằng bài toán 2-SLPWR giúp ích rất nhiều trong các lĩnh vực trong đời sống, một trong số đó có thể kể đến là Kế hoạch sản xuất như bài Problem 1 trên và sắp tới là liên quan đến Bài toán Di cư tránh nạn khẩn cấp ở Problem 2.

3 Problem 2. Stochastic Linear Program for Evacuation Planning In Disaster Responses

3.1 Mô tả Bài toán

Thảm hoạ thiên nhiên (động đất, sóng thần, cháy rừng...) hoặc các thảm hoạ khác (các cuộc khủng bố, chiến tranh) thường xảy ra 1 cách bất ngờ nhưng để lại hậu quả rất lớn cho xã hội. Nhiều nhà khoa học đã tìm ra nhiều phương pháp, các mô hình để có thể dự báo, đoán trước được các tính huống có thể xảy ra để có thể đối mặt với nó một cách hợp lý nhất. Vấn đề của Problem 2 này xoay quanh điều đó, **tạo nên 1 mô hình toán học** dựa trên tình huống thật đang xảy ra, và sử dụng mô hình đó để đưa ra các kế hoạch di tản phù hợp nhất.

Bài toán này được coi là Stochastic Programming (Lập trình ngẫu nhiên) vì có những viễn cảnh, những điều mà ta không biết trước được: ví dụ như là Capacity (Có thể hiểu là số lượng tối đa mà **xe có thể đi** trên đoạn đường đó) cũng như Time Cost (thời gian mà 1 xe tốn mất để đi từ điểm này đến điểm khác). Bên cạnh đó thì cũng còn nhiều thứ ngẫu nhiên khác: không biết đó là thảm hoạ nào, nên ta sẽ có nhiều scenerio (*Ví dụ: scenerio 1: là núi lửa, scenerio 2 là sóng thần...*)

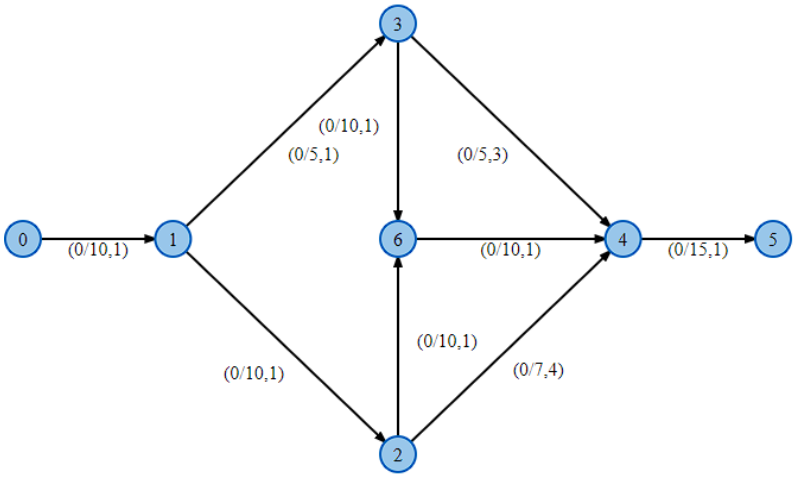


Hình 1: Mô hình mô tả về bài toán Stochastic for Evacuation

Hình 1 được trích từ cuốn *L. Wang, A two-stage stochastic programming framework for evacuation planning in disaster responses, Computers & Industrial Engineering, page 2* cho ta thấy tổng quát được bài toán này, bài toán được chia làm 2 giai đoạn. Giai đoạn 1 là giai đoạn trước thảm hoạ, ta cần phải quan sát và nhìn xem các viễn cảnh mà tương lai có thể xảy ra để có thể đưa ra kế hoạch hiệu quả nhất. Bên phải ta thấy có rất nhiều scenerio: scenerio 1, scenerio 2,.. và mỗi scenerio đều có xác suất xảy ra khác nhau, ứng với mỗi scenerio đó là các sự kiện khác nhau cho nên ta cần phải quan sát đến Stage 2 này và đưa ra kế hoạch cho Stage 1. Đó là bài toán Two-stage Stochastic Programming mà ta sẽ chuyển qua.
Theo những gì đã được đề cập ở trên thì sẽ thấy rằng bài toán ta được chuyển về như sau: Tìm con đường di chuyển hợp lý nhất từ nơi có **nguy hiểm** về nơi **an toàn**. Hợp lý nhất được hiểu là đường đi mà lượng xe lưu thông trên được là cao nhất (gọi là Max Flow) nhưng đồng thời **Time cost** phải là nhỏ nhất. Ta gọi bài toán này là Min-Cost Flow. Để giải quyết được bài toán, ta tiến hành chia nhỏ bài toán này thành 2 bài toán con, và cả 2 bài toán con này đều có thể giải quyết được bằng thuật toán **Successive shortest path** (SSP Algorithm). Bài toán này cần rất nhiều giai đoạn để có thể đưa ra kết quả cuối cùng, nhưng nhóm em đã dựa trên Đề bài tập lớn và quyết định chọn việc học, thực hiện và so sánh độ phức tạp của giải thuật SSP đã nêu trên và sẽ được trình bài ở phần 3.2. Bên cạnh đó, sau khi đã giới thiệu xong về Successive Shorest Path, nhóm em sẽ tiến hành chọn 1 giải thuật khác để so sánh về độ phức tạp để chứng minh độ hiệu quả của thuật toán SSP, và giải thuật đó là giải thuật Cycle Canceling.

3.2 Giải quyết bài Toán

Bài toán này được chia ra nhỏ ra thành nhiều bài toán con, nhưng có thể nắm được ý tưởng để thực hiện là chúng ta tiến hành thực hiện nó trên đồ thị có hướng. Chúng em sẽ tiến hành giải thích, đưa ra ví dụ cụ thể về công dụng của thuật toán này, thuật toán Successive shortest path. Cùng nhau quan sát đồ thị bên dưới để hiểu rõ thêm.



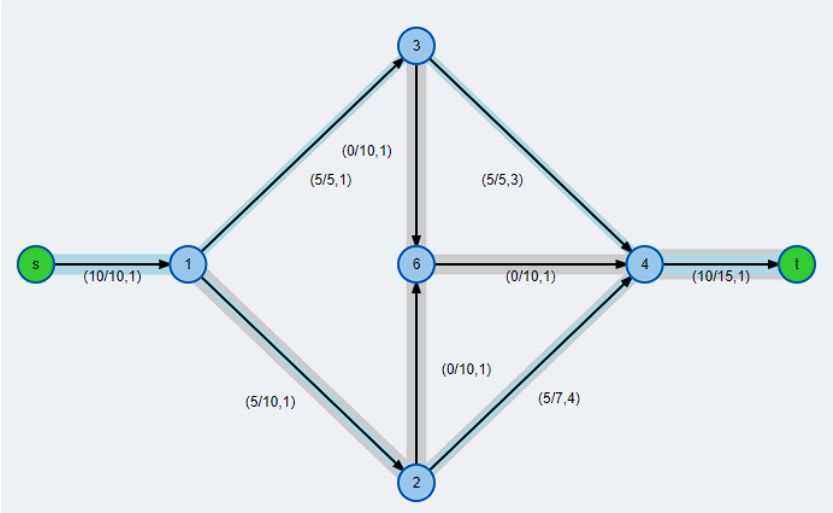
Hình 2: Đồ thị mô tả sự lưu chuyển của các xe

Hình 2 mô tả về sự di tản của các xe trên con đường là đồ thị gồm có 7 nodes, từ $0 \rightarrow 6$, trong đó:

- Node 0 là node nguồn (**source**), trong bài toán của chúng em đây là **Nơi nguy hiểm**
- Node 5 là node đích (**sink**), đây là **Nơi an toàn**.

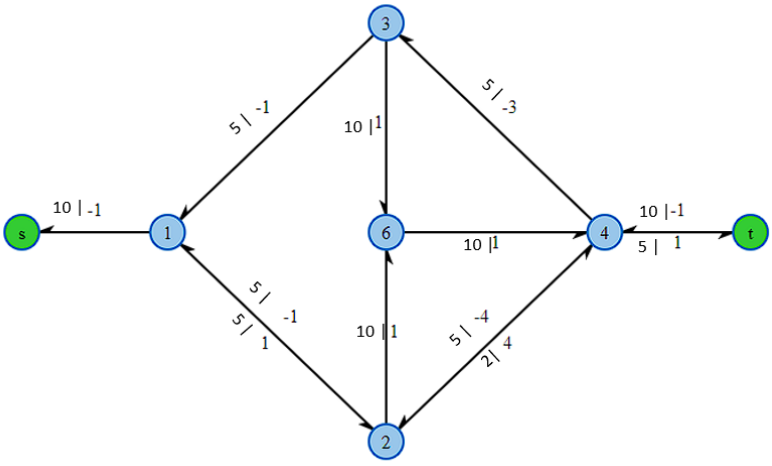
Mục tiêu của chúng em là tìm đường đi sao cho mà có **nhiều xe nhất** cùng di chuyển trên 1 đoạn đường, gọi là **Flow**, ví dụ từ $0 \rightarrow 1$ gọi là 1 Flow, đồng thời **chi phí thời gian** đi trên đoạn đường đó là thấp nhất. Để tiến hành giải quyết và tìm hiểu sâu về giải thuật Successive shortest path, trước hết chúng em sẽ giới thiệu về các điều bên dưới đây.

Residual Network: Đến với ý tưởng ban đầu, ta có đồ thị G và x là nghiệm có thể chấp nhận được của bài toán minimum cost flow. Ta giả sử cạnh (i, j) trong đồ thị đang có x_{ij} đơn vị flow, ta có thể tìm ra được residual capacity của cạnh (i, j) đó theo công thức $r_{ij} = u_{ij} - x_{ij}$. Điều này có nghĩa là, ta có thể gửi đến 1 cái Node nào đó bất kì thêm 1 cái lượng flow bằng r_{ij} nếu như ta gửi cái flow đó theo đường đi (i, j). Còn nếu ta muốn ở 1 Node nào đó mất đi bớt cái Flow units thì ta có thể gửi r_{ij} theo hướng (j, i). Tóm lại, gửi 1 đơn vị Flow nếu theo hướng (i, j) thì sẽ làm tăng flow cost lên c_{ij}, còn nếu đi theo hướng (j, i) thì sẽ làm giảm flow cost xuống c_{ij}. Cụ thể như hình sau:



Hình 3: Đồ thị mô tả sự lưu chuyển của các xe trên nhiều con đường

Ta có thể thấy như sau: 2 điểm s, t tương ứng là điểm đích và nguồn (là vị trí của **Node** 5 và **Node** 7), ở mỗi cạnh từ i đến j, đều có những kí tự số, đó lần lượt là số **đơn vị flow** (x_{ij}) trên cạnh, **capacity** (u_{ij}) và **cost** (c_{ij}). Ví dụ: Từ điểm 1 đi đến điểm *2* là (5/10, 1) thì có nghĩa là cạnh này đang có 5 đơn vị flow, và capacity của cạnh này là 10 và cost là 1.



Hình 4: Đồ thị mô tả về Residual Network

Hình trên mô tả G_x là 1 Residual Network, ta có 1 số quy tắc như sau, trên 1 cạnh giữa 2 Node với nhau, sẽ tách ra làm 2 hướng, ví dụ từ s đến 1 thì ta sẽ có 2 hướng: là từ s đến **1** và từ **1** về s, đối với hướng đi từ s đến **1**, ta sẽ có giá trị **cost** là c_{ij} và (residual) **capacity** sẽ bằng $r_{ij} = u_{ij}$ - x_{ij}, nhìn về Hình 3, lúc này u_{ij} đang là 10, x_{ij} cũng đang là 10 nên ta có capacity của hướng đi từ s đến **1** là bằng 0, bằng 0 nên ta sẽ không ghi vào đồ thị trên, còn đối với hướng đi từ **1** đến s thì **cost** sẽ bằng -c_{ij} (đối với ví dụ trên là bằng -1) và capacity sẽ bằng với x_{ij} (theo hình trên là bằng 10). Ta tương tự làm tiếp tục với các Node khác trên hình và sẽ có được Residual Network G_x.

Successive Shortest Path - Algorithm Đây là thuật toán 1 được mô tả trong bài báo của *L . Wang*, như đã mô tả ở trên, thuật toán này khởi tạo ra 1 cái Flow ban đầu, và tiến hành tìm kiếm, gia tăng các giá trị... sao cho tìm ra được kết quả tối ưu nhất. Thuật toán này được mô tả theo mô hình như sau:

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

Trong đó:

- p_{ij}: là chỉ số xuất hiện trên 1 cạnh trong Residual Network khi ta cần cân bằng lại sao cho tìm ra được kết quả
- a_{ij}: là nhân tử Larange
- S: là số scenerio có thể xảy ra, như bài toán của nhóm em, số scenerio là 1
- x_{ij}: là số flow được phân bổ cho 1 con đường nào đó (như mô tả ở Hình 3)

Đó là mô hình thứ nhất của bài toán, bài toán này được chia làm 2 Sub-problem nhỏ, bài trên là Mô hình của Sub-problem 1, đối với Sub-problem 2 thì sẽ có thêm các yếu tố biến xác định là x và y. Ta thấy biểu thức $\sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ji} = d_i$ nói về flow balance của từng cạnh bất kì trên đồ thị. Có nghĩa là bất kì Node nào nếu cho nó là cạnh nguồn (source) thì nó phải có giá trị flow là dương, nếu mà cái node đó là điểm nguồn (sink) thì giá trị flow balance ở đó sẽ là âm. Ngoài ra thì toàn bộ các flow sẽ phải tuân theo : $0 \leqslant x_{ij} \leqslant u_{ij}, \forall (i,j) \in A$ tức là số xe được phép tối đa di chuyển trên 1 đoạn đường nào đó thì phải **không được** lớn hơn Capacity trên đoạn đường đó.

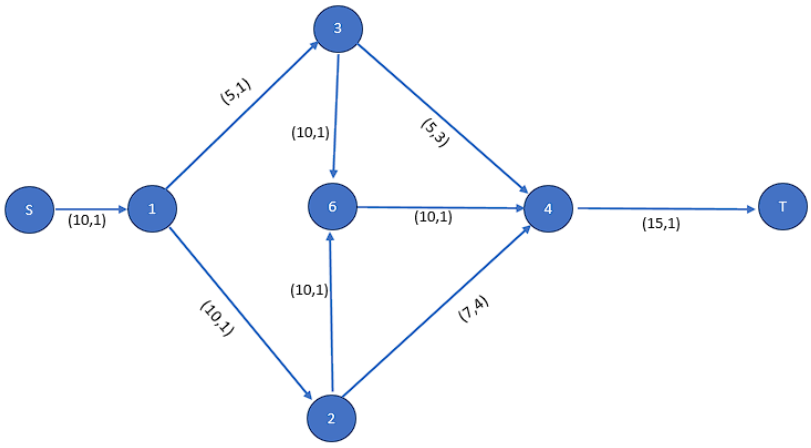
$$\begin{cases} \min SP2(\alpha) = \sum_{s=1}^{S}\sum_{(i,j)\in A}\left(\sum_{t\in\{0,1,\cdots,T\}}\mu_s \cdot c_{ij}^{S}(t) + \sum_{t\leqslant T}\alpha_{ij}^{S}(t)\right)y_{ij}^{s}(t) \\ \text{ s. } t. \\ \sum_{(it,jt')\in A_s} y_{ij}^{s}(t) - \sum_{(t',it)\in A_s} y_{ij}^{s}(t') = d_i^{s}(t), \forall i \in V, \\ t \in \{0,1,\cdots,T\}, s = 1,2,\cdots,S \\ 0 \leqslant y_{ij}^{s}(t) \leqslant u_{ij}^{s}(t), \forall (i,j) \in A, t \in \{0,1,\cdots,T\}, s = 1,2,\cdots,S \end{cases}$$

Để có thể tính **Cost** trên từng Flow, ta sẽ tìm được bằng $x_{ij}c_{ij}$. Bên trên là các ý tưởng ban đầu được mô hình lại theo kiểu Toán học, ta có thể tóm tắt ý tưởng bài toán như này:

- **Bước 1**: Dùng giải thuật đường đi ngắn nhất tìm kiếm 1 Feasible Flow từ **Nơi nguy hiểm** (Điểm s) đến **Nơi an toàn** (Điểm t) mà ở đó chi phí của Cost là thấp nhất (Giả sử ta truyền 1 Flow vào), như Hình 3, từ s $\longrightarrow$ t, thay vì ta truyền đi 10 flow, thì giờ đây ta truyền 1 flow và tính được cost nhỏ nhất của nó. Sau khi tìm ra được Feasible Flow đó xong, ta tiến hành truyền nhiều Flow nhất có thể vào con đường vừa tìm kiếm được.

- **Bước 2**: Từ flow đó, ta tiến hành tạo nên được 1 **Residual Network** được mô tả ở Hình 3, có thể chu trình sẽ bao gồm các chu trình mà **cost** < 0 dẫn đến không tìm được đường đi ngắn nhất từ điểm nguy hiểm đến nơi an toàn. Vậy nên ta sẽ tiến hành sử dụng thuật toán **Bellman-Ford** để kiểm tra xem có chu trình âm không. Khi tìm thấy chu trình âm, ta sẽ tiến hành điều chỉnh flow bằng cách xoá 1 flow của 1 cạnh trên 2 Node (i, j) bất kì về 0 và điều chỉnh các Flow còn lại sao cho hợp lý. **Lặp lại** việc kiểm tra chu trình âm và hiệu chỉnh cho đến khi không còn chu trình âm nào.
- **Bước 3**: Sử dụng giải thuật Tìm kiếm đường đi ngắn nhất và cập nhật **cost**. Giải thuật chỉ dừng lại khi **Residual Network** không còn chu trình âm nào nữa. Kết thúc bài toán thì đó cũng chính là Max flow và Min cost cần tìm.

Cho ví dụ như sau:

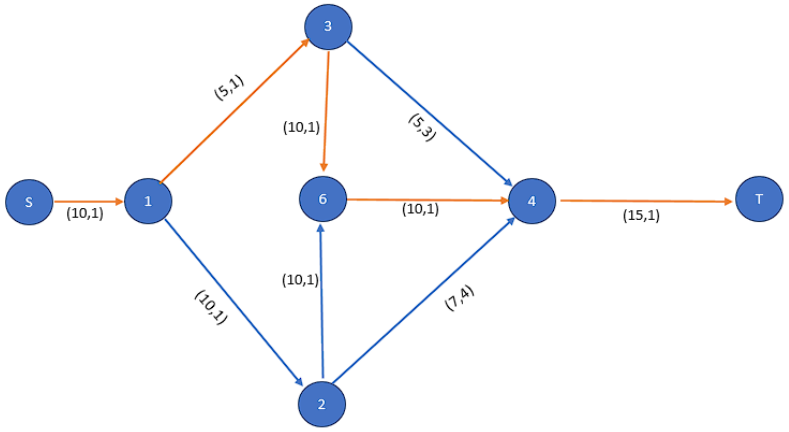


Hình 5: Shortest Path Algorithm, đồ thị minh hoạ

Hình 6: Shortest Path Algorithm, chú thích

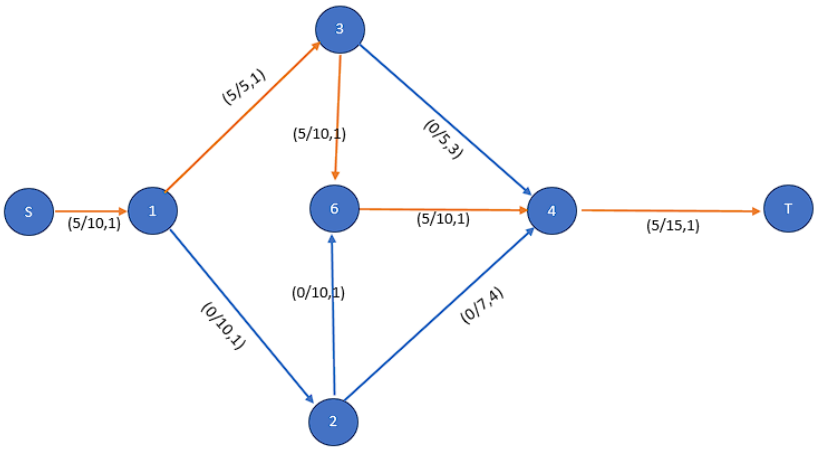
Ta tiến hành thực hiện các bước đã nêu như trên:

Bước 1: Dùng giải thuật Bellman-Ford đi từ s đến t với chi phí thấp nhất.



Hình 7: Shortest Path Algorithm, ví dụ minh hoạ

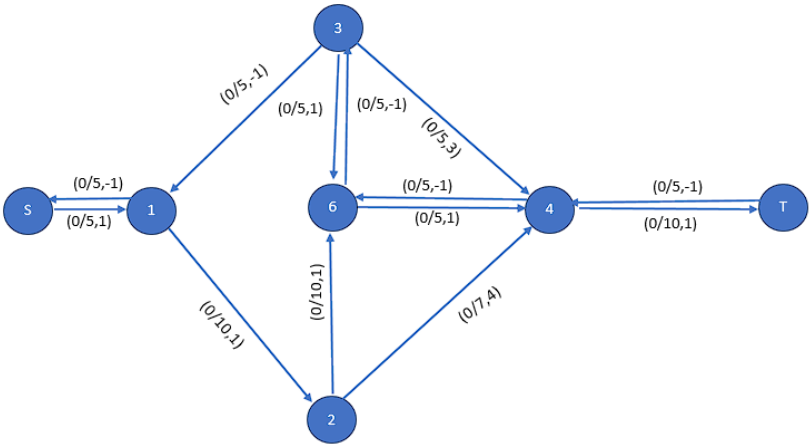
Ta thấy đường đi tô màu cam bên trên với chi phí là thấp nhất là 5, ta tiến hành truyền nhiều Flow nhất có thể vào đường đi này.



Hình 8: Shortest Path Algorithm, ví dụ minh hoạ

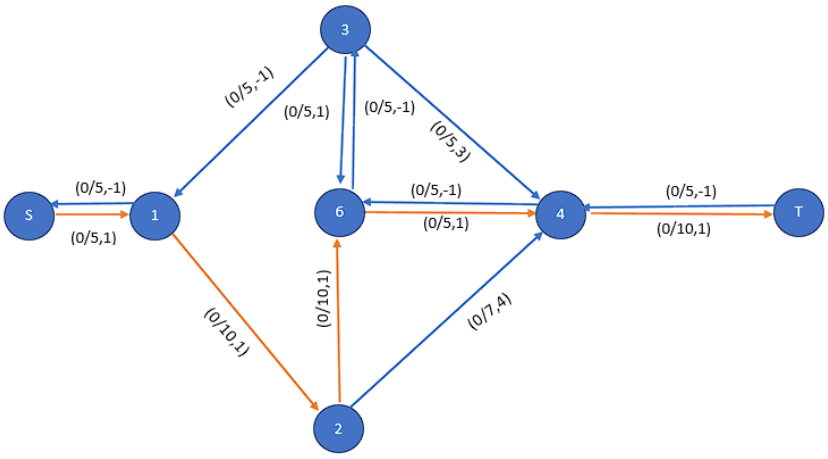
Lúc này flow hiện tại là 5, ta cũng có thể tính cost bằng $c_{ij}x_{ij}$, ta tính được ra cost là $5 \times 1 + 5 \times 1 + 5 \times 1 + 5 \times 1 + 5 \times 1 = 25$

Từ đó tiến hành tìm ra được Residual Network như đã đề cập ở Hình 4 ta được:



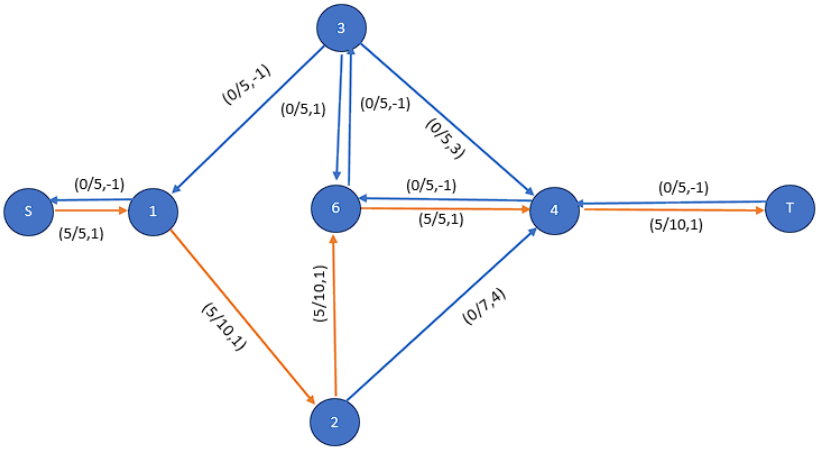
Hình 9: Shortest Path Algorithm, ví dụ minh hoạ

Dùng Bellman-Ford để kiểm tra chu trình âm, ở đây ta không thấy xuất hiện chu trình âm nào, ta tiếp tục tìm kiếm đường đi với chi phí thấp nhất. Ta thấy xuất hiện 1 đường với chi phí là 5.

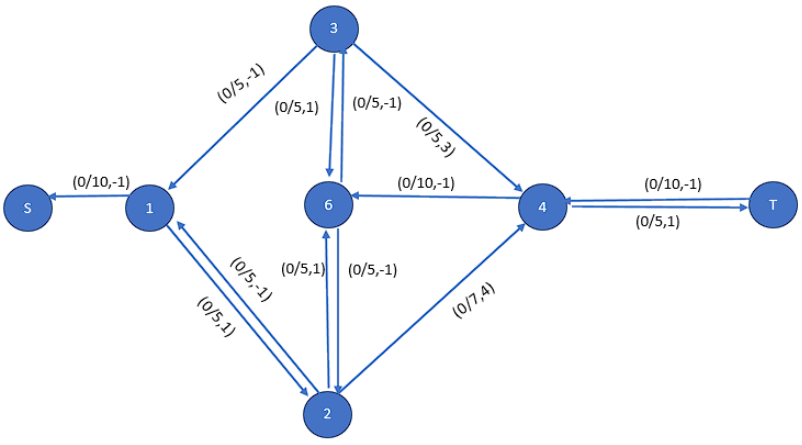


Hình 10: Shortest Path Algorithm, ví dụ minh hoạ

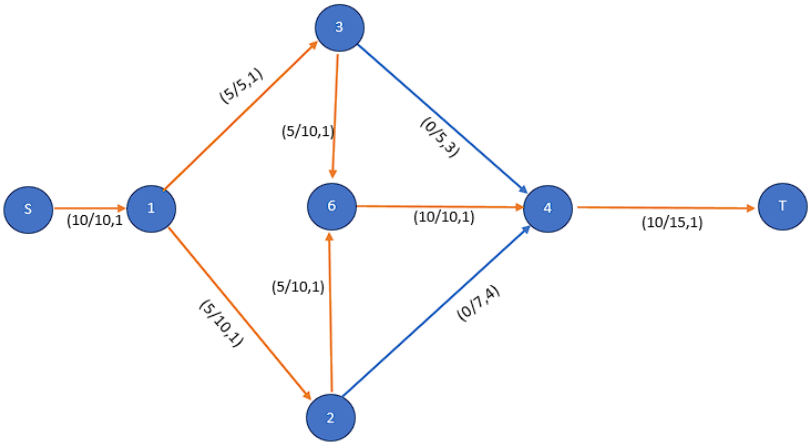
Ta lại tiếp tục truyền nhiều số flow nhất có thể vào con đường này, và đường như hình bên dưới. Flow hiện tại đã là $5 + 5 = 10$ và cost hiện tại là $25 + 25 = 50$ (Hình 11). Lúc này Residual Network cũng được cập nhật (Hình 12). Sử dụng Bellman-Ford để kiểm tra chu trình âm thì thấy rằng không tồn tại chu trình âm nào, tiếp tục tìm kiếm đường đi thấp nhất từ s đến t thì thấy không còn bất cứ đường nào. Vậy nên kết hợp cả (Hình 8 và Hình 11) ta sẽ được Hình 13. Lúc này max flow là 10 và min cost là 50, đây cũng là đáp án cuối cùng của bài toán.



Hình 11: Shortest Path Algorithm, ví dụ minh hoạ

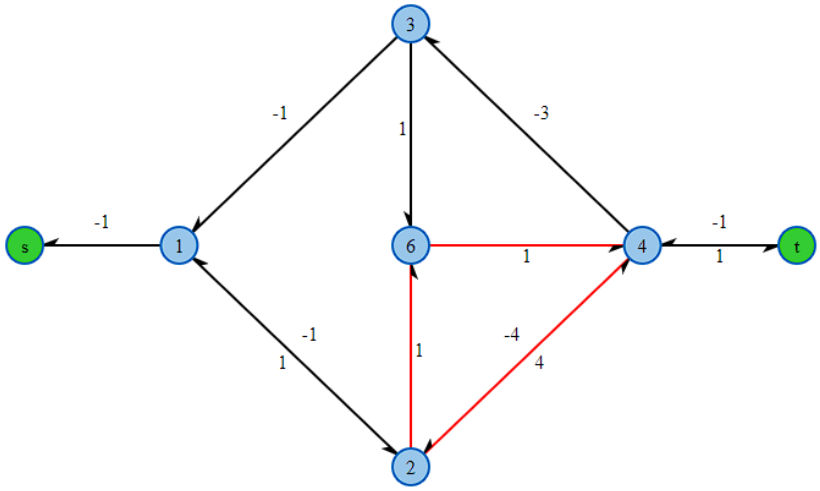


Hình 12: Shortest Path Algorithm, ví dụ minh hoạ



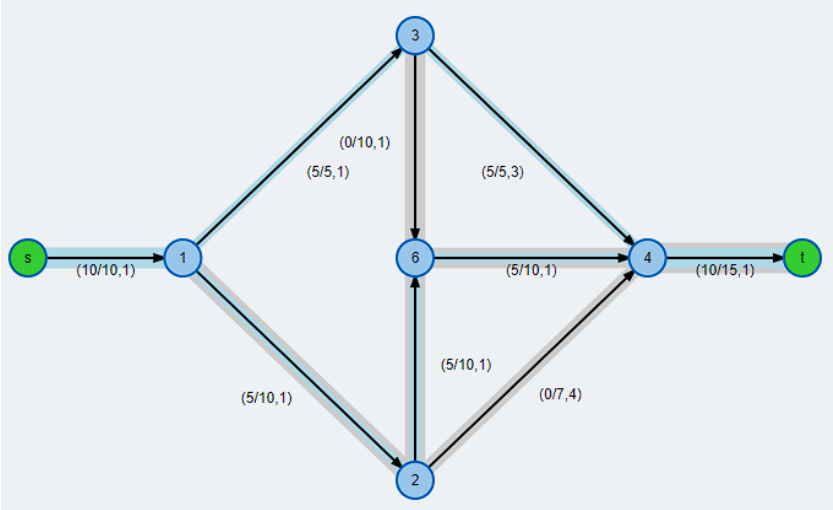
Hình 13: Shortest Path Algorithm, ví dụ minh hoạ

Cycle-Canceling Algorithm Thuật toán này xoay quanh trên các Cycle của đồ thị. Thuật toán này sẽ tìm ra được giá trị nghiệm của bài toán Min-cost flow **khi và chỉ khi** trong đồ thị *không có* bất kì chu kì âm nào (Hình 14). Ở Hình 3, ta tìm được đường đi và giá trị **cost** = 65



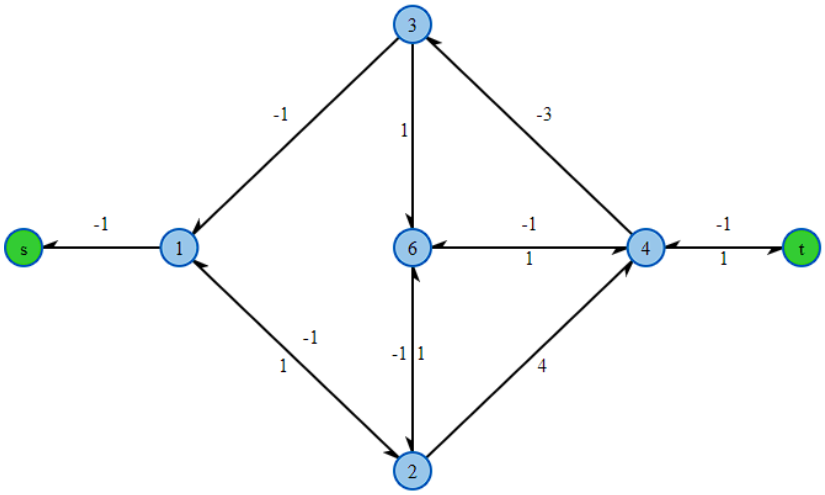
Hình 14: Cycle-Canceling, ví dụ về Chu trình âm

Tuy nhiên, ta tìm thấy được 1 chu trình âm (được tô đỏ ở Hình 14) có cost là -2 (Đi từ Node 4 $\rightarrow$ Node 2), vì thế nên giá trị cost x = 65 là không thoả mãn. Cách để giải quyết vấn đề này là ta sẽ tăng giá trị của những cái Flow trong chu trình âm này sao cho nó mất đi chu trình âm đó. Sau khi mất chu trình âm, ta lại tiến hành tìm ra chu trình khác và lặp lại cho đến khi không còn bất kì chu trình âm nào trong Đồ thị nữa. Lúc đó, giá trị **cost** tìm được sẽ là giá trị Min-cost flow. Ở hình trên thì ta sẽ loại bỏ flow đi từ Node 4 $\longrightarrow$ Node 2, vì nó khiến cho chu trình này âm. Sau khi loại bỏ xong, ta tiếp tục tiến hành ở Hình 15.

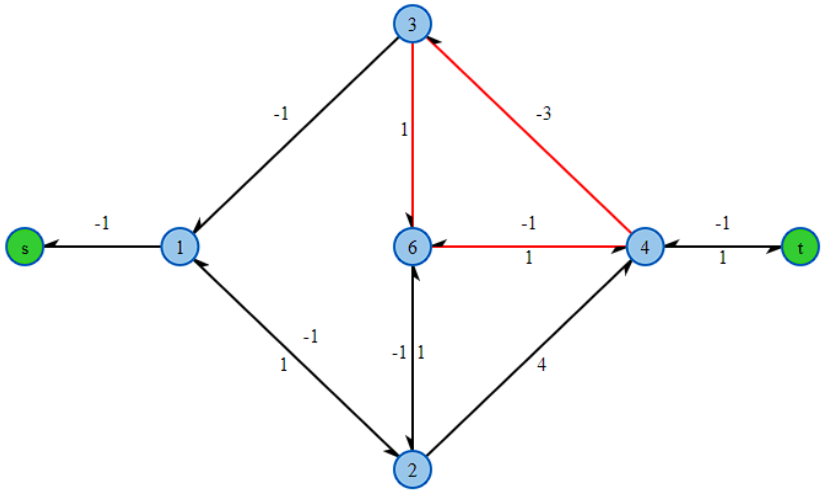


Hình 15: Cycle-Canceling, tiến hành tăng flow để loại chu trình âm

Chúng em tiến hành tìm thấy một con đường đi từ điểm nguồn đến đích (từ s đến t). Xong ta lại sử dụng giải thuật Bellman-Ford để tìm kiếm chu trình âm trong con đường ta vừa tìm thấy. Các quá trình được lặp lại cho đến khi tìm ra được kết quả, Hình 16 là chúng em sử dụng thuật toán Bellman-Ford để có thể tìm ra được các chu trình âm.

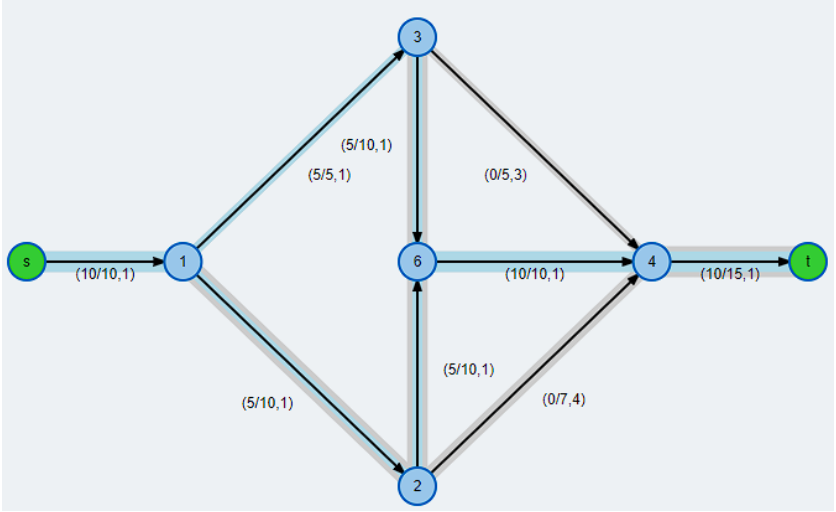


Hình 16: Cycle-Canceling, quá trình kiểm tra chu trình âm

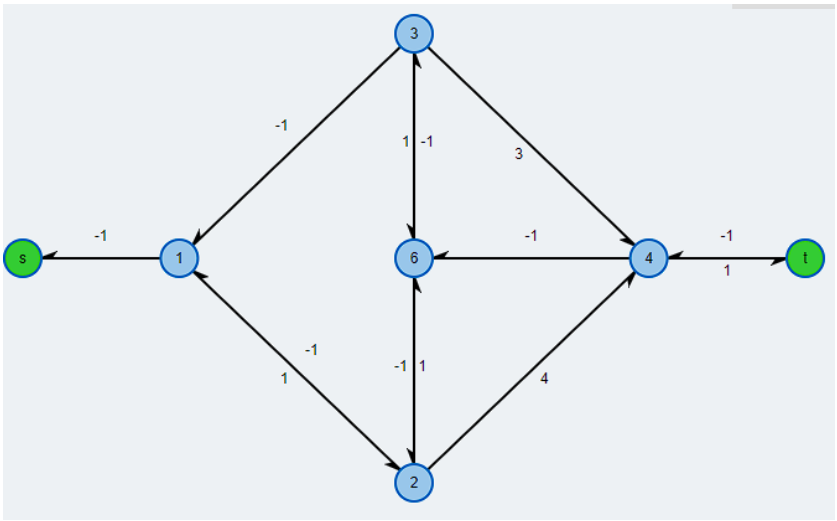


Hình 17: Cycle-Canceling, quá trình tìm thấy chu trình âm

Quan sát được xuất hiện 1 chu trình âm nữa (Đó là đường đi từ $3 \longrightarrow 6 \longrightarrow 4 \longrightarrow 3$) có **cost** là -3 + 1 + 1 = -1 < 0 nên ta sẽ loại bỏ (bão hoà) con đường gây ra chu trình âm này, đó là từ $4 \longrightarrow 3$. Ta tiếp tục tiến hành lặp lại quá trình, tìm kiếm 1 con đường đi từ **s** đến **t** như Hình 18.



Hình 18: Cycle-Canceling, quá trình Tìm ra Flow



Hình 19: Cycle-Canceling, quá trình kiểm tra chu trình âm

Hình 18 và Hình 19 mô tả lần lượt quá trình tìm ra Flow và kiểm tra xem trên Flow đó có chu trình âm hay không, ở đây ta thấy không còn chu trình âm nào tồn tại nữa, nên ta kết luận Flow của 18 là **Max Flow** ta cần tìm. Vậy **Max Flow** của đồ thị này là 10, **Min Cost** của bài này là 50. Tức là có tồn tại con đường đi từ **Nơi có nạn** đến **Nơi an toàn** mà ở đó 10 xe cùng lúc đi được, chi phí thời gian tối thiểu cần để đến đó là 50.

Thuật toán **Cycle-Canceling** được trình bày như sau:

- Đầu tiên, ta sử dụng bất kì Thuật toán maximum Flow bất kì (tìm ra đường đi có Flow lớn nhất) để tìm ra 1 cái chu trình có thể chấp nhận được trong đồ thị.
- Sau đó, thuật toán sẽ tìm kiếm những chu trình âm (chu trình có **cost** là giá trị âm) và tiến hành tăng các giá trị Flow theo chiều thuận của các chu trình này, đồng thời giảm lưu lượng trên các Flow của chiều lùi bằng 1 giá trị vừa tăng. Dưới đây là mã giả mô tả thuật toán:

```
1    Establish a feasible flow x in the network
2    while ( Gx contains a negative cycle ) do
3        identify a negative cycle W
4
5        augment  units of flow along the cycle W
6        update Gx
```

3.3 Độ phức tạp & So sánh thuật toán

Độ phức tạp của Successive Shortest Path - Algorithm được tính toán như sau: Time Complexity: O(VE^2logU), trong đó V số lượng của các đỉnh, E là số lượng các cạnh, và U là capacity tối đa trong Network.

- Ưu điểm: Dễ hiểu và dễ triển khai, có thể giải quyết đa số đồ thị đơn giản.
- Nhược điểm: Không thực sự hiệu quả đối với các đồ thị quá phức tạp, phụ thuộc quá nhiều vào cách lựa chọn con đường để tăng giá trị Flow... dẫn đến có thể sai kết quả.

Độ phức tạp của Cycle Canceling - Algorithm tương tự như thuật toán trên, chúng ta có được Time Complexity: O(VE^2U), trong đó các V, E, U cũng tương tự trên. Có thể thấy rằng, trong trường hợp tệ nhất, Cycle Canceling có Time Complexity cao hơn Successive shortest Path - Algorithm.

- Ưu điểm: Sử dụng tốt cho các capacity là số nguyên
- Nhược điểm: Một số trường hợp sẽ khiến Time Complexity của Cycle Canceling cao hơn các thuật toán khác.

Kết luận: Tuỳ theo trường hợp nhất định, các đồ thị cụ thể mà ta sẽ chọn những thuật toán khác nhau để giải quyết. Ở đây, thuật toán Successive Shortest Path vẫn là một lựa chọn ổn định trong hầu hết các trường hợp, đối với bài toán của chúng ta là tìm tuyến đường di tản hợp lý nhất thì rất phù hợp. Và sự hiệu quả của nó đã được chứng minh thông qua các điều trên.

4 Tổng kết

4.1 Lời cảm ơn

Nhóm em đã có thể hoàn thành được bài báo cáo này phần lớn xin cảm ơn các thầy trong bộ môn Mô hình hoá Toán học đã hướng dẫn chi tiết và cụ thể. Chúng em xin cảm ơn thầy Thịnh đã giải đáp các thắc mắc của chúng em trong giờ.

4.2 Code của cả 2 bài

Code sẽ được đính kèm trong file nộp trên BKEL, bao gồm 1 file .py (của problem 1), 1 file .cpp (của problem 2) (Yêu cầu để chạy được: Cài đặt các thư viện cần thiết đối với file .py)

References

[1] Faculty of Computer Science and Engineering, "MM CO2011 SEM231 Assignment Stochastic Programming and Applications v1", 2023

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