HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1

BÁO CÁO BÀI TẬP LỚN

HỌC PHẦN: TOÁN RỜI RẠC 1

GIẢNG VIÊN: NGUYỄN TẤT THẮNG

NHÓM SINH VIÊN THỰC HIỆN: NHÓM 4

TÊN THÀNH VIÊN	MÃ SINH VIÊN
Phạm Hoàng Anh	B24DCCN041
Phạm Minh Hằng	B24DCCN193
Nguyễn Quang Huy	B24DCCN281
Đặng Nguyễn Hồng Vân	B24DCCN607

Hà Nội – Tháng 11 năm 2025

MỤC LỤC

1. Giới thiệu bài toán tối ưu

1.1. Đặt vấn đề

Cho tập dữ liệu $D = \{x\}$, trong đó $x = (x_1, x_2, \dots, x_n)$, với $x_i \in D_i$. Tìm $x \in D$ sao cho $f(x) \to \min (\max)$, trong đó:

- Hàm $f(x)$ được gọi là hàm mục tiêu của bài toán.
- Mỗi phần tử $x \in D$ gọi là một phương án của bài toán.
- Phương án $X_{opt} \in D$ để hàm mục tiêu có giá trị nhỏ nhất (lớn nhất) gọi là phương án tối ưu của bài toán.
- $F_{opt} = f(X_{opt})$ được gọi là giá trị tối ưu của bài toán.

1.2. Ví dụ kinh điển: Bài toán cái túi

1.2.1. Phát biểu bài toán

Một nhà thám hiểm có một cái túi có thể chứa các đồ vật với trọng lượng không quá b. Có n đồ vật đem theo. Đồ vật thứ i có trọng lượng a_i và giá trị sử dụng c_i $(i = 1, 2, \dots, n;\ a_i, c_i \in \mathbb{Z}^+)$. Hãy tìm cách chọn đồ vật có thể đem theo cho nhà thám hiểm sao cho tổng giá trị sử dụng các đồ vật trong túi là lớn nhất.

1.2.2. Mô hình hóa bài toán cái túi

Gọi X là một phương án chọn các đồ vật trong túi, $f(X)$ là tổng giá trị sử dụng và $g(X)$ là trọng lượng các đồ vật được chọn trong phương án đó. X_{opt} là phương án tối ưu với trọng lượng các đồ vật không vượt quá b và tổng giá trị sử dụng của chúng là lớn nhất có thể. Khi đó ta có thể mô hình hóa bài toán như sau:

$$\begin{cases} f(X) = c_1x_1 + c_2x_2 + \cdots + c_nx_n \to \max \\ g(X) = a_1x_1 + a_2x_2 + \cdots + a_nx_n \\ X = (x_1, x_2, \dots, x_n); x_i \in \{0, 1\} \\ X_{opt} \in D, D = \{X \mid g(X) \le b\} \end{cases}$$

VD: Có 4 đồ vật có thể lựa chọn để cho vào túi, biết trọng lượng tối đa chiếc túi có thể chứa là 8. Trọng lượng và giá trị sử dụng của từng đồ vật được cho bởi bảng sau:

i	1	2	3	4
Trọng lượng (a_i)	3	2	5	4
Giá trị sử dụng (c_i)	5	3	10	6

Mô hình hóa bài toán trên ta được:

$$\begin{cases} 5x_1 + 3x_2 + 10x_3 + 6x_4 \to \max \\ 3x_1 + 2x_1 + 5x_3 + 4x_4 \le 8 \\ x_i \in \{0, 1\} \mid i = 1, 2, 3, 4 \end{cases}$$

1.3. Phương pháp duyệt toàn thể

1.3.1. Ý tưởng chung

Giả sử D là tập phương án, $f(X)$ là hàm mục tiêu của bài toán tối ưu. Với mỗi $X \in D$, tính $f(X)$ và so sánh với phương án tối ưu hiện có để xác định $f(X) \rightarrow \min$ (max).

1.3.2. Mã giả (Pseudo-code)

```
list X_opt = empty;          // phương án tối ưu
int F_opt = 0;               // giá trị f ứng với phương án tối ưu
int total = 2^n;             // tổng số phương án

// duyệt qua từng phương án
for (int mask: 0 -> total - 1) {
    list X = empty;          // các phần tử trong phương án hiện tại
    int f = 0;               // f: hàm mục tiêu
    for (int i: 1 -> n) {
        if (phần tử thứ i được chọn) {
            <thêm phần tử thứ i vào X>
            <cập nhật f>
        }
    }
    if (X ∈ D and f tối ưu hơn F_opt) {
        F_opt = f;
        X_opt = X;
    }
}
```

Hình 1.1. Thuật toán duyệt toàn bộ.

1.3.3. Ví dụ cụ thể

Ta áp dụng phương pháp duyệt toàn thể với bài toán đã cho ở mục 1.1.2.2:

$$\begin{cases} f(X) = 5x_1 + 3x_2 + 10x_3 + 6x_4 \rightarrow \max \\ g(X) = 3x_1 + 2x_1 + 5x_3 + 4x_4 \leq 8 \\ x_i \in \{0,1\} \mid i = 1,2,3,4 \end{cases}$$

Dữ liệu ban đầu: $n = 4,\ b = 8,\ F_{opt} = -\infty,\ X_{opt} = \emptyset$

Lập bảng:

X	$g(X)$	$g(X) \leq b$	$f(X)$	F_{opt}
0, 0, 0, 0	0	Yes	0	**0**
0, 0, 0, 1	4	Yes	6	**6**
0, 0, 1, 0	5	Yes	10	**10**
0, 0, 1, 1	9	No	-	10
0, 1, 0, 0	2	Yes	3	10
0, 1, 0, 1	6	Yes	9	10
0, 1, 1, 0	7	Yes	13	**13**

0, 1, 1, 1	11	No	–	13
1, 0, 0, 0	3	Yes	5	13
1, 0, 0, 1	7	Yes	11	13
1, 0, 1, 0	8	Yes	15	**15**
1, 0, 1, 1	12	No	–	15
1, 1, 0, 0	5	Yes	8	15
1, 1, 0, 1	9	No	–	15
1, 1, 1, 0	10	No	–	15
1, 1, 1, 1	14	No	–	15

Kết luận: $F_{opt} = 15$; $X_{opt} = (1, 0, 1, 0)$.

1.3.4. Đánh giá phương pháp duyệt toàn thể

a) Ưu điểm

- Đơn giản, dễ cài đặt.
- Có thể thực hiện trên mọi bài toán tối ưu.

b) Nhược điểm

Chi phí tính toán thường tăng rất nhanh theo n nên chỉ phù hợp với các bài toán có n nhỏ. Ví dụ với bài toán có n phần tử:

- Số lượng hoán vị: $n!$. Nếu xét $n = 15$ thì máy tính đã phải duyệt hơn 10^{12} phương án ⇒ Phương pháp duyệt toàn thể chỉ dùng cho trường hợp $n \leq 10$.
- Số cách chọn một số phần tử trong số n phần tử đã cho: 2^n ⇒ Phương pháp duyệt toàn thể chỉ dùng cho trường hợp $n \leq 23$.

Đối với những bài toán có n lớn hơn, ta cần một thuật toán có độ phức tạp tốt hơn. Ở các phần sau, chúng ta sẽ tìm hiểu về một thuật toán khác để giải quyết bài toán tối ưu: thuật toán nhánh cận.

2. Thuật toán nhánh cận

2.1. Xây dựng & chứng minh thuật toán

Giả sử chúng ta cần giải quyết bài toán tối ưu tổ hợp với mô hình tổng quát như sau:

Tìm $\min\{f(X)\colon X \in D\}$ ($|D|$ hữu hạn)

Trong đó:

- $D = \{X = (x_1, x_2, \dots, x_n) \in A_1 \times A_2 \times \dots \times A_n \mid X$ thỏa mãn tính chất $P\}$
- $A_1, A_2, \dots, A_n$ là các tập hữu hạn
- P là tính chất cho trên tích Descartes $A_1 \times A_2 \times \dots \times A_n$.

Như vậy, các bài toán chúng ta vừa trình bày ở trên đều có thể được mô tả dưới dạng trên. Với giả thiết về tập D như trên, chúng ta có thể sử dụng thuật toán quay lui để liệt kê các phương án của bài toán. Trong quá trình liệt kê theo thuật toán quay lui, ta sẽ xây dựng dần các thành phần của phương án. Ta gọi một bộ phận gồm k trong số n thành phần của x xuất hiện trong quá trình thực hiện thuật toán sẽ được gọi là phương án bộ phận cấp k.

Thuật toán nhánh cận có thể được áp dụng giải bài toán đặt ra nếu như có thể tìm được một hàm g xác định trên tập tất cả các phương án bộ phận của bài toán thoả mãn bất đẳng thức:

$$g(a_1, a_2, \dots, a_k) \leq \min\{f(X) : X \in D, x_i = a_i, i = 1, 2, \dots, k\} \qquad (*)$$

với mọi lời giải bộ phận $(a_1, a_2, \dots, a_k)$, và với mọi $k = 1, 2, \dots, n$.

Bất đẳng thức (*) có nghĩa là giá trị của hàm tại phương án bộ phận $(a_1, a_2, \dots, a_k)$ không vượt quá giá trị nhỏ nhất của hàm mục tiêu bài toán trên tập con các phương án.

$$D(a_1, a_2, \dots, a_k)\{x \in D: x_i = a_i, i = 1, 2, \dots, k\},$$

Nói cách khác, $g(a_1, a_2, \dots, a_k)$ là cận dưới của tập $D(a_1, a_2, \dots, a_k)$. Do có thể đồng nhất tập $D(a_1, a_2, \dots, a_k)$ với phương án bộ phận $(a_1, a_2, \dots, a_k)$, nên ta cũng gọi giá trị $g(a_1, a_2, \dots, a_k)$ là cận dưới của phương án bộ phận $(a_1, a_2, \dots, a_k)$.

Giả sử ta đã có được hàm g. Ta xét cách sử dụng hàm này để hạn chế khối lượng duyệt trong quá trình duyệt tất cả các phương án theo thuật toán quay lui. Trong quá trình liệt kê các phương án có thể đã thu được một số phương án của bài toán. Gọi $\overline{x}$ là phương án tốt nhất hiện có, còn $\overline{f}$ là hàm mục tiêu nhỏ nhất trong số các phương án đã duyệt, ký hiệu $\overline{f} = f(\overline{x})$. Giả sử ta có được $\overline{f}$, khi đó nếu $g(a_1, a_2, \dots, a_k) > \overline{f}$ thì từ bất đẳng thức (*) ta suy ra:

$$\overline{f} < g(a_1, a_2, \dots, a_k) \leq \min\{f(x): x \in D, x_i = a_i, i = 1, 2, \dots, k\}$$

vì thế tập con các phương án của bài toán $D(a_1, a_2, \dots, a_k)$ chắc chắn không chứa phương án tối ưu. Trong trường hợp này ta không cần phải phát triển phương án bộ phận $(a_1, a_2, \dots, a_k)$, nói cách khác là ta có thể loại bỏ các phương án trong tập $D(a_1, a_2, \dots, a_n)$ khỏi quá trình tìm kiếm.

Việc xây dựng hàm g phụ thuộc vào từng bài toán tối ưu tổ hợp cụ thể. Nhưng chúng ta cố gắng xây dựng sao cho đạt được những điều kiện dưới đây:

- Giá trị của g phải dễ tính hơn so với việc giải bài toán tổ hợp trong vế phải của (*).
- Giá trị của $g(a_1, a_2, \dots, a_k)$ phải sát với giá trị vế phải của (*).

Rất tiếc, hai yêu cầu này trong thực tế thường đối lập nhau.

2.2. Mã giả (Pseudo-code)

Thuật toán quay lui liệt kê các phương án cần sửa đổi lại như Hình 2.1:

```
Procedure Try(k){
/*Phát triển phương án bộ phận (a_1, a_2, . ., a_k-1 theo thuật
toán quay lui có kiểm tra cận dưới trước khi tiếp tục phát triển
phương án*/
    for (a_k ∈ A_k) {
        if (chấp nhận a_k){
            x_k = a_k;
            if (k == n) <UPDATE f_opt, x_opt>
            else if (g(a_1, a_2,…, a_k) <= f_opt)
                Try(k + 1);
        }
    }
}
```

Hình 2.1. Thuật toán quay lui dựa vào hàm đánh giá cận.

Khi đó, thuật toán nhánh cận được thực hiện như Hình 2.2

```
Procedure Nhanh_Can( ) {
    f_opt = INF;
    Try(1); // Thực hiện thuật toán quay lui trong Hình 2.1
    if (f_opt < INF)
        <f_opt là giá trị tối ưu, x_opt là phương án tối ưu>;
    else
        <Không có phương án nào thỏa mãn>;
}
```

Hình 2.2. Thuật toán nhánh cận.

3. Một số bài toán kinh điển và cách giải quyết sử dụng phương pháp nhánh cận

3.1. Bài toán cái túi

Giả sử có n loại đồ vật và số lượng đồ vật mỗi loại không hạn chế. Đồ vật loại j có trọng lượng a_j và giá trị sử dụng $c_j, j = (1, 2, \dots, n)$. Ta cần đưa các đồ vật này vào một cái túi sao cho:

- Tổng trọng lượng các đồ vật không vượt quá b
- Tổng giá trị sử dụng của các đồ vật trong túi là lớn nhất
- Mỗi loại đồ vật có thể lấy nhiều lần.

3.1.1. Dạng tổng quát

Bài toán cái túi có thể được phát biểu dưới dạng tổng quát như sau: Tìm giá trị max của hàm mục tiêu $f(X)$ với $X \in D$ và $f(X)$ được xác định như dưới đây:

$$D = \left\{ X = (x_1, x_2, \dots, x_n) : \sum_{i=1}^{n} a_i x_i \le b, x_i \in \mathbb{Z}^+, i = 1,2, \dots, n \right\}$$

$$f^* = max \left\{ f(X) = \sum_{i=1}^{n} c_i x_i : \sum_{i=1}^{n} a_i x_i \le b, x_i \in \mathbb{Z}^+, i = 1,2, \dots, n \right\}$$

3.1.2. Cách giải bài toán cái túi sử dụng phương pháp nhánh cận

- **Bước 1:** Sắp xếp các đồ vật thoả mãn: $\frac{c_1}{a_1} \ge \frac{c_2}{a_2} \ge \cdots \ge \frac{c_n}{a_n}$
- **Bước 2:** Lặp trên các bài toán bộ phận cấp $k = 1, 2, \dots, n$
  - Giá trị sử dụng của k đồ vật trong túi là:

$$\delta_k = \sum_{i=1}^{k} c_i x_i$$

  - Trọng lượng còn lại của túi là:

$$b_k = b - \sum_{i=1}^{k} a_i x_i$$

  - Cận trên của phương án bộ phận cấp k:

$$g(x_1, x_2, \dots, x_k) = \delta_k + b_k \frac{c_{k+1}}{a_{k+1}}$$

- **Bước 3:** Kết luận phương án tối ưu và giá trị tối ưu tìm được

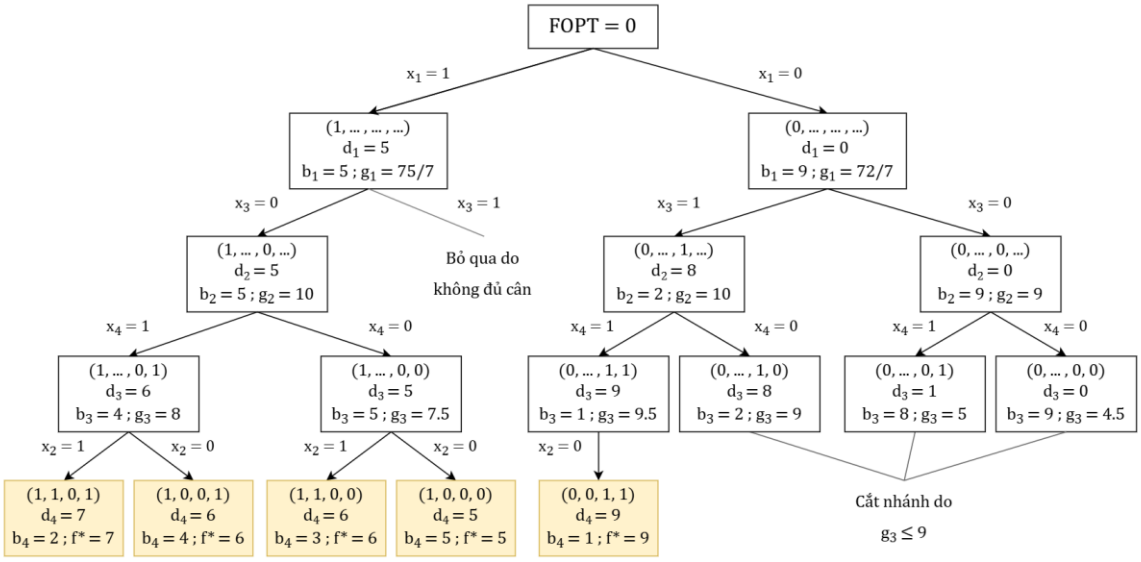
3.1.3. Sơ đồ giải bài toán cái túi

Ta có bài toán cái túi được mô hình hóa như sau:

$$\begin{cases} f(X) = 5x_1 + x_2 + 8x_3 + x_4 \to \max \\ g(X) = 4x_1 + 2x_2 + 7x_3 + x_4 \le 9 \\ x_j \in \{0, 1\}; j \in \{1, 2, 3, 4\} \end{cases}$$

Sau khi sắp xếp các món đồ theo thứ tự giảm dần giá trị $\frac{c_j}{a_j}$ $(j = 1, 2, \dots, n)$, ta xác định được thứ tự duyệt: $1 \to 3 \to 4 \to 2$.

Sơ đồ giải bài toán trên:



Hình 3.1. Sơ đồ giải bài toán cái túi.

Từ sơ đồ trên ta thu được phương án tối ưu là $(x_1, x_2, x_3, x_4) \in \{(0,0,1,1), (0,1,1,0)\}$ và giá trị tối ưu của hàm mục tiêu là $f^* = 9$.

3.2. Bài toán người đi du lịch

Một người du lịch muốn tham quan n thành phố $T_1, T_2, \dots, T_n$. Xuất phát từ một thành phố nào đó, người du lịch muốn đi qua tất cả thành phố còn lại, mỗi thành phố đúng một lần rồi quay trở lại thành phố xuất phát. Biết c_{ij} là chi phí đi từ thành phố T_i đến thành phố T_j $(i, j = 1, 2, \dots, n)$. Hãy tìm hành trình thỏa mãn điều kiện trên với tổng chi phí là nhỏ nhất.

3.2.1. Dạng tổng quát

Tìm giá trị nhỏ nhất của hàm mục tiêu $f(X)$ với $X \in D$ và $f(X)$ được xác định như sau:

$$D = \{X = (x_1, x_2, \dots, x_n): x_1 = 1 \ \wedge \ x_i \neq x_j \ \forall \ i \neq j\}$$

$$f^* = \min\left\{f(X) = \sum_{i=1}^{n-1} c[x_i, x_{i+1}] + c[x_n, x_1]: X \in D\right\}$$

3.2.2. Cách giải bài toán người đi du lịch sử dụng phương pháp nhánh cận

Ta sẽ giải bài toán trên với trường hợp người du lịch luôn xuất phát từ thành phố đầu tiên (T_1). Gọi $c_{min} = \min\{c[i,j] \mid i, j = 1, 2, \dots, n; i \neq j\}$ là giá trị nhỏ nhất của ma trận chi phí. Phương pháp đánh giá cận dưới của mỗi bài toán bộ phận cấp k được tiến hành như sau. Giả sử ta đang có hành trình bộ phận qua k thành phố:

$$T_1 \to T_{u_2} \to \cdots \to T_{u_k}$$

Khi đó, chi phí của phương án bộ phận cấp k là:

$$\delta = c[1, u_2] + c[u_2, u_3] + \cdots + c[u_{k-1}, u_k]$$

Để phát triển hành trình bộ phận này thành hành trình đầy đủ, ta cần phải qua $n - k$ thành phố nữa rồi quay trở về 1, nên số đoạn đường phải đi qua là $n - k + 1$. Vì mỗi đoạn đường đều có chi phí không nhỏ hơn c_{min} nên cận dưới của phương án bộ phận có thể được xác định theo công thức:

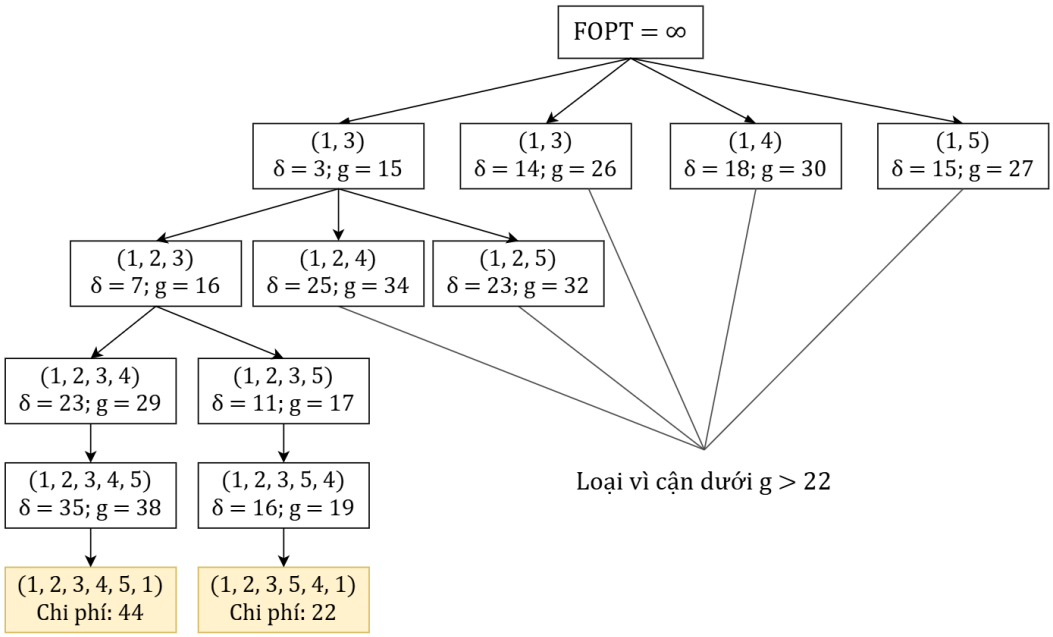
$$g(u_1, u_2, \dots, u_k) = \delta + (n - k + 1) \times c_{min}$$

3.2.3. Sơ đồ giải bài toán người đi du lịch

Ta có bài toán người đi du lịch với ma trận chi phí như sau:

0	3	14	18	15
3	0	4	22	20
17	9	0	16	4
6	3	7	0	12
9	15	11	5	0

Sơ đồ giải bài toán trên ($c_{min} = 3$):



Hình 3.2. Sơ đồ giải bài toán người du lịch.

Từ sơ đồ trên ta thu được phương án tối ưu là $1 \to 2 \to 3 \to 5 \to 4 \to 1$, với chi phí tối thiểu là 22.

4. Ứng dụng thuật toán nhánh cận để giải quyết bài toán lập lịch sản xuất

4.1. Bài toán lập lịch sản xuất

4.1.1. Phát biểu bài toán

Mỗi một chi tiết trong số n chi tiết $D_1, D_2, \dots, D_n$ cần phải được lần lượt gia công trên m máy $M_1, M_2, \dots, M_m$. Thời gian gia công chi tiết D_i trên máy M_j là t_{ij}. Biết các chi tiết phải được gia công một cách liên tục, tức là không có khoảng thời gian dừng khi chuyển từ máy này sang máy khác. Hãy sắp xếp trình tự gia công các chi tiết trên các máy sao cho việc hoàn thành gia công tất cả các chi tiết là sớm nhất có thể.

4.1.2. Phân tích

Dễ thấy mỗi một lịch gia công các chi tiết trên các máy trong tình huống như vậy sẽ tương ứng với một hoán vị $x = (x_1, x_2, \dots, x_n)$ của n số tự nhiên $1, 2, \dots, n$. Hay tập phương án có thể được biểu diễn dưới dạng tập hợp:

$$\{X = (x_1, x_2, \dots, x_n) \mid \forall i \neq j, x_i \neq x_j\}$$

Thời gian hoàn thành theo lịch trên được tính bởi hàm số

$$f(x) = \sum_{i=1}^{n-1} c_{x_i x_{i+1}} + \sum_{j=1}^{m} t_{x_n j}$$

Trong đó c_{ij} là khoảng thời gian giữa thời điểm bắt đầu gia công chi tiết i và chi tiết j nếu chi tiết j được gia công ngay sau chi tiết i. c_{ij} có thể tính theo công thức:

$$c_{ij} = \max_{1 \le k \le m} \left[\sum_{l=1}^{k} t_{il} - \sum_{l=1}^{k-1} t_{jl} \right] ;\ i, j = 1, 2, \dots, n\ (i \neq j)$$

Bài toán trở về vấn đề tối ưu tổ hợp: $\min\{f(x): x \in \mathrm{X}\}$

4.2. Cách giải bài toán sử dụng thuật toán nhánh cận

4.2.1. Xây dựng thuật toán

Để xây dựng thuật toán nhánh cận giải quyết bài toán này, trước hết ta cần đưa ra cách đánh giá cận dưới cho phương án bộ phận. Giả sử ta có phương án bộ phận $(u_{n-k+1}, u_{n-k+2}, \dots, u_n)$ ứng với một cách đặt lịch sản xuất cho k chi tiết cuối trong số n chi tiết cần gia công. Thời gian hoàn thành của phương án này:

$$\sigma = \sum_{i=n-k+1}^{n-1} c_{u_i u_{i+1}} + \sum_{j=1}^{m} t_{u_n j}$$

Để phát triển phương án bộ phận này thành quy trình sản xuất đầy đủ, ta cần gia công $n - k$ chi tiết còn lại trước khi gia công chi tiết u_{n-k+1}. Gọi $c_{min} = \min\{c_{ij} \mid i, j = 1, 2, \dots, n;\ i \neq j\}$ là chênh lệch thời gian bé nhất giữa thời điểm bắt đầu gia công 2 chi tiết. Đối với mỗi chi tiết thêm vào phương án, thời gian tăng thêm không nhỏ hơn c_{min}. Vì vậy cận dưới cho phương án bộ phận này có thể tính theo công thức:

$$g(u_{n-k+1}, u_{n-k+2}, \dots, u_n) = \sigma + (n - k)c_{min}$$

Sử dụng cách tính cận dưới nêu trên ta có thể áp dụng thuật toán nhánh cận để giải quyết bài toán lập lịch sản xuất.

4.2.2. Sơ đồ giải bài toán lập lịch sản xuất

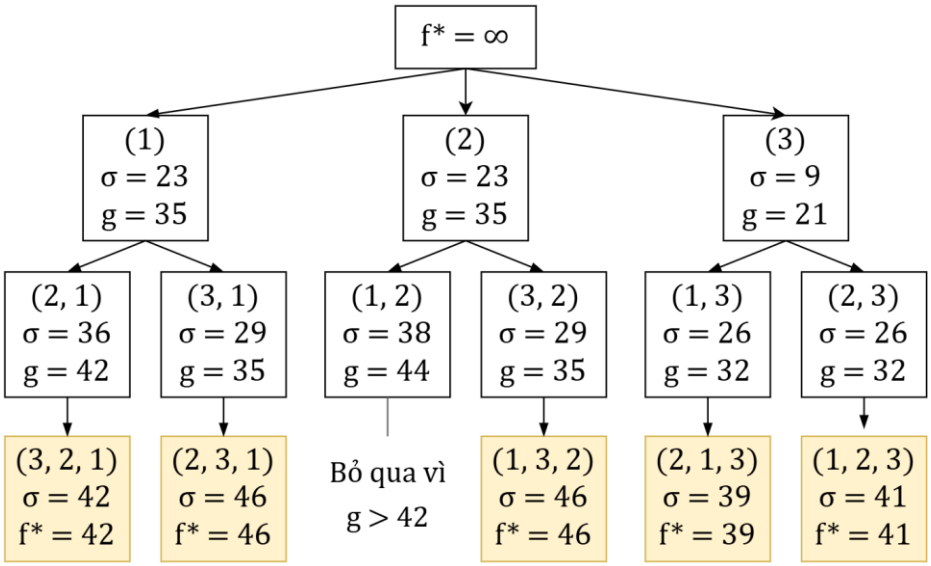
Giải bài toán lập lịch sản xuất với 3 chi tiết, 2 máy và thời gian gia công được thể hiện qua ma trận T, với t_{ij} là thời gian gia công chi tiết D_i trên máy M_j:

$$T = \begin{bmatrix} 10 & 13 \\ 8 & 15 \\ 6 & 3 \end{bmatrix}$$

Từ ma trận trên ta tính được các giá trị c_{ij} được cho bởi ma trận C, và xác định được $c_{min} = 6$.

$$C = \begin{bmatrix} \infty & 15 & 17 \\ 13 & \infty & 17 \\ 6 & 6 & \infty \end{bmatrix}$$

Dưới đây là sơ đồ mô tả quá trình thực hiện thuật toán. Thông tin về một phương án bộ phận trên cây được ghi trên từng ô bao gồm: các thành phần của phương án, thời gian hoàn thành σ và cận dưới g.



Hình 4.1. Sơ đồ giải bài toán lập lịch sản xuất.

Sau khi duyệt tất cả các nhánh, ta tìm được thời gian tối ưu là $f^* = 39$ với thứ tự gia công $(2, 1, 3)$.

4.3. Mã giả (Pseudo-code)

Để giải quyết bài toán lập lịch sản xuất khi đã biết n, m và ma trận T, trước tiên ta cần tính ma trận C xác định c_{min} cho việc tính cận dưới.

```
int c_min = INF;
void matrixC(){
```

```
    // Duyệt từng cặp i, j để tính c[i][j]
    for (int i: 1 -> n){
        for (int j: 1 -> n){
            // chi tiết i gia công sau chính nó -> không hợp lệ
            if (i == j) skip;
            else{
                int s1 = 0, s2 = 0;
                for (int k: 1 -> m){
                    <s1: th.gian gia công chi tiết i qua k máy>
                    <s2: th.gian gia công chi tiết j qua k-1 máy>
                    c[i][j] = max(c[i][j], s1 - s2);
                }
                // c_min là giá trị nhỏ nhất trong các c[i][j]
                c_min = min(c_min, c[i][j]);
            }
        }
    }
}
```

Hình 4.2. Thuật toán tính ma trận C và xác định c_{min}.

Sau khi đã có ma trận C, ta có thể dùng thuật toán nhánh cận để giải quyết bài toán như sau:

```
list X_opt = empty;          // phương án tối ưu
list X = empty;              // phương án hiện tại
int F_opt = INF;             // giá trị f ứng với phương án tối ưu
Procedure BNB(int k, int time){
/*Phát triển phương án bộ phận cấp k, thời gian thực hiện = time*/
    int g = time + (n - k) * c_min;
    if (g > f_opt) return; // nếu cận dưới > f_opt thì bỏ qua

    for (int i: 1 -> n){
        if (<chi tiết i chưa nằm trong phương án>){
            <thêm chi tiết i vào X>
            <đánh dấu i đã nằm trong X>
            if (k == 0)
                BNB(k + 1, <thời gian gia công chi tiết i>)
            else
                BNB(k + 1, time + c[i][<chi tiết sau i>])
        }
/*Thứ tự gia công trong X được viết ngược nên chi tiết được gia
công sau i là chi tiết đứng trước chi tiết i trong X*/
    }
```

```
}
// Lời gọi đầu tiên, chưa có chi tiết nào trong phương án
BNB(0, 0)
```

Hình 4.3. Thuật toán nhánh cận giải bài toán lập lịch sản xuất.

Phiên bản hoàn chỉnh của thuật toán (sử dụng ngôn ngữ C++) có thể xem trong file production_scheduling.cpp

4.4. Đánh giá thuật toán

- **Độ phức tạp không gian:** $O(n \times \max(m, n))$, tỉ lệ thuận với số phần tử ma trận T và C.
- **Độ phức tạp thời gian:** Trong trường hợp xấu nhất, ta vẫn phải duyệt qua tất cả các hoán vị $1, 2, \ldots, n$ nên độ phức tạp trong trường hợp đó là $O(n!)$. Tuy nhiên, thực tế điều này còn phụ thuộc vào dữ liệu đầu vào, và do đã loại sớm các phương án con không thể phát triển thành phương án tối ưu nên thường độ phức tạp trung bình của thuật toán này thường tốt hơn nhiều so với độ phức tạp trong trường hợp tệ nhất.