CHƯƠNG IV: BÀI TOÁN TỐI ƯU

Nội dung chính của chương này là giới thiệu các phương pháp giải quyết bài toán tối ưu đồng thời giải quyết một số bài toán có vai trò quan trọng của lý thuyết tổ hợp. Những nội dung được đề cập bao gồm:

- ✓ Giới thiệu bài toán và phát biểu bài toán tối ưu cho các mô hình thực tế.
- ✓ Phân tích phương pháp liệt kê giải quyết bài toán tối ưu.
- ✓ Phương pháp nhánh cận giải quyết bài toán tối ưu.
- ✓ Phương pháp rút gọn giải quyết bài toán tối ưu.

Bạn đọc có thể tìm thấy phương pháp giải chi tiết cho nhiều bài toán tối ưu quan trọng trong các tài liệu [1], [2].

4.1. GIỚI THIỆU BÀI TOÁN

Trong nhiều bài toán thực tế, các cấu hình tổ hợp còn được gán một giá trị bằng số đánh giá giá trị sử dụng của cấu hình đối với một mục đích sử dụng cụ thể náo đó. Khi đó xuất hiện bài toán: Hãy lựa chọn trong số tất cả các cấu hình tổ hợp chấp nhận được cấu hình có giá trị sử dụng tốt nhất. Các bài toán như vậy được gọi là bài toán tối ưu tổ hợp. Chúng ta có thể phát biểu bài toán tối ưu tổ hợp dưới dạng tổng quát như sau:

Tìm cực tiểu (hay cực đại) của phiếm hàm $f(x) = min(max)$ với điều kiện $x \in D$, trong đó D là tập hữu hạn các phần tử.

Hàm $f(x)$ được gọi là hàm mục tiêu của bài toán, mỗi phần tử $x \in D$ được gọi là một phương án còn tập D gọi là tập các phương án của bài toán. Thông thường tập D được mô tả như là tập các cấu hình tổ hợp thoả mãn một số tính chất nào đó cho trước nào đó.

Phương án $x^* \in D$ đem lại giá trị nhỏ nhất (lớn nhất) cho hàm mục tiêu được gọi là phương án tối ưu, khi đó giá trị $f^* = f(x^*)$ được gọi là giá trị tối ưu của bài toán.

Dưới đây chúng ta sẽ giới thiệu một số bài toán tối ưu tổ hợp kinh điển. Các bài toán này là những mô hình có nhiều ứng dụng thực tế và giữ vai trò quan trọng trong việc nghiên cứu và phát triển lý thuyết tối ưu hoá tổ hợp.

Bài toán Người du lịch: Một người du lịch muốn đi thăm quan n thành phố $T_1, T_2, ..., T_n$. Xuất phát từ một thành phố nào đó, người du lịch muốn đi qua tất cả các thành phố còn lại, mỗi thành phố đi qua 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 $(i,j = 1, 2,.., n)$, hãy tìm hành trình với tổng chi phí là nhỏ nhất (một hành trình là một cách đi thoả mãn điều kiện).

Rõ ràng, ta có thể thiết lập được một tương ứng 1-1 giữa hành trình $T_{\pi(1)} \to T_{\pi(2)} \to ...T_{\pi(n)} \to T_{\pi(1)})$ với một hoán vị π = *(π(1), π(2),.., π(n))* của n số tự nhiên *1,2,..., n*. Đặt:

$$f(\pi) = C_{\pi(1),\pi(2)} + C_{\pi(2),\pi(3)} + \cdots + C_{\pi(n-1),\pi(n)} + C_{\pi(n),\pi(1)},$$

kí hiệu $\prod$ là tập tất cả các hoán vị π =(π(1), π(2),..., π(n)) của n số tự nhiên *1, 2,.., n*. Khi đó bài toán người du lịch có thể phát biểu dưới dạng bài toán tối ưu tổ hợp sau:

$min\ \{ f(\pi): \pi \in \prod \}$

Có thể thấy rằng tổng số hành trình của người du lịch là n!, trong đó chỉ có (n-1)! hành trình thực sự khác nhau (bởi vì có thể xuất phát từ một thành phố bất kỳ nên có thể cố định một thành phố nào đó làm điểm xuất phát).

Bài toán cái túi: Một nhà thám hiểm cần đem theo một cái túi có trọng lượng không quá *b*. Có n đồ vật có thể đem theo. Đồ vật thứ *j* có trọng lượng a_j và giá trị sử dụng c_j *(j =1, 2,.., n)*. Hỏi nhà thám hiểm cần đem theo những đồ vật nào để cho tổng giá trị sử dụng là lớn nhất ?

Một phương án của nhà thám hiểm có thể biểu diễn như một vector nhị phân độ dài *n:* $x = (x_1,x_2,.., x_n)$, trong đó $x_i = 1$ có nghĩa là đồ vật thứ *i* được đem theo, $x_i = 0$ có nghĩa trái lại. Với phương án đem theo *x*, giá trị sử dụng các đồ vật đem theo là:

$f(x) = \sum_{i=1}^{n} c_i x_i$, tổng trọng lượng đồ vật đem theo là $g(x) = \sum_{i=1}^{n} a_i x_i$, như vậy bài toán cái túi được phát biểu dưới dạng bài toán tối ưu tổ hợp sau:

Trong số các vetor nhị phân độ dài n thoả mãn điều kiện $g(x) \le b$, hãy tìm vector x^* để hàm mục tiêu *f(x)* đạt giá trị nhỏ nhất. Nói cách khác:

$min\ \{ f(x): g(x) \le b \}$

Bài toán cho thuê máy: Một ông chủ có một cái máy để cho thuê. Đầu tháng ông ta nhận được yêu cầu thuê máy của *m* khách hàng. Mỗi khách hàng *i* sẽ cho biết tập N_i các ngày trong tháng cần sử dụng máy *(i = 1, 2,.., m)*. Ông chủ chỉ có quyền hoặc từ chối yêu cầu của khách hàng *i*, hoặc nếu nhận thì phải bố trí mãy phục vụ khách hàng *i* đúng những ngày mà khách hàng này yêu cầu. Hỏi rằng ông chủ phải tiếp nhận các yêu cầu của khách thế nào để cho tổng số ngày sử dụng máy là lớn nhất.

Ký hiệu, *I = { 1, 2,.., m }* là tập chỉ số khách hàng, *S* là tập hợp các tập con của *I*. Khi đó, tập hợp tất cả các phương án cho thuê máy là:

$D = \{J \subset S : N_k \cap N_p = \phi, \forall k \ne p \in J\ \}$. Với mỗi phương án J ∈ D $f(j) = \sum_{j \in J} |\ N_j\ |$ sẽ là tổng số ngày sử dụng máy theo phương án đó. Bài toán đặt ra có thể phát biểu dưới dạng bài toán tối ưu tổ hợp sau:

$$\max\{f(j) : j \in D\}.$$

Bài toán phân công: Có *n* công việc và *n* thợ. Biết c_{ij} là chi phí cần trả để thợ *i* hoàn thành công việc thứ *j (i, j = 1, 2,..., n)*. Cần phải thuê thợ sao cho các công việc đều hoàn thành và mỗi thợ chỉ thực hiện một công việc, mỗi công việc chỉ do một thợ thực hiện. Hãy tìm cách thuê *n* nhân công sao cho tổng chi phí thuê thợ là nhỏ nhất.

Rõ ràng, mỗi phương án bố trí thợ thực hiện các công việc tương ứng với một hoán vị $\pi = (\pi(1), \pi(2),..., \pi(n))$ của *n* số tự nhiên *{ 1, 2,.., n }*. Chi phí theo phương án trên là $f(\pi) = C_{\pi(1),1} + C_{\pi(2),2} + \cdots + C_{\pi(n),n}$.

Công việc	Thợ thực hiện
1	π(1)
2	π(2)
...	...
n	π(n)

Bài toán đặt ra được dẫn về bài toán tối ưu tổ hợp: $\min\{f(\pi) : \pi \in \Pi\}$.

Bài toán lập lịch: Mỗi một chi tiết trong số n chi tiết $D_1, D_2,.., D_n$ cần phải lần lượt được gia công trên m máy $M_1, M_2,.., M_m$. Thời gian gia công chi tiết D_i trên mãy M_j là t_{ij}. Hãy tìm lịch (trình tự gia công) các chi tiết trên các mãy sao cho việc hoàn thành gia công tất cả các chi tiết là sớm nhất có thể được. Biết rằng, các chi tiết được gia công một cách liên tục, nghĩa là quá trình gia công của mỗi một chi tiết phải được tiến hành một cách liên tục hết máy này sang máy khác không cho phép có khoảng thời gian dừng khi chuyển từ máy này sang máy khác.

Rõ ràng, mỗi một lịch gia công các chi tiết trên các máy sẽ tương ứng với một hoán vị $\pi = (\pi(1), \pi(2),.., \pi(n))$ của n số tự nhiên *1, 2,.., n*. Thời gian hoàn thành theo các lịch trên được xác định bởi hàm số:

$f(\pi) = \sum_{j=1}^{n-1} C_{\pi(j),\pi(j+1)} + \sum_{k=1}^{m} t_{k,\pi(n)}$, trong đó $c_{ij} = S_j - S_i$, S_j là thời điểm bắt đầu thực hiện việc gia công chi tiết *j (i, j = 1, 2,..., n)*. Ý nghĩa của hệ số c_{ij} có thể được giải thích như sau: nó là tổng thời gian gián đoạn (được tính từ khi bắt đầu gia công chi tiết *i*) gây ra bởi chi tiết *j* khi nó được gia công sau chi tiết i trong lịch gia công. Vì vậy, c_{ij} có thể tính theo công thức:

$c_{ij} = \max_{1 \le k \le m} \left[\sum_{l=1}^{k} t_{lj} - \sum_{l=1}^{k-1} t_{lj} \right]$, *i, j = 1, 2,..., n.* Vì vậy bài toán đặt ra dẫn về bài toán tối ưu tổ hợp sau:

$$\min \{ f(\pi): \pi \in \prod \}.$$

Trong thực tế, lịch gia công còn phải thoả mãn thêm nhiều điều kiện khác nữa. Vì những ứng dụng quan trọng của những bài toán loại này mà trong tối ưu hoá tổ hợp đã hình thành một lĩnh vực lý thuyết riêng về các bài toán lập lịch gọi là lý thuyết lập lịch hay qui hoạch lịch.

4.2. DUYỆT TOÀN BỘ

Một trong những phương pháp hiển nhiên nhất để giải bài toán tối ưu tổ hợp đặt ra là: Trên cơ sở các thuật toán lệt kê tổ hợp ta tiến hành duyệt từng phương án của bài toán, đối với mỗi phương án, ta đều tính giá trị hàm mục tiêu cho phương án đó, sau đó so sánh giá trị của hàm mục tiêu tại tất cả các phương án đã được liệt kê để tìm ra phương án tối ưu. Phương pháp xây dựng theo nguyên tắc như vậy được gọi là phương pháp duyệt toàn bộ.

Hạn chế của phương pháp duyệt toàn bộ là sự bùng nổ của các cấu hình tổ hợp. Chẳng hạn để duyệt được 15! = 1 307 674 368 000 cấu hình, trên máy có tốc độ 1 tỷ phép tính giây, nếu mỗi hoán vị cần liệt kê mất khoảng 100 phép tính, thì ta cần khoảng thời gian là 130767 giây (lớn hơn 36 tiếng đồng hồ). Vì vậy, cần phải có biện pháp hạn chế việc kiểm tra hoặc tìm kiếm trên các cấu hình tổ hợp thì mới có hy vọng giải được các bài toán tối ưu tổ hợp thực tế. Tất nhiên, để đưa ra được một thuật toán cần phải nghiên cứu kỹ tính chất của mỗi bài toán tổ hợp cụ thể. Chính nhờ những nghiên cứu đó, trong một số trường hợp cụ thể ta có thể xây dựng được thuật toán hiệu quả để giải quyết bài toán đặt ra. Nhưng chúng ta cũng cần phải chú ý rằng, trong nhiều trường hợp (bài toán người du lịch, bài toán cái túi, bài toán cho thuê máy) chúng ta vẫn chưa tìm ra được một phương pháp hữu hiệu nào ngoài phương pháp duyệt toàn bộ đã được đề cập ở trên.

Để hạn chế việc duyệt, trong quá trình liệt kê cần tận dụng triệt để những thông tin đã tìm để loại bỏ những phương án chắc chắn không phải là tối ưu. Dưới đây là một bài toán tối ưu tổ hợp rất thường hay gặp trong kỹ thuật.

Ví dụ. Duyệt mọi bộ giá trị trong tập các giá trị rời rạc.

Bài toán. Tìm:

$$\max\{f(x_1,x_2,...,x_n) : x_i \in D_i; i = 1,2,...,n\} \text{ hoặc:}$$

$$\min\{f(x_1,x_2,...,x_n) : x_i \in D_i; i = 1,2,...,n\}.$$

Trong đó, D_i là một tập hữu hạn các giá trị rời rạc thỏa mãn một điều kiện ràng buộc nào đó.

Giải.

Giả sử số các phần tử của tập giá trị rời rạc D_i là r_i *(i=1, 2,..., n)*. Gọi $R = r_1 + r_2 +... + r_n$ là số các phần tử thuộc tất cả các tập D_i *(i=1, 2,..., n)*. Khi đó, ta có tất cả *C(R, n)* bộ có thứ tự các giá trị gồm *n* phần tử trong *R* phần tử, đây chính là số các phương án ta cần duyệt. Trong số *C(R,n)* các bộ *n* phần tử, ta cần lọc ra các bộ thoả mãn điều kiện $x_i \in D_i$ *(i=1, 2,..., n)* để tính giá trị của hàm mục tiêu $f(x_1, x_2,..., x_n)$. Như vậy, bài toán được đưa về bài toán duyệt các bộ gồm *n* phần tử $(x_1, x_2,..., x_n)$ từ tập hợp gồm $R = r_1 + r_2 +.. + r_n$ phần tử thoả mãn điều kiện $x_i \in D_i$.

Ví dụ: Với tập $D_1 = (1, 2, 3)$,

$D_2 = (3, 4)$,

$D_3 = (5, 6, 7)$.

Khi đó chúng ta cần duyệt bộ các giá trị rời rạc sau:

1	3	5	2	4	5
1	3	6	2	4	6
1	3	7	2	4	7
1	4	5	3	3	5
1	4	6	3	3	6
1	4	7	3	3	7
2	3	5	3	4	5
2	3	6	3	4	6
2	3	7	3	4	7

Với cách phân tích như trên, ta có thể sử dụng thuật toán quay lui để duyệt kết hợp với việc kiểm tra thành phần $x_i \in D_i$. Dưới đây là toàn văn chương trình duyệt các bộ giá trị trong tập các giá trị rời rạc.

```
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <conio.h>
#define MAX         2000000
#define TRUE 1
#define FALSE 0
int n, k, H[100]; float *B;int *C, count =0, m;
FILE *fp;
void Init(void){
    int i,j;float x;C[0]=0;H[0]=0;
    fp=fopen("roirac.in","r");
    fscanf(fp,"%d",&n);
    printf("\n So tap con roi rac n=%d",n);
    for(i=1; i<=n; i++){
            fscanf(fp,"%d",&H[i]);
            printf("\n Hang %d co so phan tu la %d",i, H[i]);
    }
    H[0]=0;
    for (i=1; i<=n; i++){
            printf("\n");
            for(j=1; j<=H[i]; j++){
```

```
                    fscanf(fp,"%f",&x);
                    B[++k]=x;
                }
    }
    printf("\n B=");
    for(i=1; i<=k; i++){
                printf("%8.2f", B[i]);
    }
    fclose(fp);
}
int In_Set(int i){
    int canduoi=0, cantren=0,j;
    for(j=1; j<=i; j++)
                cantren = cantren + H[j];
    canduoi=cantren-H[j-1];
    if (C[i]> canduoi && C[i]<=cantren)
                return(TRUE);
    return(FALSE);
}
void Result(void){
    int i;
    count++; printf("\n Tap con thu count=%d:",count);
    for(i=1; i<=n ; i++){
                printf("%8.2f", B[C[i]]);
    }
}
void Try(int i){
    int j;
    for(j = C[i-1]+1; j<=(k-n+i); j++){
                C[i]=j;
                if(In_Set(i)){
                    if (i==n ) Result();
```

```
                    else Try(i+1);
                }
        }
}
void main(void){
    clrscr();
    B = (float *) malloc(MAX *sizeof(float));
    C = (int *) malloc(MAX *sizeof(int));
    Init();Try(1);free(B); free(C);getch();
}
```

4.3. THUẬT TOÁN NHÁNH CẬN

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

$\min\{f(x) : x \in D\}$. Trong đó D là tập hữu hạn phần tử. Ta giả thiết D được mô tả như sau:

$D = \{ x = (x_1, x_2, ..., x_n) \in A_1 \times A_2 \times ... \times A_n$; x thoả mãn tính chất $P \}$, với $A_1 \times A_2 \times ... \times A_n$ là các tập hữu hạn, P là tính chất cho trên tích đề xác $A_1 \times A_2 \times ... \times A_n$.

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 thành phần $(a_1, a_2, ..., a_k)$ 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 sau:

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

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

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

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

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

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 $\bar{x}$ là giá trị hàm mục tiêu nhỏ nhất trong số các phương án đã duyệt, ký hiệu $\bar{f} = f(\bar{x})$. Ta gọi $\bar{x}$ là phương án tốt nhất hiện có, còn $\bar{f}$ là kỷ lục. Giả sử ta có được $\bar{f}$, khi đó nếu:

$g(a_1, a_2,.., a_k) > \bar{f}$ thì từ bất đẳng thức (*) ta suy ra:

$\bar{f} < g(a_1, a_2,..., a_k) \le min\ \{ f(x): x \in D, x_i = a_i, i=1, 2,..., k \}$, vì thế tập con các phương án của bài toán $D(a_1, a_2, ..., a_k)$ chắc chắn không chứa phương án tối ưu. Trong trường hợp này ta không cần phải phát triển phương án bộ phận $(a_1, a_2,..., a_k)$, nói cách khác là ta có thể loại bỏ các phương án trong tập $D(a_1, a_2,.., a_n)$ khỏi quá trình tìm kiếm.

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

```
void Try(int k){
/*Phát triển phương án bộ phận (a1, a2,..., ak-1
theo thuật toán quay lui có kiểm tra cận dưới
Trước khi tiếp tục phát triển phương án*/
for ( ak ∈ Ak ) {
        if ( chấp nhận ak ){
            xk = ak;
            if (k == n)
< cập nhật kỷ lục>;
            else if (g(a1, a2,..., ak) ≤ f̄ )
Try (k+1);
        }
    }
}
```

Khi đó, thuật toán nhánh cận được thực hiện nhờ thủ tục sau:

```
void Nhanh_Can(void) {
    f̄ = +∞;
/* Nếu biết một phương án x̄ nào đó thì có thể đặt f̄ = f(x̄). */
    Try(1);
    if ( f̄ ≤ +∞ )
< f̄ là giá trị tối ưu, x̄ là phương án tối ưu >;
```

```
    else
< bài toán không có phương án>;
}
```

Chú ý rằng nếu trong thủ tục *Try* ta thay thế câu lệnh:

```
    if (k == n)
< cập nhật kỷ lục >;
    else if (g(a1, a2,.., ak) ≤ f )
Try(k+1);
```

bởi

```
    if (k == n)
< cập nhật kỷ lục >;
    else Try(k+1);
```

thì thủ tục *Try* sẽ liệt kê toàn bộ các phương án của bài toán, và ta lại thu được thuật toán duyệt toàn bộ. 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:

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

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

Ví dụ 1. Bài toán cái túi. Chúng ta sẽ xét bài toán cái túi tổng quát hơn mô hình đã được trình bày trong mục 4.1. Thay vì có n đồ vật, ở đây ta giả thiết rằng có n loại đồ vật và số lượng đồ vật mỗi loại là không hạn chế. Khi đó, ta có mô hình bài toán cái túi biến nguyên sau đây: Có *n* loại đồ vật, đồ vật thứ *j* có trọng lượng a_j và giá trị sử dụng c_j *(j =1, 2,.., n)*. Cần chất các đồ vật này vào một cái túi có trọng lượng là *b* sao cho tổng giá trị sử dụng của các đồ vật đựng trong túi là lớn nhất.

Mô hình toán học của bài toán có dạng sau tìm:

$$f^* = \max\left\{ f(x) = \sum_{j=1}^{n} c_j x_j : \sum_{j=1}^{n} a_j x_j \le b, x_j \in Z_+, j = 1,2,...,n \right\}, (1).$$

trong đó Z^+ là tập các số nguyên không âm.

Ký hiệu *D* là tập các phương án của bài toán *(1)*:

$$D = \left\{ x = (x_1, x_2, \cdots, x_n : \sum_{j=1}^{n} a_j x_j \le b, x_j \in Z_+, j = 1,2,\cdots,n \right\}.$$

Không giảm tính tổng quát ta giả thiết rằng, các đồ vật được đánh số sao cho bất đẳng thức sau được thoả mãn

$$\frac{c_1}{a_1} \geq \frac{c_2}{a_2} \geq \cdots \geq \frac{c_n}{a_n} \qquad (2)$$

Để xây dựng hàm tính cận dưới, cùng với bài toán cái túi *(1)* ta xét bài toán cái túi biến liên tục sau:

$$\text{Tìm:} \qquad g^* = \max\left\{\sum_{j=1}^{n} c_j x_j : \sum_{j=1}^{n} a_j x_j \leq b, x_j \geq 0, j = 1,2,\cdots,n\right\}. \qquad (3)$$

Mệnh đề. Phương án tối ưu của bài toán (3) là vector $\overline{x} = (\overline{x_1}, \overline{x_2}, \ldots, \overline{x_n})$ với các thành phần được xác định bởi công thức:

$\overline{x_1} = \dfrac{b}{a_1}$, $\overline{x_2} = \overline{x_3} = \cdots = \overline{x_n} = 0$ và giá trị tối ưu là $g^* = \dfrac{c_1 b_1}{a_1}$.

Chứng minh. Thực vậy, xét $x = (x_1, x_2, .., x_n)$ là một phương án tuỳ ý của bài toán (3). Khi đó từ bất đẳng thức (3) và do $x_j \geq 0$, ta suy ra:

$$c_j x_j \geq (c_1 / a_1) a_j x_j,\ j = 1, 2, \cdots n.$$

suy ra:

$$\sum_{j=1}^{n} c_j x_j \leq \sum_{j=1}^{n} (\frac{c_1}{a_1}) a_j x_j = (\frac{c_1}{a_1}) \sum_{j=1}^{n} a_j x_j \leq \frac{c_1}{a_1} b = g^*.$$ Mệnh đề được chứng minh.

Bây giờ ta giả sử có phương án bộ phận cấp k: $(u_1, u_2, .., u_k)$. Khi đó giá trị sử dụng của các đồ vật đang có trong túi là:

$\partial_k = c_1 u_1 + c_2 u_2 + \cdots + c_k u_k$, và trọng lượng còn lại của túi là:

$b_k = b - c_1 u_1 + c_2 u_2 + \cdots + c_k u_k$,

ta có:

$$\max\{f(x) : x \in D, x_j = u_j, j = 1,2,\cdots n\}$$

$$= \max\left\{\partial_k + \sum_{j=k+1}^{n} c_j x_j : \sum_{j=k+1}^{n} a_j x_j \leq b_k, x_j \in Z_+, j = k+1, k+2, \cdots, n\right\}$$

$$\leq \partial_k + \max\left\{\sum_{j=k+1}^{n} c_j x_j : \sum_{j=k+1}^{n} a_j x_j \leq b_k, x_j \geq 0, j = k+1, k+2, \cdots, n\right\}$$

$$= \partial_k + \frac{c_{k+1} b_k}{a_{k+1}}$$

(Theo mệnh đề giá trị số hạng thứ hai là $\frac{c_{k+1}b_k}{a_{k+1}}$)

Vậy ta có thể tính cận trên cho phương án bộ phận $(u_1, u_2,..., u_k)$ theo công thức:

$$g(u_1,u_2,\cdots,u_k) = \partial_k + \frac{c_{k+1}b_k}{a_{k+1}}\cdots$$

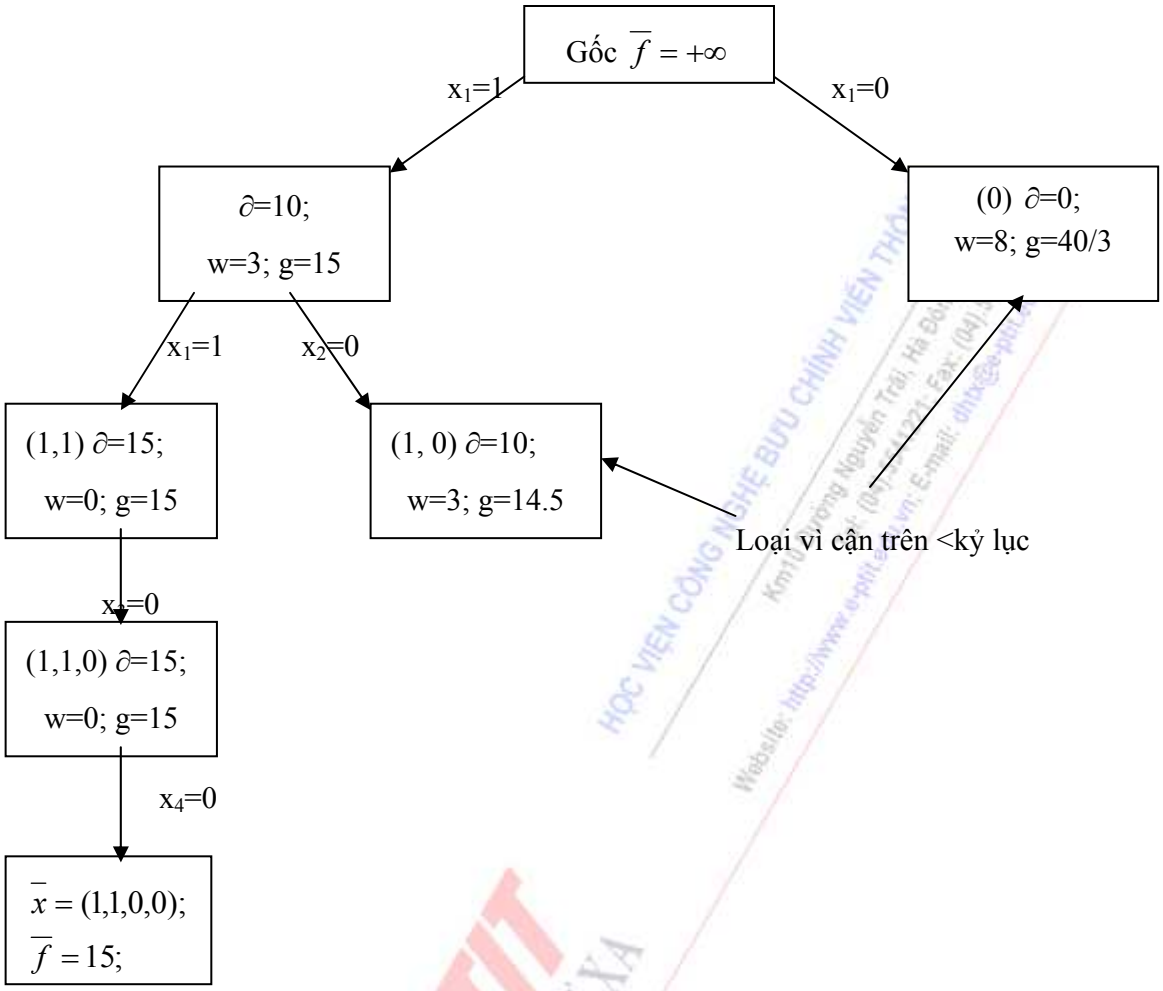
Chú ý: Khi tiếp tục xây dựng thành phần thứ $k+1$ của lời giải, các giá trị đề cử cho x_{k+1} sẽ là $0, 1,..., [b_k/a_{k+1}]$. Do có kết quả của mệnh đề, khi chọn giá trị cho x_{k+1} ta sẽ duyệt các giá trị đề cử theo thứ tự giảm dần.

Ví dụ. Giải bài toán cái túi sau theo thuật toán nhánh cận trình bày trên.

$$f(x) = 10x_1 + 5x_2 + 3x_3 + 6x_4 \to \max$$
$$5x_1 + 3x_2 + 2x_3 + 4x_4 \le 8$$
$$x_j \in Z_+, j = 1,2,3,4.$$

Giải. Quá trình giải bài toán được mô tả trong cây tìm kiếm trong hình 4.1. Thông tin về một phương án bộ phận trên cây được ghi trong các ô trên hình vẽ tương ứng theo thứ tự sau: đầu tiên là các thành phần của phương án, tiếp đến ∂ là giá trị của các đồ vật chất trong túi, w là trọng lượng còn lại của túi và g là cận trên.

Kết thúc thuật toán, ta thu được phương án tối ưu là x* =(1, 1, 0, 1), giá trị tối ưu f*= 15.



Hình 4.1. Giải bài toán cái túi theo thuật toán nhánh cận.

Chương trình giải bài toán cái túi theo thuật toán nhánh cận được thể hiện như sau:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <dos.h>
#define TRUE        1
#define FALSE       0
#define MAX         100
int x[MAX], xopt[MAX];
float       fopt, cost, weight;
```

```
void Init(float *C, float *A, int *n, float *w){
    int i;FILE *fp;
    fopt=0; weight=0;
    fp=fopen("caitui.in","r");
    if(fp==NULL){
            printf("\n Khong co file input");
            delay(2000); return;
    }
    fscanf(fp,"%d %f", n,w);
    for(i=1; i<=*n;i++) xopt[i]=0;
    printf("\n So luong do vat %d:", *n);
    printf("\n Gioi han tui %8.2f:", *w);
    printf("\n Vecto gia tri:");
    for(i=1; i<=*n; i++) {
            fscanf(fp,"%f", &C[i]);
            printf("%8.2f", C[i]);
    }
    printf("\n Vector trong luong:");
    for(i=1; i<=*n; i++){
            fscanf(fp,"%f", &A[i]);
            printf("%8.2f", A[i]);
    }
    fclose(fp);
}
void swap(int n){
    int i;
    for(i=1; i<=n; i++)
            xopt[i]=x[i];
}
void Update_Kyluc(int n){
    if(cost>fopt){
            swap(n);
```

```
            fopt=cost;
    }
}
void Try(float *A, float *C, int n, float w, int i){
    int j, t=(w-weight)/A[i];
    for(j=t; j>=0;j--){
            x[i]=j;
            cost = cost + C[i]*x[i];
            weight = weight + x[i]*A[i];
            if(i==n) Update_Kyluc(n);
            else if(cost + C[i+1]*(w-weight)/A[i+1]> fopt){
                    Try(A, C, n, w, i+1);
            }
            weight = weight-A[i]*x[i];
            cost = cost-C[i]*x[i];

    }
}
void Result(int n){
    int i;
    printf("\n Gia tri do vat %8.2f:", fopt);
    printf("\n Phuong an toi uu:");
    for(i=1; i<=n; i++)
            printf("%3d", xopt[i]);
}
void main(void){
    int     n;
    float   A[MAX], C[MAX], w;
    clrscr();Init(C, A, &n, &w);
    Try(C, A, n, w,1);Result(n);
    getch();
}
```

Ví dụ 2. Bài toán Người du lịch. Một người du lịch muốn đi thăm quan n thành phố $T_1, T_2, ..., T_n$. Xuất phát từ một thành phố nào đó, người du lịch muốn đi qua tất cả các thành phố còn lại, mỗi thành phố đi qua 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 = 1, 2,.., n)*, hãy tìm hành trình với tổng chi phí là nhỏ nhất (một hành trình là một cách đi thoả mãn điều kiện).

Giải. Cố định thành phố xuất phát là T_1. Bài toán Người du lịch được đưa về bài toán: Tìm cực tiểu của phiếm hàm:

$$f(x_1, x_2, \cdots, x_n) = c[1, x_2] + c[x_2, x_3] + \cdots + c[x_{n-1}, x_n] + c[x_n, x_1] \to \min$$

với điều kiện

$c_{\min} = \min\{c[i, j], i, j = 1,2,\cdots,n; i \neq j\}$ là chi phí đi lại giữa các thành phố.

Giả sử ta đang có phương án bộ phận $(u_1, u_2,..., u_k)$. Phương án tương ứng với hành trình bộ phận qua k thành phố:

$$T_1 \to T(u_2) \to \cdots \to T(u_{k-1}) \to T(u_k)$$

Vì vậy, chi phí phải trả theo hành trình bộ phận này sẽ là tổng các chi phí theo từng node của hành trình bộ phận.

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

Để phát triển hành trình bộ phận này thành hành trình đầy đủ, ta còn phải đi qua $n-k$ thành phố còn lại rồi quay trở về thành phố T_1, tức là còn phải đi qua $n-k+1$ đoạn đường nữa. Do chi phí phải trả cho việc đi qua mỗi trong $n-k+1$ đoạn đường còn lại đều không nhiều hơn c_{min}, nên cận dưới cho phương án bộ phận $(u_1, u_2,..., u_k)$ có thể được tính theo công thức

$g(u_1, u_2,..., u_k) = \partial + (n - k + 1)\, c_{min}$.

Chẳng hạn ta giải bài toán người du lịch với ma trận chi phí như sau

$$C = \begin{vmatrix} 0 & 3 & 14 & 18 & 15 \\ 3 & 0 & 4 & 22 & 20 \\ 17 & 9 & 0 & 16 & 4 \\ 6 & 2 & 7 & 0 & 12 \\ 9 & 15 & 11 & 5 & 0 \end{vmatrix}$$

Ta có $c_{min} = 2$. Quá trình thực hiện thuật toán được mô tả bởi cây tìm kiếm lời giải được thể hiện trong hình 4.2.

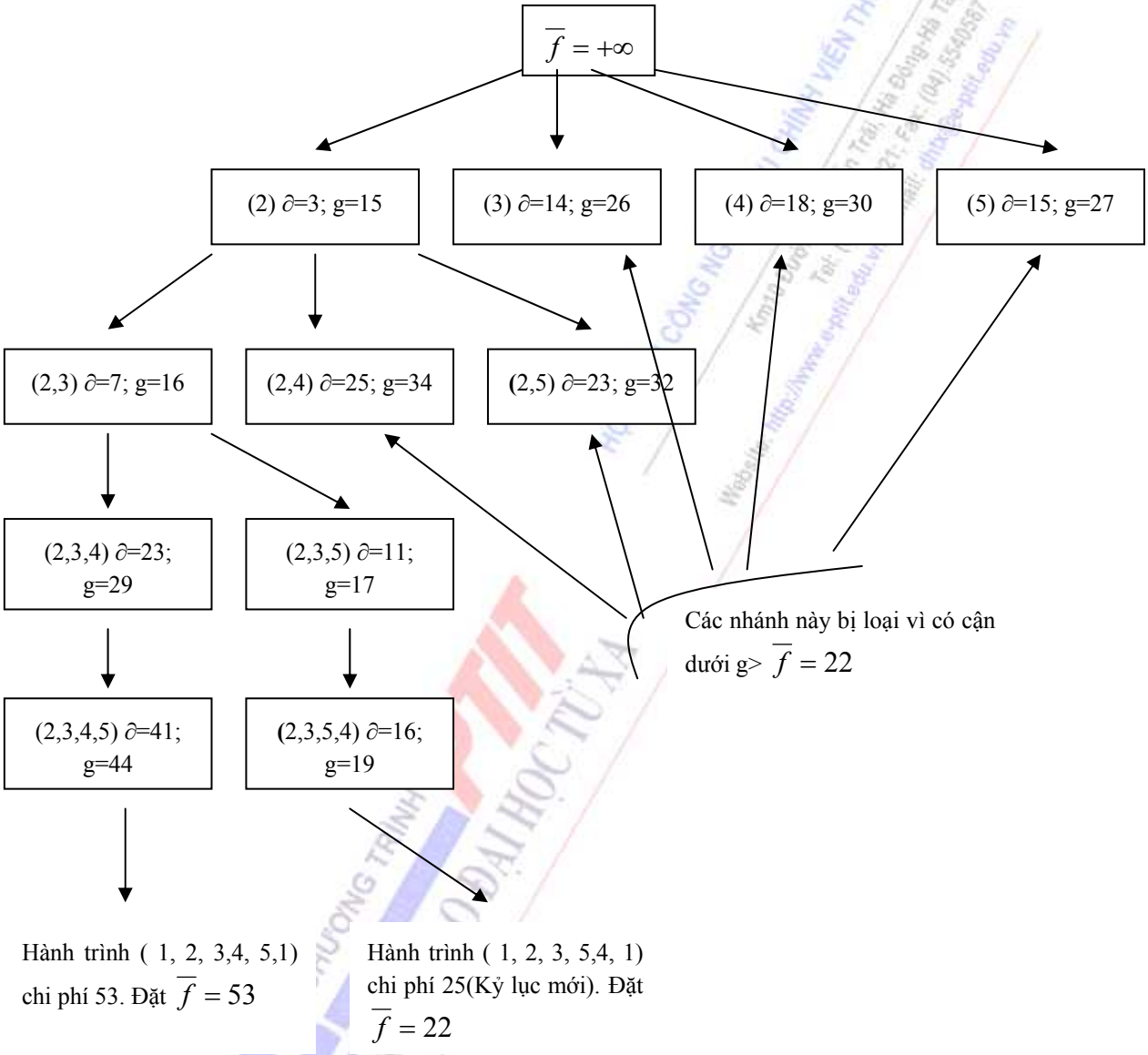
Thông tin về một phương án bộ phận trên cây được ghi trong các ô trên hình vẽ tương ứng theo thứ tự sau:

- Đầu tiên là các thành phần của phương án
- Tiếp đến ∂ là chi phí theo hành trình bộ phận

- g là cận dưới

Kết thúc thuật toán, ta thu được phương án tối ưu (1, 2, 3, 5, 4, 1) tương ứng với phương án tối ưu với hành trình:

$T_1 \to T_2 \to T_3 \to T_5 \to T_4 \to T_1$ và chi phí nhỏ nhất là 22



Hình 4.2. Cây tìm kiếm lời giải bài toán người du lịch.

Chương trình giải bài toán theo thuật toán nhánh cận được thể hiện như sau:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
```

```
#include <io.h>
#define MAX 20
int n, P[MAX], B[MAX], C[20][20], count=0;
int A[MAX], XOPT[MAX];
int can, cmin, fopt;
void Read_Data(void){
    int i, j;FILE *fp;
    fp = fopen("dulich.in","r");
    fscanf(fp,"%d", &n);
    printf("\n So thanh pho: %d", n);
    printf("\n Ma tran chi phi:");
    for (i=1; i<=n; i++){
            printf("\n");
            for(j=1; j<=n; j++){
                    fscanf(fp,"%d",&C[i][j]);
                    printf("%5d", C[i][j]);
            }
    }
}
int Min_Matrix(void){
    int min=1000, i, j;
    for(i=1; i<=n; i++){
            for(j=1; j<=n; j++){
                    if (i!=j && min>C[i][j])
                            min=C[i][j];
            }
    }
    return(min);
}
void Init(void){
    int i;
    cmin=Min_Matrix();
```

```
    fopt=32000;can=0; A[1]=1;
    for (i=1;i<=n; i++)
            B[i]=1;
}
void Result(void){
    int i;
    printf("\n Hanh trinh toi uu %d:", fopt);
    printf("\n Hanh trinh:");
    for(i=1; i<=n; i++)
            printf("%3d->", XOPT[i]);
    printf("%d",1);
}
void Swap(void){
    int i;
    for(i=1; i<=n;i++)
            XOPT[i]=A[i];
}
void Update_Kyluc(void){
    int sum;
    sum=can+C[A[n]][A[1]];
    if(sum<fopt) {
            Swap();
            fopt=sum;
    }
}
void Try(int i){
    int j;
    for(j=2; j<=n;j++){
            if(B[j]){
                    A[i]=j; B[j]=0;
                    can=can+C[A[i-1]][A[i]];
                    if (i==n) Update_Kyluc();
```

```
                    else if( can + (n-i+1)*cmin< fopt){
                            count++;
                            Try(i+1);
                    }
                    B[j]=1;can=can-C[A[i-1]][A[i]];
            }
        }
    }
    void main(void){
        clrscr();Read_Data();Init();
        Try(2);Result(); getch();
    }
```

4.4. KỸ THUẬT RÚT GỌN GIẢI QUYẾT BÀI TOÁN NGƯỜI DU LỊCH

Thuật toán nhánh cận là phương pháp chủ yếu để giải các bài toán tối ưu tổ hợp. Tư tưởng cơ bản của thuật toán là trong quá trình tìm kiếm lời giải, ta sẽ phân hoạch tập các phương án của bài toán thành hai hay nhiều tập con biểu diễn như một node của cây tìm kiếm và cố gắng bằng phép đánh giá cận các node, tìm cách loại bỏ những nhánh cây (những tập con các phương án của bài toán) mà ta biết chắc chắn không phương án tối ưu. Mặc dù trong trường hợp tồi nhất thuật toán sẽ trở thành duyệt toàn bộ, nhưng trong những trường hợp cụ thể nó có thể rút ngắn đáng kể thời gian tìm kiếm. Mục này sẽ thể hiện khác những tư tưởng của thuật toán nhánh cận vào việc giải quyết bài toán người du lịch.

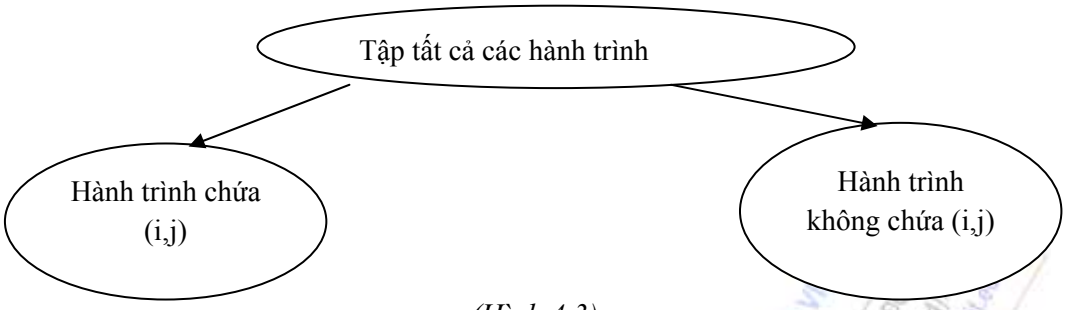
Xét bài toán người du lịch như đã được phát biểu. Gọi $C = \{ c_{ij}: i, j = 1, 2,...n\}$ là ma trận chi phí. Mỗi hành trình của người du lịch:

$T_{\pi(1)} \rightarrow T_{\pi(2)} \rightarrow ... \rightarrow T_{\pi(n)} \rightarrow T_{\pi(1)}$ có thể viết lại dưới dạng:

$(\pi(1), \pi(2), \pi(2), \pi(3),..., \pi(n-1), \pi(n), \pi(n), \pi(1)$, trong đó mỗi thành phần:

$\pi(j-1), \pi(j)$ sẽ được gọi là một cạnh của hành trình.

Khi tiến hành tìm kiếm lời giải bài toán người du lịch chúng ta phân tập các hành trình thành 2 tập con: Tập những hành trình chứa một cặp cạnh (i, j) nào đó còn tập kia gồm những hành trình không chứa cạnh này. Ta gọi việc làm đó là sự phân nhánh, mỗi tập con như vậy được gọi là một nhánh hay một node của cây tìm kiếm. Quá trình phân nhánh được minh hoạ bởi cây tìm kiếm như trong hình 4.3.



(Hình 4.3)

Việc phân nhánh sẽ được thực hiện dựa trên một qui tắc heuristic nào đó cho phép ta rút ngắn quá trình tìm kiếm phương án tối ưu. Sau khi phân nhánh và tính cận dưới giá trị hàm mục tiêu trên mỗi tập con. Việc tìm kiếm sẽ tiếp tục trên tập con có giá trị cận dưới nhỏ hơn. Thủ tục này được tiếp tục cho đến khi ta nhận được một hành trình đầy đủ tức là một phương án cuả bài toán. Khi đó ta chỉ cần xét những tập con các phương án nào có cận dưới nhỏ hơn giá trị của hàm mục tiêu tại phương án đã tìm được. Quá trình phân nhánh và tính cận trên tập các phương án của bài toán thông thường cho phép rút ngắn một cách đáng kể quá trình tìm kiếm do ta loại được rất nhiều tập con chắc chắn không chứa phương án tối ưu. Sau đây, là một kỹ thuật nữa của thuật toán.

4.4.1.Thủ tục rút gọn

Rõ ràng tổng chi phí của một hành trình của người du lịch sẽ chứa đúng một phần tử của mỗi dòng và đúng một phần tử của mỗi cột trong ma trận chi phí C. Do đó, nếu ta cộng hay trừ bớt mỗi phần tử của một dòng (hay cột) của ma trận C đi cùng một số α thì độ dài của tất cả các hành trình đều giảm đi α vì thế hành trình tối ưu cũng sẽ không bị thay đổi. Vì vậy, nếu ta tiến hành bớt đi các phần tử của mỗi dòng và mỗi cột đi một hằng số sao cho ta thu được một ma trận gồm các phần tử không âm mà trên mỗi dòng, mỗi cột đều có ít nhất một số 0, thì tổng các số trừ đó cho ta cận dưới của mọi hành trình. Thủ tục bớt này được gọi là thủ tục rút gọn, các hằng số trừ ở mỗi dòng (cột) sẽ được gọi là hằng số rút gọn theo dòng(cột), ma trận thu được được gọi là ma trận rút gọn. Thủ tục sau cho phép rút gọn ma trận một ma trận A kích thước $k \times k$ đồng thời tính tổng các hằng số rút gọn.

```
float Reduce( float A[][max], int k) {
    sum =0;
    for (i = 1; i≤k; i++){
            r[i] = < phần tử nhỏ nhất của dòng i >;
            if (r[i] > 0 ) {
                    <Bớt mỗi phần tử của dòng i đi r[i] >;
                    sum = sum + r[i];
            }
    }
```

```
        for (j=1; j≤ k; j++) {
                s[j]:= <Phần tử nhỏ nhất của cột j>;
                if (s[j] > 0 )
                        sum = sum + S[j];
        }
        return(sum);
}
```

Ví dụ. Giả sử ta có ma trận chi phí với n= 6 thành phố sau:

	1	2	3	4	5	6	r[i]
1	∞	3	93	13	33	9	3
2	4	∞	77	42	21	16	4
3	45	17	∞	36	16	28	16
4	39	90	80	∞	56	7	7
5	28	46	88	33	∞	25	25
6	3	88	18	46	92	∞	3
	0	0	15	8	0	0	

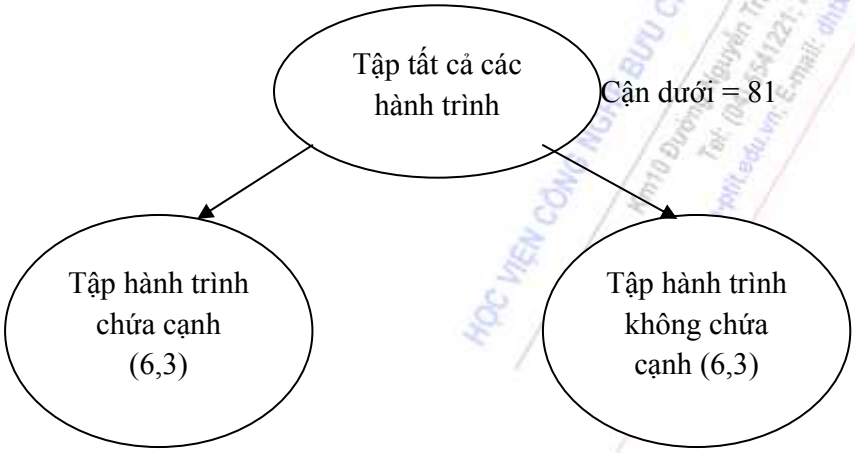
Đầu tiên trừ bớt mỗi phần tử của các dòng 1, 2, 3, 4, 5, 6 cho các hằng số rút gọn tương ứng là (3, 4, 16, 7, 25, 3), sau đó trong ma trận thu được ta tìm được phần tử nhỏ khác 0 của cột 3 và 4 tương ứng là (15, 8). Thực hiện rút gọn theo cột ta nhận được ma trận sau:

	1	2	3	4	5	6
1	∞	0	75	2	30	6
2	0	∞	58	30	17	12
3	29	1	∞	12	0	12
4	32	83	58	∞	49	0
5	3	21	48	0	∞	0
6	0	85	0	35	89	∞

Tổng các hằng số rút gọn là 81, vì vậy cận dưới cho tất cả các hành trình là 81 (không thể có hành trình có chi phí nhỏ hơn 81).

Bây giờ ta xét cách phân tập các phương án ra thành hai tập. Giả sử ta chọn cạnh (6, 3) để phân nhánh. Khi đó tập các hành trình được phân thành hai tập con, một tập là các hành trình chứa cạnh (6,3), còn tập kia là các hành trình không chứa cạnh (6,3). Vì biết cạnh (6, 3) không tham gia

vào hành trình nên ta cấm hành trình đi qua cạnh này bằng cách đặt C[6, 3] = ∞. Ma trận thu được sẽ có thể rút gọn bằng cách bớt đi mỗi phần tử của cột 3 đi 48 (hàng 6 giữ nguyên). Như vậy ta thu được cận dưới của hành trình không chứa cạnh (6,3) là 81 + 48 = 129. Còn đối với tập chứa cạnh (6, 3) ta phải loại dòng 6, cột 3 khỏi ma trận tương ứng với nó, bởi vì đã đi theo cạnh (6, 3) thì không thể đi từ 6 sang bất sang bất cứ nơi nào khác và cũng không được phép đi bất cứ đâu từ 3. Kết quả nhận được là ma trận với bậc giảm đi 1. Ngoài ra, do đã đi theo cạnh (6, 3) nên không được phép đi từ 3 đến 6 nữa, vì vậy cần cấm đi theo cạnh (3, 6) bằng cách đặt C(3, 6) = ∞. Cây tìm kiếm lúc này có dạng như trong hình 4.4.



Hình 4.4

Cận dưới =81

	1	2	4	5	6
1	∞	0	2	30	6
2	0	∞	30	17	12
3	29	1	12	0	∞
4	32	83	∞	49	0
5	3	21	0	∞	0

Cận dưới = 129

	1	2	3	4	5	6
1	∞	0	27	2	30	6
2	0	∞	10	30	17	12
3	29	1	∞	12	0	12
4	32	83	10	∞	49	0
5	3	21	0	0	∞	0
6	0	85	∞	35	89	∞

Cạnh (6,3) được chọn để phân nhánh vì phân nhánh theo nó ta thu được cận dưới của nhánh bên phải là lớn nhất so với việc phân nhánh theo các cạnh khác. Qui tắc này sẽ được áp dụng ở để phân nhánh ở mỗi đỉnh của cây tìm kiếm. Trong quá trình tìm kiếm chúng ta luôn đi theo nhánh bên trái trước. Nhánh bên trái sẽ có ma trận rút gọn với bậc giảm đi 1. Trong ma trận của nhánh bên phải ta thay một số bởi ∞, và có thể rút gọn thêm được ma trận này khi tính lại các hằng số rút gọn theo dòng và cột tương ứng với cạnh phân nhánh, nhưng kích thước của ma trận vẫn giữ nguyên.

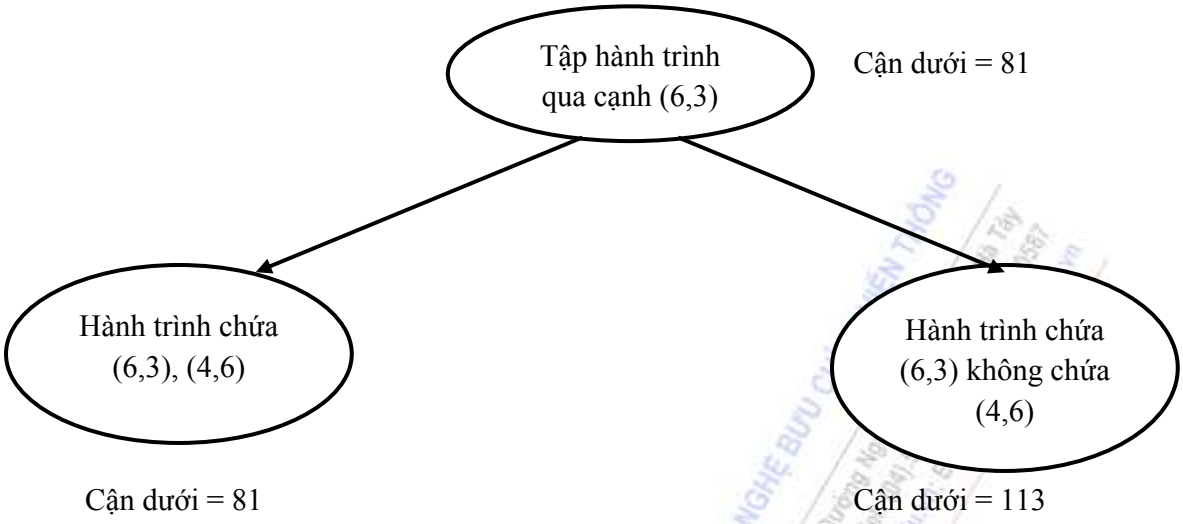
Do cạnh chọn để phân nhánh phải là cạnh làm tăng cận dưới của nhánh bên phải lên nhiều nhất, nên để tìm nó ta sẽ chọn số không nào trong ma trận mà khi thay nó bởi ∞ sẽ cho ta tổng hằng số rút gọn theo dòng và cột chứa nó là lớn nhất. Thủ tục đó có thể được mô tả như sau để chọn cạnh phân nhánh (r, c).

4.4.2.Thủ tục chọn cạnh phân nhánh (r,c)

```
void BestEdge(A, k, r, c, beta)
Đầu vào: Ma trận rút gọn A kích thước k × k
Kết quả ra: Cạnh phân nhánh (r,c) và tổng hằng số rút gọn theo dòng r cột c là beta.
{
    beta = -∞;
    for ( i = 1; i≤ k; i++){
    for (j = 1; j≤ k; j++) {
                    if (A[i,j] == 0){
                            minr = <phần tử nhỏ nhất trên dòng i khác với A[i,j];
                            minc = <phần tử nhỏ nhất trên cột j khác với A[i,j];
                            total = minr + minc;
                            if (total > beta ) {
                                    beta = total;
                                    r = i; /* Chỉ số dòng tốt nhất*/
                                    c = j; /* Chỉ số cột tốt nhất*/
                            }
                    }
            }
    }
}
```

Trong ma trận rút gọn 5 × 5 của nhánh bên trái hình 5.4, số không ở vị trí (4, 6) sẽ cho tổng hằng số rút gọn là 32 (theo dòng 4 là 32, cột 6 là 0). Đây là hệ số rút gọn có giá trị lớn nhất đối với các số không của ma trận này. Việc phân nhánh tiếp tục sẽ dựa vào cạnh (4, 6). Khi đó cận dưới của nhánh bên phải tương ứng với tập hành trình đi qua cạnh (6,3) nhưng không đi qua cạnh (4, 6) sẽ là 81 + 32 = 113. Còn nhánh bên trái sẽ tương ứng với ma trận 4 × 4 (vì ta phải loại bỏ dòng 4 và cột 6). Tình huống phân nhánh này được mô tả trong hình 4.5.

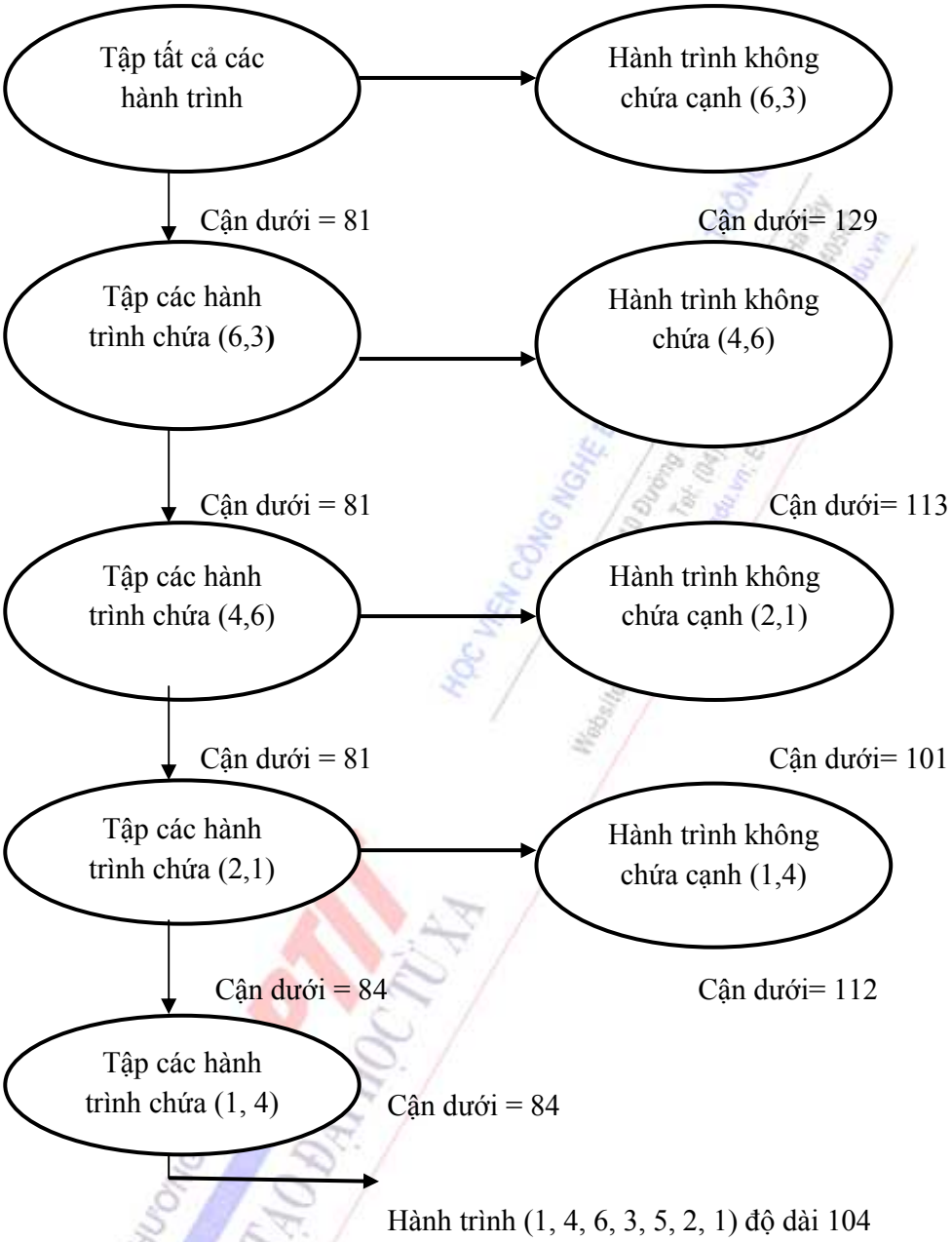
Nhận thấy rằng vì cạnh (4, 6) và (6, 3) đã nằm trong hành trình nên cạnh (3, 4) không thể đi qua được nữa (nếu đi qua ta sẽ có một hành trình con từ những thành phố này). Để ngăn cấm việc tạo thành các hành trình con ta sẽ gán cho phần tử ở vị trí (3, 4) giá trị ∞.



Hình 4.5

Ngăn cấm tạo thành hành trình con:

Tổng quát hơn, khi phân nhánh dựa vào cạnh (i_u, i_v) ta phải thêm cạnh này vào danh sách các cạnh của node bên trái nhất. Nếu iu là đỉnh cuối của một đường đi $(i_1, i_2,.., i_u)$ và j_v là đỉnh đầu của đường đi $(j_1, j_2,.., j_k)$ thì để ngăn ngừa khả năng tạo thành hành trình con ta phải ngăn ngừa khả năng tạo thành hành hành trình con ta phải cấm cạnh (j_k, i_1). Để tìm i_1 ta đi ngược từ iu, để tìm j_k ta đi xuôi từ j_1 theo danh sách các cạnh đã được kết nạp vào hành trình.



Hình 4.6 mô tả quá trình tìm kiếm giải pháp tối ưu

Tiếp tục phân nhánh từ đỉnh bên trái bằng cách sử dụng cạnh (2,1) vì số không ở vị trí này có hằng số rút gọn lớn nhất là 17 + 3 = 20 (theo dòng 2 là 17, theo cột 1 là 3). Sau khi phân nhánh theo cạnh (2, 1) ma trận của nhánh bên trái có kích thước là 3 × 3. Vì đã đi qua (2, 1) nên ta cấm cạnh (2, 1) bằng cách đặt C[1, 2] = ∞, ta thu được ma trận sau:

		2	4	5	
1		∞	2	30	
3		1	∞	0	
5		21	0	∞	

Ma trận này có thể rút gọn được bằng cách bớt 1 tại cột 1 và bớt 2 đi ở dòng 1 để nhận được ma trận cấp 3:

		2	4	5	
1		∞	0	28	
3		0	∞	0	
5		20	0	∞	

Ta có cận dưới của nhánh tương ứng là 81 + 1 + 2 = 84. Cây tìm kiếm cho đến bước này được thể hiện trong hình 4.6.

Chú ý rằng, sau khi đã chấp nhận n-2 cạnh vào hành trình thì ma trận còn lại sẽ có kích thước là 2 × 2. Hai cạnh còn lại của hành trình sẽ không phải chọn lựa nữa mà được kết nạp ngay vào chu trình (vì nó chỉ còn sự lựa chọn duy nhất). Trong ví dụ trên sau khi đã có các cạnh (6, 3), (4,6), (2, 1), (1,4) ma trận của nhánh bên trái nhất có dạng:

	2	5
3	∞	0
5	0	∞

Vì vậy ta kết nạp nốt cạnh (3, 5), (5, 2) vào chu trình và thu được hành trình:

1, 4, 6, 3, 5, 2, 1 với chi phí là 104.

Trong quá trình tìm kiếm, mỗi node của cây tìm kiếm sẽ tương ứng với một ma trận chi phí A. Ở bước đầu tiên ma trận chi phí tương ứng với gốc chính là ma trận C. Khi chuyển động từ gốc xuống nhánh bên trái xuống phĩa dưới, kích thước của các ma trận chi phí A sẽ giảm dần. Cuối cùng khi ma trận A có kích thước 2× 2 thì ta chấm dứt việc phân nhánh và kết nạp hai cạnh còn lại để thu được hành trình của người du lịch. Dễ dàng nhận thấy ma trận cuối cùng rút gọn chỉ có thể ở một trong hai dạng sau:

	w	x
u	∞	0
v	0	∞

	w	x
u	0	∞
v	∞	0

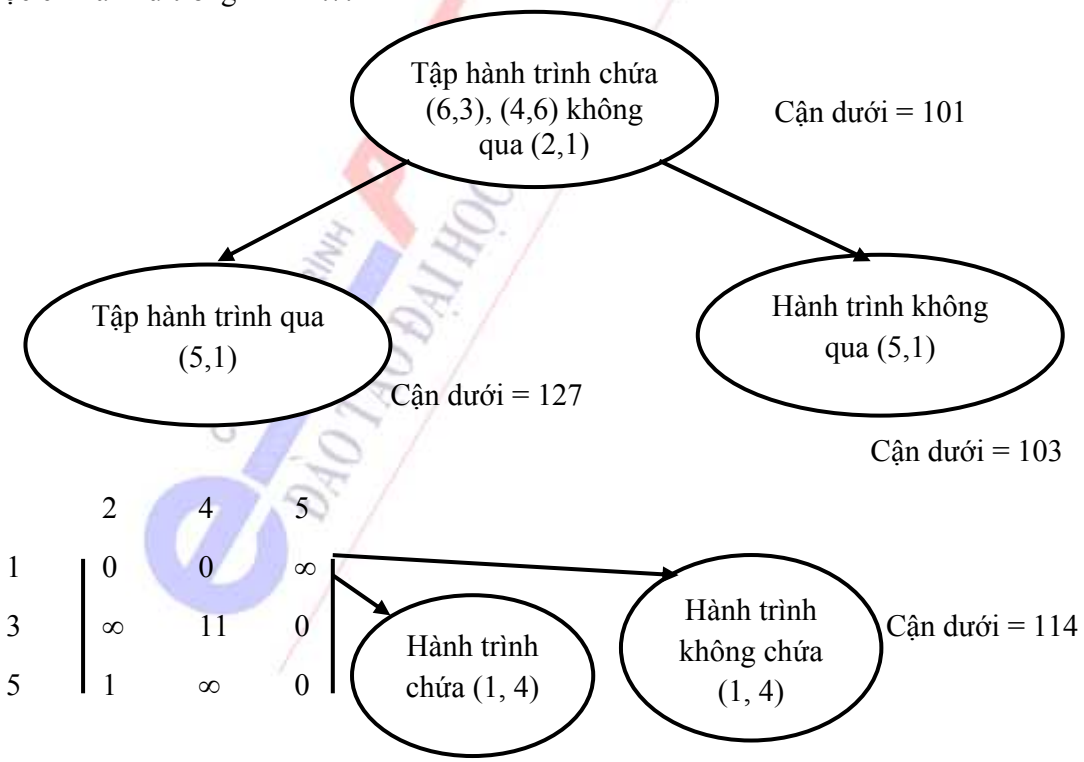
Trong đó u, v, x, y có thể là 4 đỉnh khác nhau hoặc 3 đỉnh khác nhau. Để xác định xem hai cạnh nào cần được nạp vào hành trình ta chỉ cần xét một phần tử của ma trận A:

```
if A[1, 1] = ∞ then
        <Kết nạp cạnh (u, x), (v, w)>
else
        < Kết nạp cạnh (u, w), ( v, x) >;
```

Bây giờ tất cả các node có cận dưới lớn hơn 104 có thể bị loại bỏ vì chúng không chứa hành trình rẻ hơn 104. Trên hình 4.6 chúng ta thấy chỉ có node có cận dưới là 101 < 104 là cần phải xét tiếp. Node này chứa các cạnh (6, 3), (4, 6) và không chứa cạnh (2, 1). Ma trận chi phí tương ứng với đỉnh này có dạng:

	1	2	4	5
1	∞	0	2	30
2	∞	∞	13	0
3	26	1	∞	0
5	0	21	0	∞

Việc phân nhánh sẽ dựa vào cạnh (5, 1) với tổng số rút gọn là 26. Quá trình rẽ nhánh tiếp theo được chỉ ra như trong hình 4.7.



	2	4	5
1	0	0	∞
3	∞	11	0
5	1	∞	0

Hình 4.7. Duyệt hành trình có cận dưới là 101.

Hành trình 1, 4, 6, 3, 2, 5, 1 ; Độ dài 104.

Như vậy chúng ta thu được hai hành trình tối ưu với chi phí là 104. Ví dụ trên cho thấy bài toán người du lịch có thể có nhiều phương án tối ưu. Trong ví dụ này hành trình đầu tiên nhận được đã là tối ưu, tuy nhiên điều này không thể mong đợi đối với những trường hợp tổng quát. Trong ví dụ trên chúng ta chỉ cần xét tới 13 node, trong khi tổng số hành trình của người du lịch là 120.

4.4.3.Thuật toán nhánh cận giải bài toán người du lịch

Các bước chính của thuật toán nhánh cận giải bài toán người du lịch được thể hiện trong thủ tục TSP. Thủ tục TSP xét hành trình bộ phận với Edges là cạnh đã được chọn và tiến hành tìm kiếm tiếp theo. Các biến được sử dụng trong thủ tục này là:

Edges - Số cạnh trong hành trình bộ phận;

A - Ma trận chi phí tương ứng với kích thước (n-edges, n-edges)

cost - Chi phí của hành trình bộ phận.

Mincost - Chi phí của hành trình tốt nhất đã tìm được.

Hàm Reduce(A, k), BestEgde(A, k, r, c,beta) đã được xây dựng ở trên.

```
void TSP( Edges, cost, A) {

    cost=cost + Reduce(A, n-Edges);
    if (cost <MinCost){
        if (edges == n-2){
            <bổ xung nốt hai cạnh còn lại>;
            MinCost:=Cost;
        }
        else {
            BestEdge(A, n-eges, r, c, beta);
            LowerBound = Cost + beta;
            <Ngăn cấm tạo thành hành trình con>;
            NewA = < A loại bỏ dòng r cột c>;
            TSP(edges+1, cost, NewA);/*đi theo nhánh trái*/
            <Khôi phục A bằng cách bổ xung dòng r cột c>;
            if (LowerBound < MinCost){
                /* đi theo nhánh phải*/
                A[r, c] =∞;
```

```
                                        TSP (edges, cost, A);
                                        A[r,c]:=0;
                                    }
                                }
                                < Khôi phục ma trận A>;/* thêm lại các hằng số rút gọn vào
                                                        các dòng và cột tương ứng*/
                            }
                        }/* end of TSP*/;
```

NHỮNG NỘI DUNG CẦN GHI NHỚ

Bạn đọc cần ghi nhớ một số nội dung quan trọng dưới đây:

- ✓ Thế nào là một bài toán tối ưu? Ý nghĩa của bài toán tối ưu trong các mô hình thực tế.
- ✓ Phân tích ưu điểm, nhược điểm của phương pháp liệt kê.
- ✓ Hiểu phương pháp nhánh cận, phương pháp xây dựng cận và những vấn đề liên quan.
- ✓ Hiểu phương pháp rút gọn ma trận trong giải quyết bài toán người du lịch.

BÀI TẬP CHƯƠNG 4

Bài 1. Giải bài toán cái túi sau:

$$\begin{cases} 5x_1 + x_2 + 9x_3 + 3x_4 \to \max, \\ 4x_1 + 2x_2 + 7x_3 + 3x_4 \le 10, \\ x_j \ge 0 \quad nguyên, j = 1,2,3,4. \end{cases}$$

Bài 2. Giải bài toán cái túi sau:

$$\begin{cases} 7x_1 + 3x_2 + 2x_3 + x_4 \to \max, \\ 5x_1 + 3x_2 + 6x_3 + 4x_4 \le 12, \\ x_j \ge 0, \quad nguyên, j = 1,2,3,4 \end{cases}$$

Bài 3. Giải bài toán cái túi sau:

$$\begin{cases} 5x_1 + x_2 + 9x_3 + 3x_4 \to \max, \\ 4x_1 + 2x_2 + 7x_3 + 3x_4 \le 10, \\ x_j \in \{0,1\}, j = 1,2,3,4. \end{cases}$$

Bài 4. Giải bài toán cái túi sau:

$$\begin{cases} 7x_1 + 3x_2 + 2x_3 + x_4 \to \max, \\ 5x_1 + 3x_2 + 6x_3 + 4x_4 \le 12, \\ x_j \in \{0,1\}, j = 1,2,3,4 \end{cases}$$

Bài 5. Giải bài toán cái túi sau:

$$\begin{cases} 30x_1 + 19x_2 + 13x_3 + 38x_4 + 20x_5 + 6x_6 + 8x_7 + 19x_8 + 10x_9 + 11x_{10} \to \max, \\ 15x_1 + 12x_2 + 9x_3 + 27x_4 + 15x_5 + 5x_6 + 8x_7 + 20x_8 + 12x_9 + 15x_{10} \le 62 \\ x_j \in \{0,1\}, j = 1,2\cdots,10. \end{cases}$$

Bài 6. Áp dụng thuật toán nhánh cận giải bài toán người du lịch với ma trận chi phí sau:

$$\begin{vmatrix} 00 & 08 & 05 & 22 & 11 \\ 04 & 00 & 09 & 17 & 27 \\ 15 & 07 & 00 & 12 & 35 \\ 05 & 27 & 17 & 00 & 29 \\ 23 & 21 & 19 & 07 & 00 \end{vmatrix}$$

Bài 7. Áp dụng thuật toán nhánh cận giải bài toán người du lịch với ma trận chi phí sau:

$$\begin{vmatrix} 00 & 05 & 37 & 21 & 29 \\ 42 & 00 & 31 & 07 & 33 \\ 31 & 27 & 00 & 31 & 08 \\ 49 & 33 & 14 & 00 & 39 \\ 06 & 41 & 32 & 38 & 00 \end{vmatrix}$$

Bài 6. Áp dụng thuật toán nhánh cận giải bài toán người du lịch với ma trận chi phí sau:

$$\begin{vmatrix} 00 & 08 & 05 & 22 & 11 \\ 04 & 00 & 09 & 17 & 27 \\ 15 & 07 & 00 & 12 & 35 \\ 05 & 27 & 17 & 00 & 29 \\ 23 & 21 & 19 & 07 & 00 \end{vmatrix}$$

Bài 8. Giải bài toán người du lịch với ma trận chi phí như sau:

$$\begin{vmatrix} \infty & 31 & 15 & 23 & 10 & 17 \\ 16 & \infty & 24 & 07 & 12 & 12 \\ 34 & 03 & \infty & 25 & 54 & 25 \\ 15 & 20 & 33 & \infty & 50 & 40 \\ 16 & 10 & 32 & 03 & \infty & 23 \\ 18 & 20 & 13 & 28 & 21 & \infty \end{vmatrix}$$

Bài 9. Giải bài toán người du lịch với ma trận chi phí như sau:

$$\begin{vmatrix} \infty & 03 & 93 & 13 & 33 & 09 \\ 04 & \infty & 77 & 42 & 21 & 16 \\ 45 & 17 & \infty & 36 & 16 & 28 \\ 39 & 90 & 80 & \infty & 56 & 07 \\ 28 & 46 & 88 & 33 & \infty & 25 \\ 03 & 88 & 18 & 46 & 92 & \infty \end{vmatrix}$$