



ĐẠI HỌC BÁCH KHOA HÀ NỘI  
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

# TỐI ƯU LẬP KẾ HOẠCH

## Quy hoạch tuyển sinh

# Nội dung

---

- Bài toán quy hoạch tuyến tính
- Bài toán quy hoạch nguyên tuyến tính
- Sơ đồ thuật toán Branch-and-Cut
- Ví dụ minh họa
- Bài toán phân công giảng dạy
- Bài toán TSP

# Quy hoạch tuyến tính (QHTT)

$$f(x) = c_1x_1 + c_2x_2 + \dots + c_nx_n \rightarrow \min(\max)$$

$$a_{1,1}x_1 + a_{1,2}x_2 + \dots + a_{1,n}x_n \leq b_1$$

$$a_{2,1}x_1 + a_{2,2}x_2 + \dots + a_{2,n}x_n \leq b_2$$

...

$$a_{m,1}x_1 + a_{m,2}x_2 + \dots + a_{m,n}x_n \leq b_m$$

$x_1, x_2, \dots, x_n$  là các biến thực

# Bài toán QHTT dạng chính tắc

$$f(x) = c_1x_1 + c_2x_2 + \dots + c_nx_n \rightarrow \min$$

$$a_{1,1}x_1 + a_{1,2}x_2 + \dots + a_{1,n}x_n = b_1$$

$$a_{2,1}x_1 + a_{2,2}x_2 + \dots + a_{2,n}x_n = b_2$$

...

$$a_{m,1}x_1 + a_{m,2}x_2 + \dots + a_{m,n}x_n = b_m$$

$x_1, x_2, \dots, x_n$  là các biến thực không âm

- Bài toán QHTT tổng quát luôn có thể chuyển về bài toán QHTT dạng chính tắc bằng các biến đổi dấu và đặt thêm biến phụ:
  - Với ràng buộc  $a \leq b$  thì thêm biến phụ  $y$ :  $a + y = b$
  - Biến  $x_i$  không có ràng buộc về dấu sẽ được thay bằng hiệu của 2 biến không âm:  $x_i = x_i^+ - x_i^-$ ,  $x_i^+, x_i^- \geq 0$
  - Ràng buộc  $a \geq b$  sẽ được thay bằng ràng buộc  $-a \leq -b$

# Quy hoạch nguyên tuyến tính

## Integer Linear Program (ILP)

$$f(x) = c_1x_1 + c_2x_2 + \dots + c_nx_n \rightarrow \min(\max)$$

$$a_{1,1}x_1 + a_{1,2}x_2 + \dots + a_{1,n}x_n \leq b_1$$

$$a_{2,1}x_1 + a_{2,2}x_2 + \dots + a_{2,n}x_n \leq b_2$$

...

$$a_{m,1}x_1 + a_{m,2}x_2 + \dots + a_{m,n}x_n \leq b_m$$

$x_1, x_2, \dots, x_n$  là các biến nguyên

# Quy hoạch hỗn hợp (tuyến tính)

## Mixed Integer Program (MIP)

$$f(x) = c_1x_1 + c_2x_2 + \dots + c_nx_n \rightarrow \min(\max)$$

$$a_{1,1}x_1 + a_{1,2}x_2 + \dots + a_{1,n}x_n \leq b_1$$

$$a_{2,1}x_1 + a_{2,2}x_2 + \dots + a_{2,n}x_n \leq b_2$$

...

$$a_{m,1}x_1 + a_{m,2}x_2 + \dots + a_{m,n}x_n \leq b_m$$

$x_1, x_2, \dots, x_j$  là các biến nguyên

$x_{j+1}, x_{j+2}, \dots, x_n$  là các biến thực

# Thuật toán đơn hình

Bài toán quy hoạch tuyến tính (QHTT) dạng chính tắc có dạng:

$$f(x) = c_1x_1 + c_2x_2 + \dots + c_nx_n \rightarrow \min$$

$$a_{1,1}x_1 + a_{1,2}x_2 + \dots + a_{1,n}x_n = b_1$$

$$a_{2,1}x_1 + a_{2,2}x_2 + \dots + a_{2,n}x_n = b_2$$

...

$$a_{m,1}x_1 + a_{m,2}x_2 + \dots + a_{m,n}x_n = b_m$$

$x_1, x_2, \dots, x_n$  là các biến không âm

# Thuật toán đơn hình

- Thuật toán đơn hình (Simplex Method, Dantzig, 1947)
- Dịch chuyển từ 1 phương án sang một phương án mới tốt hơn cho đến khi đạt được phương án tối ưu
- Thuật toán là hữu hạn nhưng độ phức tạp hàm mũ
- Các ký hiệu
  - $x = (x_1, x_2, \dots, x_n)^T$  – vector biến số
  - $c = (c_1, c_2, \dots, c_n)^T$  – vector hệ số hàm mục tiêu
  - $A = (a_{i,j})_{m \times n}$  – ma trận ràng buộc
  - $b = (b_1, b_2, \dots, b_m)^T$  – vector ràng buộc (về phải)

# Thuật toán đơn hình

- Bài toán viết lại dưới dạng

$$f(x) = c^T x \rightarrow \min$$

$$Ax = b, x \geq 0$$

hay

$$\min\{f(x) = c^T x: Ax = b, x \geq 0\}$$

# Thuật toán đơn hình

- **Ký hiệu các tập chỉ số:**

- $J = \{1, 2, \dots, n\}$  - tập chỉ số của các biến số
- $I = \{1, 2, \dots, m\}$  - tập chỉ số của các ràng buộc

- **Khi đó ta sử dụng các ký hiệu sau**

$x = x(J) = \{x_j; j \in J\}$  – vector biến số;

$c = c(J) = \{c_j; j \in J\}$  – vector hệ số hàm mục tiêu;

$A = A(I, J) = \{a_{i,j}; i \in I, j \in J\}$  - ma trận ràng buộc

$A_j = (a_{i,j}; i \in I)$  – vector cột thứ  $j$  của ma trận  $A$ .

- **Hệ phương trình ràng buộc của bài toán QHTT dạng chính tắc còn có thể viết dưới dạng:**

$$A_1x_1 + A_2x_2 + \dots + A_nx_n = b$$

# Thuật toán đơn hình

- Tập  $D = \{x: Ax = b, x \geq 0\}$  được gọi là miền ràng buộc (hay miền chấp nhận được)
- Mỗi phương án  $x \in D$  được gọi là *phương án chấp nhận được*
- Phương án chấp nhận được  $x^*$  tại đó giá trị hàm  $f$  đạt nhỏ nhất:

$$f(x^*) \leq f(x) \text{ với mọi } x \in D$$

được gọi là phương án tối ưu của bài toán.

Khi đó giá trị  $f^* = c^T x^*$  được gọi là giá trị tối ưu của bài toán

# Thuật toán đơn hình

- **Định nghĩa 1.** Ta gọi cơ sở của ma trận  $A$  là một bộ gồm  $m$  vector cột độc lập tuyến tính của nó.
- Giả sử  $B = A(I, J_B)$  trong đó  $J_B = \{j_1, \dots, j_m\}$  là một cơ sở của ma trận  $A$ . Khi đó vector  $x = (x_1, \dots, x_n)$  thoả mãn:
  - $x(J_B) = B^{-1}b$
  - $x_j = 0, j \in J_N = J \setminus J_B$sẽ được gọi là *phương án cơ sở* (pacs) tương ứng với cơ sở  $B$ .
- Các biến  $x_j, j \in J_B$  được gọi là *biến cơ sở*
- Các biến  $x_j, j \in J_N$  được gọi là *biến phi cơ sở*.

# Thuật toán đơn hình

- Ký hiệu  $x_B = x(J_B)$ ,  $x_N = x(J_N)$
- Phương án cơ sở  $x$  tương ứng với cơ sở  $B$  có thể xác định nhờ thủ tục sau:
  - Đặt  $x_N = 0$ .
  - Xác định  $x_B$  từ hệ phương trình  $Bx_B = b$ .
- Bài toán luôn có phương án cơ sở nếu  $\text{rank}(A) = m$ .

# Thuật toán đơn hình

- Giả sử  $x = (x_B, x_N)$  là phương án cơ sở tương ứng với cơ sở  $B$ . Khi đó bài toán QHTT dạng chính tắc có thể viết lại như sau:

$$f(x_B, x_N) = c_B x_B + c_N x_N \rightarrow \min$$

$$Bx_B + Nx_N = b,$$

$$x_B, x_N \geq 0$$

trong đó  $N = (A_j : j \in J_N)$  được gọi là ma trận phi cơ sở.

# Thuật toán đơn hình

Xét bài toán QHTT

$$6x_1 + 2x_2 - 5x_3 + x_4 + 4x_5 - 3x_6 + 12x_7 \rightarrow \min$$

$$x_1 + x_2 + x_3 + x_4 = 4$$

$$x_1 + x_5 = 2$$

$$x_3 + x_6 = 3$$

$$3x_2 + x_3 + x_7 = 6$$

$$x_1, x_2, x_3, x_4, x_5, x_6, x_7 \geq 0$$

# Thuật toán đơn hình

- Dưới dạng ma trận:

$$c = (6, 2, -5, 1, 4, -3, 12)^T;$$

$$b = (4, 2, 3, 6)^T;$$

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 3 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$A_1 \ A_2 \ A_3 \ A_4 \ A_5 \ A_6 \ A_7$

# Thuật toán đơn hình

- Xét cơ sở

$$B = \{A_4, A_5, A_6, A_7\} = E_4$$

- Phương án cơ sở  $x = (x_1, x_2, \dots, x_7)$  được xác định như sau:

- Biến phi cơ sở:  $x_1 = 0, x_2 = 0, x_3 = 0$

- Biến cơ sở:  $x_B = (x_4, x_5, x_6, x_7) = (4, 2, 3, 6)$  (giải hệ phương trình  $E_4 x_B = b$ ).

- Vậy pacs tương ứng với cơ sở  $B$  là

$$x = (0, 0, 0, 4, 2, 3, 6)$$

# Thuật toán đơn hình

- **Định nghĩa.** Phương án cơ sở được gọi là phương án cơ sở chấp nhận được (pacscnd) nếu như nó là phương án chấp nhận được
- Một bài toán QHTT có hữu hạn pacscnd:  
 $\text{số pacscnd} \leq \text{số cơ sở} \leq C(m,n)$

# Thuật toán đơn hình

- Công thức số gia hàm mục tiêu
- Giả sử  $x$  là pacscnd với cơ sở tương ứng là  $B = (A_j; j \in J_B)$ . Ký hiệu:

$J_B = \{j_1, j_2, \dots, j_m\}$  – tập chỉ số biến cơ sở;

$J_N = J \setminus J_B$  – tập chỉ số biến phi cơ sở;

$B = (A_j; j \in J_B)$  – ma trận cơ sở;

$N = (A_j; j \in J_N)$  – ma trận phi cơ sở;

$x_B = x(J_B) = \{x_j; j \in J_B\}$  – vector biến cơ sở,

$x_N = x(J_N) = \{x_j; j \in J_N\}$  – vector biến phi cơ sở;

$c_B = c(J_B) = \{c_j; j \in J_B\}$  – vector hệ số hàm mục tiêu của biến cơ sở,

$c_N = c(J_N) = \{c_j; j \in J_N\}$  – vector hệ số hàm mục tiêu của biến phi cơ sở;

# Thuật toán đơn hình

- Công thức số gia hàm mục tiêu
- Xét pacnd  $z = x + \Delta x$ , trong đó  $\Delta x = (\Delta x_1, \Delta x_2, \dots, \Delta x_n)$  – vector gia số của biến số. Ta tìm công thức tính số gia của hàm mục tiêu:

$$\Delta f = c^T z - c^T x = c^T \Delta x.$$

- Do  $x, z$  đều là pacnd nên  $Ax = b$  và  $Az = b$ . Vì vậy gia số  $\Delta x$  phải thoả mãn điều kiện  $A\Delta x = 0$ , hay là:

$$B\Delta x_B + N\Delta x_N = 0,$$

trong đó  $\Delta x_B = (\Delta x_j : j \in J_B)$ ,  $\Delta x_N = (\Delta x_j : j \in J_N)$ .

# Thuật toán đơn hình

- Suy ra  $\Delta x_B = -B^{-1}N\Delta x_N$ .
- Từ đó ta có  $\Delta f = c_B \Delta x_B + c_N \Delta x_N = -(c_B B^{-1}N - c_N) \Delta x_N$ .
- Ký hiệu:

$u = c_B B^{-1}$  – vector thế vị

$\Delta_N = (\Delta_j; j \in J_N) = uN - c_N$  – vectơ ước lượng.

ta thu được công thức:

$$\Delta f = c^T z - c^T x = -\Delta_N \Delta x_N = -\sum_{j \in J_N} \Delta_j \Delta x_j$$

- Công thức thu được gọi là **công thức số gia hàm mục tiêu**

# Thuật toán đơn hình

- **Định nghĩa.** Pacscnd  $x$  được gọi là *không thoái hoá* nếu như tất cả các thành phần cơ sở của nó là khác không. Bài toán QHTT được gọi là không thoái hoá nếu như tất cả các pacscnd của nó là không thoái hoá
- **Định lý 2.** (Tiêu chuẩn tối ưu) Bất đẳng thức

$$\Delta_N \leq 0 \quad (\Delta_j \leq 0, j \in J_N)$$

là điều kiện đủ và trong trường hợp không thoái hoá cũng là điều kiện cần để pacscnd  $x$  là tối ưu.

# Thuật toán đơn hình

- Điều kiện đủ.
  - $x$  là ptcsnd nên  $x_N = 0$ .
  - Khi đó với mọi phương án chấp nhận được  $z = x + \Delta x$ , ta có  $\Delta x_N = z_N - x_N = z_N \geq 0$
  - Từ công thức số gia hàm mục tiêu  $\Delta f = f(z) - f(x) = -\Delta_N \Delta x_N \geq 0 \rightarrow x$  là phương án tối ưu

# Thuật toán đơn hình

- **Điều kiện cần.** Giả sử  $x$  là pacscnđ không thoái hóa tối ưu. Khi đó  $x_B > 0$ .
- Giả sử bất đẳng thức trong điều kiện của định lý không được thực hiện. Khi đó tìm được chỉ số  $j_0 \in J_N$  sao cho

$$\Delta_{j_0} > 0$$

- Xây dựng vector  $z = x + \Delta x$ , trong đó  $\Delta x$  được xác định như sau:
  - $\Delta x_{j_0} = \theta \geq 0, \Delta x_j = 0, j \neq j_0, j \in J_N,$
  - Số giá của các biến cơ sở được xác định bởi công thức

$$\Delta x_B = -B^{-1}N\Delta x_N = -\theta B^{-1}A_{j_0}$$

# Thuật toán đơn hình

- Rõ ràng khi đó vector z thỏa mãn ràng buộc cơ bản của bài toán  $Az = A(x + \Delta x) = b$
- Ngoài ra,  $z_N = z(J_N) = x_N + \Delta x_N = \Delta x_N \geq 0$ , với mọi  $\theta \geq 0$
- $z_B = z(J_B) = x_B + \Delta x_B = x_B - \theta B^{-1} A_{j_0}$
- Do  $x_B > 0$ , nên với  $\theta \geq 0$  đủ nhỏ ta có  $z_B > 0$  và khi đó ta có z là pacnd
- Mặt khác theo công thức số gia hàm mục tiêu,  
$$\Delta f = c^T z - c^T x = -\Delta_N \Delta x_N = -\sum_{j \in J_N} \Delta_j \Delta x_j = -\theta \Delta_{j_0} < 0$$
- Điều này mâu thuẫn với giả thiết x là phương án tối ưu.

# Thuật toán đơn hình

- Giả sử với pacscnd x, tiêu chuẩn tối ưu không được thực hiện, tức là tìm được chỉ số  $j_0 \in J_N$  sao cho  $\Delta_{j_0} > 0$ .
- Xét trường hợp khi  $B^{-1}A_{j_0} \leq 0$ ,  $x_B - \theta B^{-1}A_{j_0} \geq 0$ ,  $\forall \theta > 0$ . Đồng thời  $\Delta f = -\theta \Delta_{j_0} < 0$ :  $\theta$  càng lớn thì  $\Delta f$  càng nhỏ. Khi này, bài toán có hàm mục tiêu không bị chặn dưới (không tìm được phương án tối ưu)

# Thuật toán đơn hình

$c_j$ cơ sở	Cơ sở	Phương án	$c_1$	...	$c_j$	...	$c_n$	$\theta$
			$A_1$	...	$A_j$	...	$A_n$	
$c_{j_1}$	$A_{j_1}$	$x_{j_1}$			$x_{j_1 j}$			$\theta_{j_1}$
...	...	...			...			...
$c_i$	$A_i$	$x_i$			$x_{i j}$			$\theta_i$
...	...	...			...			...
$c_{j_m}$	$A_{j_m}$	$x_{j_m}$			$x_{j_m j}$			$\theta_{j_m}$
$\Delta$			$\Delta_1$	...	$\Delta_j$	...	$\Delta_n$	

# Thuật toán đơn hình

- Cột đầu tiên ghi hệ số hàm mục tiêu của các biến cơ sở.
- Cột thứ hai dành để ghi tên của các cột cơ sở.
- Cột thứ ba ghi các giá trị của các biến cơ sở (các thành phần của vectơ  $x_B = \{x_j : j \in J_B\} = B^{-1}b$ ).
- Các phần tử  $x_{i,j}$ ,  $i \in J_B$  trong các cột tiếp theo được tính theo công thức:  $\{x_{i,j} : i \in J_B\} = B^{-1}A_j$ ,  $j = 1, 2, \dots, n$ .
- Cột cuối cùng để ghi các tỷ số  $\theta_i$ :

- $$\theta_i = \begin{cases} x_i / x_{i,j_0}, & \text{nếu } x_{i,j_0} > 0, \\ +\infty, & \text{nếu } x_{i,j_0} \leq 0 \text{ và } i \notin J_B. \end{cases}$$
- Dòng đầu tiên của bảng ghi hệ số hàm mục tiêu của các biến ( $i \in J_B$ ).
  - Dòng tiếp theo ghi tên của các cột  $A_1, \dots, A_n$ .
  - Dòng cuối cùng gọi là dòng ước lượng:

$$\Delta_j = \sum_{i \in J_B} c_i x_{i,j} - c_j, \quad j = 1, 2, \dots, n.$$

- Có thể thấy rằng  $\Delta_j = 0$ ,  $j \in J_B$ .

# Thuật toán đơn hình

- Với bảng đơn hình xây dựng được ta có thể tiến hành thực hiện một bước lặp đơn hình đối với phương án cơ sở chấp nhận được x như sau.
- *Kiểm tra tiêu chuẩn tối ưu:* Nếu các phần tử của dòng ước lượng là không dương ( $\Delta_j \leq 0, j=1,2,\dots,n$ ) thì phương án cơ sở chấp nhận được đang xét là tối ưu, thuật toán kết thúc.
- *Kiểm tra điều kiện đủ để hàm mục tiêu không bị chặn dưới:* Nếu có ước lượng  $\Delta_{j_0} > 0$  mà các phần tử trong bảng đơn hình trên cột ứng với nó đều không dương ( $x_{j,j_0} \leq 0, j \in J_B$ ) , thì hàm mục tiêu của bài toán là không bị chặn dưới, thuật toán kết thúc.
- *Tìm cột xoay:* Tìm  $\Delta_{j_0} = \max \{\Delta_j : j = 1,2,\dots,n\} > 0$ .  
Cột  $A_{j_0}$  gọi là cột xoay (cột đưa vào cơ sở), còn biến  $x_{j_0}$  gọi là biến đưa vào.

# Thuật toán đơn hình

$c_j$ cơ sở	Cơ sở	Phương án	$c_1$	...	$c_{j_0}$	...	$c_n$	$\theta$
			$A_1$	...	$A_{j_0}$	...	$A_n$	
$c_{j_1}$	$A_{j_1}$	$x_{j_1}$			$x_{j_1 j_0}$			$\theta_{j_1}$
...	...	...			...			...
$c_i$	$A_i$	$x_i$			$x_{i j_0}$			$\theta_i$
...	...	...			...			...
$c_{j_m}$	$A_{j_m}$	$x_{j_m}$			$x_{j_m j_0}$			$\theta_{j_m}$
$\Delta$			$\Delta_1$	...	$\Delta_{j_0}$	...	$\Delta_n$	

# Thuật toán đơn hình dạng bảng giải bài toán QHTT dạng chính tắc

- Tìm dòng xoay

$$\theta_i = \begin{cases} x_i/x_{i,j_0}, & \text{nếu } x_{i,j_0} > 0, \\ +\infty, & \text{nếu } x_{i,j_0} \leq 0, i \in J_B. \end{cases}$$
$$\theta_0 = \theta_{i_0} = x_{i_0}/x_{i_0,j_0} = \min\{\theta_i : i \in J_B\}.$$

- Dòng  $A_{i_0}$  gọi là dòng xoay ( $A_{i_0}$  - cột đưa ra cơ sở), còn biến  $x_{i_0}$  gọi là biến đưa ra. Phần tử nằm trên giao của dòng xoay và cột xoay của bảng đơn hình được gọi là *phần tử xoay*.
- Thực hiện phép biến đổi đơn hình chuyển từ pacscnd x sang pacscnd z: Bảng đơn hình tương ứng với z (gọi là bảng mới) có thể thu được từ bảng đơn hình tương ứng với x (gọi là bảng cũ) theo quy tắc biến đổi sau

# Thuật toán đơn hình

$c_j$ cơ sở	Cơ sở	Phương án	$c_1$	...	$c_{j_0}$	...	$c_n$	$\theta$
			$A_1$	...	$A_{j_0}$	...	$A_n$	
$c_{j_1}$	$A_{j_1}$	$x_{j_1}$			$X_{j_1, j_0}$			$\theta_{j_1}$
...	...	...			...			...
$c_{i_0}$	$A_{i_0}$	$x_{i_0}$			$x_{i_0, j_0}$			$\theta_{i_0}$
...	...	...			...			...
$c_{j_m}$	$A_{j_m}$	$x_{j_m}$			$x_{j_m, j_0}$			$\theta_{j_m}$
$\Delta$			$\Delta_1$	...	$\Delta_{j_0}$	...	$\Delta_n$	

# Thuật toán đơn hình

- Các phần tử ở vị trí dòng xoay trong bảng mới ( $\bar{x}_{i_0,j}$ ) bằng các phần tử tương ứng trong bảng cũ chia cho phần tử xoay:
- Các phần tử ở vị trí cột xoay trong bảng mới, ngoại trừ phần tử nằm trên vị trí phần tử xoay bằng 1, còn tất cả là bằng 0.
- Các phần tử cần tính còn lại trong bảng mới được tính từ các phần tử tương ứng trong bảng cũ theo công thức sau:

$$\bar{x}_{i_0} = x_{i_0}/x_{i_0,j_0}, \quad \bar{x}_{i_0j} = x_{i_0,j}/x_{i_0,j_0}, j \in J.$$

$$\bar{x}_{i,j} = x_{i,j} - x_{i_0,j}x_{i,j_0}/x_{i_0,j_0}, i \in J_B \ (i \neq i_0), j \in J \ (j \neq j_0),$$

$$\bar{\Delta}_j = \Delta_j - x_{i_0,j}\Delta_{j_0}/x_{i_0,j_0}, j \in J \ (j \neq j_0).$$

# Thuật toán đơn hình

- Giải bài toán QHTT sau đây:

$$x_1 - 6x_2 + 32x_3 + x_4 + x_5 + 10x_6 + 100x_7 \rightarrow \min$$

$$x_1 + x_4 + 6x_6 = 9$$

$$3x_1 + x_2 - 4x_3 + 2x_6 + x_7 = 2$$

$$x_1 + 2x_2 + x_5 + 2x_6 = 6$$

$$x_i \geq 0, i = 1, 2, \dots, 7$$

# Thuật toán đơn hình

$c_j$ cơ sở	Cơ sở	Phương án	1	-6	32	1	1	10	100	$\theta$
			$A_1$	$A_2$	$A_3$	$A_4$	$A_5$	$A_6$	$A_7$	
1	$A_4$	9	1	0	0	1	0	6	0	9
100	$A_7$	2	3	1	-4	0	0	2	1	2/3
1	$A_5$	6	1	2	0	0	1	2	0	6
$\Delta$			301	108	-432	0	0	198	0	

- Chọn cột xoay là cột có ước lượng lớn nhất

# Thuật toán đơn hình

$c_j$ cơ sở	Cơ sở	Phương án	1	-6	32	1	1	10	100	$\theta$
			$A_1$	$A_2$	$A_3$	$A_4$	$A_5$	$A_6$	$A_7$	
1	$A_4$	9	1	0	0	1	0	6	0	9
100	$A_7$	2	3	1	-4	0	0	2	1	2/3
1	$A_5$	6	1	2	0	0	1	2	0	6
$\Delta$			301	108	-432	0	0	198	0	

Chọn dòng xoay là dòng có tỉ số  $\theta_i$  nhỏ nhất

# Thuật toán đơn hình

$c_j$ cơ sở	Cơ sở	Phương án	1	-6	32	1	1	10	100	$\theta$
			$A_1$	$A_2$	$A_3$	$A_4$	$A_5$	$A_6$	$A_7$	
1		$A_4$								
1		$A_1$	2/3	1	1/3	-4/3	0	0	2/3	1/3
1		$A_5$								
$\Delta$										

- Biến đổi bảng: Các phần tử trên dòng xoay ở bảng mới = Các phần tử tương ứng trong bảng cũ chia cho phần tử xoay.

# Thuật toán đơn hình

$c_j$	Cơ sở	Phương án	1	-6	32	1	1	10	100	$\theta$
cơ sở			$A_1$	$A_2$	$A_3$	$A_4$	$A_5$	$A_6$	$A_7$	
1	$A_4$									
1	$A_1$	2/3	1	1/3	-4/3	0	0	2/3	1/3	
1	$A_5$									
$\Delta$										

- Biến đổi bảng: Các phần tử trên các cột cơ sở là các vector đơn vị

# Thuật toán đơn hình

$c_j$ cơ sở	Cơ sở	Phương án	1	-6	32	1	1	10	100	$\theta$
			$A_1$	$A_2$	$A_3$	$A_4$	$A_5$	$A_6$	$A_7$	
1	$A_4$		0			1	0			
1	$A_1$	2/3	1	1/3	-4/3	0	0	2/3	1/3	
1	$A_5$		0			0	1			
$\Delta$			0			0	0			

- Biến đổi bảng: Các phần tử trên các cột cơ sở là các vector đơn vị

# Thuật toán đơn hình

$c_j$ cơ sở	Cơ sở	Phương án	1	-6	32	1	1	10	100	$\theta$
			$A_1$	$A_2$	$A_3$	$A_4$	$A_5$	$A_6$	$A_7$	
1	$A_4$	25/3	0	-1/3	4/3	1	0	16/3	-1/3	
1	$A_1$	2/3	1	1/3	-4/3	0	0	2/3	1/3	
1	$A_5$	16/3	0	5/3	4/3	0	1	4/3	-1/3	
$\Delta$			0	23/3	-92/3	0	0	-8/3	-301/3	

- Biến đổi bảng: Các phần tử còn lại tính theo quy tắc hình chữ nhật

# Thuật toán đơn hình

$c_j$ cơ sở	Cơ sở	Phương án	1	-6	32	1	1	10	100	$\theta$
			$A_1$	$A_2$	$A_3$	$A_4$	$A_5$	$A_6$	$A_7$	
1	$A_4$	25/3	0	-1/3	4/3	1	0	16/3	-1/3	$\infty$
1	$A_1$	2/3	1	1/3	-4/3	0	0	2/3	1/3	2
1	$A_5$	16/3	0	5/3	4/3	0	1	4/3	-1/3	16/5
$\Delta$			0	23/3	-92/3	0	0	-8/3	-301/3	

- Biến đổi bảng: Các phần tử còn lại tính theo quy tắc hình chữ nhật

# Thuật toán đơn hình

$c_j$ cơ sở	Cơ sở	Phương án	1	-6	32	1	1	10	100	$\theta$
			$A_1$	$A_2$	$A_3$	$A_4$	$A_5$	$A_6$	$A_7$	
1	$A_4$	25/3	0	-1/3	4/3	1	0	16/3	-1/3	$\infty$
1	$A_1$	2/3	1	1/3	-4/3	0	0	2/3	1/3	2
1	$A_5$	16/3	0	5/3	4/3	0	1	4/3	-1/3	16/5
$\Delta$			0	23/3	-92/3	0	0	-8/3	-301/3	

- Biến đổi bảng: Các phần tử còn lại tính theo quy tắc hình chữ nhật

# Thuật toán đơn hình

$c_j$ cơ sở	Cơ sở	Phương án	1	-6	32	1	1	10	100	$\theta$
			$A_1$	$A_2$	$A_3$	$A_4$	$A_5$	$A_6$	$A_7$	
1	$A_4$	9	1	0	0	1	0	6	0	
-6	$A_2$	2	3	1	-4	0	0	2	1	
1	$A_5$	2	-5	0	8	0	1	-2	-2	
$\Delta$			-23	0	0	0	0	-18	-108	

- Tiêu chuẩn tối ưu được thỏa mãn, thuật toán kết thúc.

Phương án tối ưu  $x^* = (0,2,0,9,2,0,0)$ , giá trị tối ưu  $f^* = -1$

# Thuật toán đơn hình

- **Định nghĩa.** Thuật toán giải bài toán tối ưu hoá được gọi là hữu hạn nếu như nó cho phép sau một số lần hữu hạn phép tính tìm được phương án tối ưu của bài toán.
- Do mỗi bước lặp của thuật toán đơn hình có thể thực hiện xong sau một số hữu hạn phép tính, để chứng minh tính hữu hạn của thuật toán đơn hình ta sẽ chứng minh rằng nó phải kết thúc sau hữu hạn bước lặp.
- **Định nghĩa.** Bài toán QHTT được gọi là không thoái hoá nếu như tất cả các phương án cơ sở chấp nhận được của nó là không thoái hoá, trong trường hợp ngược lại bài toán được gọi là thoái hoá.
- **Định lý 1.4.** Giả sử bài toán QHTT là không thoái hoá và có phương án tối ưu. Khi đó với mọi phương án cơ sở chấp nhận được xuất phát thuật toán đơn hình là hữu hạn.

# Bài toán quy hoạch hỗn hợp tuyến tính

- Bài toán MIP

$$\max f(x,y) = kx + hy$$

$$Ax + Gy \leq b$$

$$(x, y) \in R_+^n \times Z_+^p$$

$$x = (x_1, \dots, x_n), y = (y_1, \dots, y_p)$$

- Ký hiệu

- $P$ : tập các ràng buộc tuyến tính trong  $Ax + Gy \leq b$  và các ràng buộc  $x, y \geq 0$
- Ta cũng dùng ký hiệu  $P$  để chỉ tập các bộ  $(x, y)$  thỏa mãn ràng buộc trong  $P$  (nếu không có nhầm lẫn)
- $D_1 = \{(x, y) : Ax + Gy \leq b; (x, y) \in R_+^n \times R_+^p\}$ ,
- $D_2 = \{(x, y) : (x, y) \in R_+^n \times Z_+^p\}$
- $z = (x, y)$ ,  $c = (k, h)$ ,  $X = \{(x, y) : Ax + Gy \leq b, (x, y) \in R_+^n \times Z_+^p\}$ , bài toán viết lại dưới dạng  $\max f(z) = \{cz : z \in X\}$

# LP-based Branch-and-Bound

- Bài toán xuất phát, ký hiệu MIPROB
  - $\max f(x,y) = kx + hy$
  - $(x,y) \in \{(x,y) : (x,y) \in P \text{ and } (x,y) \in R_+^n \times Z_+^p\}$ ,
- Nới lỏng ràng buộc nguyên, chuyển bài toán MIP về bài toán LP, ký hiệu LP(MIPROB)
  - $\max f(x,y) = kx + hy$
  - $(x,y) \in \{(x,y) : (x,y) \in P \text{ and } (x,y) \in R_+^n \times R_+^p\}$ ,

# LP-based Branch-and-Bound

- Giải bài toán LP(MIPROB)
  - Nếu vô nghiệm thì bài toán MIPROB đã cho vô nghiệm
  - Ngược lại, giả sử thu được phương án tối ưu  $(x_{LP}^*, y_{LP}^*)$ .
    - Nếu  $y_{LP}^*$  là nguyên thì  $(x_{LP}^*, y_{LP}^*)$  là phương án tối ưu của bài toán MIPROB đã cho
    - Ngược lại nếu tồn tại  $y_j^*$  thuộc  $y_{LP}^*$  không nguyên, khi đó thực hiện phân nhành, tạo ra 2 bài toán con
      - Bài toán con thứ nhất LP1: thêm ràng buộc  $y_j \geq \lceil y_j^* \rceil$  vào bài toán LP(MIPROB)
      - Bài toán con thứ hai LP2: thêm ràng buộc  $y_j \leq \lfloor y_j^* \rfloor$  vào bài toán LP(MIPROB)
      - Lặp lại việc giải LP1 và LP2, lời giải tối ưu với biến  $y$  là nguyên tốt nhất (nếu tồn tại) của LP1 và LP2 sẽ là lời giải tối ưu của bài toán MIPROB đặt ra

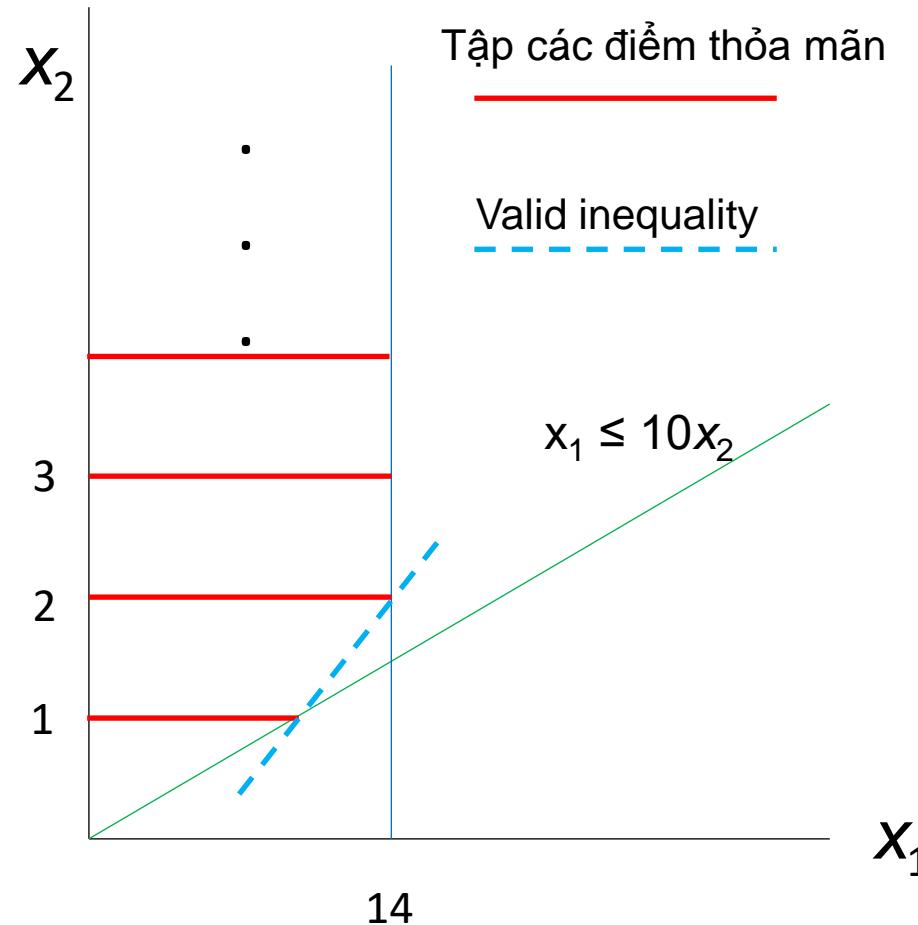
# Branch-and-Cut

---

- Mặt phẳng cắt
  - Bài toán MIP  $\max\{cz: z \in X\}$
  - Bất đẳng thức (BĐT) tuyến tính  $\pi z \leq \pi_0$  được gọi là 1 valid inequality nếu  $\pi z \leq \pi_0$  đúng với mọi  $z \in X$
  - Việc tìm ra các valid inequality giúp thu hẹp không gian lời giải, chuyển dần bài toán MIP về bài toán LP mà lời giải tối ưu của bài toán LP chính là lời giải tối ưu của bài toán MIP

# Branch-and-Cut

- Ví dụ, xét bài toán MIP với tập  $X = \{(x_1, x_2): x_1 \leq 10x_2, 0 \leq x_1 \leq 14, x_2 \in Z_+^1\}$
- Các đoạn đở biểu diễn tập  $X$
- Việc bổ sung thêm ràng buộc  $x_1 \leq 6 + 4x_2$  sẽ không làm mất phương án trong  $X$  do đó ràng buộc  $x_1 \leq 6 + 4x_2$  là một valid inequality (biểu diễn bởi đường nét đứt)



# Integer Rounding

- Xét vùng  $X = P \cap \mathbb{Z}^3$  trong đó  $P = \{x \in \mathbb{R}_+^3 : 5x_1 + 9x_2 + 13x_3 \geq 19\}$
- Từ  $5x_1 + 9x_2 + 13x_3 \geq 19$  ta có  $x_1 + \frac{9}{5}x_2 + \frac{13}{5}x_3 \geq \frac{19}{5}$   
 $\rightarrow x_1 + 2x_2 + 3x_3 \geq \frac{19}{5}$
- Vì  $x_1, x_2, x_3$  nguyên, suy ra

$$x_1 + 2x_2 + 3x_3 \geq \lceil \frac{19}{5} \rceil = 4$$

(đây chính là 1 inequality cho  $X$ )

# Gomory Cut

- (*IP*)  $\max \{cx: Ax = b, x \geq 0 \text{ and integer}\}$
- Giải bài toán nói lỏng (*LP*)  $\max \{cx: Ax = b, x \geq 0\}$
- Giả sử với phương án tối ưu cơ sở, bài toán *LP* được viết lại dưới dạng

$$\overline{a_{00}} + \sum_{j \in JN} \overline{a_{0j}}x_j \rightarrow \max$$

$$x_{B_u} + \sum_{j \in JN} \overline{a_{uj}}x_j = \overline{a_{u0}}, u = 1, 2, \dots, m$$

$$x \geq 0 \text{ and integer}$$

Trong đó  $\overline{a_{0j}} \leq 0$  (hệ số này tương ứng với lời giải tối ưu, maximizer), and  $\overline{a_{u0}} \geq 0$

# Gomory Cut

- Nếu phương án cơ sở tối ưu  $x^*$  không nguyên, khi đó tồn tại 1 hàng u với  $\overline{a_{u0}}$  không nguyên

→ Tạo ra một Gomory cut  $x_{B_u} + \sum_{j \in J_N} \lfloor \overline{a_{uj}} \rfloor x_j \leq \lfloor \overline{a_{u0}} \rfloor$  (1)

→ Viết lại dưới dạng (vì  $x_{B_u}$  nguyên)

$$\sum_{j \in J_N} (\overline{a_{uj}} - \lfloor \overline{a_{uj}} \rfloor) x_j \geq \overline{a_{u0}} - \lfloor \overline{a_{u0}} \rfloor$$

hoặc

$$\sum_{j \in J_N} f_{u,j} x_j \geq f_{u,0} \quad (2)$$

Với  $f_{u,j} = \overline{a_{uj}} - \lfloor \overline{a_{uj}} \rfloor$  and  $f_{u,0} = \overline{a_{u0}} - \lfloor \overline{a_{u0}} \rfloor$

- Vì  $0 \leq f_{u,j} < 1$  và  $0 < f_{u,0} < 1$  và  $x_j^* = 0, \forall j \in J_N \rightarrow (2)$  loại trừ  $x^*$ .

# Gomory Cut

- Sự khác nhau giữa vế trái (LHS) và vế phải (RHS) của (1) là số nguyên (do  $x$  nguyên)  $\rightarrow$  sự khác biệt giữa LHS và RHS của (2) cũng là số nguyên
- $\rightarrow$  viết lại (2) dưới dạng  $s = \sum_{j \in J_N} f_{u,j} x_j - f_{u,0}$  trong đó biến  $s$  là biến nguyên không âm

# Gomory Cut

Xét bài toán (IP)

$$\begin{aligned} f(x_1, x_2, x_3, x_4, x_5) = \quad & x_1 + x_2 \rightarrow \max \\ 2x_1 + x_2 + x_3 &= 8 \\ 3x_1 + 4x_2 + x_4 &= 24 \\ x_1 - x_2 + x_5 &= 2 \\ x_1, x_2, x_3, x_4, x_5 &\geq 0 \text{ and integer} \end{aligned}$$

# Gomory Cut

Giải bài toán nổi lỏng LP

$$\begin{aligned} f(x_1, x_2, x_3, x_4, x_5) = & \quad x_1 + x_2 \rightarrow \max \\ & 2x_1 + x_2 + x_3 = 8 \\ & 3x_1 + 4x_2 + x_4 = 24 \\ & x_1 - x_2 + x_5 = 2 \\ & x_1, x_2, x_3, x_4, x_5 \geq 0 \end{aligned}$$

→ Thu được lời giải tối ưu  $(1.6, 4.8, 0, 0, 5.2)$  với JB =  $(1,2,5)$  và JN =  $(3, 4)$

Viết lại bài toán IP ban đầu:

$$\begin{aligned} f(x_1, x_2, x_3, x_4, x_5) = & \quad 6.4 - 0.2x_3 - 0.2x_4 \rightarrow \max \\ & x_1 + 0.8x_3 - 0.2x_4 = 1.6 \\ & x_2 - 0.6x_3 + 0.4x_4 = 4.8 \\ & x_5 - 1.4x_3 + 0.6x_4 = 5.2 \\ & x_1, x_2, x_3, x_4, x_5 \geq 0 \text{ and integer} \end{aligned}$$

# Gomory Cut

- Consider  $x_1 + 0.8x_3 - 0.2x_4 = 1.6$   
→  $x_1 + 0x_3 - x_4 \leq \lfloor 1.6 \rfloor = 1$   
→ Add slack variable  $x_6$  and constraint (gomory cut):  $0.8x_3 + 0.8x_4 - x_6 = 0.6$
- Consider  $x_2 - 0.6x_3 + 0.4x_4 = 4.8$   
→  $x_2 - x_3 + 0.x_4 \leq \lfloor 4.8 \rfloor = 4$   
→ Add slack variable  $x_7$  and constraint (gomory cut):  $0.4x_3 + 0.4x_4 - x_7 = 0.8$
- Consider  $x_5 - 1.4x_3 + 0.6x_4 = 5.2$   
→  $x_5 - 2x_3 + 0x_4 \leq \lfloor 5.2 \rfloor = 5$   
→ Add slack variable  $x_8$  and constraint (gomory cut)  $0.6x_3 + 0.6x_4 - x_8 = 0.2$

# Gomory Cut

Thu được bài toán IP tương đương

$$f(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = x_1 + x_2 \rightarrow \max$$

$$2x_1 + x_2 + x_3 = 8$$

$$3x_1 + 4x_2 + x_4 = 24$$

$$x_1 - x_2 + x_5 = 2$$

$$0.8x_3 + 0.8x_4 - x_6 = 0.6$$

$$0.4x_3 + 0.4x_4 - x_7 = 0.8$$

$$0.6x_3 + 0.6x_4 - x_8 = 0.2$$

$$x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8 \geq 0 \text{ and integer}$$

# Gomory Cut

Giải bài toán nới lỏng LP tương ứng

$$f(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = x_1 + x_2 \rightarrow \max$$

$$2x_1 + x_2 + x_3 = 8$$

$$3x_1 + 4x_2 + x_4 = 24$$

$$x_1 - x_2 + x_5 = 2$$

$$0.8x_3 + 0.8x_4 - x_6 = 0.6$$

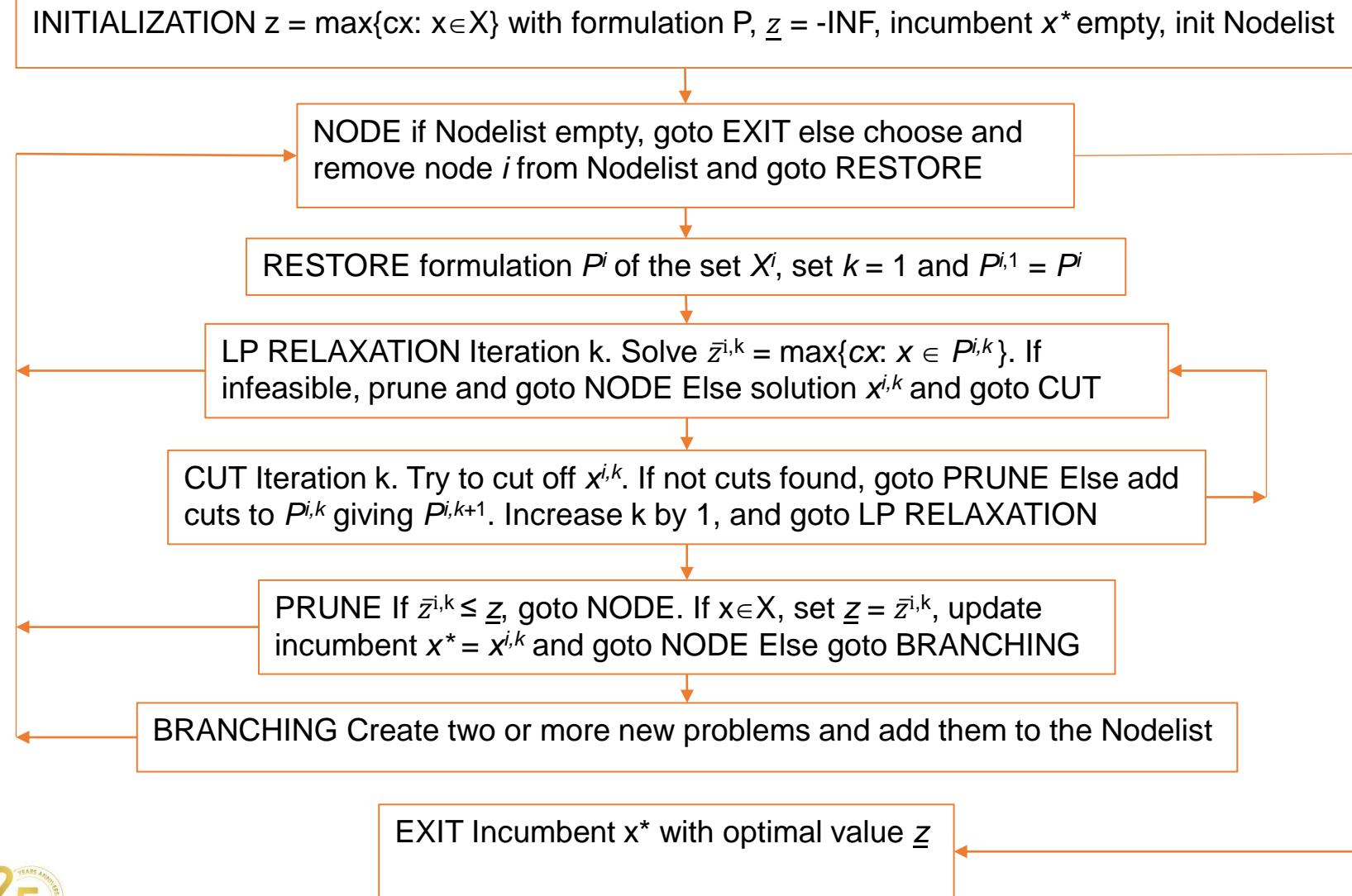
$$0.4x_3 + 0.4x_4 - x_7 = 0.8$$

$$0.6x_3 + 0.6x_4 - x_8 = 0.2$$

$$x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8 \geq 0$$

- Thu được lời giải tối ưu  $x^* = (0, 6, 2, 0, 8, 1, 0, 1)$  và là lời giải nguyên.
- Do đó  $(0,6,2,0,8)$  là phương án tối ưu của bài toán ban đầu với giá trị hàm mục tiêu tối ưu là  $0+6 = 6$

# Branch and Cut [Wolsey, 98]



# Ví dụ MIP solver

- Giải bài toán MIP sau

$$\max x_1 + x_2$$

thỏa mãn:

$$x_1 \leq 10x_2 + 7$$

$$2x_1 + 3x_2 \leq 20$$

$$0 \leq x_1 \leq 14, 0 \leq x_2 \leq 20, x_1 \in R, x_2 \in Z$$

# Ví dụ MIP solver

- Nới lỏng ràng buộc  $x_2$  nguyên  $\rightarrow$  giải bài toán LP sau

$$\max x_1 + x_2$$

thỏa mãn:

$$x_1 \leq 10x_2 + 7$$

$$2x_1 + 3x_2 \leq 20$$

$$0 \leq x_1 \leq 14, 0 \leq x_2 \leq 20, x_1 \in \mathbb{R}, x_2 \in \mathbb{R}$$

# Ví dụ MIP solver

- Nới lỏng ràng buộc  $x_2$  nguyên  $\rightarrow$  giải bài toán LP sau

$$\max x_1 + x_2$$

thỏa mãn:

$$x_1 \leq 10x_2 + 7$$

$$2x_1 + 3x_2 \leq 20$$

$$0 \leq x_1 \leq 14, 0 \leq x_2 \leq 20, x_1 \in \mathbb{R}, x_2 \in \mathbb{R}$$

$\rightarrow$  thu được nghiệm:

$$x_1 = 9.608695652173914,$$

$$x_2 = 0.26086956521739124$$

# Ví dụ MIP solver

- Nới lỏng ràng buộc  $x_2$  nguyên  $\rightarrow$  giải bài toán LP sau

$$\max x_1 + x_2$$

thỏa mãn:

$$x_1 \leq 10x_2 + 7$$

$$2x_1 + 3x_2 \leq 20$$

$$0 \leq x_1 \leq 14, 0 \leq x_2 \leq 20, x_1 \in \mathbb{R}, x_2 \in \mathbb{R}$$

$\rightarrow$  Phân nhánh bằng cách thêm 2 ràng buộc:

- $x_2 \leq 0$  (thu được bài toán LP1)
- $x_2 \geq 1$  (thu được bài toán LP2)

# Ví dụ MIP solver

- Giải bài toán LP1 sau

$$\max x_1 + x_2$$

thỏa mãn:

$$x_1 \leq 10x_2 + 7$$

$$2x_1 + 3x_2 \leq 20$$

$$x_2 \leq 0$$

$$0 \leq x_1 \leq 14, 0 \leq x_2 \leq 20, x_1 \in R, x_2 \in R$$

→ thu được nghiệm

$x_1 = 7, x_2 = 0$  (phương án chấp nhận được của bài toán MIP ban đầu), hàm mục tiêu  $f_1^* = 7$

# Ví dụ MIP solver

- Giải bài toán LP2 sau:

$$\max x_1 + x_2$$

thỏa mãn:

$$x_1 \leq 10x_2 + 7$$

$$2x_1 + 3x_2 \leq 20$$

$$x_2 \geq 1$$

$$0 \leq x_1 \leq 14, 0 \leq x_2 \leq 20, x_1 \in R, x_2 \in R$$

→ thu được nghiệm

$x_1 = 8.5, x_2 = 1$  (phương án chấp nhận được của bài toán MIP ban đầu), hàm mục tiêu  $f_2^* = 9.5$

# Ví dụ MIP solver

- Lời giải tối ưu của bài toán LP2 tốt hơn lời giải tối ưu của bài toán LP1 và 2 lời giải này đều là lời giải thỏa mãn ràng buộc của bài toán MIP ban đầu
- Lời giải tối ưu của bài toán MIP ban đầu chính là lời giải tối ưu của bài toán LP2, giá trị tối ưu hàm mục tiêu là  $f^* = 9.5$  và lời giải tối ưu là  $x^* = (8.5, 1)$

# Ví dụ MIP solver

```
public class ExampleMIPChap2 {  
    static{  
        System.LoadLibrary("jniortools");  
    }  
    public static void main(String[] args) {  
        double INF = java.lang.Double.POSITIVE_INFINITY;  
        MPSolver solver = new MPSolver("SimpleMIP",  
            MPSolver.OptimizationProblemType.CBC_MIXED_INTEGER_PROGRAMMING);  
        MPVariable x1 = solver.makeNumVar(0,14,"x1");  
        MPVariable x2 = solver.makeIntVar(0, 20, "x2");  
  
        MPConstraint c1 = solver.makeConstraint(-INF, 0);  
        c1.setCoefficient(x1, 1);  
        c1.setCoefficient(x2, -10);  
  
        MPConstraint c2 = solver.makeConstraint(0,20);  
        c2.setCoefficient(x1, 2);  
        c2.setCoefficient(x2, 3);  
    }  
}
```

# Ví dụ MIP solver

```
MPOjective obj = solver.objective();
obj.setCoefficient(x1, 1);
obj.setCoefficient(x2, 1);
obj.setMaximization();

MPSolver.ResultStatus rs = solver.solve();
if(rs != MPSolver.ResultStatus.OPTIMAL){
    System.out.println("Cannot find optimal solution");
} else{
    System.out.println("Objective value = " + obj.value());
    System.out.println("x1 = " + x1.solutionValue());
    System.out.println("x2 = " + x2.solutionValue());
}
}
```

# Bài toán phân công giảng dạy

- Có N lớp 0, 1, 2, ..., N-1 cần được phân cho M giáo viên 0, 1, 2, ..., M-1.
- Lớp i có số tín chỉ là  $c(i)$
- Mỗi giáo viên chỉ có thể dạy 1 số lớp nhất định tùy thuộc chuyên ngành của giáo viên và được thể hiện bởi cấu trúc  $tc(i,j)$  trong đó  $tc(i,j) = 1$  có nghĩa giáo viên i có thể được phân công giảng dạy lớp j và  $tc(i,j) = 0$  có nghĩa giáo viên i không thể được phân công giảng dạy lớp j, với  $i = 0, 1, \dots, N-1, j = 0, 1, \dots, M-1$
- N lớp đã được xếp thời khóa biểu từ trước và sẽ có tình trạng có 2 lớp được xếp trùng kíp thời khóa biểu (2 lớp này sẽ không thể phân cho cùng 1 giáo viên) và được thể hiện bởi cấu trúc  $f(i,j)$  trong đó  $f(i,j) = 1$  có nghĩa lớp i và j trùng kíp thời khóa biểu, với  $i, j = 0, 1, \dots, N-1$
- Cần lập kế hoạch phân công giảng dạy cho sao số tín chỉ lớn nhất các lớp phân cho mỗi giáo viên là nhỏ nhất

# Bài toán phân công giảng dạy

- Ví dụ

Lớp	0	1	2	3	4	5	6	7	8	9	10	11	12
Số tiết	3	3	4	3	4	3	3	3	4	3	3	4	4

Giáo viên	Danh sách lớp học có thể dạy
0	0, 2, 3, 4, 8, 10
1	0, 1, 3, 5, 6, 7, 8
2	1, 2, 3, 7, 9, 11, 12

Cặp lớp trùng tiết

0	2
0	4
0	8
1	4
1	10
3	7
3	9
5	11
5	12
6	8
6	12

# Bài toán phân công giảng dạy

- Mô hình hóa (quy hoạch nguyên tuyến tính)
  - Biến
    - $x[0..N-1, 0..M-1]$ :  $x[i,j]$  là biến nhị phân  $D(x[i,j]) = \{0,1\}$  với ý nghĩa
      - $x[i,j] = 1$ : lớp  $i$  được phân cho GV  $j$
      - $x[i,j] = 0$ : lớp  $i$  không được phân công cho GV  $j$
    - $y$ : là số tiết của các lớp phân cho giáo viên lớn nhất
  - Ràng buộc
    - Với mọi cặp  $i,j$  sao cho  $tc(i,j) = 0$ , đưa ra ràng buộc  $x[j,i] = 0$
    - Với mọi cặp 2 lớp  $i$  và  $j$  sao cho  $f(i,j) = 1$ , với mọi  $k = 0, \dots, M-1$ , đưa ra ràng buộc
$$x[i,k] + x[j,k] \leq 1$$
    - Mỗi lớp chỉ có thể được phân công cho đúng 1 giáo viên
$$x[i,0] + x[i,1] + \dots + x[i,M-1] = 1, \text{ với mọi } i = 0, \dots, N-1$$
    - $\sum_{i=0..N-1} x[i,j]*c[i] \leq y$ , với mọi  $j = 0, \dots, M-1$
- Hàm mục tiêu:  $y \rightarrow \min$

# Bài toán phân công giảng dạy

```
import com.google.ortools.linearsolver.MPConstraint;
import com.google.ortools.linearsolver.MPOjective;
import com.google.ortools.linearsolver.MPSolver;
import com.google.ortools.linearsolver.MPSolver.ResultStatus;
import com.google.ortools.linearsolver.MPVariable;

public class BCA {
    static {
        System.LoadLibrary("jniortools");
    }
    // input data structures
    int M = 3;// so giao vien 0,1,...,M-1
    int N = 13;// so mon hoc 0,1,...,N-1
    int[][] teachClass = { { 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0 },
                           { 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0 },
                           { 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1 }, };
    int[] credits = { 3, 3, 4, 3, 4, 3, 3, 4, 3, 3, 4, 4 };
}
```

# Bài toán phân công giảng dạy

```
int[][] conflict = { { 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0 },  
    { 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0 },  
    { 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  
    { 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0 },  
    { 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1 },  
    { 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 },  
    { 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  
    { 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0 },  
    { 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  
    { 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },  
    { 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 },  
    { 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0 } };
```

# Bài toán phân công giảng dạy

```
public void solve() {  
    MPSolver solver = new MPSolver("BCA",  
        MPSolver.OptimizationProblemType.CBC_MIXED_INTEGER_PROGRAMMING);  
    MPVariable[][] x = new MPVariable[N][M];  
    for (int i = 0; i < N; i++)  
        for (int j = 0; j < M; j++)  
            x[i][j] = solver.makeIntVar(0, 1, "x[" + i + "," + j + "]");  
    MPVariable[] load = new MPVariable[M];  
    int totalCredits = 0;  
    for (int i = 0; i < credits.length; i++)  
        totalCredits += credits[i];  
    for (int i = 0; i < M; i++)  
        load[i] = solver.makeIntVar(0, totalCredits, "load[" + i + "]");  
  
    MPVariable y = solver.makeIntVar(0, totalCredits, "y");
```

# Bài toán phân công giảng dạy

```
for (int i = 0; i < N; i++)  
    for (int j = 0; j < M; j++)  
        if (teachClass[j][i] == 0) {  
            MPConstraint c = solver.makeConstraint(0, 0);  
            c.setCoefficient(x[i][j], 1);  
        }  
    for (int i = 0; i < N; i++)  
        for (int j = 0; j < N; j++)  
            if (conflict[i][j] == 1) {  
                for (int k = 0; k < M; k++) {  
                    MPConstraint c = solver.makeConstraint(0, 1);  
                    c.setCoefficient(x[i][k], 1);  
                    c.setCoefficient(x[j][k], 1);  
                }  
            }  
        }
```

# Bài toán phân công giảng dạy

```
for (int i = 0; i < N; i++) {  
    MPConstraint c = solver.makeConstraint(1, 1);  
    for (int j = 0; j < M; j++)  
        c.setCoefficient(x[i][j], 1);  
}  
  
for (int i = 0; i < M; i++) {  
    MPConstraint c = solver.makeConstraint(0, 0);  
    for (int j = 0; j < N; j++)  
        c.setCoefficient(x[j][i], credits[j]);  
    c.setCoefficient(load[i], -1);  
}  
  
for (int i = 0; i < M; i++) {  
    MPConstraint c = solver.makeConstraint(0, totalCredits);  
    c.setCoefficient(load[i], -1);c.setCoefficient(y, 1);  
}
```

# Bài toán phân công giảng dạy

```
MPObjective obj = solver.objective();
obj.setCoefficient(y, 1); obj.setMinimization();
ResultStatus rs = solver.solve();
if (rs != ResultStatus.OPTIMAL) {
    System.out.println("cannot find optimal solution");
} else {
    System.out.println("obj= " + obj.value());
    for (int i = 0; i < M; i++) {
        System.out.print("teacher " + i + ": ");
        for (int j = 0; j < N; j++)
            if (x[j][i].solutionValue() == 1)
                System.out.print(j + " ");
        System.out.println(", Load = " + Load[i].solutionValue());
    }
}
```

# Bài toán phân công giảng dạy

```
public static void main(String[] args) {  
    // TODO Auto-generated method stub  
    BCA app = new BCA();  
    app.solve();  
}
```

# Bài toán TSP

- Có N thành phố  $0, 1, \dots, N-1$ , biết rằng chi phí đi từ thành phố  $i$  đến thành phố  $j$  là  $c(i,j)$ , với  $i, j = 0, 1, \dots, N-1$ . Hãy tìm hành trình xuất phát từ thành phố 0, đi qua tất cả các thành phố khác, mỗi thành phố đúng 1 lần và quay về thành phố 0 có tổng chi phí nhỏ nhất

# Bài toán TSP

- Biến
  - $X(i,j) = 1$ , nếu chu trình đi từ thành phố  $i$  đến thành phố  $j$ ,  $X(i,j) = 0$ , nếu chu trình ko đi từ  $i$  đến  $j$  (với mọi  $i,j = 0,1,\dots,N-1$  và  $i \neq j$ ).
- Ràng buộc
  - Mỗi thành phố sẽ có 1 đường đi vào và 1 đường đi ra
    - $\sum_{i \in \{0,1,\dots,N-1\} \setminus \{j\}} X(i,j) = \sum_{i \in \{0,1,\dots,N-1\} \setminus \{j\}} X(j,i) = 1$ , với mọi  $j = 0,1,\dots,N-1$
  - Ràng buộc cấm tạo chu trình con (SEC)
    - $\sum_{i,j \in S, i \neq j} X(i,j) \leq |S| - 1$ , với mọi  $S \subset \{0,1,2,\dots,N-1\}$ ,  $|S| \geq 2$
- Hàm mục tiêu:  $\sum_{i,j \in \{0,1,\dots,N-1\}, i \neq j} X(i,j) c(i,j) \rightarrow \min$

# Bài toán TSP

```
// generate subsets of {0,1,...,N-1)
public class SubSetGenerator{
    int N;
    int[] X;// represents binary sequence
    public SubSetGenerator(int N){
        this.N = N;
    }
    public HashSet<Integer> first(){
        X = new int[N];
        for(int i = 0; i < N; i++)
            X[i] = 0;
        HashSet<Integer> S = new
            HashSet<Integer>();
        for(int i = 0; i < N; i++)
            if(X[i] == 1) S.add(i);
        return S;
    }
}
```

```
public HashSet<Integer> next(){
    int j = N-1;
    while(j >= 0 && X[j] == 1){
        X[j] = 0; j--;
    }
    if(j >= 0){
        X[j] = 1;
        HashSet<Integer> S = new
            HashSet<Integer>();
        for(int i = 0; i < N; i++)
            if(X[i] == 1) S.add(i);
        return S;
    }else{
        return null;
    }
}
```

# Bài toán TSP

```
import com.google.ortools.linearsolver.MPConstraint;
import com.google.ortools.linearsolver.MPOjective;
import com.google.ortools.linearsolver.MPSolver;
import com.google.ortools.linearsolver.MPVariable;
public class TSP {
    static {
        System.LoadLibrary("jniortools");
    }
    int N = 5;
    int[][] c = { {0,4,2,5,6},
                  {2,0,5,2,7},
                  {1,2,0,6,3},
                  {7,5,8,0,3},
                  {1,2,4,3,0}};
    double inf = java.lang.Double.POSITIVE_INFINITY;
    MPSolver solver;
    MPVariable[][] x;
```

# Bài toán TSP

```
public void solve(){
    if(N > 10){
        System.out.println("N = 10 is too high to apply this solve method, use
                           solveDynamicAddSubTourConstraint");
        return;
    }
    solver = new MPSolver("TSP solver",
                          MPSolver.OptimizationProblemType.
                                         valueOf("CBC_MIXED_INTEGER_PROGRAMMING"));
    X = new MPVariable[N][N];
    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++)  if(i != j)
            X[i][j] = solver.makeIntVar(0, 1, "X[" + i + "," + j + "]");
    MPOjective obj = solver.objective();
    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++) if(i != j){
            obj.setCoefficient(X[i][j], c[i][j]);
        }
}
```

# Bài toán TSP

```
// flow constraint
for(int i = 0; i < N; i++){
    // \sum X[i,j] = 1, \forall j \in {0,...,N-1}\{i}
    MPConstraint fc1 = solver.makeConstraint(1,1);
    for(int j = 0; j < N; j++) if(j != i){
        fc1.setCoefficient(X[i][j], 1);
    }

    // \sum X[j][i] = 1, \forall j\in {0,1,...,N-1}\{i}
    MPConstraint fc2 = solver.makeConstraint(1,1);
    for(int j = 0; j < N; j++) if(j != i){
        fc2.setCoefficient(X[j][i], 1);
    }
}
```

# Bài toán TSP

```
// sub-tour elimination constraints
SubSetGenerator generator = new SubSetGenerator(N);
HashSet<Integer> S = generator.first();
while(S != null){
    if(S.size() > 1 && S.size() < N){
        MPConstraint sc = solver.makeConstraint(0,S.size() -1);
        for(int i: S){
            for(int j: S)if(i != j){
                sc.setCoefficient(X[i][j], 1);
            }
        }
    }
    S = generator.next();
}
```

# Bài toán TSP

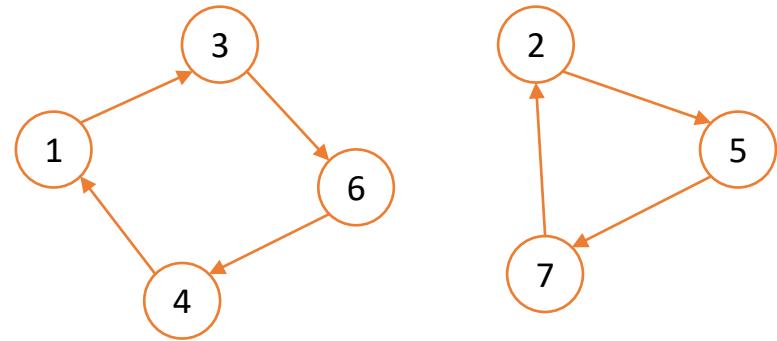
```
final MPSolver.ResultStatus resultStatus = solver.solve();
if (resultStatus != MPSolver.ResultStatus.OPTIMAL) {
    System.err.println("The problem does not have an optimal solution!");
    return;
}
System.out.println("Problem solved in " + solver.wallTime() + "
                    milliseconds");
// The objective value of the solution.
System.out.println("Optimal objective value = " +
                    solver.objective().value());
}
public static void main(String[] args) {
    TSP app = new TSP();
    app.solve();
}
}
```

# Bài toán TSP: Thêm dần SEC

- Khi  $N$  lớn, việc thêm tất cả các ràng buộc SEC sẽ không hiệu quả (số lượng rất lớn)
- Cách khắc phục
  - Ban đầu tạm nới lỏng tất cả các ràng buộc SEC (không đưa vào) → mô hình MIPR
  - **Bước lặp:** Giải mô hình MIPR
    - Nếu lời giải không có chu trình con → kết quả tối ưu, kết thúc chương trình
    - Nếu tồn tại chu trình con  $v_1 \rightarrow v_2 \rightarrow \dots, v_k \rightarrow v_1$ . Khi đó, ta bổ sung SEC ứng với  $S = \{v_1, v_2, \dots, v_k\}$  vào MIPR và lặp lại **Bước lặp** trên

# Bài toán TSP: Thêm dần SEC

```
SolveDynSEC(n,c){  
    SEC = []; // list cac SEC  
    while True do{  
        x = SolveDynSEC(SEC);  
        for v = 1 → n do mark[v] = False  
        for s = 1 → n do if not mark[s] then{  
            C = ExtractCycle(x,s);  
            if size(C) = n then{  
                return C;  
            }else{  
                SEC.add(C);  
                for e in C do mark[e] = True;  
            }  
        }  
    }  
}
```



$x[1,3] = 1, x[3,6] = 1, x[6,4] = 1, x[4,1] = 1, x[2,5] = 1,$   
 $x[5,7] = 1, x[7,2] = 1,$

$\text{SEC} = [[1,3,6,4], [2,5,7]]$

# Bài toán TSP: Thêm dần SEC

```
public class TSPDynSEC {  
    static {  
        System.LoadLibrary("jniortools");  
    }  
  
    int N = 5;  
    int[][] c = { {0,4,2,5,6},  
                  {2,0,5,2,7},  
                  {1,2,0,6,3},  
                  {7,5,8,0,3},  
                  {1,2,4,3,0}};  
    double inf = java.lang.Double.POSITIVE_INFINITY;  
    MPSolver solver;  
    MPVariable[][] x;
```

# Bài toán TSP: Thêm dần SEC

```
private int findNext(int s){  
    for(int i = 0; i < N; i++) if(i != s && X[s][i].solutionValue() > 0) return i;  
    return -1;  
}  
  
public ArrayList<Integer> extractCycle(int s){  
    ArrayList<Integer> L = new ArrayList<Integer>();  
    int x = s;  
    while(true){  
        L.add(x); x = findNext(x);  
        int rep = -1;  
        for(int i = 0; i < L.size(); i++)if(L.get(i) == x){  
            rep = i; break; }  
        if(rep != -1){  
            ArrayList<Integer> rL = new ArrayList<Integer>();  
            for(int i = rep; i < L.size(); i++) rL.add(L.get(i));  
            return rL;  
        }  
    }  
    //return rL;  
}
```

# Bài toán TSP: Thêm dần SEC

```
private void createVariables(){
    solver = new MPSolver("TSP solver",
        MPSolver.OptimizationProblemType.valueOf("CBC_MIXED_INTEGER_PROGRAMMING"));
    X = new MPVariable[N][N];
    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            if(i != j){
                X[i][j] = solver.makeIntVar(0, 1, "X[" + i + "," + j + "]");
            }
        }
    }
}
```

# Bài toán TSP: Thêm dần SEC

```
private void createObjective(){
    MPOjective obj = solver.objective();
    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            if(i != j){
                obj.setCoefficient(X[i][j], c[i][j]);
            }
        }
    }
}
```

# Bài toán TSP: Thêm dần SEC

```
private void createFlowConstraint(){
    // flow constraint
    for(int i = 0; i < N; i++){
        // \sum X[j][i] = 1, \forall j \in {0,1,...,N-1}\{i}
        MPConstraint fc1 = solver.makeConstraint(1,1);
        for(int j = 0; j < N; j++) if(j != i){
            fc1.setCoefficient(X[j][i], 1);
        }
        // \sum X[i][j] = 1, \forall j \in {0,1,...,N-1}\{i}
        MPConstraint fc2 = solver.makeConstraint(1,1);
        for(int j = 0; j < N; j++) if(j != i){
            fc2.setCoefficient(X[i][j], 1);
        }
    }
}
```

# Bài toán TSP: Thêm dần SEC

```
private void createSEC(HashSet<ArrayList<Integer>> S){  
    for(ArrayList<Integer> C: S){  
        MPConstraint sc = solver.makeConstraint(0, C.size() -1);  
        for(int i: C){  
            for(int j : C) if(i != j){  
                sc.setCoefficient(X[i][j], 1);  
            }  
        }  
    }  
}  
  
private void createSolverWithSEC(HashSet<ArrayList<Integer>> S){  
    createVariables();  
    createObjective();  
    createFlowConstraint();  
    createSEC(S);  
}
```

# Bài toán TSP: Thêm dần SEC

```
public void solveDynamicAddSubTourConstraint(){
    HashSet<ArrayList<Integer>> S = new HashSet();
    boolean[] mark = new boolean[N];
    boolean found = false;
    while(!found){
        createSolverWithSEC(S);
        final MPSolver.ResultStatus resultStatus = solver.solve();
        if (resultStatus != MPSolver.ResultStatus.OPTIMAL) {
            System.err.println("The problem does not have an optimal solution!");
            return;
        }
        System.out.println("obj = " + solver.objective().value());
    }
}
```

# Bài toán TSP: Thêm dần SEC

```
for(int i = 0; i < N; i++) mark[i] = false;  
for(int s = 0; s < N; s++){  
    ArrayList<Integer> C = extractCycle(s);  
    if(C.size() < N){// sub-tour detected  
        System.out.print("SubTour deteted, C = ");  
        for(int i: C) System.out.print(i + " "); System.out.println();  
        S.add(C);  
        for(int i: C) mark[i] = true;  
    }else{  
        System.out.println("Global tour detected, solution found!!!!");  
        found = true; break;  
    }  
}  
}  
ArrayList<Integer> tour = extractCycle(0);  
for(int i = 0; i < tour.size(); i++) System.out.print(tour.get(i) + " -> ");  
System.out.println(tour.get(0));  
}
```

# Bài toán TSP: Thêm dần SEC

```
public static void main(String[] args) {  
    TSPDynSEC app = new TSPDynSEC();  
    app.solveDynamicAddSubTourConstraint();  
}  
}
```