

Chương 4

BÀI TOÁN TỐI ƯU TỔ HỢP

Nội dung

- 1. Phát biểu bài toán
- 2. Duyệt toàn bộ
- 3. Thuật toán nhánh cận



1. Phát biểu bài toán[?]

- 1.1. Bài toán tổng quát
- 1.2. Bài toán người du lịch
- 1.3. Bài toán cái túi
- 1.4. Bài toán đóng thùng

- Trong rất nhiều vấn đề ứng dụng thực tế của tổ hợp, các cấu hình tổ hợp được gán cho một giá trị bằng số đánh giá giá trị sử dụng của cấu hình đối với mục đích sử dụng cụ thể nào đó. Khi đó xuất hiện bài toán: Hãy lựa chọn trong số các cấu hình tổ hợp chấp nhận được cấu hình có giá trị sử dụng tốt nhất. Các bài toán như vậy chúng ta sẽ gọi là bài toán tối ưu tổ hợp.

Phát biểu bài toán

- Dạng tổng quát bài toán tối ưu tổ hợp có thể phát biểu nh sau:

Tìm cực tiểu (hay cực đại) của phiếm hàm

$$f(x) \rightarrow \min (\max),$$

với điều kiện

$$x \in D,$$

trong đó D là tập hữu hạn phần tử.

Các thuật ngữ

- $f(x)$ - hàm mục tiêu của bài toán,
- $x \in D$ - phương án
- D - tập các phương án của bài toán.
- Thông thường tập D được mô tả nh là tập các cấu hình tổ hợp thoả mãn một số tính chất 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 u**, khi đó giá trị $f^* = f(x^*)$ được gọi là **giá trị tối u** của bài toán.

1. Phát biểu bài toán[?]

- 1.1. Bài toán tổng quát
- **1.2. Bài toán người du lịch**
- 1.3. Bài toán cái túi
- 1.4. Bài toán đóng thùng

Bài toán người du lịch

(Traveling Salesman Problem – TSP)

- Một người du lịch muốn đi tham quan n thành phố T_1, T_2, \dots, T_n .
- *Hành trình là cách đi xuất phát từ một thành phố nào đó đi qua tất cả các 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 ($i, j = 1, 2, \dots, n$),
- **Tìm hành trình với tổng chi phí là nhỏ nhất.**

Sơ lược về lịch sử

- The origins of the TSP are obscure. In the 1920's, the mathematician and economist Karl Menger publicized it among his colleagues in Vienna.
- In the 1930's, the problem reappeared in the mathematical circles of Princeton.
- In the 1940's, it was studied by statisticians (Mahalanobis (1940), Jessen (1942), Gosh (1948), Marks (1948)) in connection with an agricultural application and the mathematician Merrill Flood popularized it among his colleagues at the RAND Corporation. Eventually, the TSP gained notoriety as the prototype of a hard problem in combinatorial optimization: examining the tours one by one is out of the question because of their large number, and no other idea was on the horizon for a long time.
- New history with George Dantzig, Ray Fulkerson, and Selmer Johnson's 1954 breakthrough.

- Ta có tương ứng 1-1 giữa một hành trình

$$T_{\pi(1)} \rightarrow T_{\pi(2)} \rightarrow \dots \rightarrow T_{\pi(n)} \rightarrow T_{\pi(1)}$$

với một hoán vị $\pi = (\pi(1), \pi(2), \dots, \pi(n))$ của n số tự nhiên $1, 2, \dots, n$.

Đặt

$$f(\pi) = c_{\pi(1), \pi(2)} + \dots + c_{\pi(n-1), \pi(n)} + c_{\pi(n), \pi(1)}.$$

Ký hiệu:

Π - tập tất cả các hoán vị của n số tự nhiên $1, 2, \dots, n$.

- Khi đó bài toán người du lịch có thể phát biểu dưới dạng bài toán tối ưu tổ hợp sau:

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

- 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 là thành phố xuất phát).

1. Phát biểu bài toán[?]

- 1.1. Bài toán tổng quát
- 1.2. Bài toán người du lịch
- **1.3. Bài toán cái túi**
- 1.4. Bài toán đóng thùng

Bài toán cái túi (Knapsack Problem)

- 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 là a_j và
 - giá trị sử dụng là c_j ($j = 1, 2, \dots, n$).
- Hỏi rằng nhà thám hiểm cần đem theo các đồ vật nào để cho tổng giá trị sử dụng của các đồ vật đem theo là lớn nhất?

Phát biểu bài toán

- Một phương án đem đồ của nhà thám hiểm có thể biểu diễn bởi vectơ nhị phân độ dài n : $x = (x_1, x_2, \dots, x_n)$, trong đó $x_j = 1$ nếu đồ vật thứ j được đem theo và $x_j = 0$ nếu trái lại.
- Với phương án x , giá trị đồ vật đem theo là

$$f(x) = \sum_{j=1}^n c_j x_j,$$

tổng trọng lượng đồ vật đem theo là

$$g(x) = \sum_{j=1}^n a_j x_j$$

Bài toán cái túi

- Bài toán cái túi có thể phát biểu dưới dạng bài toán tối ưu tổ hợp sau:

Trong số các vectơ nhị phân độ dài n thoả mãn điều kiện $g(x) \leq b$, hãy tìm vectơ x^* cho giá trị lớn nhất của hàm mục tiêu $f(x)$:

$$\max \{ f(x): x \in B^n, g(x) \leq b \}.$$

1. Phát biểu bài toán[?]

- 1.1. Bài toán tổng quát
- 1.2. Bài toán người du lịch
- 1.3. Bài toán cái túi
- **1.4. Bài toán đóng thùng**

Bài toán đóng thùng

(Bin Packing)

- Có n đồ vật với trọng lượng là w_1, w_2, \dots, w_n . Cần tìm cách xếp các đồ vật này vào các cái thùng có cùng dung lượng là b sao cho số thùng cần sử dụng là nhỏ nhất có thể được.

Phát biểu bài toán

- Ta có thể giả thiết là

$$w_i \leq b, i = 1, 2, \dots, n.$$

- Do đó số thùng cần sử dụng để chứa tất cả các đồ vật là không quá n . Vấn đề là cần số thùng ít nhất. Ta sẽ mở sẵn n cái thùng. Bài toán đặt ra là hãy xác định xem mỗi một trong số n đồ vật cần được xếp vào cái thùng nào trong số n cái thùng đã mở để cho số thùng chứa đồ là ít nhất.

Bài toán đóng thùng

- Đưa vào biến Bun

$x_{ij} = 1$, nếu đồ vật i được xếp vào thùng j ,
0, nếu trái lại.

Khi đó bài toán đóng thùng có thể phát biểu dưới dạng:

$$\sum_{j=1}^n \text{sign}(\sum_{i=1}^n x_{ij}) \rightarrow \min,$$

$$\sum_{j=1}^n x_{ij} = 1, \quad i = 1, 2, \dots, n$$

$$\sum_{i=1}^n w_i x_{ij} \leq b, \quad j = 1, 2, \dots, n;$$

$$x_{ij} \in \{0, 1\}, \quad i, j = 1, 2, \dots, n.$$

2. DUYỆT TOÀN BỘ

NỘI DUNG

2.1. Mô tả phương pháp

2.2. Ví dụ áp dụng: Bài toán cái túi

Mô tả phương pháp

- 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 liệt 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 tại nó, sau đó so sánh giá trị hàm mục tiêu tại tất cả các phương án được liệt kê để tìm ra phương án tối ưu.
- Phương pháp xây dựng theo nguyên tắc như vậy có tên gọi là phương pháp duyệt toàn bộ.

NỘI DUNG

2.1. Mô tả phương pháp

2.2. Ví dụ áp dụng: Bài toán cái túi

Ví dụ: Giải bài toán cái túi

- Xét bài toán cái túi:

$$\max \{f(x) = \sum_{j=1}^n c_j x_j : x \in D\},$$

$$\text{trong đó } D = \{x = (x_1, x_2, \dots, x_n) \in B^n : \sum_{j=1}^n w_j x_j \leq b\}$$

- c_j, w_j, b là các số nguyên dương, $j=1, 2, \dots, n$.
- Cần có thuật toán liệt kê các phần tử của D

Thuật toán quay lui liệt kê các phương án chất đồ

➤ **Xây dựng S_k :**

$S_1 = \{0, t_1\}$, với $t_1 = 1$ nếu $b \geq w_1$; $t_1 = 0$, nếu trái lại

➤ Giả sử đã có phương án (x_1, \dots, x_{k-1}) . Khi đó

Dung lượng còn lại là:

$$b_{k-1} = b - w_1x_1 - \dots - w_{k-1}x_{k-1}$$

Giá trị của các đồ vật chất vào túi là

$$f_{k-1} = c_1x_1 + \dots + c_{k-1}x_{k-1}$$

Do đó: $S_k = \{0, t_k\}$, với $t_k = 1$ nếu $b_{k-1} \geq w_k$; $t_k = 0$, nếu trái lại

➤ **Mô tả S_k :**

For $y := 0$ to t_k do

Chương trình trên Pascal

type

arrint= array[1..20] of integer;

var

x, xopt, c, w: arrint;

n,b, bk, fk, fopt: integer;

procedure Nhapdl;

var i: integer;

begin

{Nhập vào n, c, w, b}

end;

procedure Inkq;

var j;

begin

{Phương án tối ưu: xopt;

Giá trị tối ưu: fopt }

end;

```

procedure KP(i: integer);
var j, t: integer;
begin
    if bk ≥ w[i] then t:=1 else t:=0;
    for j := t downto 0 do begin
        x[i] := j; bk:= bk-w[i]*x[i];
        fk:= fk + c[i]*x[i];
        if i = n then begin
            if fk>fopt then begin
                xopt:=x; fopt:=fk;
            end
            end else KP(i+1);
        bk:= bk+w[i]*x[i];
        fk:= fk - c[i]*x[i];
    end;
end;

```

```

BEGIN {Main program}
    Nhapdl;
    bk:=b;
    fk:= 0;
    fopt:= 0;
    KP(1);
    InKq
END.

```

Bình luận

- Duyệt toàn bộ là khó có thể thực hiện được ngay cả trên những máy tính điện tử hiện đại nhất. Ví dụ để liệt kê hết

$$15! = 1\ 307\ 674\ 368\ 000$$

hoán vị trên máy tính điện tử với tốc độ tính toán 1 tỷ phép tính một giây, nếu để liệt kê một hoán vị cần phải làm 100 phép tính, thì ta cần một khoảng thời gian là 130767 giây > 36 tiếng đồng hồ!

- $20! \implies 7645 \text{ năm}$

- Vì vậy cần phải có những biện pháp nhằm hạn chế việc tìm kiếm thì mới có hy vọng giải được các bài toán tối ưu tổ hợp thực tế. Tất nhiên để có thể đề ra những biện pháp như vậy cần phải nghiên cứu kỹ tính chất của bài toán tối ưu tổ hợp cụ thể.
- Nhờ những nghiên cứu như vậy, trong một số trường hợp cụ thể ta có thể xây dựng những thuật toán hiệu quả để giải bài toán đặt ra.

- Tuy nhiên phải nhấn mạnh rằng trong nhiều trường hợp (ví dụ trong các bài toán ngời du lịch, bài toán cái túi, bài toán đóng thùng) chúng ta **cha thể xây dựng đợc phương pháp hữu hiệu** nào khác ngoài phương pháp duyệt toàn bộ.

- Khi đó, một vấn đề đặt ra là trong quá trình liệt kê lời giải ta cần tận dụng các thông tin đã tìm được để loại bỏ những phương án chắc chắn không phải là tối ưu.
- Trong mục tiếp theo chúng ta sẽ xét một sơ đồ tìm kiếm nh vậy để giải các bài toán tối ưu tổ hợp mà trong tài liệu tham khảo đọc biết đến với tên gọi: thuật toán nhánh cận.

3. THUẬT TOÁN NHÁNH CẬN

(Branch and Bound Algorithm)

NỘI DUNG

- **3.1. Sơ đồ chung**
- 3.2. Bài toán cái túi
- 3.3. Bài toán người du lịch

Sơ đồ chung

- Thuật toán bao gồm hai thủ tục:
 - Phân nhánh (Branching Procedure)
 - Tính cận (Bounding Procedure)
- **Phân nhánh:** *Quá trình phân hoạch tập các phương án ra thành các tập con với kích thước càng ngày càng nhỏ cho đến khi thu được phân hoạch tập các phương án ra thành các tập con một phần tử*
- **Tính cận:** *Cần đưa ra cách tính cận cho giá trị hàm mục tiêu của bài toán trên mỗi tập con A trong phân hoạch của tập các phương án.*

Sơ đồ chung

- Ta sẽ mô tả tổng của thuật toán trên mô hình bài toán tối ưu tổ hợp tổng quát sau

$$\min \{ f(x) : x \in D \},$$

trong đó D là tập hữu hạn phần tử.

Giả thiết rằng tập D được mô tả như sau

$$D = \{ x = (x_1, x_2, \dots, x_n) \in A_1 \times A_2 \times \dots \times A_n :$$

$$x \text{ thoả mãn tính chất } P \},$$

với A_1, A_2, \dots, A_n là các tập hữu hạn, còn P là tính chất cho trên tích Đềcac $A_1 \times A_2 \times \dots \times A_n$.

Nhận xét

- Nhận thấy rằng, các bài toán vừa trình bày ở mục 1 đều có thể mô tả dưới dạng bài toán trên.
- Yêu cầu về mô tả của tập D là để có thể sử dụng thuật toán quay lui để liệt kê các phương án của bài toán.

- Bài toán

$$\max \{f(x): x \in D\}$$

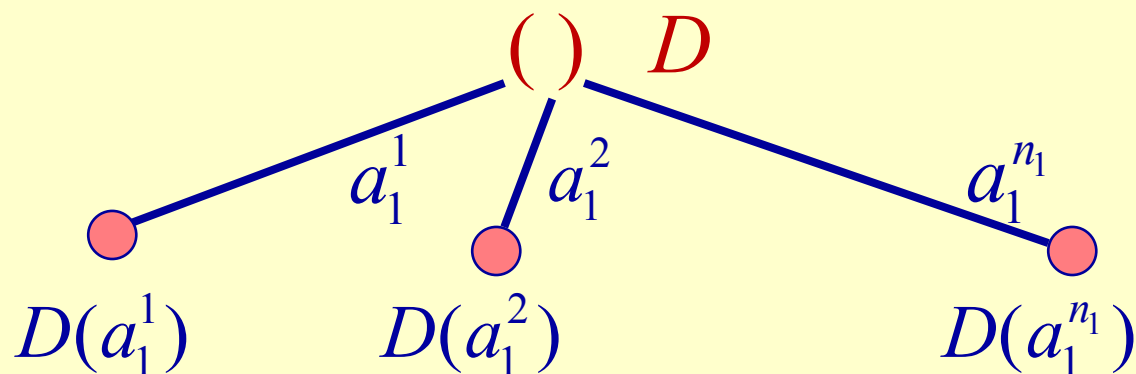
là tương đương với bài toán

$$\min \{g(x): x \in D\}, \text{ trong đó } g(x) = -f(x)$$

- Do đó ta có thể hạn chế ở việc xét bài toán min.

Phân nhánh

- Quá trình phân nhánh được thực hiện nhờ thuật toán quay lui:



trong đó $D(a_1^i) = \{x \in D : x_1 = a_1^i\}$, $i = 1, 2, \dots, n_1$

là tập các phương án có thể phát triển từ pabp (a_1^i)

$$D = D(a_1^1) \cup D(a_1^2) \cup \dots \cup D(a_1^{n_1})$$

Phân nhánh

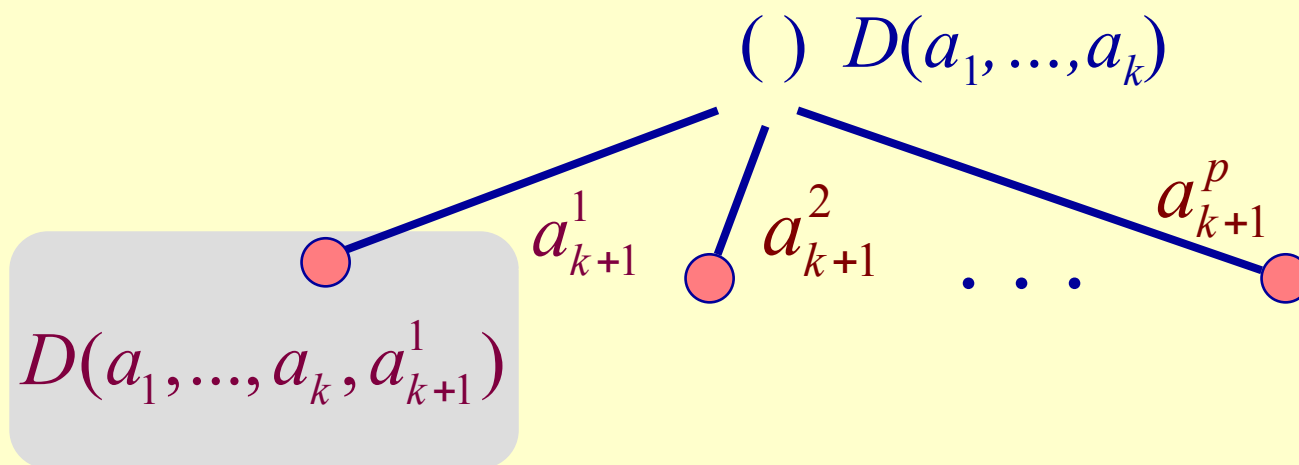
- Như vậy ta có thể đặt tương ứng mỗi phương án bộ phận (a_1, a_2, \dots, a_k) với một tập con các phương án của bài toán:

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

- Ở bước tổng quát của thuật toán quay lui ta sẽ làm việc với phương án bộ phận (a_1, a_2, \dots, a_k) và xét các cách tiếp tục phát triển phương án này.
- Điều đó tương đương với việc phân hoạch tập D ra thành các tập con nhỏ hơn.

Phân nhánh

- Quá trình phân nhánh có thể diễn tả như sau:



- Ta có phân hoạch:

$$D(a_1, \dots, a_k) = \bigcup_{i=1}^p D(a_1, \dots, a_k, a_{k+1}^i)$$

Tính cận

- Cần có hàm g xác định trên tập tất cả các phương án bộ phận của bài toán thoả mãn bất đẳng thức sau:

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

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

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

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

hay nói một cách khác, $g(a_1, a_2, \dots, a_k)$ là **cận dưới** của giá trị hàm mục tiêu trên tập $D(a_1, a_2, \dots, a_k)$.

- Vì lẽ đó, hàm g được gọi là **hàm cận đối**, và giá trị $g(a_1, a_2, \dots, a_k)$ được gọi là cận đối của tập $D(a_1, a_2, \dots, a_k)$.
- Do có thể đồng nhất tập $D(a_1, \dots, a_k)$ với phương án bộ phận (a_1, \dots, a_k) , nên ta cũng gọi giá trị $g(a_1, \dots, a_k)$ là **cận đối của phương án bộ phận (a_1, \dots, a_k)** .

Cắt nhánh nhờ sử dụng cận dưới

- Giả sử đã có hàm g . Ta xét cách sử dụng hàm này để giảm bớt 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 phương án với giá trị hàm mục tiêu nhỏ nhất trong số các phương án đã tìm được, ký hiệu $\bar{f} = f(\bar{x})$.
- Ta sẽ gọi
 - \bar{x} là **phương án tốt nhất hiện có**,
 - còn \bar{f} là **kỷ lục**.

Cắt nhánh nhờ sử dụng cận dưới

- Giả sử đã có \bar{f} , khi đó nếu

$$g(a_1, a_2, \dots, a_k) > \bar{f},$$

thì từ bất đẳng thức (*) suy ra

$$\bar{f} < g(a_1, \dots, a_k) \leq \min\{f(x) : x \in D(a_1, \dots, a_k)\},$$

- Vì thế tập $D(a_1, \dots, a_k)$ chắc chắn không chứa phương án tối ưu và có thể loại bỏ khỏi quá trình duyệt.

Thuật toán nhánh cận

procedure Branch(k);

(* Phát triển phương án bộ phận $(x_1, x_2, \dots, x_{k-1})$ *)

begin

for $a_k \in A_k$ **do**

if $a_k \in S_k$ **then**

begin

$x_k := a_k;$

if $(k = n)$ **then** < Cập nhật kỷ lục >

else

if $g(x_1, \dots, x_k) \leq \bar{f}$ **then** Branch(k+1)

end;

end;

Thuật toán nhánh cận

procedure BranchAndBound;

begin

$\bar{f} := +\infty;$

(* Nếu biết p/án \bar{x} nào đó thì đặt $\bar{f} = f(\bar{x})$ *)

Branch(1);

if $\bar{f} < +\infty$ then

$\langle \bar{f}$ là giá trị tối ưu, \bar{x} là p/án tối ưu \rangle

else \langle bài toán không có phương án \rangle ;

end;

Chú ý: Sơ đồ duyệt toàn bộ

- Chú ý rằng nếu trong thủ tục Branch ta thay câu lệnh

if $(k = n)$ **then** < Cập nhật kỷ lục>
else

if $g(a_1, \dots, a_k) \leq \bar{f}$ **then** Branch(k+1)
bởi

if $(k = n)$ **then** < Cập nhật kỷ lục>
else Branch(k+1)

thì ta thu được thuật toán **duyet toàn bộ**.

Chú ý:

- Việc xây dựng hàm g phụ thuộc vào từng bài toán tối ưu tổ hợp cụ thể. Thông thường ta cố gắng xây dựng nó sao cho:
 - Việc tính giá trị của g phải đơn giản hơn việc giải bài toán tối ưu tổ hợp ở vế phải của (*).
 - Giá trị của $g(a_1, \dots, a_k)$ phải sát với giá trị của vế phải của (*).
- Rất tiếc là hai yêu cầu này trong thực tế thường đối lập nhau.

NỘI DUNG

- 3.1. Sơ đồ chung
- **3.2. Bài toán cái túi**
- 3.3. Bài toán người du lịch

Bài toán cái túi

- Có n loại đồ vật.
- Đồ vật loại j có
 - trọng lượng a_j và
 - giá trị sử dụng là c_j ($j = 1, 2, \dots, n$) .
- Cần chắt các đồ vật này vào một cái túi có trọng lượng là b sao cho tổng giá trị sử dụng của các đồ vật chắt trong túi là lớn nhất.

Bài toán cái túi (KP)

- Đưa vào biến số

x_j – số lượng đồ vật loại j được chắt vào túi,
 $j=1,2, \dots, n$

- Mô hình toán học của bài toán có dạng sau: Tìm

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

trong đó Z_+ là tập các số nguyên không âm.

- Ký hiệu D là tập các phương án của bài toán:

$$D = \{x = (x_1, \dots, x_n) : \sum_{j=1}^n a_j x_j \leq b, x_j \in Z_+, j = 1, 2, \dots, n\}$$

- Giả thiết rằng các đồ vật được đánh số sao cho bất đẳng thức sau được thoả mãn

$$c_1 / a_1 \geq c_2 / a_2 \geq \dots \geq c_n / a_n .$$

(có nghĩa là các đồ vật được xếp theo thứ tự không tăng của giá trị một đơn vị trọng lượng)

Xây dựng hàm cận trên

- Để xây dựng hàm tính cận trên, cùng với bài toán cái túi (KP) ta xét bài toán cái túi biến liên tục (KPC) sau đây: Tìm

$$g^* = \max \left\{ f(x) = \sum_{j=1}^n c_j x_j : \sum_{j=1}^n a_j x_j \leq b, x_j \geq 0, j = 1, 2, \dots, n \right\}$$

- **Mệnh đề.** *Phương án tối u của bài toán KPC là vectơ $\bar{x} = (\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n)$ với các thành phần được xác định bởi công thức:*

$$\bar{x}_1 = b / a_1, \bar{x}_2 = \bar{x}_3 = \dots = \bar{x}_n = 0.$$

và giá trị tối u là $g^ = c_1 b / a_1$.*

- **Chứng minh.** Xét $x = (x_1, \dots, x_n)$ là một phương án tùy ý của bài toán KPC. Khi đó

$$c_j \leq (c_1 / a_1) a_j, j = 1, 2, \dots, n$$

do $x_j \geq 0$, ta suy ra

$$c_j x_j \leq (c_1 / a_1) a_j x_j, j = 1, 2, \dots, n$$

- Từ đó ta có

$$\sum_{j=1}^n c_j x_j \leq \sum_{j=1}^n (c_1 / a_1) a_j x_j$$

$$= (c_1 / a_1) \sum_{j=1}^n a_j x_j$$

$$\leq (c_1 / a_1) b = g^*$$

- Mệnh đề được chứng minh.

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

$$\sigma_k = c_1u_1 + c_2u_2 + \dots + c_ku_k$$

và trọng lượng còn lại của cái túi là

$$b_k = b - (a_1u_1 + a_2u_2 + \dots + a_ku_k).$$

Tính cận trên

➤ Ta có

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

$$= \max \left\{ \sigma_k + \sum_{j=k+1}^n c_j x_j : \sum_{j=k+1}^n a_j x_j \leq b_k, x_j \in \mathbb{Z}_+, j = k+1, k+2, \dots, n \right\}$$

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

$$= \sigma_k + c_{k+1} b_k / a_{k+1}.$$

➤ Vậy ta có thể tính cận trên cho phương án bộ phận (u_1, u_2, \dots, u_k) bởi công thức

$$g(u_1, u_2, \dots, u_k) = \sigma_k + c_{k+1} b_k / a_{k+1}.$$

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

Ví dụ

- Giải bài toán cái túi sau theo thuật toán nhánh cận vừa trình bày

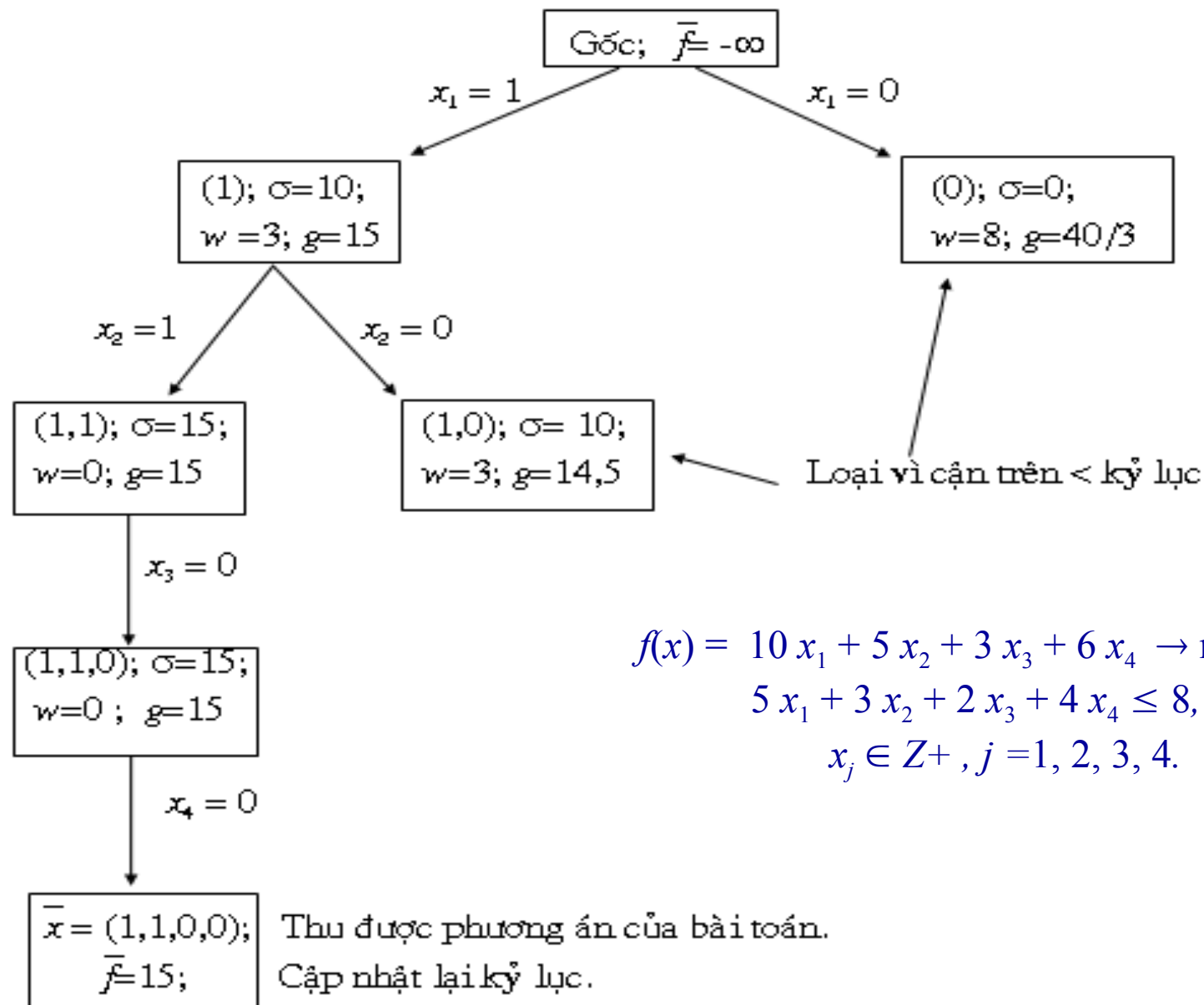
$$f(x) = 10x_1 + 5x_2 + 3x_3 + 6x_4 \rightarrow \max,$$

$$5x_1 + 3x_2 + 2x_3 + 4x_4 \leq 8,$$

$$x_j \in \mathbb{Z}_+, j = 1, 2, 3, 4.$$

- *Chú ý: Trong ví dụ đang xét, các đồ vật đã được xếp theo thứ tự không tăng của giá trị một đơn vị trọng lượng.*

- Quá trình giải bài toán đọc mô tả trong cây tìm kiếm trong hình 1. Thông tin về một phương án bộ phận trên cây đọc ghi trong các ô trên hình vẽ tương ứng theo thứ tự sau:
- các thành phần của phương án,
 - σ - giá trị của các đồ vật đang chất trong túi,
 - w - trọng lượng còn lại của túi
 - g - cận trên.



$$f(x) = 10x_1 + 5x_2 + 3x_3 + 6x_4 \rightarrow \max,$$

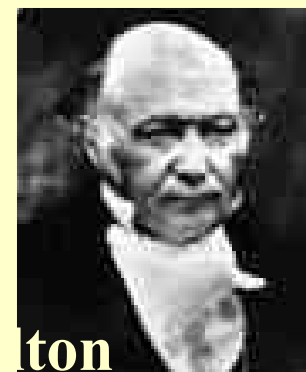
$$5x_1 + 3x_2 + 2x_3 + 4x_4 \leq 8,$$

$$x_j \in \mathbb{Z}^+, j = 1, 2, 3, 4.$$

- Kết thúc thuật toán, ta thu được:
 - Phương án tối ưu: $x^* = (1, 1, 0, 0)$,
 - Giá trị tối ưu: $f^* = 15$.

NỘI DUNG

- 3.1. Sơ đồ chung
- 3.2. Bài toán cái túi
- **3.3. Bài toán người du lịch**



ton

- Cố định thành phố xuất phát là T_1 , bài toán ngời du lịch dẫn về bài toán:

Tìm cực tiểu của hàm

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

với điều kiện

$(1, x_2, x_3, \dots, x_n)$ là hoán vị của các số $1, 2, \dots, n$.

Hàm cận dưới

➤ Ký hiệu

$$c_{\min} = \min \{ c[i, j] , i, j = 1, 2, \dots, n, i \neq j \}$$

là chi phí đi lại nhỏ nhất giữa các thành phố.

➤ Cần đánh giá cận dưới cho phương án bộ phận $(1, u_2, \dots, u_k)$ tương ứng với hành trình bộ phận qua k thành phố:

$$T_1 \rightarrow T(u_2) \rightarrow \dots \rightarrow T(u_{k-1}) \rightarrow T(u_k).$$

Hàm cận dưới

- Chi phí phải trả theo hành trình bộ phận này là

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

- Để phát triển thành hành trình đầy đủ, ta còn phải đi qua $n-k+1$ đoạn đường nữa, mỗi đoạn có chi phí không ít hơn c_{min} , nên cận dưới cho phương án bộ phận $(1, u_2, \dots, u_k)$ có thể tính theo công thức

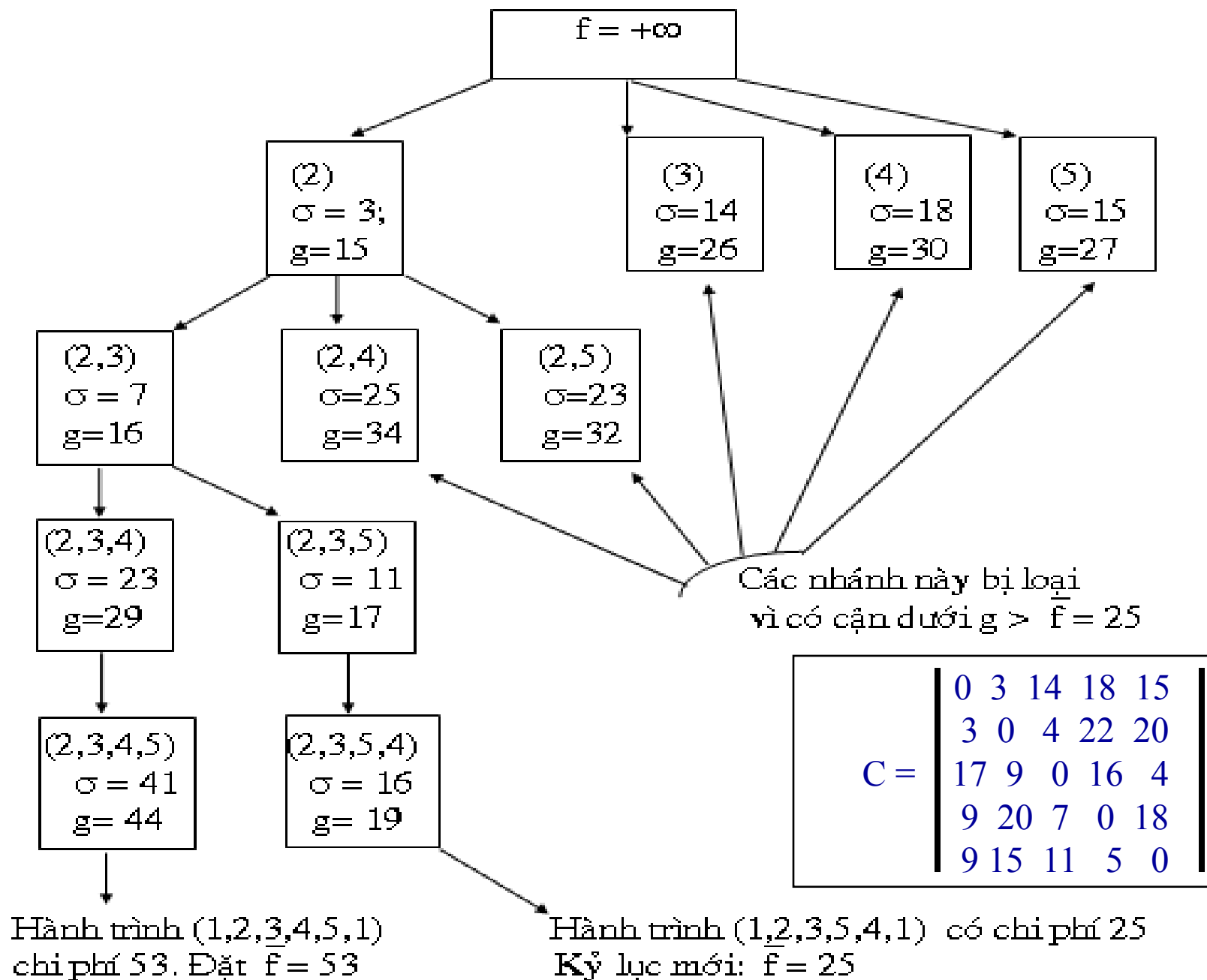
$$g(1, u_2, \dots, u_k) = \sigma + (n-k+1) c_{min}.$$

Ví dụ

- Giải bài toán người du lịch với ma trận chi phí sau:

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

- Ta có $c_{min} = 3$. Quá trình thực hiện thuật toán được mô tả bởi cây tìm kiếm lời giải.
- Thông tin được ghi trong các ô trên hình vẽ theo thứ tự sau:
 - các thành phần của phương án,
 - σ là chi phí theo hành trình bộ phận
 - g - cận dưới.



Kết quả

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

$$T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_5 \rightarrow T_4 \rightarrow T_1 ,$$

- Chi phí nhỏ nhất là 25.

Kỷ lục về giải bài toán người du lịch

Kỷ lục

(Kích thước TSP giải được)

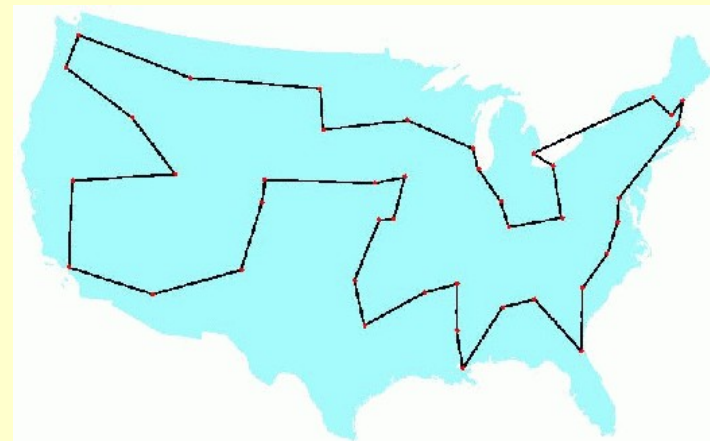
1954	1962	1977	1987	1987	1987	1994	1998	2001	2004
49	33	120	532	666	2392	7397	13509	15112	24978

<http://www.tsp.gatech.edu/index.html>

Year	Research Team	Size of Instance
1954	G. Dantzig, R. Fulkerson, and S. Johnson	49 cities
1971	M. Held and R.M. Karp	64 cities
1975	P.M. Camerini, L. Fratta, and F. Maffioli	67 cities
1977	M. Grötschel	120 cities
1980	H. Crowder and M.W. Padberg	318 cities
1987	M. Padberg and G. Rinaldi	532 cities
1987	M. Grötschel and O. Holland	666 cities
1987	M. Padberg and G. Rinaldi	2,392 cities
1994	D. Applegate, R. Bixby, V. Chvátal, and W. Cook	7,397 cities
1998	D. Applegate, R. Bixby, V. Chvátal, and W. Cook	13,509 cities
2001	D. Applegate, R. Bixby, V. Chvátal, and W. Cook	15,112 cities
2004	D. Applegate, R. Bixby, V. Chvátal, W. Cook, and K. Helsgaun	24,978 cities

The First Big TSP

Dantzig, Ray Fulkerson, and Selmer Johnson (1954) published a description of a method for solving the TSP and illustrated the power of this method by solving an instance with 49 cities, an impressive size at that time. They created this instance by picking one city from each of the 48 states in the U.S.A. (Alaska and Hawaii became states only in 1959) and adding Washington, D.C.; the costs of travel between these cities were defined by road distances. Rather than solving this problem, they solved the 42-city problem obtained by removing Baltimore, Wilmington, Philadelphia, Newark, New York, Hartford, and Providence. As it turned out, an optimal tour through the 42 cities used the edge joining Washington, D.C. to Boston; since the shortest route between these two cities passes through the seven removed cities, this solution of the 42-city problem yields a solution of the 49-city problem.



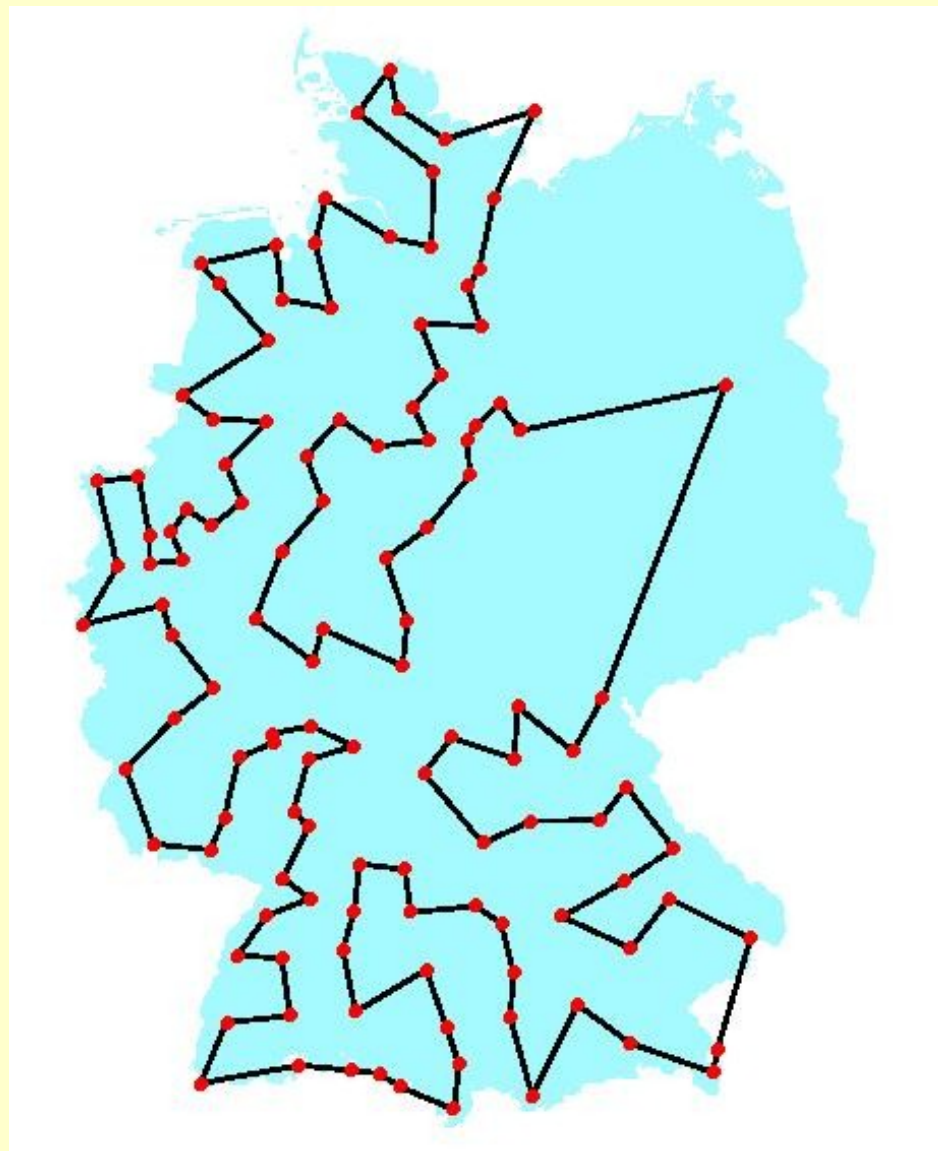
Proctor and Gamble's Contest

- Proctor and Gamble ran a contest in 1962. The contest required solving a TSP on a specified 33 cities. There was a tie between many people who found the optimum. An early TSP researcher, Professor Gerald Thompson of Carnegie Mellon University, was one of the winners.



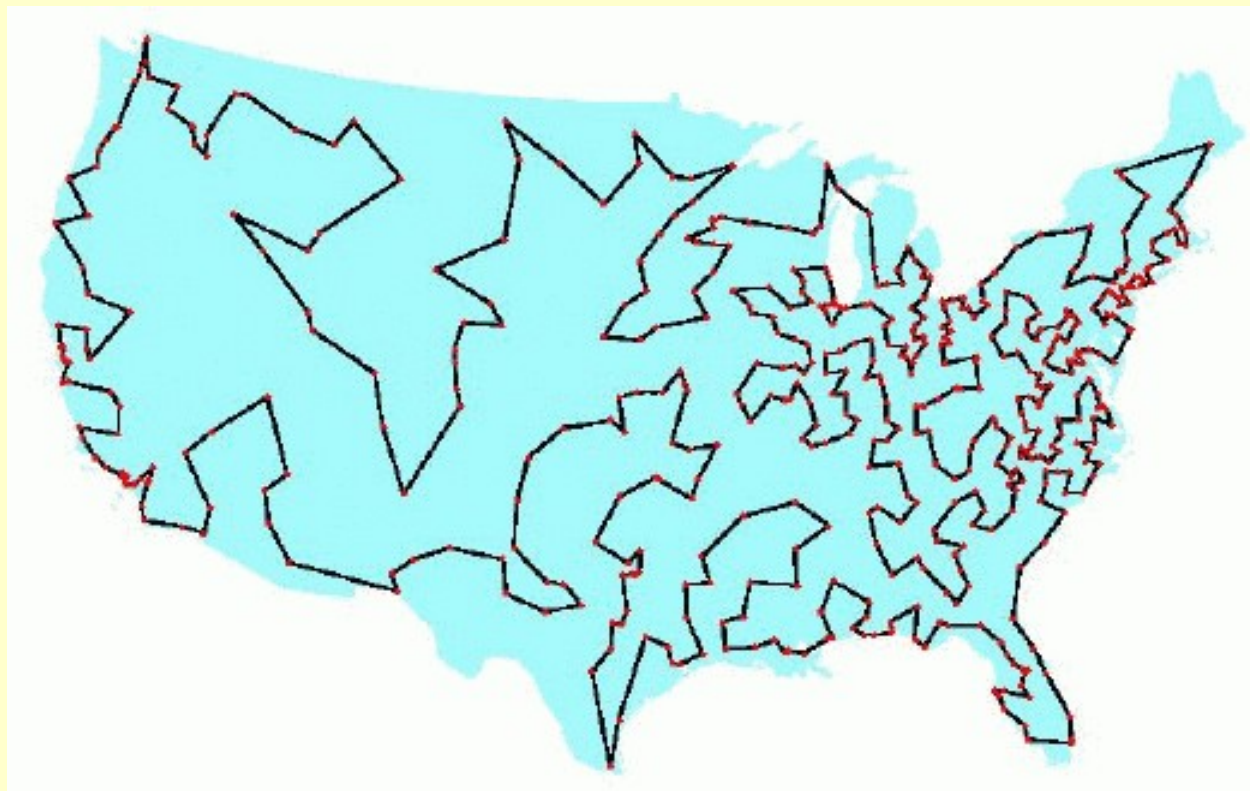
120 Western German Cities

- Groetschel (1977) found the optimal tour of 120 cities from what was then West Germany.



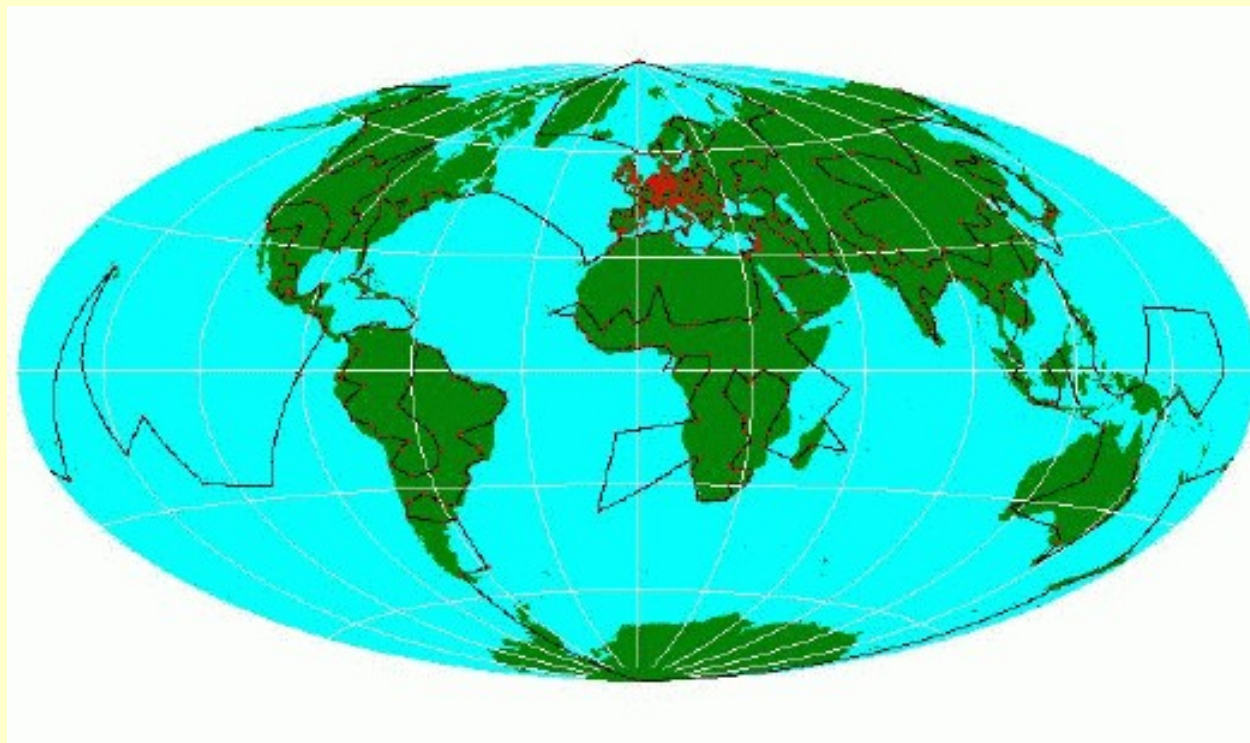
532 Locations in America

- Padberg and Rinaldi (1987) found the optimal tour of 532 AT&T switch locations in the USA.



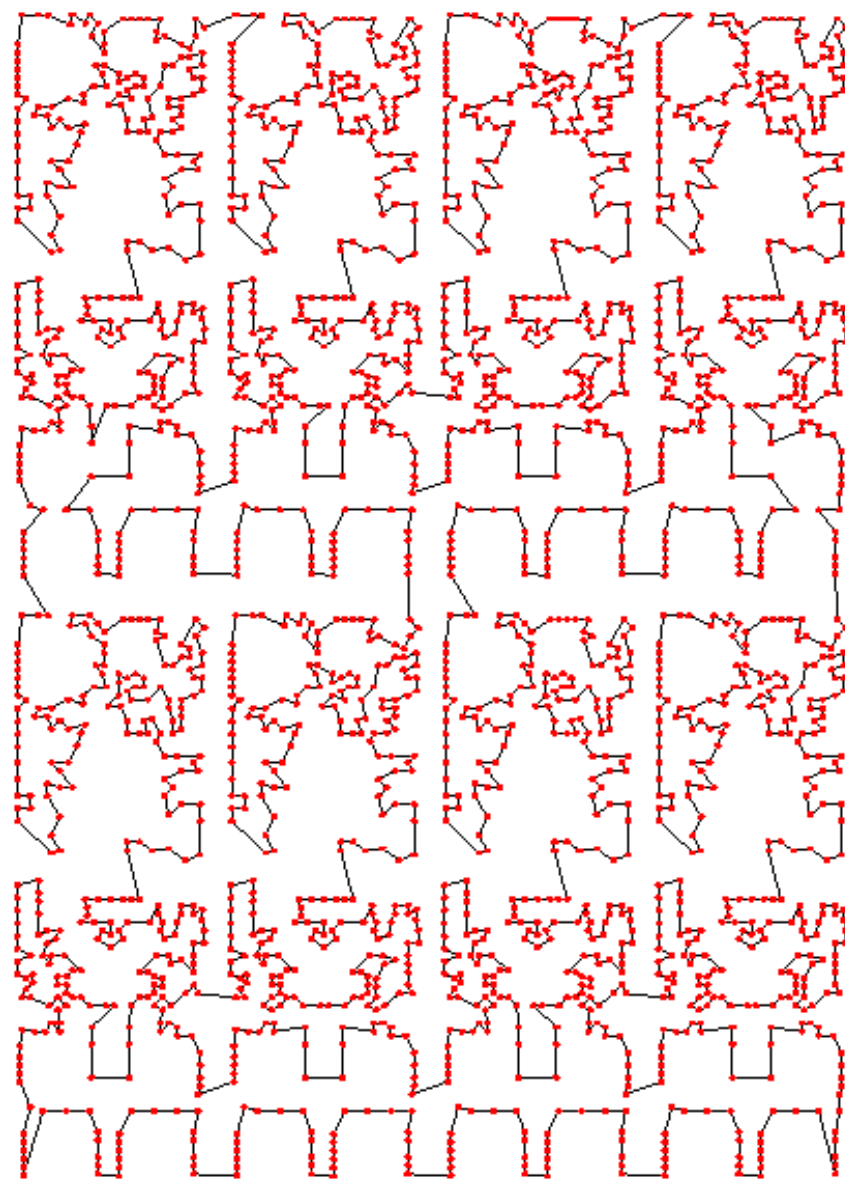
666 Cities Worldwide

- Groetschel and Holland (1987) found the optimal tour of 666 interesting places in the world.



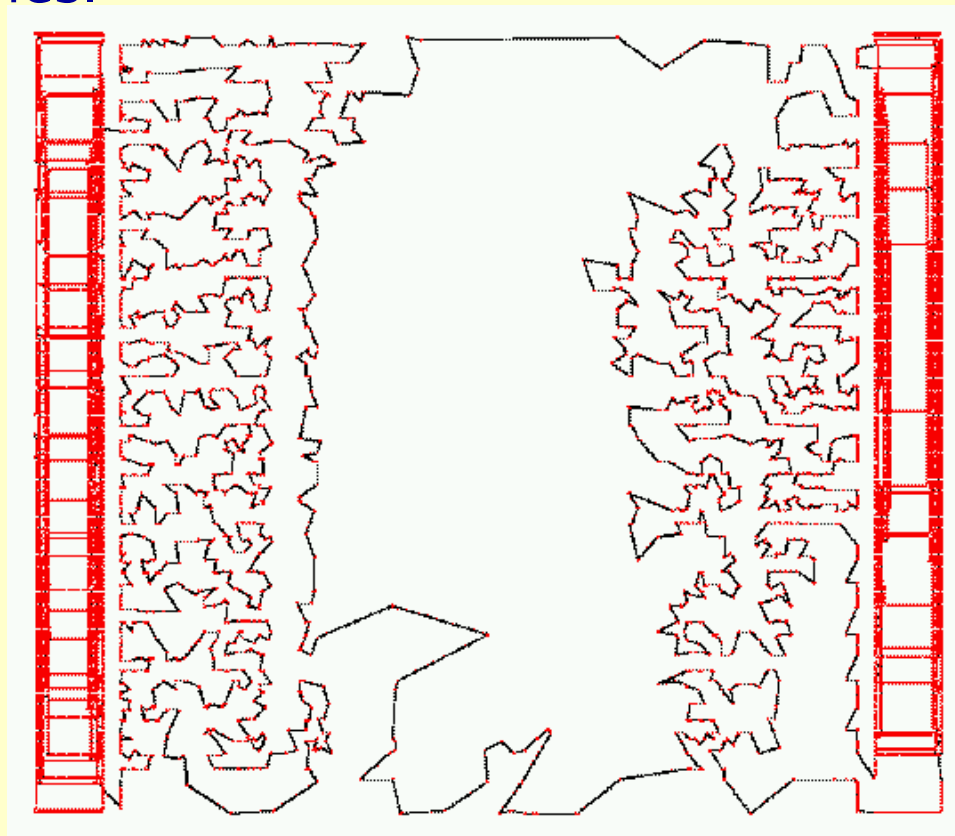
2,392 Points

- Padberg and Rinaldi (1987) found the optimal tour through a layout of 2,392 points obtained from Tektronics Incorporated.



7,397-city TSP

- Applegate, Bixby, Chvátal, and Cook (1994) found the optimal tour for a 7,397-city TSP that arose in a programmable logic array application at AT&T Bell Laboratories.



13509 Cities in the USA

- Applegate, Bixby, Chvátal, and Cook (1998) found the optimal tour of the 13,509 cities in the USA with populations greater than 500.



15112 Cities in Germany

- Applegate, Bixby, Chvátal, and Cook (2001) found the optimal tour of 15,112 cities in Germany.



24978 Swedish Cities

- Applegate, Bixby, Chvátal, Cook, and Helsgaun (2004) found the optimal tour of 24,978 cities in Sweden.



Optimal Tour of Sweden

- In May 2004, the traveling salesman problem of visiting all 24,978 cities in Sweden was solved: a tour of length 855,597 TSPLIB units (approximately 72,500 kilometers) was found and it was proven that no shorter tour exists. This is currently the largest solved TSP instance, surpassing the previous record of 15,112 cities through Germany set in April 2001.

Optimal Tour of Sweden

➤ Research Team

- David Applegate, AT&T Labs - Research
- Robert Bixby, ILOG and Rice University
- Vašek Chvátal, Rutgers University
- William Cook, Georgia Tech
- Keld Helsgaun, Roskilde University

➤ Support for this research was provided by the following grants

- Office of Naval Research Grant N00014-03-1-0040, "Experimental Modules for Combinatorial Optimization and Mixed-Integer Programming"
- National Science Foundation, Grant DMI-0245609, "Local Cuts in Discrete Optimization and Mixed-Integer Programming"

Finding Sweden Tour

- The traveling salesman problem (TSP) asks for the cheapest possible tour through a given collection of cities. Solving the problem means to not only find the best tour but also to prove that no cheaper tour is possible. Early work on the TSP in the 1950s focused exclusively on the this full solution of the problem.
- Starting in the mid-1960s researchers began to study the relaxed version of the TSP where we ask only for a tour of low cost. This task is much easier, but performing it well is an important ingredient in a full (exact) solution method, as well as being an interesting problem in its own right. Indeed, tour finding is a very popular topic, having a large and growing literature devoted to its various aspects. And like the TSP itself, tour finding has led researchers to discover general purpose search techniques that have found application in many domains.
- The Sweden TSP was attacked by a number of groups with some of the top tour-finding methods that have been developed to date. Information on the improvements in the best known tour length can be found in the Sweden Computation Log; the results are summarized in the following table.

Finding Sweden Tour

855618	September 4, 2001	Tour Merging	Cook and Seymour
855612	September 20, 2001	LKH	Helsgaun
855610	September 30, 2001	LKH Merge	Helsgaun
855602	March 16, 2003	Hybrid Genetic	Hung Dinh Nguyen
855597	March 18, 2003	LKH	Helsgaun

- The final improvement in the tour length was made by Keld Helsgaun using a version of his LKH code. This 855,597 value was proved to be optimal by the Concorde TSP code.

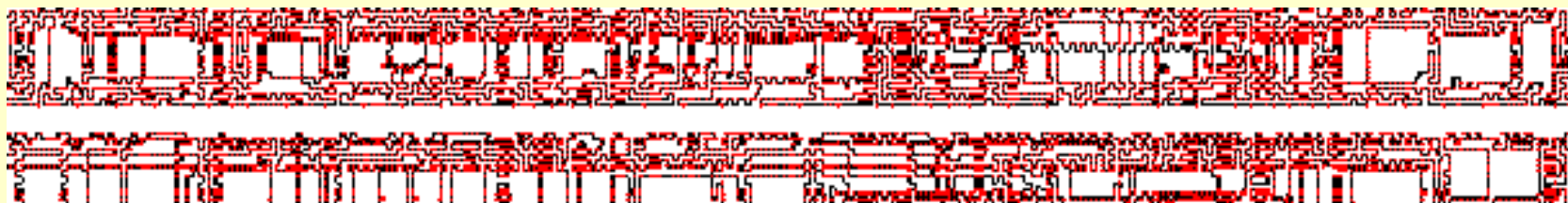
Finding Sweden Tour

- The Concorde solver can accept as an input parameter the value of the best known tour for a TSP instance if one is available. As a full (exact) TSP solver, Concorde is designed to find optimal solutions regardless of the quality of the estimate, but knowledge of a good tour allows for better tuning of parameters that are set in the computer code.
- In the case of the Sweden TSP, the results of the tour-finding attacks guided our choices in approaching the full solution of the problem. Most importantly, the final stages that improved the lower bound from 855,595 up to the optimal value 855,597 required approximately 8 years of computation time (running in parallel on a network of Linux workstations) and without knowledge of the 855,597 tour we would not have made the decision to carry out this final computation.

New record: 85900 cities, 2006

- The largest solved instance of the traveling salesman problem consists of a tour through 85,900 cities in a VLSI application that arose in Bell Laboratories in the late 1980s.
- The computation with Concorde was carried out in 2005/06 and reported in the book *The Traveling Salesman Problem: A Computational Study*. The instance is called pla85900 in Gerd Reinelt's TSPLIB; the shortest possible tour for the problem has length 142,382,641 units.
- With the solution of pla85900, the complete TSPLIB collection of challenge problems has now been successfully solved with the Concorde code.
- <http://www.tsp.gatech.edu/index.html>

Picture of *pla85900* tour



15 year race for better tours

➤	Date	Tour Length	Research Team	Method
➤	07.06.1991	142,514,146	David S. Johnson	Iterated Lin-Kernighan
➤	29.03.1996	142,487,006	Concorde Tour Merging	
➤	23.09.1997	142,482,068	Concorde Tour Merging	
➤	14.10.1998	142,416,327	Keld Helsgaun	LKH
➤	22.10.1999	142,409,553	Concorde Tour Merging	
➤	18.06.2001	142,406,493	Keld Helsgaun	LKH
➤	27.06.2001	142,405,532	Keld Helsgaun	LKH
➤	31.08.2001	142,395,130	Concorde Tour Merging with LKH	
➤	14.12.2001	142,393,738	Keld Helsgaun	LKH
➤	15.09.2002	142,385,237	Hisao Tamaki	Approximate Tour Merging
➤	12.12.2002	142,383,704	Keld Helsgaun	LKH
➤	19.03.2003	142,383,467	Nguyen Dinh Hung	Hybrid Genetic
	Algorithm			
➤	28.04.2003	142,383,189	Keld Helsgaun	LKH
➤	23.12.2003	142,383,011	Keld Helsgaun	LKH
➤	02.05.2004	142,382,641	Keld Helsgaun	LKH

Questions?

Merci à tous !



