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

Discrete Mathematics

Course preparation group:
Nguyễn Khánh Phương
Đỗ Phan Thuận
Phạm Quang Dũng
Huỳnh Thanh Bình
Trần Vĩnh Đức
Bùi Quốc Trung
Đinh Viết Sang
Ban Hà Bằng

Contents of Part 1

PART 1
COMBINATORIAL THEORY
(Lý thuyết tổ hợp)

PART 2
GRAPH THEORY
(Lý thuyết đồ thị)

CONTENTS

1. **Introduction to problem**
2. Brute force
3. Branch and bound

1. Introduction to problem

1.1. General problem

1.2. Traveling salesman problem

1.3. Knapsack problem

State the combinatorial optimization problem

The combinatorial optimization problem in general could be stated as follows:

> Find the min (or max) of function
>
> $f(x) \to \min (\max)$,
>
> with condition:
>
> $x \in D$,
>
> where D is a finite set.

Terminologies:

- $f(x)$ – objective function of problem,
- $x \in D$ - a solution
- D – set of solutions of problem.
- Set D is often described as a set of combinatorial configurations that satisfy given properties.
- Solution $x^* \in D$ having minimum (maximum) value of the objective function is called ***optimal solution***, and the value $f^*=f(x^*)$ is called ***optimzal value*** of the problem.

1.1. General problem

- In many practical application problems of combinatorics, each configuration is assigned to a value equal to the rating of the worth using of the configuration for a particular use purpose.
- Then it appears the problem: Among possible combination configurations, determine the one that the worth using is the best. Such kind of problems is called the combinatorial optimization problem.

1. Introduction to problem

1.1. General problem

1.2. Traveling salesman problem

1.3. Knapsack problem

Traveling Salesman Problem – TSP
(Bài toán người du lịch)

- A salesman wants to travel n cities: 1, 2, 3,…, n.
- *Itinerary is a way of starting from a city, and going through all the remaining cities, each city exactly once, and then back to the starting city.*
- Given c_{ij} is the cost of going from city i to city j (i, j = 1, 2, ..., n),
- Find the itinerary with minimum total cost.

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Traveling Salesman Problem – TSP

- Then, the TSP could be stated as the following combinatorial optimization problem:

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

- One could see that the number of possible itineraries is $n!$, but there are only $(n-1)!$ itineraries if the starting city is fixed.

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Traveling Salesman Problem – TSP

We have a 1-1 correspondence between a *itinerary*

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

and a permutation $\pi = (\pi(1), \pi(2),\dots, \pi(n))$ of n natural numbers 1, 2,..., n.

Set the cost of itinerary:

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

Denote:

Π - set of all permutations of n natural numbers 1, 2, ..., n.

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

1. Introduction to problem

1.1. General problem

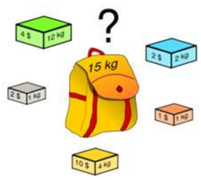
1.2. Traveling salesman problem

1.3. Knapsack problem

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

1.3. Knapsack Problem

- Problem Definition
  - Want to carry essential items in one bag
  - Given a set of items, each has
    - An weight (i.e., 12kg)
    - A value (i.e., 4$)



- Goal
  - To determine the # of each item to include in a collection so that
    - The total weight is less than some given weight that the bag can carry
    - And the total value is as large as possible

0/1 Knapsack Problem

- Problem: John wishes to take n items on a trip
  - The weight of item i is w_i and items are all different (0/1 Knapsack Problem)
  - The items are to be carried in a knapsack whose weight capacity is c
    - When sum of item weights $\leq$ c, all n items can be carried in the knapsack
    - When sum of item weights > c, some items must be left behind
- Which items should be taken/left?

1.3. Knapsack Problem

- Three Types:
  - 0/1 Knapsack Problem
    - restricts the number of each kind of item to zero or one
  - Bounded Knapsack Problem
    - restricts the number of each item to a specific value
  - Unbounded Knapsack Problem
    - places no bounds on the number of each item
- Complexity Analysis
  - The general knapsack problem is known to be NP-hard
    - No polynomial-time algorithm is known for this problem

0/1 Knapsack Problem

- John assigns a profit p_i to item i
  - All weights and profits are positive numbers
- John wants to select a subset of the n items to take
  - The weight of the subset should not exceed the capacity of the knapsack (constraint)
  - Cannot select a fraction of an item (constraint)
  - The profit of the subset is the sum of the profits of the selected items (optimization function)
  - The profit of the selected subset should be maximum (optimization criterion)
- Let $x_i = 1$ when item i is selected and $x_i = 0$ when item i is not selected
  - Because this is a 0/1 Knapsack Problem, you can choose the item or not choose it.

0/1 Knapsack Problem

- A subset of items that John can carry with him can be represented by a binary vector of length n: $x = (x_1, x_2,..., x_n)$, where $x_j = 1$ when item j is selected and $x_j = 0$ when item j is not selected, $j = 1,..,n$
- For each solution x, the profit of carried items is

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

The weight of carried items is

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

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

CONTENTS

1. Introduction to problem
2. **Brute force**
3. Branch and bound

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

0/1 Knapsack Problem

0/1 Knapsack problem could be stated in the form of the following combinatorial optimization problem:

Among binary vectors of length n that satisfy the condition $g(x) \le b$, determine the vector x^* giving the maximum value of objective function $f(x)$:

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

$$A^n = \{(a_1, ..., a_n): a_i \in \{0, 1\}, i=1, 2, ..., n\}.$$

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Method description

- One of the most obvious methods to solve the combinatorial optimization problem is: On the basis of the combinatorial enumeration algorithms, we go through each solution of the problem, and for each solution, we calculate its value of objective function; then compare values of objective functions of all solutions to find the optimal solution whose objective function is minimal (maximal).
- The approach based on such principles is called the brute force.

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Example: 0/1 knapsack problem

- Consider 0/1 knapsack problem

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

$$\text{where } D=\{x=(x_1,x_2,\dots,x_n)\in A^n : \sum_{j=1}^{n} w_j x_j \le b\}$$

- ➢ v_j , w_j, b are positive integers, j=1,2,…, n.
- ➢ Need algorithm to enumerate all elements of set D

Comment

- Brute force is difficult to do even on the most morden super computer. Example to enumerate all

15! = 1 307 674 368 000

permutations on the machine with the calculation speed of 1 billions operations per second, and if to enumerate one permutation requires 100 operations, then we need 130767 seconds > 36 hours!

20! ===> 7645 years

Backtracking: enumerate all possible solutions

- **Construct set S_k:**
  - S_1={ 0, t_1 }, where t_1=1 if $b \ge w_1$; t_1 = 0, otherwise
  - Assume the current partial solution is (x_1, …, x_{k-1}). Then:
    - The remaining capacity of the bag is:

      $b_{k-1}= b - w_1x_1 - \dots - w_{k-1}x_{k-1}$
    - The value of items already in the bag is:

      $f_{k-1}= v_1x_1 + \dots + v_{k-1}x_{k-1}$

  Therefore: S_k = {0, t_k}, where t_k=1 if $b_{k-1} \ge w_k$; t_k = 0, otherwise
- **Implement S_k?**

```
for (y = 0; y++; y <= tk)
```


Comment

- Then, a problem arises that in the process of enumerating all solutions, we need to make use of the found information to eliminate solutions that are definitely not optimal.
- In the next section, we will look at such a search approach to solve the combinatorial optimization problems. In literature, it is called Branch and bound algorithm.

CONTENTS

3.1. General diagram

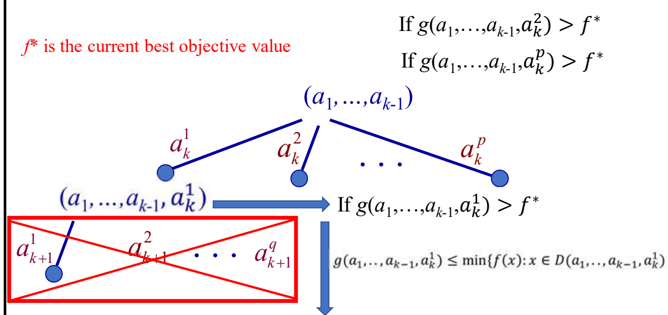
- Branch and bound algorithm consists of 2 procedures:
  - Branching Procedure
  - Bounding Procedure
- **Branching procedure:** The process of partitioning the set of solutions into subsets of size decreasing gradually until the subsets consists only one element.
- **Bounding procedure:** It is necessary to give an approach to calculate the bound for the value of the objective function on each subset A in the partition of the set of solutions.

3. Branch and bound

Cut branch by using lower bound

- Assume we already have function g defined as above. We will use this function to reduce the amount of searching during the process to consider all possible solutions in the backtracking algorithm.
- In the process to enumerate solutions, assume we already obtain some solutions. Thus, denote x^* the solution with objective function is minimum among all solutions obtained so far, and denote $f^* = f(x^*)$
- We call
  - x^* is the current best solution (optimal solution),
  - f^* is the current best value of objective function (optimal objective value).

Cut branch by using lower bound



→all solutions with first k elements as $(a_1,\ldots,a_{k-1},a_k^1)$ certainly have the objective value > f^* ➔ we do not need to browse this branch

$g(a_1,\ldots, a_k)$ is lower bound of partial solution $(a_1,\ldots, a_k)$

Note:

$g(a_1,\ldots, a_k) \le \min\{f(x): x \in D(a_1, \ldots, a_k)\}$ (*)

The construction of g function depends on each specific combinatorial optimization problem. Usually we try to build it so that:

- Calculating the value of g must be simpler than solving the combinatorial optimization problem on the right side of (*).
- The value of $g(a_1, \ldots, a_k)$ must be close to the value of the right side of (*).

Unfortunately, these two requirements are often contradictory in practice.

Branch and bound

```
void  Branch(int k) {
    //Construct x_k from partial soluion (x_1, x_2, ..., x_k-1)
   for  a_k∈A_k
      if   (a_k∈S_k )
      {
          x_k =  a_k;
          if   (k == n)  <Update Record>;
          else if   (g(x_1,..., x_k)  ≤ f*) Branch(k+1);
       }
}
void  BranchAndBound ( ) {
    f* = +∞;
   //if you know any solution x* then set f* = f(x*)
   Branch(1);
   if  (f*  <  +∞)
            <f* is the optimal objective value, x* is optimal solution >
    else  <  problem does not have any solutions >;
}
```


3. Branch and bound

3.1. General diagram

3.2. Example

3.2.1. Traveling salesman problem

3.2.2. Knapsack problem

Sir William Rowan Hamilton
1805 - 1865

Traveling Salesman Problem – TSP

- A salesman wants to travel n cities: 1, 2, 3,…, n.
- *Itinerary is a way of starting from a city, and going through all the remaining cities, each city exactly once, and then back to the starting city.*
- Given c_{ij} is the cost of going from city i to city j (i, j = 1, 2, ..., n),
- Find the itinerary with minimum total cost.

Lower bound function

- Denote

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

  the smallest cost between all pairs of cities.
- We need to evaluate the lower bound for the partial solution $(1, u_2, ..., u_k)$ corresponding to the partial journey that has passed through k cities

  $1 \to u_2 \to . . . \to u_{k-1} \to u_k$

3.2.1. Traveling salesman problem

Fix the starting city as city 1, the TSP leads to the problem:

- Determine the minimum value of

  $f(1,x_2,..., x_n) = c[1,x_2]+c[x_2,x_3]+...+c[x_{n-1},x_n] + c[x_n,1]$

  where

  $(x_2, x_3, ..., x_n)$ is permutation of natural numbers 2, ..., n.

Lower bound function

- The cost need to pay for this partial solution is

  $\sigma = c[1,u_2] + c[u_2, u_3] + ... + c[u_{k-1}, u_k]$.
- To develop it as the complete journey:

1 2 3

$$\underbrace{1 \to u_2 \to . . . \to u_{k-1} \to u_k}_{\text{Cost: } \sigma} \underbrace{\to u_{k+1} \to u_{k+2} \to ..... \to u_n \to 1}_{\text{Cost: } (n-k+1)c_{min}}$$

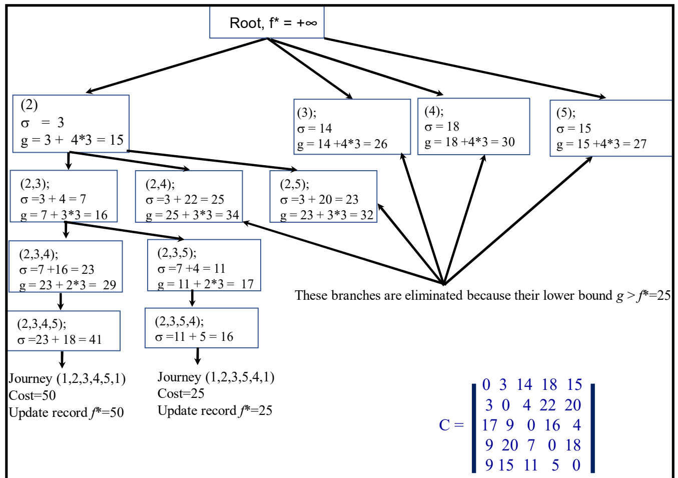
We still need to go through n-k+1 segments, each segment with the cost at least c_{min}, thus the lower bound of the partial solution $(1, u_2, ..., u_k)$ can be calculated by the formula:

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

Example

Give 5 cities {1, 2, 3, 4, 5}. Solve the TSP where the salesman starts from the city 1, and the cost matrix:

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



Example

- We have c_{min} = 3. The process executing the algorithm is described by the solution search tree.
- Information written in each box is the following in order:
  - elements of partial solution,
  - σ is the cost of partial solution,
  - *g* – lower bound of partial solution.

Result

Terminate the algorithm, we obtain:

- Optimal solution (1, 2, 3, 5, 4, 1) correspond to the journey

$$1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 4 \rightarrow 1,$$

- The minimum cost is 25.

3. Branch and bound

3.1. General diagram

3.2. Example

3.2.1. Traveling salesman problem

3.2.2. Knapsack problem

Sir William Rowan Hamilton
1805 - 1865

3.2.2. Knap sack problem

- We have the variable

  x_j – number items type j loaded in the bag, j=1,2, ..., n
- Mathematical model of problem: Find

$$f^* = \max \{ f(x) = \sum_{j=1}^{n} p_j x_j : \sum_{j=1}^{n} w_j x_j \le c, x_j \in Z_+, j = 1, 2, ..., n \}$$

where Z_+ is the set of nonnegative integers

Knapsack problem with integer variables

- Denote D the set of solutions to the problem:

$$D = \{ x = (x_1, ..., x_n) : \sum_{j=1}^{n} p_j x_j \le c, x_j \in Z_+, j = 1, 2, ..., n \}$$

3.2.2. Knap sack problem

- There are n types of items.
- Item type j has
  - weight w_j and
  - profit p_j (j = 1, 2,..., n) .
- We need to select subsets of these items to put it into the bag of capacity c such that the total profit obtained from items loaded in the bag is maximum.

Construct upper bound

- Assume we index the item in the order such that the following inequality is satisfied:

$$v_1 / w_1 \ge v_2 / w_2 \ge . . . \ge v_n / w_n .$$

  (it means items are ordered in descending order of profit per one unit of weight)
- To construct the upper bound function, we consider the following Knapsack continuous variables (KPC): Find

$$g^* = \max \{ f(x) = \sum_{j=1}^{n} p_j x_j : \sum_{j=1}^{n} w_j x_j \le c, \boxed{x_j \ge 0,} j = 1, 2, ..., n \}$$

Construct upper bound function

Proposition. *The optimal solution to the KPC is vector $(x^* = x_1^*, x_2^*, \ldots, x_n^*)$ where elements are determined by the formula:*

$x_1^* = c/w_1,\ x_2^* = x_3^* = \cdots = x_n^* = 0$

and the optimal value is $g^* = v_1 b / w_1$.

Proof. Consider $x = (x_1, \ldots, x_n)$ as a solution to the KPC. Then

$$p_j \le (p_1 / w_1)\, w_j,\ j = 1, 2, \ldots, n$$

as $x_j \ge 0$, we have

$$p_j x_j \le (p_1 / w_1)\, w_j x_j,\ j = 1, 2, \ldots, n.$$

- Therefore

$$\sum_{j=1}^{n} p_j x_j \le \sum_{j=1}^{n} (p_1 / w_1) w_j x_j$$
$$= (p_1 / w_1) \sum_{j=1}^{n} w_j x_j$$
$$\le (p_1 / w_1) c = g^*$$

Proposition is proved.

Calculate the upper bound

- **Note:** When continuing build the $(k+1)$th element of solution, candidates for x_{k+1} are $0, 1, \ldots, [c_k / w_{k+1}]$
- Using the result of the proposition, when selecting value for x_{k+1}, we browse candidates for x_{k+1} in the descending order: $[c_k / w_{k+1}], [c_k / w_{k+1}]-1, \ldots, 1, 0$

Calculate the upper bound

- Now we have the k-level partial solution: $(u_1, u_2, \ldots, u_k)$, then the profit of items currently loaded in the bag is

  $\sigma_k = p_1 u_1 + p_2 u_2 + \ldots + p_k u_k$

  and the remaining capacity of the bag is

  $c_k = c - (w_1 u_1 + w_2 u_2 + \ldots + w_k u_k)$

- We have:

$$\max\{f(x) : x \in D, x_j = u_j, j = 1, 2, \ldots, k\}$$
$$= \max\ \{\sigma_k + \sum_{j=k+1}^{n} p_j x_j : \sum_{j=k+1}^{n} w_j x_j \le c_k,\ x_j \in Z_+, j = k+1, k+2, \ldots, n\}$$
$$\le \sigma_k + \max\ \{\sum_{j=k+1}^{n} p_j x_j : \sum_{j=k+1}^{n} w_j x_j \le c_k,\ x_j \ge 0, j = k+1, k+2, \ldots, n\}$$
$$= \sigma_k + p_{k+1} c_k / w_{k+1}$$

- Thus, we can calculate the upper bound for the partial solution $(u_1, u_2, \ldots, u_k)$ by formula $g(u_1, u_2, \ldots, u_k) = \sigma_k + p_{k+1} c_k / w_{k+1}$

Example

- Solve the knap sack problem using the branch and bound algorithm:

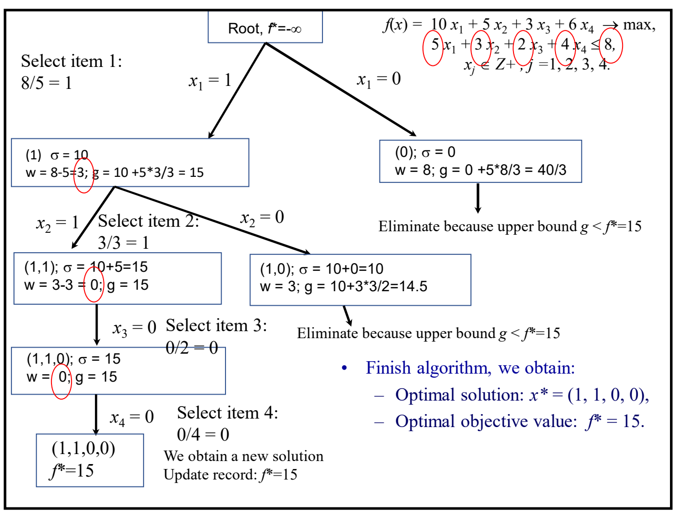
$$f(x) = 10x_1 + 5x_2 + 3x_3 + 6x_4 \to \max,$$
$$5x_1 + 3x_2 + 2x_3 + 4x_4 \le 8,$$
$$x_j \in Z_+,\ j = 1, 2, 3, 4.$$

- Note that in this example, all four items are already sorted in descending order of profit on an unit weight

$$\frac{10}{5} = 2 > \frac{5}{3} \approx 1,66 > \frac{3}{2} = 1,5 = \frac{6}{4}$$

Example

- The process executing the algorithm is described by the solution search tree.
- Information written in each box is the following in order:
  - elements of partial solution,
  - σ is the cost of partial solution (profit of items currently loaded in the bag),
  - w : remaining capacity of the bag,
  - g : upper bound of partial solution.



- Finish algorithm, we obtain:
  - Optimal solution: $x^* = (1, 1, 0, 0)$,
  - Optimal objective value: $f^* = 15$.