Discrete Mathematics

Combinatorial Optimization Problem

Pham Quang Dung

Hanoi, 2012

Outline

- $X = \{x = (x_1, \ldots, x_n) \mid x_i \in A_i, \forall i = 1, \ldots, n\}$ is a set of configurations
- Each configuration is associated with a value representing the quality of that configuration
- Among configurations having a property $\mathcal{P}$, find the configuration with the highest quality

Definition

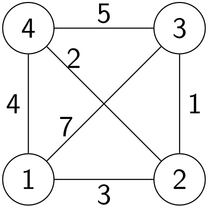
$$\begin{array}{ll} \min(\max) & f(x) \\ & x \in D \\ & D \subseteq X \text{ is the set configurations } x \in X \text{ having property } \mathcal{P} \end{array}$$

Introduction

- Function f is called objective function
- $x^* \in D$ such that $f(x^*)$ is minimal (maximal) is called optimal solution
- $f^* = f(x^*)$ is called optimal objective value

Example: Traveling Salesman Problem

- Given a list of n cities with pairwise distances
- Find the shortest route that visits each city exactly once and returns to the origin city
- $x = (x_1, \ldots, x_n)$, route is $x_1 \to x_2 \to \cdots \to x_n \to x_1$
- $f(x) = c(x_1, x_2) + c(x_2, x_3) + \cdots + c(x_n, x_1)$



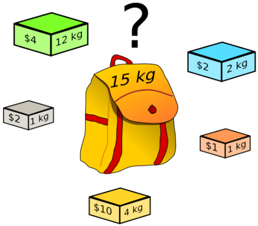
$$c = \begin{pmatrix} 0 & 3 & 7 & 4 \\ 3 & 0 & 1 & 2 \\ 7 & 1 & 0 & 5 \\ 4 & 2 & 5 & 0 \end{pmatrix}$$

Example: Knapsack Problem

- Given a set of n kind of items 1, ..., n. Each item kind i has a weight a_i and a value c_i
- Compute how many items of each kind to take such that the total weight does not exceed a given value b and the total value is maximal

Definition

$$\begin{array}{ll} \text{max} & \sum_{i=1}^{n} c_i x_i \\ \text{s.t.} & \sum_{i=1}^{n} a_i x_i \leq b,\ x_i \in N, \forall i \in \{1, \ldots, n\} \end{array}$$



source: http://en.wikipedia.org/wiki/Knapsack_problem

Assignment Problem

- There n agents and n tasks. Each agent i can perform a task j with a cost $c(i, j)$
- All tasks must be performed, each by exactly one agent and an agent must perform exactly one task
- Find an assignment agent-task for accomplishing all the tasks such that the total cost is minimal

Definition

- a configuration is $x = (x_1, \ldots, x_n)$: agent i performs taks x_i
- cost $f(x) = \sum_{i=1}^{n} c(i, x_i)$

Outline

Backtracking algorithm

Algorithm 1: BackTracking(k)

Construct a candidate set S_k;
foreach $y \in S_k$ **do**
    $a_k \leftarrow y$;
    if $(a_1, \ldots, a_k)$ *is a complete configuration* **then**
        $\overline{f} \leftarrow f(a_1, \ldots, a_k)$;
        if $\overline{f} < f^*$ **then**
            $f^* \leftarrow \overline{f}$;
    else
        BackTracking$(k+1)$;

Algorithm 2: Main()

$f^* \leftarrow \infty$;
BackTracking(1);

Branch-and-Bound algorithm

- Base on Backtracking algorithm
- Suppose we have a partial solution $\overline{x} = (a_1, \ldots, a_k)$, we must decide whether or not to continue to extend $\overline{x}$ until complete solutions
  - Identify a lower bound function $g(\overline{x})$:

$$g(a_1, \ldots, a_k) \leq min\{f(x) \mid x \in D, x_i = a_i, \forall i = 1, \ldots, k\}$$

  - If $g(\overline{x}) < f^*$ then continue to extend. Otherwise, backtrack.

Branch-and-Bound algorithm

Algorithm 3: BackTracking(k)

Construct a candidate set S_k;
foreach $y \in S_k$ **do**
    $a_k \leftarrow y$;
    if $(a_1, \ldots, a_k)$ *is a complete configuration* **then**
        $\overline{f} \leftarrow f(a_1, \ldots, a_k)$;
        if $\overline{f} < f^*$ **then**
            $f^* \leftarrow \overline{f}$;
    else
        if $g(a_1, \ldots, a_k) < f^*$ **then**
            BackTracking$(k+1)$;

Algorithm 4: Main()

$f^* \leftarrow \infty$;
BackTracking(1);

Knapsack Problem

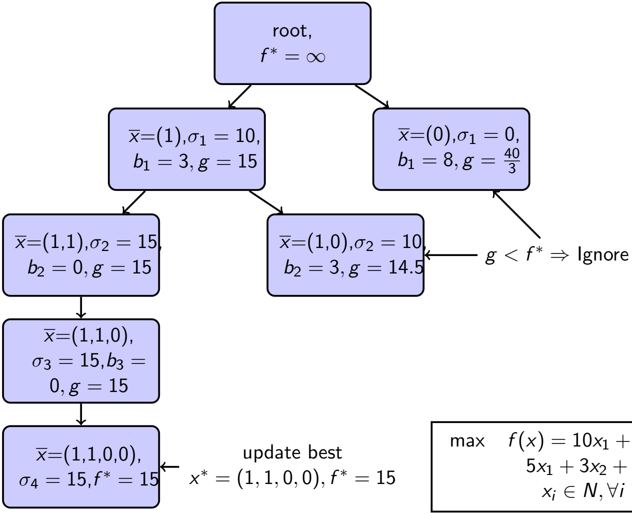
Definition

max $\sum_{i=1}^{n} c_i x_i$

s.t. $\sum_{i=1}^{n} a_i x_i \leq b, x_i \in N, \forall i \in \{1, \dots, n\}$

- Without loss of generality, suppose that $\frac{c_1}{a_1} \geq \frac{c_2}{a_2} \geq \cdots \geq \frac{c_n}{a_n}$
- Observation:
  $\max\{f(x) = \sum_{i=1}^{n} c_i x_i : \sum_{i=1}^{n} a_i x_i \leq b, x_i \in N\} \leq$
  $\max\{f(x) = \sum_{i=1}^{n} c_i x_i : \sum_{i=1}^{n} a_i x_i \leq b, x_i \geq 0\} = \frac{b * c_1}{a_1}$
- $\sigma_k = \sum_{i=1}^{k} c_i \alpha_i, \; b_k = b - \sum_{i=1}^{k} a_i \alpha_i$
- $g(\alpha_1, \dots, \alpha_k) =$
  $\max\{f(x) : \sum_{i=1}^{n} a_i x_i \leq b, x \geq 0, x_i = \alpha_i, \forall i \in \{1, \dots, k\}\} =$
  $\sigma_k + \frac{b_k * c_{k+1}}{a_{k+1}}$

Knapsack Problem

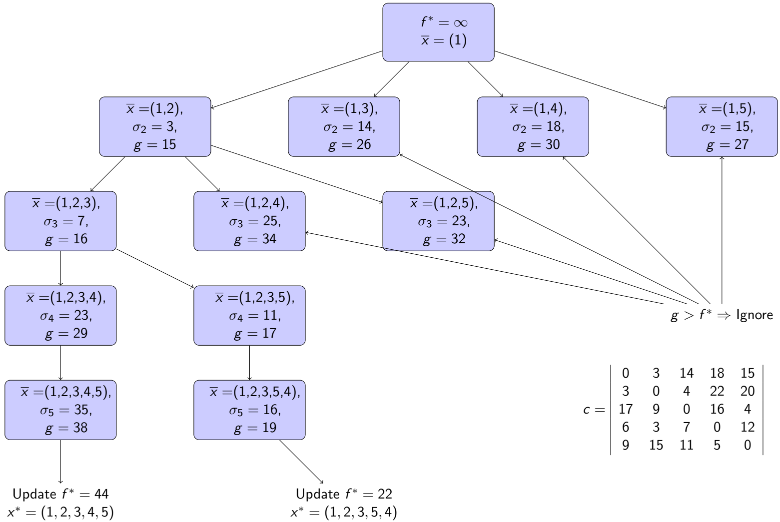


$$\begin{aligned} \max \quad & f(x) = 10x_1 + 5x_2 + 3x_3 + 6x_4 \\ & 5x_1 + 3x_2 + 2x_3 + 4x_4 \leq 8 \\ & x_i \in N, \forall i \in \{1, \ldots, 4\} \end{aligned}$$

TSP

- $Cmin = \min\{c(i,j) : \forall i \neq j \in \{1,\dots,n\}\}$
- Suppose we have a partial solution $\overline{x} = (\alpha_1,\dots,\alpha_k)$
- $\sigma_k = c(\alpha_1,\alpha_2) + c(\alpha_2,\alpha_3) + \cdots + c(\alpha_{k-1},\alpha_k)$
- $g(\alpha_1,\dots,\alpha_k) = \sigma_k + (n-k+1) * Cmin$

TSP

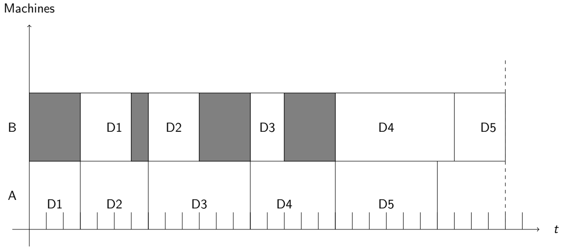


$$c = \begin{vmatrix} 0 & 3 & 14 & 18 & 15 \\ 3 & 0 & 4 & 22 & 20 \\ 17 & 9 & 0 & 16 & 4 \\ 6 & 3 & 7 & 0 & 12 \\ 9 & 15 & 11 & 5 & 0 \end{vmatrix} \qquad Cmin = 3$$

Outline

Johnson Algorithm for scheduling on two machines

- There are n jobs, each job must be processed consecutively on two machines: A and then B
- a_i and b_i are processing time of the job i on machine A and B
- Compute the processing schedule such that the completion time is minimized

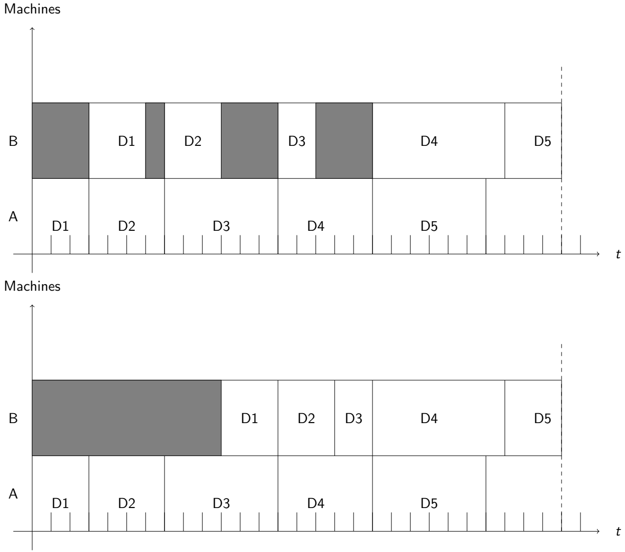


	D_1	D_2	D_3	D_4	D_5
A	3	4	6	5	6
B	3	3	2	7	3

Johnson Algorithm for scheduling on two machines problem

- A schedule is modeled by a permutation $\pi = (\pi_1, \dots, \pi_n)$ of $1, \dots, n$
- Denote s_{jX}, t_{jX} respectively the starting and finishing times of the processing of the job j on machine X, $\forall j \in \{1, \dots, n\}, X \in \{A, B\}$
- Constraints
  1. $t_{\pi_k A} \leq s_{\pi_{k+1} A}, \forall k = 1, \dots, n-1$
  2. $t_{\pi_k A} \leq s_{\pi_k B}, \forall k = 1, \dots, n$
  3. $s_{\pi_k B} \geq \max\{t_{\pi_k A}, t_{\pi_{k-1} B}\}$
- Completion time $T(\pi) = t_{\pi_n B}$
- Optimality is achieved iff the equalities in constraints 1, 2, and 3 are achieved

Johnson Algorithm for scheduling on two machines



Johnson Algorithm for scheduling on two machines

- $T(\pi) = d_B(\pi) + \sum_{j=1}^{n} b_j$
- $d_B(\pi)$ is the sum of idle times of the machine B plus $t_{\pi_1 A}$
- $$d_B(\pi) = \max_{1 \leq u \leq n} \Delta_u(\pi)$$

  where:

  - $$\Delta_1(\pi) = a_{\pi_1}$$
  - $$\Delta_u(\pi) = \sum_{j=1}^{u} a_{\pi_j} - \sum_{j=1}^{u-1} b_{\pi_j}, \forall u = 2, \ldots, n$$
- The original problem is reduced to $\min\{d_B(\pi) : \pi \in P\}$, P is the set of permutations of $1, \ldots, n$

Johnson Algorithm for scheduling on two machines problem

Lemma

$\pi = (\pi_1, \ldots, \pi_{k-1}, \pi_k, \pi_{k+1}, \ldots, \pi_n)$ and $\pi' = (\pi_1, \ldots, \pi_{k-1}, \pi_{k+1}, \pi_k, \ldots, \pi_n)$. If $min(a_{\pi_k}, b_{\pi_{k+1}}) \leq min(a_{\pi_{k+1}}, b_{\pi_k})$, then $T(\pi) \leq T(\pi')$

Proof.

- π and π' are different at only position k and $k+1$, so $\Delta_u(\pi) = \Delta_u(\pi'), \forall u = 1, \ldots, k-1, k+2, \ldots, n$
- To prove the lemma, we prove that
  - $\max(\Delta_k(\pi), \Delta_{k+1}(\pi)) \leq \max(\Delta_k(\pi'), \Delta_{k+1}(\pi'))$
  - $\Leftrightarrow \max(\Delta_k(\pi) - \delta, \Delta_{k+1}(\pi) - \delta) \leq \max(\Delta_k(\pi') - \delta, \Delta_{k+1}(\pi') - \delta)$ with $\delta = \sum_{j=1}^{k+1} a_{\pi(j)} - \sum_{j=1}^{k-1} b_{\pi(j)}$
  - $\Leftrightarrow \max(-a_{\pi(k+1)}, -b_{\pi(k)}) \leq \max(-a_{\pi(k)}, -b_{\pi(k+1)})$
  - $\Leftrightarrow \min(a_{\pi(k)}, b_{\pi(k+1)}) \leq \min(a_{\pi(k+1)}, b_{\pi(k)})$ (this is the original hypothesis)

□

Johnson Algorithm for scheduling on two machines problem

Lemma

If $min(a_i, b_j) \leq min(a_j, b_i)$ *and* $min(a_j, b_k) \leq min(a_k, b_j)$, *then* $min(a_i, b_k) \leq min(a_k, b_i)$

Proof.

The lemma can trivially proved by considering all 16 possible cases □

Johnson Algorithm for scheduling on two machines problem

Theorem

$T(\pi)$ is minimized when
$min(a_{\pi_k}, b_{\pi_{k+1}}) \leq min(a_{\pi_{k+1}}, b_{\pi_k}), \forall k = 1, \ldots, n-1$

Proof.

- Suppose that $\pi' = (\pi'_1, \pi'_2, \ldots, \pi'_n)$ is an optimal solution
- If there exists $1 \leq k \leq n-1$ such that $\min(a_{\pi'_k}, b_{\pi'_{k+1}}) > \min(a_{\pi'_{k+1}}, b_{\pi'_k})$, then by swapping π'_k and π'_{k+1}, we obtain a new solution which is not worse than π' and is thus an optimal solution
- The process is repeated until we obtain an optimal solution an optimal solution π having $\min(a_{\pi_k}, b_{\pi_{k+1}}) \leq \min(a_{\pi_{k+1}}, b_{\pi_k}), \forall k = 1, \ldots, n-1$

$\square$

Johnson Algorithm for scheduling on two machines

1. Divide the jobs into two groups:
   - N_1 contains jobs i such that $a_i \leq b_i$
   - N_2 contains jobs i such that $a_i > b_i$
2. Sort jobs of N_1 in the increasing order of a_i which results in a sequence of jobs L_1
3. Sort jobs of N_2 in the decreasing order of b_i which results in a sequence of jobs L_2
4. The concatenation L_1::L_2 is an optimal solution to the problem

Example

	D_1	D_2	D_3	D_4	D_5
A	3	4	6	5	6
B	3	3	2	7	3

- $N_1 = \{D_1, D_4\}$, $N_2 = \{D_2, D_3, D_5\}$
- Sort: $L_1 = \langle D_1, D_4 \rangle$, $L_2 = \langle D_2, D_5, D_3 \rangle$
- Optimal solution is $\langle D_1, D_4, D_2, D_5, D_3 \rangle$, completion time $= 26$