

# Lecture 4: Dynamic Programming

(3 units)

## Outline

- ▶ Shortest paths
- ▶ Principle of optimality
- ▶ 0-1 knapsack problem
- ▶ Integer knapsack problem
- ▶ Uncapacitated lot-sizing

# Shortest paths

- Consider a digraph  $G = (V, A)$  with nonnegative arc distance  $c_{ij}$ .

Problem: find the shortest path from  $s$  to  $t$ .

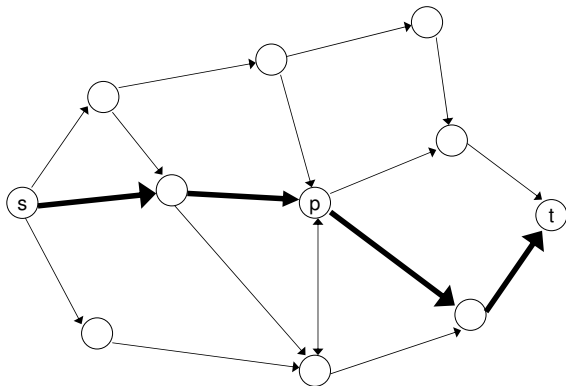


Figure: s-t shortest path

- ▶ Naive approach:
  - ▶ Find all path from  $s$  to  $t$ ;
  - ▶ Evaluate the length of each  $s - t$  path
  - ▶ Find the minimum length
  - ▶ The number of  $s - t$  path increases exponentially. Infeasible for large graph
- ▶ **Observation:** If the  $s - t$  shortest path passes by node  $p$ , the subpaths  $(s, p)$  and  $(p, t)$  are shortest path from  $s$  to  $p$  and from  $p$  to  $t$  respectively.
- ▶  $d(v)$ =length of shortest path from  $s$  to  $v$ . Then

$$d(v) = \min_{i \in V^-(v)} \{d(i) + c_{iv}\}.$$

- ▶ For  $i \neq s$ , the **complexity** of calculating  $d(v)$  is  $O(m)$ .
- ▶ Suppose  $|V| = n$  and  $|A| = m$ . Order the nodes such that  $i < j$  for all  $(i, j) \in A$ .
- ▶  $D_k(j)$  = length of shortest path from  $s$  to  $j$  containing at most  $k$  arcs. Then

$$D_k(j) = \min\{D_{k-1}(j), \min_{i \in V^-(j)} [D_{k-1}(i) + c_{ij}]\}.$$

- ▶ Increasing  $k$  from 1 to  $n - 1$ , we obtain the shortest path.  
**Complexity** of the algorithm:  $O(mn)$ .

Example: Find the shortest path from *A* to *J*.

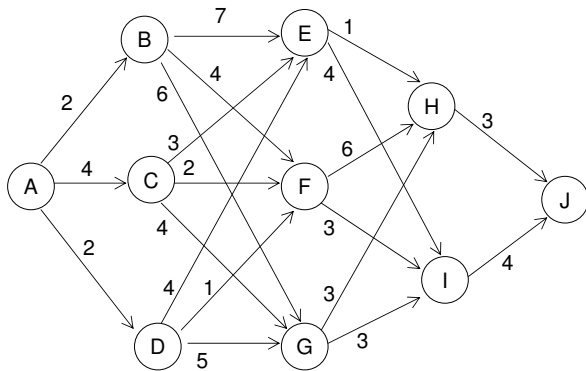


Figure: Shortest path from *A* to *J*

- Forward dynamic programming. Let  $j$  be a node at stage  $k$ ,  $D_k(j)$  be the shortest distance from  $A$  to  $j$ .

$$D_k(j) = \min\{D_{k-1}(i) + c_{ij} \mid i \in \text{stage } k-1\},$$

- $k = 0, 1, 2, 3, 4$ ,  $f^* = D_4(J)$ .

$$D_0(A) = 0.$$

$$D_1(B) = 2, D_2(C) = 4, D_2(D) = 2.$$

$$D_2(E) = \min_{j \in V-(E)}(D_1(j) + c_{jE}) = \min\{2+7, 4+3, 2+4\} = 6.$$

$$D_2(F) = \min_{j \in V-(F)}(D_1(j) + c_{jF}) = \min\{2+4, 4+2, 2+1\} = 3.$$

$$D_2(G) = \min_{j \in V-(G)}(D_1(j) + c_{jG}) = \\ \min\{2+6, 4+4, 2+5\} = 7.$$

$$D_3(H) = \min_{j \in V-(H)}(D_2(j) + c_{jH}) = \\ \min\{6+1, 3+6, 7+3\} = 7.$$

$$D_3(I) = \min_{j \in V-(I)}(D_2(j) + c_{jI}) = \min\{6+4, 3+3, 7+3\} = 6.$$

$$D_4(J) = \min_{j \in V-(J)}(D_3(j) + c_{jJ}) = \min\{7+3, 6+4\} = 10.$$

- ▶ Starting from  $D_4(J)$ , we have:

$$\begin{aligned} D_4(J) &= D_3(H) + C_{HJ} = D_2(E) + c_{EH} + c_{HJ} \\ &= D_1(D) + c_{DE} + c_{EH} + c_{HJ} \\ &= c_{AD} + c_{DE} + c_{EH} + c_{HJ}; \\ \text{or } D_4(J) &= D_3(I) + c_{IJ} = D_2(F) + c_{FI} + c_{IJ} \\ &= D_1(D) + c_{DF} + c_{FI} + c_{IJ} \\ &= c_{AD} + c_{DF} + c_{FI} + c_{IJ}. \end{aligned}$$

- ▶ We can backtrack the **Shortest path**:
  - (1)  $J \leftarrow H \leftarrow E \leftarrow D \leftarrow A$ .
  - (2)  $J \leftarrow I \leftarrow F \leftarrow D \leftarrow A$ .
- ▶ How does it compare with total enumeration?  
 $18 * 3 + 17 = 71 > 5 * 5 + 3 = 28$

- ▶ Backward dynamic programming. Let  $D_k(j)$  be the shortest distance from node  $j$  at stage  $k$  to the destination  $J$ .

$$D_k(j) = \min\{D_{k+1}(i) + c_{ji} \mid i \in \text{stage } k + 1\},$$

$k = 4, 3, 2, 1, 0$ .

$f^* = D_0(A)$ ,  $D_4(J) = 0$ .

(Exercise: Try to solve the example by backward DP.)



# Principle of optimality

- ▶ **Richard Bellman** (1952) introduced the principle of optimality and dynamic programming.

*Richard Bellman on the Birth of Dynamic Programming.*

*Stuart Dreyfus. Operations Research, Vol. 50, No. 1, 48-51, 2002*



**Figure:** Richard Bellman (1920-1984)

- ▶ **Principle of optimality:**  
*Given an optimal sequence of decisions or choices, each subsequence must also be optimal.*
- ▶ Principle of optimality applies to multi-stage decision making problem
- ▶ A large number of optimization problems satisfy this principle.
- ▶ It applies to a problem (not an algorithm)
- ▶ **States:** nodes for which values need to be calculated ( $j$ )
- ▶ **Stages:** steps which define the ordering

## 0-1 Knapsack Problem

- ▶ 0-1 Knapsack Problem

$$f^* = \max\{c^T x \mid a^T x \leq b, x \in \{0, 1\}^n\}.$$

- ▶ Define  $1 \leq k \leq n$  as stage,  $0 \leq \lambda \leq b$  as state
- ▶ Optimal value function:

$$f_k(\lambda) = \max\left\{\sum_{j=1}^k c_j x_j \mid \sum_{j=1}^k a_j x_j \leq \lambda, x \in \{0, 1\}^k\right\}$$

- ▶  $f^* = f_n(b)$ .
- ▶ Recursive equation:

$$f_k(\lambda) = \max\{f_{k-1}(\lambda), c_k + f_{k-1}(\lambda - a_k)\}.$$

- ▶ Initial conditions:  $f_0(\lambda) = 0$ , or  $f_1(\lambda) = 0$  if  $0 \leq \lambda \leq a_1$  and  $f_1(\lambda) = \max(c_1, 0)$  for  $\lambda \geq a_1$ .
- ▶ How to find the optimal solution? Let  $p_k(\lambda) = 0$  if  $f_k(\lambda) = f_{k-1}(\lambda)$ , or 1 otherwise.
- ▶ Complexity:  $O(nb)$ .

## An example of 0-1 KP

$$\begin{aligned} \max & 10x_1 + 7x_2 + 25x_3 + 24x_4 \\ \text{s.t. } & 2x_1 + x_2 + 6x_3 + 5x_4 \leq 7, \\ & x \in \{0, 1\}^4 \end{aligned}$$

	$f_1$	$f_2$	$f_3$	$f_4$	$p_1$	$p_2$	$p_3$	$p_4$
$\lambda = 0$	0	0	0	0	0	0	0	0
1	0	7	7	7	0	1	0	0
2	10	10	10	10	1	0	0	0
3	10	17	17	17	1	1	0	0
4	10	17	17	17	1	1	0	0
5	10	17	17	24	1	1	0	1
6	10	17	25	31	1	1	1	1
7	10	17	32	34	1	1	1	1

Thus  $f_4(7) = 34$ .  $p_4(7) = 1$ , so  $x_4^* = 1$ ,  
 $p_3(7 - 5) = p_3(2) = p_2(2) = 0$ , so  $x_3^* = x_2^* = 0$ ,  $p_1(2) = 1$ , so  
 $x_1^* = 1$ .

# Integer Knapsack Problem

- ▶ Consider the **integer** knapsack problem:

$$f^* = \max \left\{ \sum_{j=1}^n c_j x_j \mid \sum_{j=1}^n a_j x_j \leq b, x \in \mathbb{Z}_+^n \right\},$$

where  $c_j > 0$ ,  $a_j > 0$ ,  $j = 1, \dots, n$ .

- ▶ Define

$$g_r(\lambda) = \max \left\{ \sum_{j=1}^r c_j x_j \mid \sum_{j=1}^r a_j x_j \leq \lambda, x \in \mathbb{Z}_+^r \right\}.$$

Then  $z = g_n(b)$ .

- ▶ Recursive equation:

$$g_r(\lambda) = \max_{t=0,1,\dots,\lfloor \lambda/a_r \rfloor} \{c_r t + g_{r-1}(\lambda - t a_r)\}.$$

For  $r = 1, \dots, n$  and  $\lambda = 1, \dots, b$ , calculating  $g_r(\lambda)$  needs at most  $O(b)$ , so the **complexity** of computing  $g_n(b)$  is  $O(nb^2)$ .

- ▶ **Question:** Can we do better?
- ▶ We have the following two cases for  $x_r^*$ :
  - (i)  $x_r^* = 0$ ,  $g_r(\lambda) = g_{r-1}(\lambda)$ ;
  - (ii)  $x_r^* \geq 1$ , then  $x_r^* = 1 + t$  with  $t \geq 0$  integer  
 $\Rightarrow (x_1^*, \dots, x_{r-1}^*, t)$  is optimal to the knapsack problem with  $r$  stages and  $\lambda - a_r$  resource  
 $\Rightarrow g_r(\lambda) = c_r + g_r(\lambda - a_r)$ . Thus

$$g_r(\lambda) = \max\{g_{r-1}(\lambda), c_r + g_r(\lambda - a_r)\}.$$

- ▶ **Complexity:**  $O(nb)$ .

## An example of integer knapsack problem

$$\begin{aligned} \max \quad & 7x_1 + 9x_2 + 2x_3 + 15x_4 \\ \text{s.t.} \quad & 3x_1 + 4x_2 + x_3 + 7x_4 \leq 10, \\ & x \in \mathbb{Z}_+^4. \end{aligned}$$

	$g_1$	$g_2$	$g_3$	$g_4$
$\lambda = 0$	0	0	0	0
1	0	0	2	2
2	0	0	4	4
3	7	7	7	7
4	7	9	9	9
5	7	9	11	11
6	14	14	14	14
7	14	16	16	16
8	14	18	18	18
9	21	21	21	21
10	21	23	23	23

$$f^* = g_4(10) = 23 \text{ with } x^* = (2, 1, 0, 0)^T.$$

# Uncapacitated lot-sizing

- Uncapacitated lot-sizing problem is to determine a minimum cost production and inventory holding schedule for a product so as to satisfy its demand over a finite discrete-time planning horizon.





- ▶  $f_t$ : setup (fixed) cost of production in period  $t$ ;
- ▶  $p_t$ : the unit production cost in period  $t$ ;
- ▶  $h_t$ : the unit storage cost in period  $t$ ;
- ▶  $d_t$ : the demand in period  $t$ ;
- ▶  $x_t$ : the production in period  $t$ ;
- ▶  $s_t$ : the stock at the end of period  $t$ ,  $s_0 = 0$ ,  $s_n = 0$ .

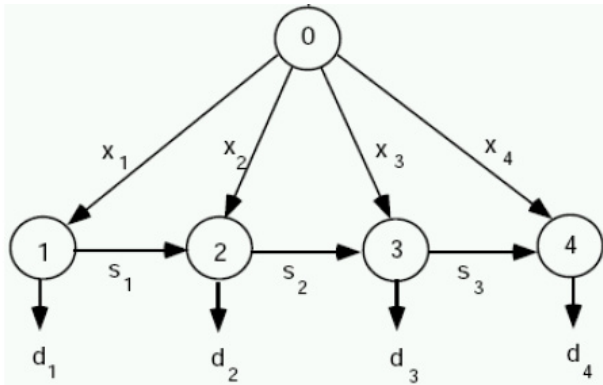


Figure: Uncapacitated lot-sizing network ( $n = 4$ )

- ▶ ULS can be viewed as a **fixed-charge network flow** problem in which one must choose which arcs  $(0, t)$  to open (which are the production periods), and then find the minimum cost flow through the network.
- ▶ Integer program model:

$$\begin{aligned}
 \min \quad & \sum_{t=1}^n p_t x_t + \sum_{t=1}^n h_t s_t + \sum_{t=1}^n f_t y_t \\
 \text{s.t.} \quad & s_{t-1} + x_t = d_t + s_t, \quad t = 1, \dots, n, \\
 & x_t \leq M y_t, \quad t = 1, \dots, n, \\
 & s_0 = 0, s_n = 0, \quad s_t, x_t \geq 0, y_t \in \{0, 1\}, \quad t = 1, \dots, n.
 \end{aligned}$$

where  $M > 0$  is a sufficiently large number.

► Basic properties:

- There exists an optimal solution with  $s_{t-1}x_t = 0$ .
- There exists an optimal solution such that if  $x_t > 0$ , then  $x_t = \sum_{i=t}^{t+k} d_i$  for some  $k$ .

- Denote  $d_{it} = \sum_{j=i}^t d_j$ . Since  $s_t = \sum_{i=1}^t x_i - \sum_{i=1}^t d_i$ , the objective function can be expressed as

$$\sum_{t=1}^n p_t x_t + \sum_{t=1}^n h_t s_t + \sum_{t=1}^n f_t y_t = \sum_{t=1}^n c_t x_t + \sum_{t=1}^n f_t y_t - \sum_{t=1}^n h_t d_{1t},$$

where  $c_t = p_t + \sum_{i=t}^n h_i$ .

- Since  $-\sum_{t=1}^n h_t d_{1t}$  is a constant, it can be removed from the objective.

- ▶ Let  $H(k)$  be the minimum cost of a solution for period  $1, \dots, k$ . If  $t \leq k$  is the **last** period in which production occurs, then  $x_t = \sum_{i=t}^k d_i$ , and the cost is  $H(t-1) + f_t + c_t d_{tk}$ .
- ▶ Forward recursion (Wagner-Whitin algorithm):

$$H(k) = \min_{1 \leq t \leq k} \{H(t-1) + f_t + c_t d_{tk}\},$$

with  $H(0) = 0$ .

- ▶ Calculating  $H(k)$  for  $k = 1, 2, \dots, n$ , we obtain the optimal value  $H(n)$ .
- ▶ The **complexity** of calculating  $H(n)$  is  $O(n^2)$ .

- Example:  $n = 4$ ,  $d = (2, 4, 5, 1)$ ,  $p = (3, 3, 3, 3)$ ,  
 $h = (1, 2, 1, 1)$ ,  $f = (12, 20, 16, 8)$ . Then  $c = (8, 7, 5, 4)$ .  
 $H(1) = H(0) + f_1 + c_1 d_{11} = 0 + 12 + 8 * 2 = 28$ .

$$\begin{aligned} H(2) &= \min \begin{cases} H(0) + f_1 + c_1 d_{12} = 0 + 12 + 8 * 6 = 60 \\ H(1) + f_2 + c_2 d_{22} = 28 + 20 + 7 * 4 = 76 \end{cases} \\ &= 60. \end{aligned}$$

$$\begin{aligned} H(3) &= \min \begin{cases} H(0) + f_1 + c_1 d_{13} = 0 + 12 + 8 * 11 = 100 \\ H(1) + f_2 + c_2 d_{23} = 28 + 20 + 7 * 9 = 111 \\ H(2) + f_3 + c_3 d_{33} = 60 + 16 + 5 * 5 = 101 \end{cases} \\ &= 100. \end{aligned}$$

$$\begin{aligned} H(4) &= \min \begin{cases} H(0) + f_1 + c_1 d_{14} = 0 + 12 + 8 * 12 = 108 \\ H(1) + f_2 + c_2 d_{24} = 28 + 20 + 7 * 10 = 118 \\ H(2) + f_3 + c_3 d_{34} = 60 + 16 + 5 * 6 = 106 \\ H(3) + f_4 + c_4 d_{44} = 100 + 8 + 4 * 1 = 112 \end{cases} \\ &= 106. \end{aligned}$$

So the optimal solution is:  $y = (1, 0, 1, 0)$ ,  $x = (6, 0, 6, 0)$ .