

Dynamic Programming

Jesper Larsen & Jens Clausen & David Pisinger
jesla, jenc1, pisinger@man.dtu.dk

DTU Management
Technical University of Denmark

The 0-1 Knapsack problem

Let us consider the 0-1 Knapsack problem.

$$\begin{array}{ll} \max & \sum_{j=1}^n p_j x_j \\ \text{s.t.} & \sum_{j=1}^n w_j x_j \leq c \\ & x \in \{0, 1\}^n \end{array}$$

where the p_j , w_j and c are positive integers.

j	1	2	3	4	5	
p_j	5	5	2	6	1	$c = 14$
w_j	6	7	3	7	2	

Dynamic Programming

General **paradigm**, that can be used when

- Problem can be split into subproblems
- Overlapping subproblems

By tabulating solutions to subproblems, we can improve time complexity for solving the problem

DP for 0-1 Knapsack

- Now define the related problem $P_r(\lambda)$ as:

$$\begin{array}{ll} f_r(\lambda) = & \max \sum_{j=1}^r p_j x_j \\ \text{s.t.} & \sum_{j=1}^r w_j x_j \leq \lambda \\ & x \in \{0, 1\}^r \end{array}$$

Here λ can take on the values $1, \dots, c$.

- Observe that $f_n(c) = z$, that is, $f_n(c)$ gives us the optimal solution to the 0-1 knapsack problem.
- Observe that $f_0(\lambda) = 0$, for all λ

j	1	2	3	4	5	
p_j	5	5	2	6	1	$c = 14$
w_j	6	7	3	7	2	

Knapsack Problem, example

Instance

j	1	2	3	4	5
p_j	5	5	2	6	1
w_j	6	7	3	7	2

$c = 14$

Dynamic programming table

λ	$r = 0$	$r = 1$	$r = 2$	$r = 3$	$r = 4$	$r = 5$
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	1
3	0	0	0	2	2	2
4	0	0	0	2	2	2
5	0	0	0	2	2	3
6	0	0	5	5	5	5
7	0	5	5	5	6	6
8	0	5	5	5	6	7
9	0	5	5	7	7	8
10	0	5	5	7	8	8
11	0	5	5	7	8	9
12	0	5	5	7	8	10
13	0	5	10	10	11	11
14	0	5	10	10	11	11

Running time

λ	$r = 0$	$r = 1$	$r = 2$	$r = 3$	$r = 4$	$r = 5$
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	1
3	0	0	0	2	2	2
4	0	0	0	2	2	2
5	0	0	0	2	2	3
6	0	5	5	5	5	5
7	0	5	5	5	6	6
8	0	5	5	5	6	7
9	0	5	5	7	7	8
10	0	5	5	7	8	8
11	0	5	5	7	8	9
12	0	5	5	7	8	10
13	0	5	10	10	11	11
14	0	5	10	10	11	11

Table size $O(nc)$

Each entry is calculated in $O(1)$

Running time $O(nc)$

Polynomial?

Recursion for $P_r(\lambda)$

- We need to define a recursion that allows us to calculate $f_r(\lambda)$ in terms of $f_{r-1}(\mu)$ for $\mu \leq \lambda$.
- What if x^* is optimal for $P_r(\lambda)$ with the value $f_r(\lambda)$?
 - $x_r^* = 0$: $f_r(\lambda) = f_{r-1}(\lambda)$
 - $x_r^* = 1$: $f_r(\lambda) = p_r + f_{r-1}(\lambda - w_r)$

Thus we get:

$$f_r(\lambda) = \max\{f_{r-1}(\lambda), p_r + f_{r-1}(\lambda - w_r)\}$$

Starting condition:

$$f_0(\lambda) = 0 \text{ for } \lambda = 0, \dots, c$$

Finding optimal solution

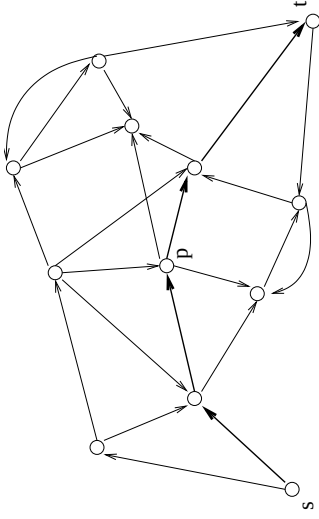
λ	$r = 0$	$r = 1$	$r = 2$	$r = 3$	$r = 4$	$r = 5$
0	0 $\leftarrow$	0 $\leftarrow$	0 $\leftarrow$	0 $\leftarrow$	0 $\leftarrow$	0 $\leftarrow$
1						1 $\uparrow$
2			2 $\uparrow$	2 $\leftarrow$	2 $\leftarrow$	2 $\leftarrow$
3						
4						
5	5 $\uparrow$	5 $\leftarrow$	5 $\leftarrow$	5 $\leftarrow$	5 $\leftarrow$	3 $\uparrow$
6		5 $\uparrow$	5 $\leftarrow$	6 $\uparrow$	6 $\leftarrow$	5 $\leftarrow$
7						7 $\uparrow$
8			7 $\uparrow$	7 $\leftarrow$	8 $\uparrow$	8 $\leftarrow$
9			7 $\uparrow$	8 $\uparrow$	8 $\leftarrow$	9 $\uparrow$
10						10 $\uparrow$
11			10 $\uparrow$	10 $\leftarrow$	11 $\uparrow$	11 $\leftarrow$
12						
13						
14						

j	1	2	3	4	5
p_j	5	5	2	6	1
w_j	6	7	3	7	2

- $x_r^* = 0$: $f_r(\lambda) = f_{r-1}(\lambda)$ ($\leftarrow$)
- $x_r^* = 1$: $f_r(\lambda) = p_r + f_{r-1}(\lambda - w_r)$ ($\uparrow$)

For each entry, mark which part of recursion was used, follow marks backward

Shortest Path I



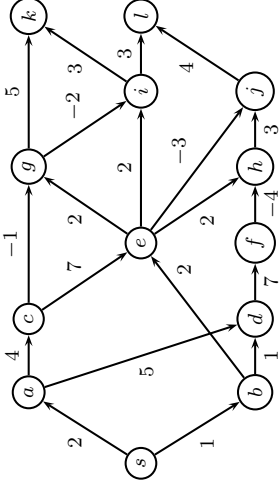
Observation: If the shortest path from s to t passes by p the sub-paths (s, p) and (p, t) are shortest paths from s to p , and p to t respectively.

Shortest Path for Acyclic graphs

- $G = (V, A)$ Directed Acyclic Graph (DAG), n nodes, m arcs
- Order the nodes so that $i < j$ for all arcs $(i, j) \in A$.
- Calculate in node order $1, 2, 3, \dots, n$.

$$d(v) = \min_{i \in V^-(j)} \{d(i) + w_{ij}\}$$

- Running time: $O(m)$

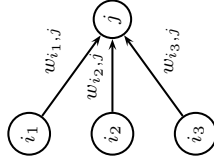


Shortest Path II

- **Observation:** Let $d(j)$ denote the length of a shortest path from s to j . Then

$$d(j) = \min_{i \in V^-(j)} \{d(i) + w_{ij}\}$$

- The observations does not give us an algorithm for a general digraph, but for the special case of an acyclic graph we get a method.

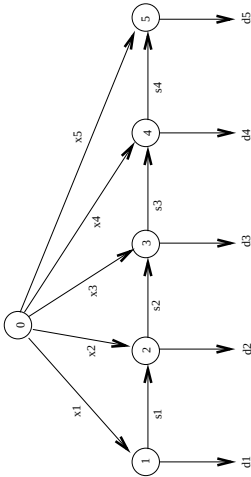


Uncapacitated Lot-sizing (ULS)

The problem is to decide on a production plan for an n -period horizon for a single product. The basic model can be viewed as having data:

- time horizon $t \in \{0, 1, \dots, n\}$.
- f is the fixed cost of producing in period t .
- p is the unit production cost in period t .
- h is the unit holding (storage) cost in period t .
- d_t is the demand in period t .

ULS as a network problem



Variables

- x_t production in period t
- s_t stock at end of period t

We can ignore production cost p since constant

Uncapacitated Lot Sizing

MIP model

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

where $y_t = 1$ if produce in period t

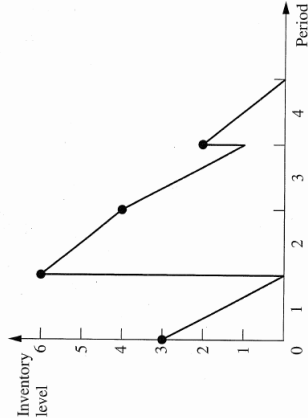
$$x_t > 0 \Leftrightarrow y_t = 1$$

Example

Demand

i	1	2	3	4
d_i	3	2	3	2

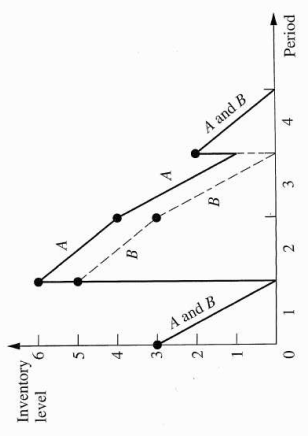
- fixed startup cost $f = 20$
- holding cost $h = 2$



Wagner-Within property

$$s_{t-1} \cdot x_t = 0$$

We only produce ($x_t > 0$) if stock is empty ($s_{t-1} = 0$)

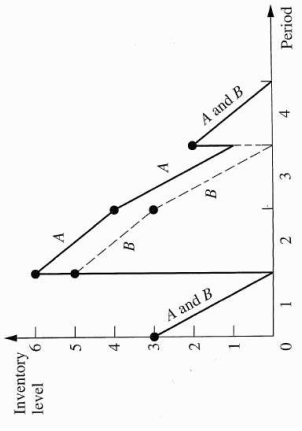


Dynamic programming

Let C_i be the cheapest production cost for periods $t, t + 1, \dots, n$ where period t starts with empty stock

$$C_t = \min_{j=t, \dots, n} C_{j+1} + f + h \sum_{k=t+1}^j (k - t) \cdot d_k$$

j is period where stock is empty when production starts t



Example

$$C_t = \min_{j=t, \dots, n} C_{j+1} + f + h \sum_{k=t+1}^j (k - t) \cdot d_k \quad f = 20, h = 2$$

$$\begin{aligned} C_5 &= 0 \\ C_4 &= \min \left\{ \begin{aligned} &C_5 + f \\ &C_4 + f \\ &C_5 + f + h(1 \cdot 2) \end{aligned} \right. = \begin{aligned} &0 + 20 = 20 \\ &20 + 20 = 40 \\ &0 + 20 + 4 = 24 \end{aligned} \\ C_3 &= \min \left\{ \begin{aligned} &C_3 + f \\ &C_4 + f + h(1 \cdot 3) \\ &C_5 + f + h(1 \cdot 3 + 2 \cdot 2) \end{aligned} \right. = \begin{aligned} &24 + 20 = 44 \\ &20 + 20 + 6 = 46 \\ &0 + 20 + 14 = 34 \end{aligned} \\ C_2 &= \min \left\{ \begin{aligned} &C_2 + f \\ &C_3 + f + h(1 \cdot 2) \\ &C_4 + f + h(1 \cdot 2 + 2 \cdot 3) \\ &C_5 + f + h(1 \cdot 2 + 2 \cdot 3 + 3 \cdot 2) \end{aligned} \right. = \begin{aligned} &34 + 20 = 54 \\ &24 + 20 + 4 = 48 \\ &20 + 20 + 16 = 56 \\ &0 + 20 + 28 = 48 \end{aligned} \\ C_1 &= \min \left\{ \begin{aligned} &C_2 + f \\ &C_3 + f + h(1 \cdot 2) \\ &C_4 + f + h(1 \cdot 2 + 2 \cdot 3) \\ &C_5 + f + h(1 \cdot 2 + 2 \cdot 3 + 3 \cdot 2) \end{aligned} \right. = \begin{aligned} &48 \\ &48 \\ &56 \\ &48 \end{aligned} \end{aligned}$$

Example

Two optimal solutions 48

$$\bullet f + (7 + 5 + 2) \cdot h = 20 + 14 \cdot 2 = 48$$

i	1	2	3	4
d_i	3	2	3	2
x_i	10	0	0	0
s_i	7	5	2	0

$$\bullet 2f + (2 + 2) \cdot h = 40 + 4 \cdot 2 = 48$$

i	1	2	3	4
d_i	3	2	3	2
x_i	5	0	5	0
s_i	2	0	2	0

Multiplying a sequence of matrices

- Suppose we want to multiply a long sequence of matrices $A \times B \times C \dots$
- Multiplying an $x \times y$ matrix by a $y \times z$ matrix (using the common algorithm) takes xyz multiplications.

$$\begin{pmatrix} 2 & 3 \\ 3 & 4 \\ 4 & 5 \end{pmatrix} \begin{pmatrix} 2 & 3 & 4 \\ 3 & 4 & 5 \end{pmatrix}$$

$$\bullet \text{ Here we need } 3 \times 2 \times 3 = 18 \text{ multiplications.}$$

Matrix multiplication

- We would like to avoid big intermediate matrices, and since matrix multiplication is associative, we can parenthesise however we want.
- Matrix multiplication is not commutative, so we cannot permute the order of the matrices without changing the result.

Dynamic Programming – p. 21

Different parenthesizations

As can be seen the order makes a huge difference. How do we find the best order?

- Let $M(i, j)$ be the *minimum* number of multiplications necessary to compute $\Pi_{k=i}^j A_k$.
- Two key observations are important:
 1. The outermost parentheses partition the chain of matrices (i, j) at some k .
 2. The optimal parenthesization order has optimal ordering on either side of k .

Dynamic Programming – p. 23

Different parenthesizations

Consider $A \times B \times C \times D$, where A is 30×1 , B is 1×40 , C is 40×10 and D is 10×25 . Here there are three possible parenthesizations:

- $((A \times B) \times C) \times D = 30 \times 1 \times 40 + 30 \times 40 \times 10 + 30 \times 10 \times 25 = 20700$
- $(A \times B) \times (C \times D) = 30 \times 1 \times 40 + 40 \times 10 \times 25 + 30 \times 40 \times 25 = 41200$
- $A \times ((B \times C) \times D) = 1 \times 40 \times 10 + 1 \times 10 \times 25 + 30 \times 1 \times 25 = 1400$

Dynamic Programming – p. 22

Recursion formula

- We get the following recurrence:

$$\begin{aligned} M(i, j) &= \min_{i \leq k \leq j-1} \{M(i, k) + M(k+1, j) + d_{i-1}d_kd_j\} \\ M(i, i) &= 0 \end{aligned}$$

- Note that a direct recursive implementation of this will be exponential, since there is a lot of duplicated work.

Dynamic Programming – p. 24

Storage instead of recalculation

Important:

- there are only $\binom{n+1}{2}$ substrings between 1 and n .
- thus it requires only $O(n^2)$ space to store the optimal cost for each of them giving us a polynomial algorithm.

Algorithm for computing optimal sequence

```

for  $i \leftarrow 1$  to  $n$  do  $M(i, i) = 0$ 
for  $l \leftarrow 2$  to  $n$ 
    for  $i \leftarrow 1$  to  $n - l + 1$  do
         $j \leftarrow i + l - 1$ 
         $M(i, j) \leftarrow +\infty$ 
        for  $k \leftarrow i$  to  $j - 1$  do
             $q \leftarrow M(i, k) + M(k + 1, j) + d_{i-1}d_kd_j$ 
            if  $q < M(i, j)$  then
                 $M(i, j) \leftarrow q, s(i, j) \leftarrow k$ 

return  $M(1, n)$ 
    
```

Matrix multiplication, example

Problem: multiply $A_1A_2A_3A_4A_5A_6$

	A_i	d_{i-1}	d_i
A_1	30	35	
A_2		35	15
A_3		15	5
A_4		5	10
A_5		10	20
A_6		20	25

$i \setminus j$	1	2	3	4	5	6
1	0	15750	7875	9375	11875	15125
2		0	2626	4375	7125	10500
3			0	750	2500	5375
4				0	1000	3500
5					0	5000
6						0

$i \setminus j$	1	2	3	4	5	6
1		1	1	3	3	3
2			2	3	3	3
3				3	3	3
4					4	5
5						5
6						

$$m[2, 5] = \min \begin{cases} m[2, 2] + m[3, 5] + d_1d_2d_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 = 13000 \\ m[2, 3] + m[4, 5] + d_1d_3d_5 = 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125 \\ m[2, 4] + m[5, 5] + d_1d_4d_5 = 4375 + 0 + 35 \cdot 10 \cdot 20 = 11375 \end{cases}$$

Common characteristics

- The problem can be divided into *stages* with a *decision* required in each stage. Each stage has a number of *states* associated with it.
- The decision at one stage transforms one state into a state in the next stage.
- Given the current state, the optimal decision for each of the remaining states does not depend on the previous states or decisions.
- There exists a recursive relationship that identifies the optimal decision for stage j , given that stage $j + 1$ has already been solved.
- The final stage must be solvable by itself.

Dynamic Programming in a Nutshell

- The big challenge in dynamic programming, and the art involved, is to take a problem and determine stages and states so that all of the above hold. If you can, then the recursive relationship makes finding the values relatively easy.
- We can express the solution to a problem in terms of solutions to smaller problems.

Dynamic Programming – p. 29

Development of a DP algorithm

The development of a dynamic programming algorithm can be broken into four steps:

1. Characterize the structure of an optimal solution.
2. Recursively define the value of an optimal solution.
3. Compute the value of an optimal solution in a bottom-up fashion.
4. Construct an optimal solution from computed information.

Dynamic Programming – p. 30