Dynamic Programming

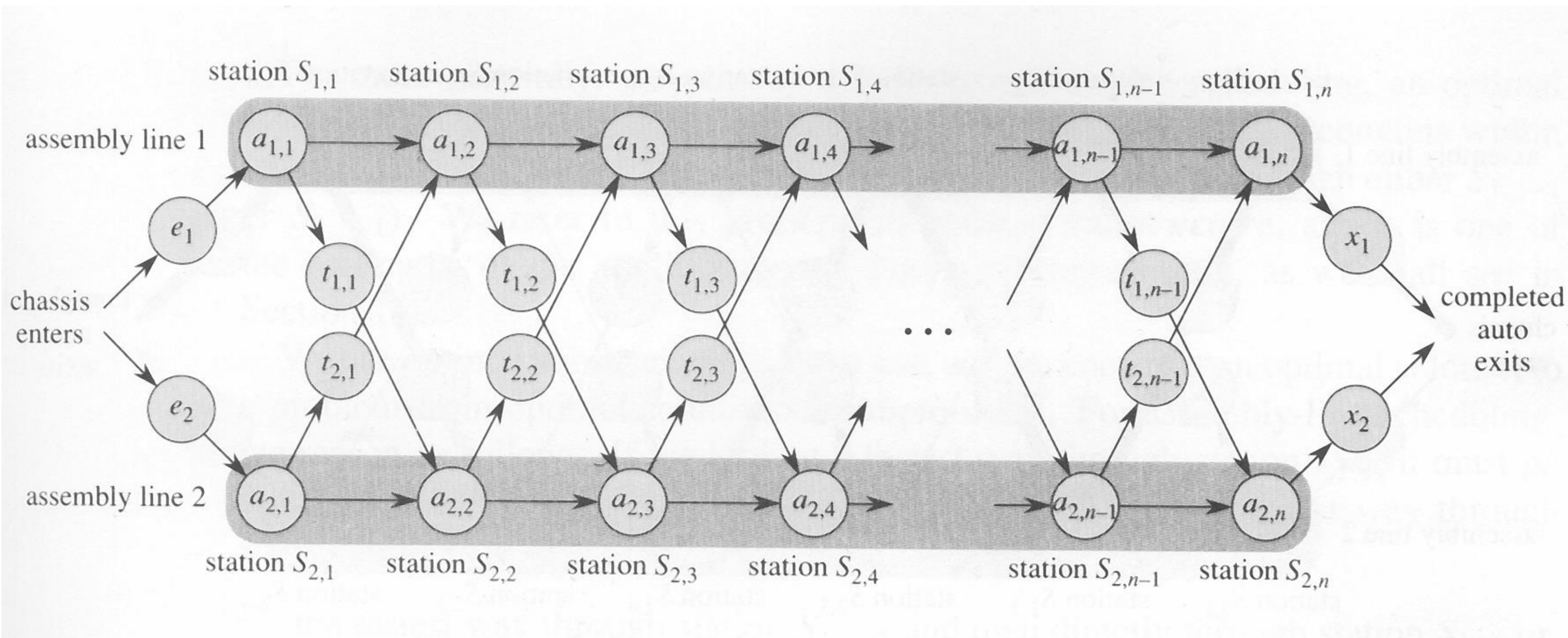
Introduction

- *Dynamic programming* is typically applied to optimization problems.
- In such problem there can be *many solutions*. Each solution has a value, and we wish to find *a solution* with the optimal value.

Assembly-line scheduling

Automobile factory with two assembly lines.

- Each line has n stations: $S_{1,1}, \ldots, S_{1,n}$ and $S_{2,1}, \ldots, S_{2,n}$.
- Corresponding stations $S_{1,j}$ and $S_{2,j}$ perform the same function but can take different amounts of time $a_{1,j}$ and $a_{2,j}$.
- Entry times e_1 and e_2.
- Exit times x_1 and x_2.
- After going through a station, can either
  - stay on same line; no cost, or
  - transfer to other line; cost after $S_{i,j}$ is $t_{i,j}$. ($j = 1, \ldots, n-1$. No $t_{i,n}$, because the assembly line is done after $S_{i,n}$.)



station $S_{1,1}$
station $S_{1,2}$
station $S_{1,3}$
station $S_{1,4}$
station $S_{1,n-1}$
station $S_{1,n}$
assembly line 1
$a_{1,1}$
$a_{1,2}$
$a_{1,3}$
$a_{1,4}$
$a_{1,n-1}$
$a_{1,n}$
e_1
$t_{1,1}$
$t_{1,2}$
$t_{1,3}$
$t_{1,n-1}$
x_1
chassis enters
completed auto exits
$t_{2,1}$
$t_{2,2}$
$t_{2,3}$
$t_{2,n-1}$
e_2
x_2
assembly line 2
$a_{2,1}$
$a_{2,2}$
$a_{2,3}$
$a_{2,4}$
$a_{2,n-1}$
$a_{2,n}$
station $S_{2,1}$
station $S_{2,2}$
station $S_{2,3}$
station $S_{2,4}$
station $S_{2,n-1}$
station $S_{2,n}$

Brute force Solution

- Steps:
  - List all possible sequences,
  - For each sequence of n stations, compute the passing time. (the computation takes $\Theta(n)$ time.)
  - Record the sequence with smaller passing time.
  - However, there are total 2^n possible sequences.

Dynamic Programming

The development of a dynamic programming algorithm can be broken into a sequence of 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.

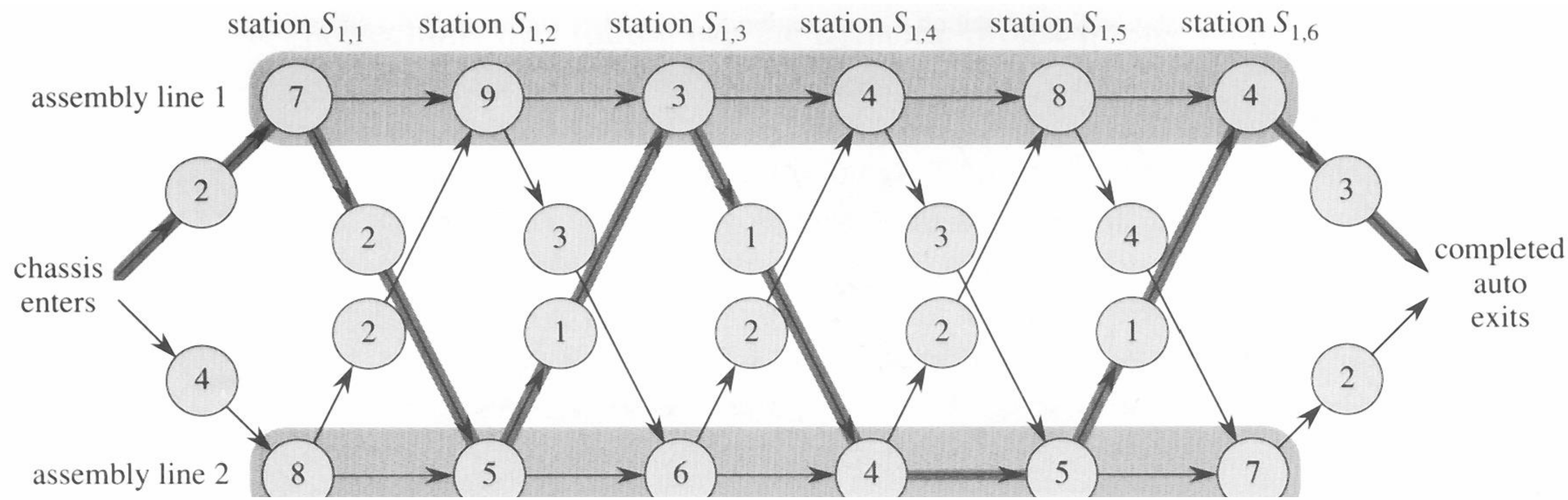
- Optimal substructure

  An optimal solution to a problem (fastest way through $S_{1,j}$) contains within it an optimal solution to subproblems.

  (fastest way through $S_{1,j-1}$ or $S_{2,j-1}$).

- Use optimal substructure to construct optimal solution to problem from optimal solutions to subproblems.

To solve problems of finding a fastest way through $S_{1,j}$ and $S_{2,j}$, solve subproblems of finding a fastest way through $S_{1,j-1}$ and $S_{2,j-1}$.



j	1	2	3	4	5	6
$f_1[j]$	9	18	20	24	32	35
$f_2[j]$	12	16	22	25	30	37

$f^* = 38$

j	2	3	4	5	6
$l_1[j]$	1	2	1	1	2
$l_2[j]$	1	2	1	2	2

$l^* = 1$

Step 1: The structure of the fastest way through the factory

Think about fastest way from entry through $S_{1,j}$.

- If $j = 1$, easy: just determine how long it takes to get through $S_{1,1}$.
- If $j \geq 2$, have two choices of how to get to $S_{1,j}$:
  - Through $S_{1,j-1}$, then directly to $S_{1,j}$.
  - Through $S_{2,j-1}$, then transfer over to $S_{1,j}$.

Step 2: A recursive solution

- ***$f_i[j]$*:** the fastest possible time to get a chassis from the starting point through station $S_{i,j}$
- ***f^**:** the fastest time to get a chassis all the way through the factory.

$$f^* = \min(f_1[n] + x_1, f_2[n] + x_2)$$

$$f_1[j] = \begin{cases} e_1 + a_{1,1} & \text{if} \quad j = 1, \\ \min(f_1[j-1] + a_{1,j}, f_2[j-1] + t_{2,j-1} + a_{1,j}) & \text{if} \quad j \geq 2 \end{cases}$$

$$f_2[j] = \begin{cases} e_2 + a_{2,1} & \text{if} \quad j = 1, \\ \min(f_2[j-1] + a_{2,j}, f_1[j-1] + t_{1,j-1} + a_{2,j}) & \text{if} \quad j \geq 2 \end{cases}$$

- $l_i[j]$ = line # (1 or 2) whose station $j-1$ is used in fastest way through $S_{i,j}$.
- $S_{l_i[j],\, j-1}$ precedes $S_{i,j}$.
- Defined for $i = 1,2$ and $j = 2, \ldots, n$.
- l^* = line # whose station n is used.

j	1	2	3	4	5	6
$f_1[j]$	9	18	20	24	32	35
$f_2[j]$	12	16	22	25	30	37

$f^* = 38$

j	2	3	4	5	6
$l_1[j]$	1	2	1	1	2
$l_2[j]$	1	2	1	2	2

$l^* = 1$

Step 3: computing an optimal solution

- Let $r_i(j)$ be the number of references made to $f_i[j]$ in a recursive algorithm.

$$r_1(n) = r_2(n) = 1$$

$$r_1(j) = r_2(j) = r_1(j+1) + r_2(j+1)$$

- The total number of references to all $f_i[j]$ values is $\Theta(2^n)$.
- We can do much better if we compute the $f_i[j]$ values in different order from the recursive way. Observe that for $j \geq 2$, each value of $f_i[j]$ depends only on the values of $f_1[j-1]$ and $f_2[j-1]$.

Step 3: computing an optimal solution

FASTEST-WAY(a, t, e, x, n)

$f_1[1] \leftarrow e_1 + a_{1,1}$

$f_2[1] \leftarrow e_2 + a_{2,1}$

for $j \leftarrow 2$ to n

  do if $f_1[j\text{-}1] + a_{1,j} \leq f_2[j\text{-}1] + t_{2,j\text{-}1} + a_{1,j}$

    then $f_1[j] \leftarrow f_1[j\text{-}1] + a_{1,j}$

      $l_1[j] \leftarrow 1$

    else $f_1[j] \leftarrow f_2[j\text{-}1] + t_{2,j\text{-}1} + a_{1,j}$

      $l_1[j] \leftarrow 2$

  if $f_2[j\text{-}1] + a_{2,j} \leq f_1[j\text{-}1] + t_{1,j\text{-}1} + a_{2,j}$

then $f_2[j] \leftarrow f_2[j-1] + \mathrm{a}_{2,j}$

$l2[j] \leftarrow 2$

else $f_2[j] \leftarrow f_1[\mathrm{j}-1] + t_{1,j\text{-}1} + a_2,j$

$l_2[j] \leftarrow 1$

if $f_1[n] + x_1 \leq f_2[n] + x_2$

then $f^* = f_1[n] + x_1$

$l^* = 1$

else $f^* = f_2[n] + x_2$

$l^* = 2$

constructing the fastest way through the factory

```
PRINT-STATIONS(l, l*, n)
i ← l*
print "line" i ",station" n
for j ← n downto 2
        do i ← l_i[j]
            print "line" i ",station" j − 1
```

output

line 1, station 6

line 2, station 5

line 2, station 4

line 1, station 3

line 2, station 2

line 1, station 1

15.2 Matrix-chain multiplication

- A product of matrices is fully parenthesized if it is either a single matrix, or a product of two fully parenthesized matrix product, surrounded by parentheses.

Illustration

- How to compute where A_i is a matrix for every i.
- Example:

$$A_1 A_2 A_3 A_4$$

$$(A_1(A_2(A_3 A_4))) \quad (A_1((A_2 A_3)A_4))$$

$$((A_1 A_2)(A_3 A_4)) \quad ((A_1(A_2 A_3))A_4)$$

$$(((A_1 A_2)A_3)A_4)$$

A_1 is a 10×100 matrix A_2 is a 100×5 matrix, and A_3 is a 5×50 matrix

Then $((A_1 A_2) A_3)$

takes $10 \times 100 \times 5 + 10 \times 5 \times 50 = 7500$ time.

However $(A_1 (A_2 A_3))$

takes $100 \times 5 \times 50 + 10 \times 100 \times 50 = 75000$ time.

The matrix-chain multiplication problem:

- Given a chain $\langle A_1, A_2, \ldots, A_n \rangle$ of n matrices, where for $i=0,1,\ldots,n$, matrix A_i has dimension $p_{i-1} \times p_i$, fully parenthesize the product $A_1 A_2 \ldots A_n$ in a way that minimizes the number of scalar multiplications.

MATRIX MULTIPLY

MATRIX MULTIPLY(A, B)

1 if columns$[A] \neq$ column$[B]$

2 then error "incompatible dimensions"

3 else for $i \leftarrow 1$ to rows$[A]$

4 do for $j \leftarrow 1$ to columns$[B]$

5 do $c[i, j] \leftarrow 0$

6 for $k \leftarrow 1$ to columns$[A]$

7 do $c[i, j] \leftarrow c[i, j] + A[i, k] B[k, j]$

8 return C

Counting the number of parenthesizations:

$$P(n) = \begin{cases} 1 & if\, n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & if\, n \geq 2 \end{cases}$$

$$P(n) = C(n-1)$$

Step 1: The structure of an optimal parenthesization

Optimal

$$((A_1 A_2 \dots A_k)(A_{k+1} A_{k+2} \dots A_n))$$

Combine

Step 2: A recursive solution

- Define $m[i, j]$= minimum number of scalar multiplications needed to compute the matrix $A_{i..j} = A_i A_{i+1} .. A_j$
- goal $m[1, n]$
- $$m[i, j] = \begin{cases} 0 & i = j \\ \min_{i \le k < j} \{m[i,k] + m[k+1, j] + p_{i-1} p_k p_j\} & i \ne j \end{cases}$$

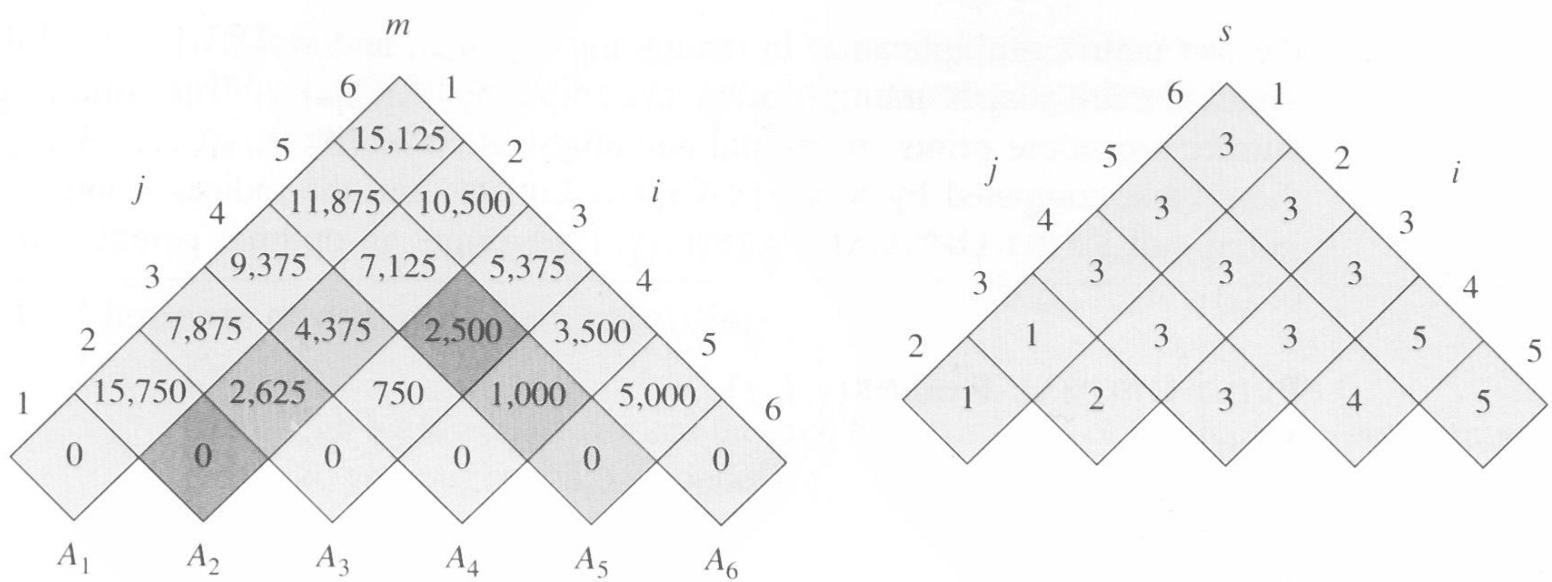
MATRIX_CHAIN_ORDER

```
MATRIX_CHAIN_ORDER(p)
n ← length[p] −1
for i ← 1 to n
    do m[i, i] ← 0
for l ← 2 to n
    do for i ← 1 to n − l + 1
            do j ← i + l − 1
               m[i, j] ← ∞
               for k ← i to j − 1
                   do q ← m[i, k] + m[k+1, j]+ p_{i-1} p_k p_j
                      if q < m[i, j]
                         then m[i, j] ← q
                              s[i, j] ← k
return m and s
```

Example

$$\begin{array}{lll} A_1 & 30\times 35 & = p_0 \times p_1 \\ A_2 & 35\times 15 & = p_1 \times p_2 \\ A_3 & 15\times 5 & = p_2 \times p_3 \\ A_4 & 5\times 10 & = p_3 \times p_4 \\ A_5 & 10\times 20 & = p_4 \times p_5 \\ A_6 & 20\times 25 & = p_5 \times p_6 \end{array}$$

the m and s table computed by MATRIX-CHAIN-ORDER for n=6



$$
\begin{aligned}
&m[2,5]= \\
&\min\{ \\
&\quad m[2,2]+m[3,5]+p_1p_2p_5=0+2500+35\times15\times20=13000, \\
&\quad m[2,3]+m[4,5]+p_1p_3p_5=2625+1000+35\times5\times20=7125, \\
&\quad m[2,4]+m[5,5]+p_1p_4p_5=4375+0+35\times10\times20=11374 \\
&\} \\
&=7125
\end{aligned}
$$

PRINT_OPTIMAL_PARENS(s, i, j)

```
if j = i
    then print "A"_i
     else  print "("
            PRINT_OPTIMAL_PARENS(s,  i,  s[i,j])
            PRINT_OPTIMAL_PARENS(s, s[i,j]+1, j)
            print ")"
```

- PRINT_OPTIMAL_PARENS(s, 1, 6)
  Output: $((A_1(A_2A_3))((A_4A_5)A_6))$