Chapter 15. Dynamic Programming

Dynamic Programming typically applies to optimization problems in which a set of choices must be made in order to get an optimal solution (for a maximization or a minimization problem). As choices are made, subproblems of the **same form** often arise. Dynamic programming is effective when the given subproblem may arise from more than one partial set of choices.

The key technique is to store the solution to each such subproblem in case it could re-appear (programming in this context refers to a tabular method). Therefore, dynamic programming is applicable when the subproblems are not **independent**.

15.2 Four Steps of Developing DP Algorithms

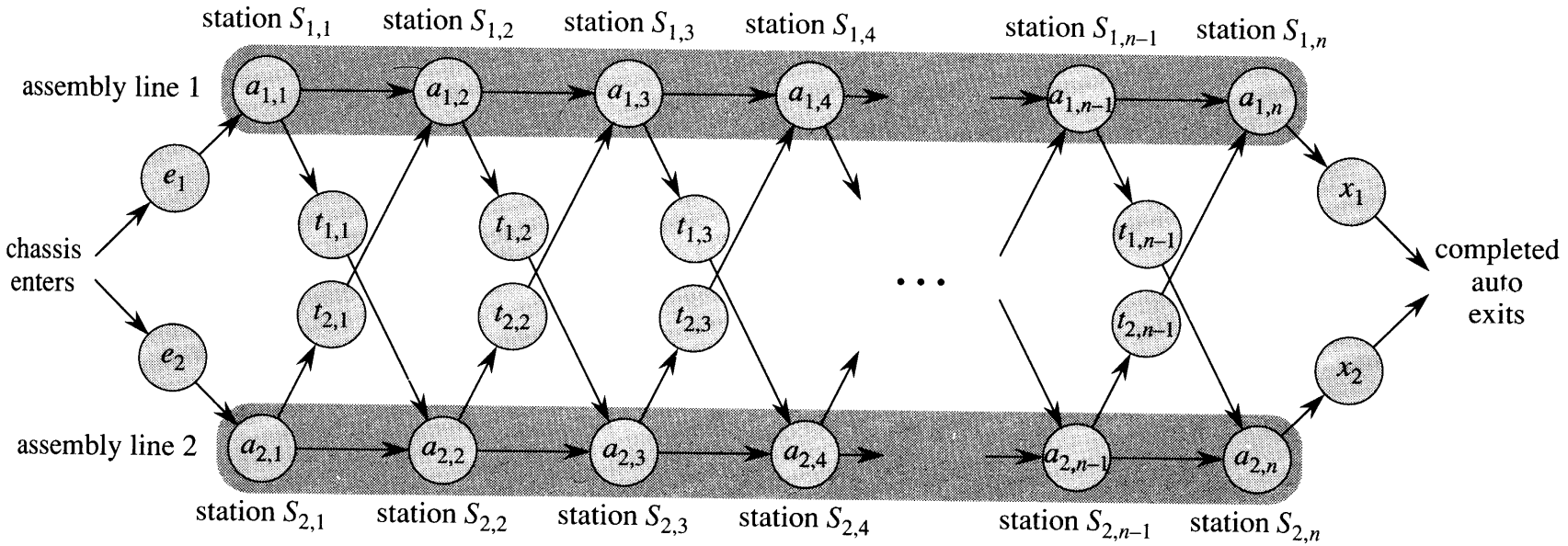
The development of a DP algorithm consists of the following four steps.

1. Characterize the structure of an optimal solution
2. Define the value of the optimal solution **recursively**
3. Compute the value of the optimal solution in **a bottom-up fashion**
4. Construct the optimal solution from the computed information (stored in tables).

Steps 1 - 3 form the basis of a DP solution to an optimization problem.

15.3 Assembly-line scheduling problem

There are two parallel assembly lines for assembling a car. Each of them performs the same sequence of operations but at different rates. It is also possible, at some cost of time, to transfer the car from one assembly line to the other. All the values in circles are times:



The problem is to find a shortest path from start to finish.

Assembly-line problem (continued)

Each assembly line has n stations. Notation:

- $S_{i,j}$ is the jth station in line i
- $a_{i,j}$ is the processing time at station $S_{i,j}$
- $t_{i,j}$ is the time to transfer from $S_{i,j}$ to $S_{i',j+1}$ $(i' \neq i)$
- e_i is the entry time for line i
- x_i is the exit time for line i

$i = 1, 2$ and $j = 1, 2, \ldots, .$

15.3 Assembly-line problem (continued)

Step 1: The structure of the fastest way (optimal solution) through the factory.

The car must pass through each stage of assembly. Stage j can be reached either from stage $j-1$ in the same assembly line, or by transferring (with a cost) from stage $j-1$ in the other assembly line.

Thus, the fastest way to complete $S_{1,j}$ is either

- through $S_{1,j-1}$ and then passing through $S_{1,j}$, or
- through $S_{2,j-1}$, a transfer from line 2 to line 1, and then passing through $S_{1,j}$.

15.3 Assembly-line problem (continued)

Step 2: A recursive solution.

Let $f_i[j]$ be the least possible time to get a car from the starting point through station $S_{i,j}$. Then we have:

$$f_1[1] = e_1 + a_{1,1},$$

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

$$f_1[j] = \min\{f_1[j-1] + a_{1,j}, f_2[j-1] + t_{2,j-1} + a_{1,j}\}, \qquad (2 \le j \le n)$$

$$f_2[j] = \min\{f_2[j-1] + a_{2,j}, f_1[j-1] + t_{1,j-1} + a_{2,j}\}, \qquad (2 \le j \le n)$$

The overall shortest time to complete the car is

$$\min\{f_1[n] + x_1, f_2[n] + x_2\}.$$

15.3 Assembly-line problem (continued)

Step 3: Apply the recurrence bottom-up.

If the recurrence is applied in the order

$$f_1[1], f_2[1], f_1[2], f_2[2], \ldots, f_1[n], f_2[n]$$

then we always have the values that are required by the right-hand side.

Thus $f_1[n]$ and $f_2[n]$ are obtained in $O(n)$ time, and from these the solution is found.

To find the actual shortest path, just record which of the two choices gave the minimum at each step, then work back from the finishing point.

15.3 Assembly-line problem (continued)

In the calculation of $f_i[j]$, we have two choices, that is, which value we should choose for $f_i[j]$.

We need to store where the chosen value comes from, and we use an array $l_i[j]$ to memorize from where the value comes.

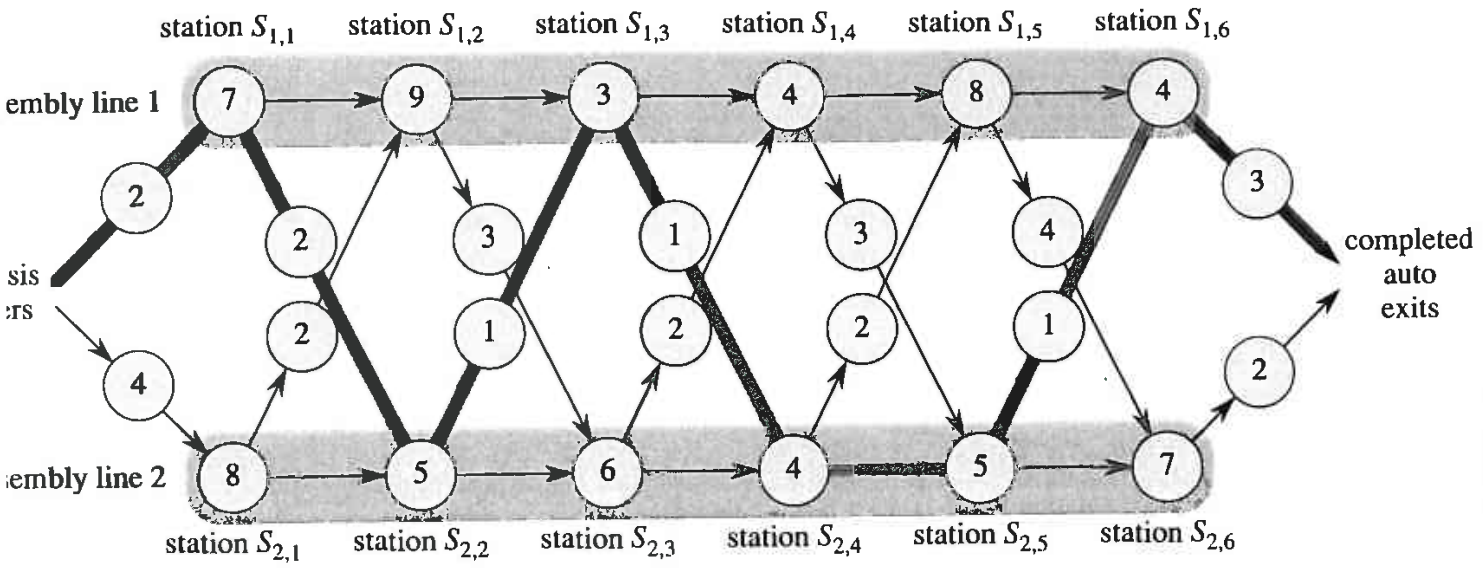
For example, when calculate

$$
\begin{aligned}
f_2[2] &= \min\{f_1[1] + t_{1,2} + a_{2,2}, f_2[1] + a_{2,2}\} && (1)\\
&= \min\{9 + 2 + 5, 12 + 5\} && (2)\\
&= \min\{16, 17\} = 16, \text{the value from line 1} && (3)
\end{aligned}
$$

Thus, $l_2[2] = 1$.

The array $f_i[j]$ will be used to find the optimal solution in Step 4.

15.3 Assembly-line problem (continued)



(a)

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

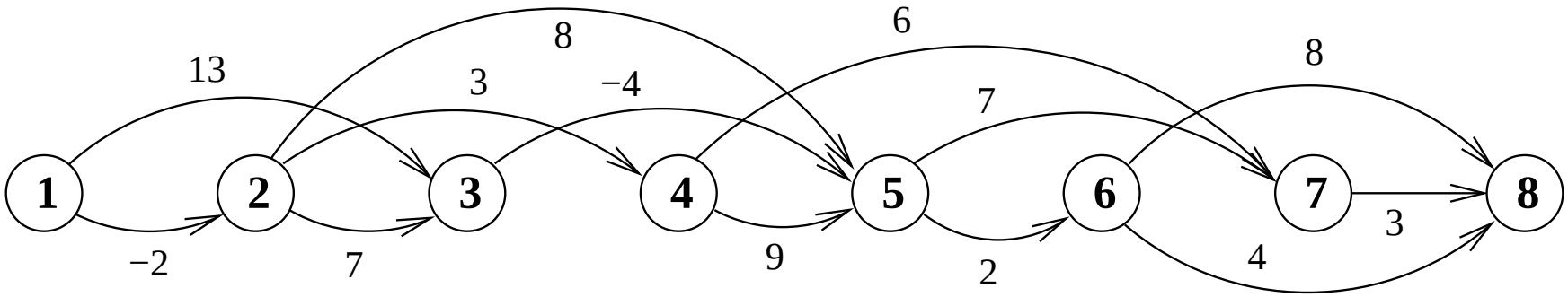
$l^* = 1$

(b)

Shortest path in a directed acyclic graph

A large fraction of applications of dynamic programming are actually special cases of the problem of shortest paths through a Directed Acyclic Graph (DAG for short) $G = (V, E)$ where $V = \{v_1, v_2 \ldots, v_n\}$.

- **Given:** Nodes $v_1, v_2, \ldots, v_n$ and some arcs (directed edges) with given lengths. Each arc is directed from left to right.
- **To find:** The shortest path from v_1 to v_n.



Directed Acyclic Graph (continued)

Key observation: A shortest path from v_1 to v_j consists of a shortest path from v_1 to v_i (for some $i < j$) followed by a single arc from v_i to v_j.

Define arc lengths

$$L[i,j] = \text{length of arc from } v_i \text{ to } v_j \text{ (if any).}$$

Define path lengths

$$P[i] = \text{length of shortest path from } v_1 \text{ to } v_i \quad (1 \leq i \leq n)$$

Then

$$P[1] = 0,$$

$$P[j] = \min_{\langle v_i, v_j \rangle \in E} \{ P[i] + L[i,j] \}, \quad (j = 1, 2, \ldots, n)$$

where the minimum is taken over all i for which arc $v_i \to v_j$ exists. If there is no such arc, the value of the minimum is ∞.

Directed Acyclic Graph (continued)

Now the recurrence can be applied in the order

$$P[1], P[2], \ldots, P[n].$$

Time complexity. Suppose that for each v_i, we have a list of incoming arcs.

The formula for $P[j]$ requires time at most $b \times d_j$, where $d_j = |E_j|$ is the number of arcs into v_j, i.e., $E_j = \cup_{(v_i,v_j)\in E}\{(v_i, v_j)\}$, and b is constant.

Therefore, the total time is at most

$$an + b\sum_j d_j \;=\; an + bm \;=\; O(n+m),$$

where $m = |E|$ is the number of arcs altogether, $\sum_{j=1}^{n} d_j = m$, and a is a constant.

In the worst case we need to look at each node and each arc at least once, so the worst case complexity of the algorithm is $\Theta(n+m)$.

Directed Acyclic Graph (continued)

Exercise 1. Given a DAG $G = (V, E)$ and a pair of nodes u and v with $u, v \in V$, assume that there is a positive weight associated with each arc in E, devise an algorithm for finding a longest path from u to v, and analyze the time complexity of your algorithm. (*Hint: use dynamic programming*)

Exercise 2. Given a sequence of n elements, find a longest increasing subsequence from the sequence. (*Hint: use dynamic programming*)

Directed Acyclic Graph (continued)

Exercise 2. Given a sequence of n elements, find a longest increasing subsequence from the sequence. (*Hint: use dynamic programming*)

Solution: We construct a directed graph $G = (V, E)$, each element in the sequence is a node in V, there is a directed edge in E from a node v_i to another node v_j if the element of v_i is less than the element v_j, and the weight of this directed edge is assigned 1. Clearly, G is a DAG. Then, Add a virtual node v_0 and add a directed edge from v_0 to v for each $v \in V$, and assign the edge with weight 0. Let G' be the final DAG.

Find a longest path in G' from v_0. This path corresponds to the longest increasing subsequence, which takes $O(n+m) = O(n^2)$ time as $m = O(n^2)$.