Dynamic Programming Methods.S1
Forward Recursion

Instead of starting at a final state and working backwards, for many problems it is possible to determine the optimum by an opposite procedure called forward recursion. Backward recovery is then used to identify the optimal path. The procedure may be more convenient for some classes of problems, but for those in which the final state set is unknown, it is the only feasible approach. Forward recursion also leads to a very powerful procedure called reaching, described in the next section.

Backward Decision Set, Transition Function and Decision Objective

The backward decision set is denoted for each state by $\boldsymbol{D}_b(\mathbf{s})$ with the subscript "b" being used to distinguish it from the comparable set $\boldsymbol{D}(\mathbf{s})$ previously defined for backward recursion. $\boldsymbol{D}_b(\mathbf{s})$ is the set of decisions that can be used to enter state **s**. For the simple rotated path problem depicted in Fig. 3, $\boldsymbol{D}_b(\mathbf{s}) = \{-1, +1\}$, where $d = -1$ implies entering **s** from above and $d = +1$ implies entering **s** from below. Note that the elements of $\boldsymbol{D}(\mathbf{s})$ and $\boldsymbol{D}_b(\mathbf{s})$ are the same for this path problem but their meaning is quite different; $\boldsymbol{D}(\mathbf{s})$ is the set of decisions that lead out of state **s** and $\boldsymbol{D}_b(\mathbf{s})$ is the set of decisions that lead in.

The backward transition function is defined as

$$\mathbf{s}_b = T_b(\mathbf{s}, \mathbf{d}), \text{ where } \mathbf{d} \quad \boldsymbol{D}_b(\mathbf{s}).$$

The function T_b determines the state that came before **s** when the decision made to reach state **s** is **d.** We use $\mathbf{s}_b$ to denote the previous state. Very often the backward transition function can be obtained directly from the forward transition function. For some problems, the forward transition equation $\mathbf{s}' = T(\mathbf{s}, \mathbf{d})$ can be solved for **s** in terms of **s**' and **d**. Substituting first $\mathbf{s}_b$ for **s** and then **s** for **s**', one obtains the backward transition function. For instance, the rotated path problem has the forward transition function components

$$s_1' = s_1 + 1 \text{ and } s_2' = s_2 + d.$$

Solving these equations for **s** in terms of **s**' yields

$$s_1 = s_1' - 1 \text{ and } s_2 = s_2' - d.$$

Substituting first $\mathbf{s}_b$ for **s** and then **s** for **s**', we obtain

$$s_{b1} = s_1 - 1 \text{ and } s_{b2} = s_2 - d.$$

These equations give the backward transition function for the rotated path problem.

For a given definition of the state variables, it is not always true that there exists both forward and backward transition functions for a particular problem. For instance, consider the path problem with turn penalties (Section 12.5). The components of its forward transition function are

$$s_1' = s_1 + 1, \;\; s_2' = s_2 + d, \;\; s_3' = d.$$

Thus it is not possible to solve for s_3 in terms of $\mathbf{s}'$ and d, because s_3 does not appear in these equations. A little reflection will show that it is impossible to know what the previous state was (in terms of the third state variable) given the current state and the decision made to reach it. Defining the set of state variables differently, though, will provide both forward and backward transition functions. For the turn penalty problem, the first two state variables remain unchanged, but the third state variable s_{b3} should be defined as the direction to be traveled rather than the direction last traveled.

The backward decision objective, $z_b(\mathbf{s}_b, \mathbf{d}, \mathbf{s})$ is the immediate cost of decision $\mathbf{d}$ when it leads to state $\mathbf{s}$ from state $\mathbf{s}_b$. For most problems of interest, z_b is a simple function of $\mathbf{s}_b$ and $\mathbf{d}$.

Forward Recursive Equation

The forward recursive equation is written

$$f_b(\mathbf{s}) = \text{Minimize}\{R_b(\mathbf{s}, \mathbf{d}) : \mathbf{d} \quad \boldsymbol{D}_b(\mathbf{s})\} \tag{4}$$

where $R_b(\mathbf{s}, \mathbf{d}) = z_b(\mathbf{d}, \mathbf{s}) + f_b(\mathbf{s}_b)$

and $\mathbf{s}_b = T_b(\mathbf{s}, \mathbf{d})$.

The factors in these relationships have analogous definitions to their counterparts in the backward recursive equation. In particular, $f_b(\mathbf{s})$ is the cost of the optimal path arriving at $\mathbf{s}$ from an initial state as a consequence of decision $\mathbf{d}$. Sometimes it is convenient to express the decision return as an explicit function of $\mathbf{s}_b$. In that case we use the notation $z_b(\mathbf{s}_b, \mathbf{d}, \mathbf{s})$.

The first step in solving (4) is to assign values to the initial states (those with no predecessors). This is equivalent to specifying the

boundary conditions. Then the equation can be solved for each state that is an immediate successor of the initial states. The procedure continues in the forward direction until all states have been considered. The policy function $\mathbf{d}^*(\mathbf{s})$ now holds the information on the optimal decision entering each state $\mathbf{s} \in \boldsymbol{S}$.

The optimum is identified by a backward recovery procedure. Starting at the final state associated with the minimum cost, the policy function identifies the optimal decision that led to that state. The backward transition function then identifies the predecessor state in the optimal path. The policy function, in turn, identifies the optimal decision entering that state. The process continues sequentially using the optimal policy function and the backward transition function until an initial state is encountered. At this point the optimal path to the specified final state from an initial state has been found. Note that the forward recursion procedure implicitly discovers the optimal path to every state from an initial state.

Example 2

For the rotated path problem described in the previous section, Table 5 provides the additional definitions associated with forward recursion.

Table 5. Forward recursion model for rotated path problem

Component	Description
Backward decision set	$\boldsymbol{D}_b(\mathbf{s}) = \{+1, -1\}$
Backward transition function	$\mathbf{s}_b = T_b(\mathbf{d}, \mathbf{s})$ $s_{b1} = s_1 - 1$ and $s_{b2} = s_2 - d$
Backward decision objective	$z_b(\mathbf{s}, \mathbf{d}) = a(\mathbf{s}, d)$, where $a(\mathbf{s}, d)$ is the length of the arc entering $\mathbf{s}$ from decision d, taken from Fig. 3.
Initial value function	$f(\mathbf{s}) = 0$ for $\mathbf{s} \in \boldsymbol{I}$
Forward recursive equation	$f_b(\mathbf{s}) = \text{Minimize}\{z_b(\mathbf{s}, \mathbf{d}) + f_b(\mathbf{s}_b) : \mathbf{d} \in \boldsymbol{D}_b(\mathbf{s})\}$

The calculations are shown in Table 6. Note that this table is constructed by listing the states in the order in which they must be considered for forward recursion, that is, in lexicographic order. All the calculations

required to obtain $f_b(\mathbf{s})$ and $\mathbf{d}^*(\mathbf{s})$, the optimal path value and the optimum policy, respectively, are included.

To better understand how the calculations are performed, assume that we are at $\mathbf{s} = (3, 1)$ which is node 8 in Fig. 3. The decision set $\boldsymbol{D}_b((3, 1)) = \{1,-1\}$ and the backward transition function for each of the two state components is $s_{b1} = s_1 - 1$ and $s_{b2} = s_2 - d$. To solve the recursive equation (4) if is first necessary to evaluate the return function for all $d \quad \boldsymbol{D}_b((3, 1))$. The two cases are as follows.

$d = 1 \qquad \mathbf{s}_b = (2,0)$ and

$$R_b((3, 1), 1) = a((2,0), 1) + f((2,0)) = 3 + 11 = 14$$

$d = -1 \qquad \mathbf{s}_b = (2,2)$ and

$$R_b((3, 1), -1) = a((2,2), -1) + f((2,2)) = 2 + 13 = 15$$

Combining these results leads to

$$f((3, 1)) = \text{Minimize} \left\{ \begin{matrix} R_b((3, 1), 1) = 14 \\ R_b((3, 1), -1) = 15 \end{matrix} \right\} = 14$$

so the optimal decision $\mathbf{d}^*((3, 1)) = (1)$ which means it is best to reach (3, 1) from (2,0). Figure 5 depicts the optimal policy for each state, which is the arc that should be traversed to enter that state. By implication, an optimal path is available for every state *from* the initial state.

Table 6. Forward recursion results for the rotated path problem

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
Index	**s**	**d**	$\mathbf{s}_b$	z_b	$f_b(\mathbf{s}_b)$	$R_b(\mathbf{s}, \mathbf{d})$	$f_b(\mathbf{s})$	$\mathbf{d}^*(\mathbf{s})$
1	(0, 0)	—					0	∞
2	(1, –1)	–1	(0, 0)	5	0	5	5	–1
3	(1, 1)	1	(0, 0)	8	0	8	8	1
4	(2, –2)	–1	(1, –1)	4	5	9	9	–1
5	(2, 0)	1	(1, –1)	7	5	12		
		–1	(1, 1)	3	8	11	11	–1
6	(2, 2)	1	(1, 1)	5	8	13	13	1
7	(3, –1)	1	(2, –2)	9	9	18		
		–1	(2, 0)	5	11	16	16	–1
8	(3, 1)	1	(2, 0)	3	11	14	14	1
		–1	(2, 2)	2	13	15		
9	(4, 0)	1	(3, –1)	6	16	22	22	1
		–1	(3, 1)	9	14	23		

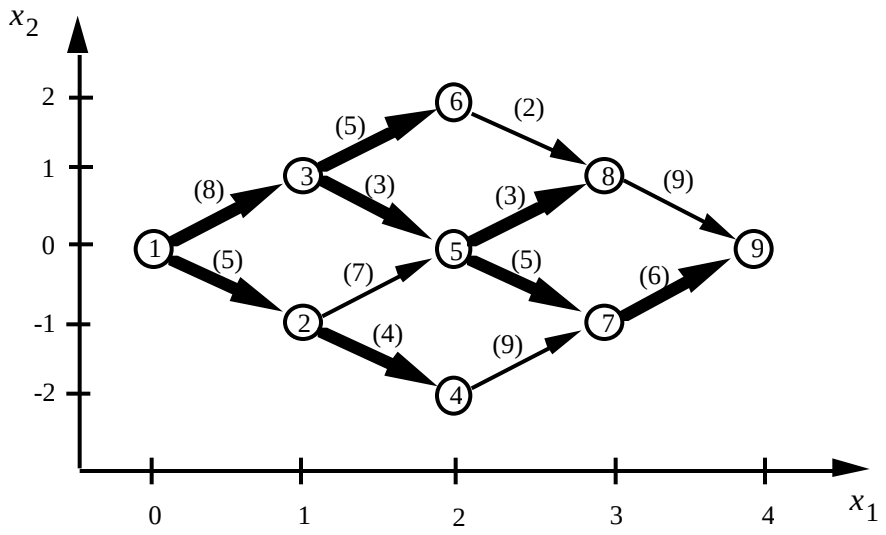

Figure 5. Optimal policy using forward recursion

The length of the optimal path is the last entry in column (8) of Table 6; in particular, $f_b((4, 0)) = 22$. The optimal solution is identified by using a *backward recovery* algorithm. The procedure starts at the final state and follows the optimal policy backward until an initial state is encountered. The computations are given in Table 7. The last column lists the cumulative path length, confirming once again that when we arrive at the initial state, $z(\mathbf{P}^*) = 22$.

Table 7. Backward recovery

$\mathbf{s}$	$\mathbf{d}^*(\mathbf{s})$	$\mathbf{s}_b$	$z(\mathbf{P}^*)$
(4, 0)	1	(3, –1)	—
(3, –1)	–1	(2, 0)	6
(2, 0)	–1	(1, 1)	11
(1, 1)	1	(0, 0)	14
(0, 0)	—	—	22