MA423 - Fundamentals of Operations Research
Lecture 9: Dynamic Programming

Katerina Papadaki[1]

Academic year 2017/18

[1]London School of Economics and Political Science. Houghton Street London WC2A 2AE (`k.p.papadaki@lse.ac.uk`)

Contents

1 Dynamic Programming **2**
1.1 Introduction 2
1.1.1 The Stagecoach Problem 2
1.2 General Dynamic Programming Problems 4
1.2.1 How does dynamic programming save time? 8
1.3 Distribution of effort 9
1.3.1 Distributing tasks over time 9
1.3.2 Distributing resources to groups 12
1.4 Stochastic Dynamic Programming 14
1.4.1 Mathematical Formulation of Stochastic DPs 16
1.4.2 Stock option example 16

Chapter 1

Dynamic Programming

1.1 Introduction

Dynamic programming (DP) is a general mathematical technique to solve complex optimisation problems by determining the optimal sequence of interrelated decisions. The problem is decomposed into smaller subproblems, and we build up the solution by devising the optimal decisions stage-by-stage. In contrast to linear programming (LP), there is no standard form of dynamic programming. DP is a flexible approach, that can be applied in different contexts. We introduce DP and its main characteristics via a series of examples.

1.1.1 The Stagecoach Problem

The Stagecoach Problem is a standard illustration of DP. It is a shortest path problem, where a 19th century traveller has to reach from his embarkation point A (Missouri) his destination F (California) across the dangerous wild west as shown in Figure 1.1. The possible intermediate stops decompose into layers, called *stages*. The numbers on the edges represent the lengths of these routes. The traveller wishes to find a shortest route.

The DP algorithm for the stagecoach network identifies the shortest path from each node to the sink F, proceeding backwards through the layers. We start with the three *states* (the DP terminology for nodes) in stage E. What is the cost of the shortest routes from each of $E1$, $E2$ and $E3$ to F? This is easy: 7, 5 and 5 respectively.

Let us now move on to stage D. What are the shortest routes from $D1$, $D2$ and $D3$ to F? From $D1$ we have 3 choices for the next step:

- we can go to $E1$. This involves a distance of 8, and the minimum distance from $E1$ to F is 7. Hence the minimum distance from $D1$ via $E1$ to F is $8 + 7 = 15$.
- we can go to $E2$. This involves a distance of 8, and the minimum distance from $E2$ to F is 5. Hence the minimum distance from $D1$ via $E2$ to F is $8 + 5 = 13$.
- we can go to $E3$. This involves a distance of 9, and the minimum distance from $E3$ to F is 5. Hence the minimum distance from $D1$ via $E3$ to F is $9 + 5 = 14$.

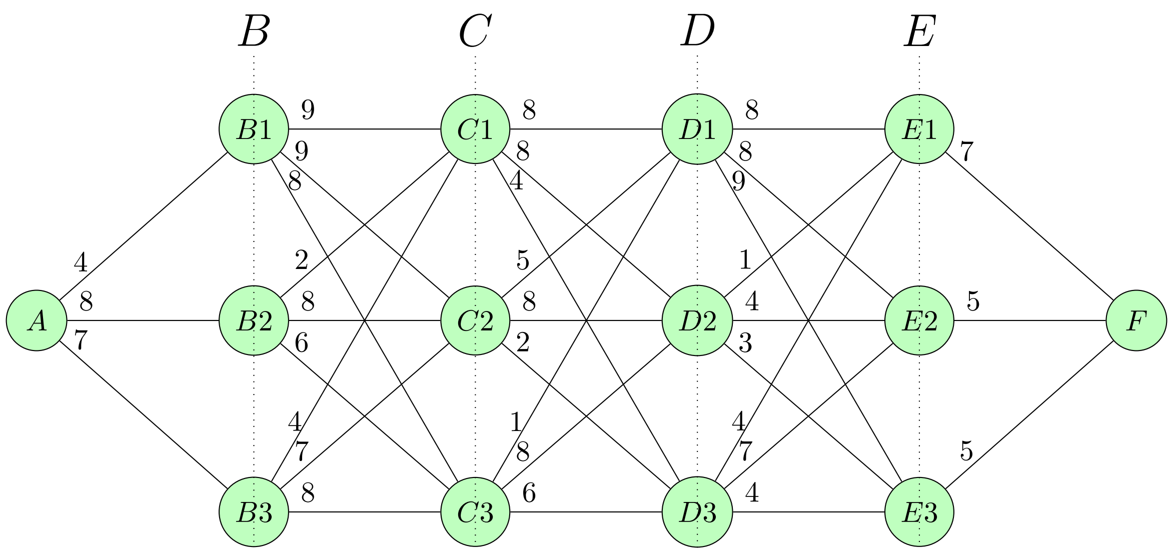

Figure 1.1: Network of the stagecoach problem

Thus the optimal policy at $D1$ is to go to $E2$ (so that the sign-post at $D1$ is towards $E2$), and the shortest path from $D1$ to F is 13.

Now repeat this procedure for nodes D2 and D3. You should find that:

- the optimal policy at $D2$ is to go either to $E1$ or $E3$ (both of these choices are equally good) and the shortest path to F is 8.
- the optimal policy at $D3$ is to go to $E3$ and the shortest path to F is 9.

We now move to layer C and find the optimal policy at nodes $C1$, $C2$ and $C3$. And then the same procedure with $B1$, $B2$ and $B3$, and so on until we find the optimal policy at (i.e. the shortest route from) point A. For each node we calculate the shortest path length to the destination F. Figure 1.2 shows the pair (a, b) next to each node, where a is the optimal distance to F from that node and b is the node to head to in the next stage.

The procedure above implemented the *Backward (Pass) DP* algorithm. We can divide it into the following steps.

- Divide the problem into stages, numbered from the end: stage (5) is being at point A and getting ready to go to layer B; state (4) is being at layer B and going to layer C; stage (3) is being at layer C and going to layer D; stage (2) is being at layer D and going to layer E; finally stage (1) is being at layer E and going to point F.
- Start at the penultimate stage of the problem, which is stage (2) in our example:
- *Repeat:*

  (1) find the optimal policy from each state of the current stage to the ultimate destination, making use of our knowledge of the shortest route for later stages which we have already worked out;

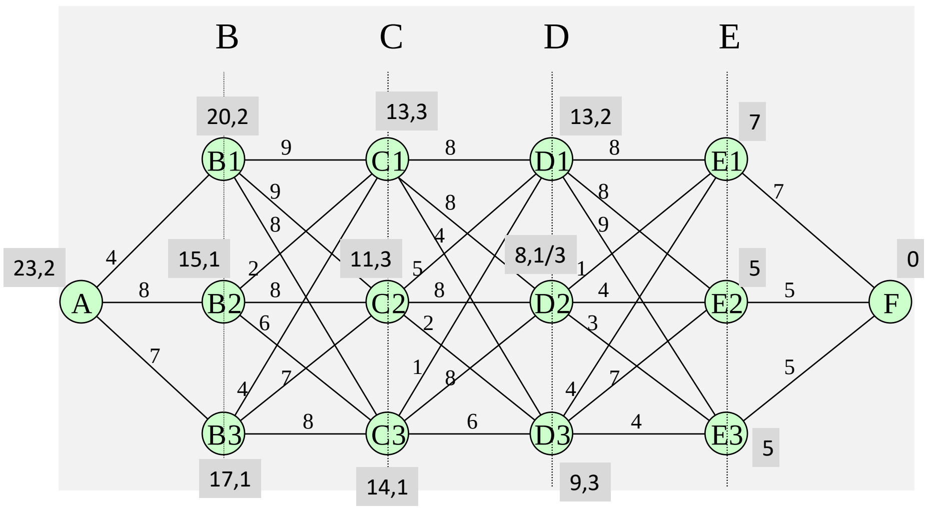

Figure 1.2: Stagecoach problem with (a, b) next to each node, where a is the optimal distance to F from that node and b is the node to head to in the next stage.

(2) record the optimal policy and the minimum cost at each state;

(3) go back one stage.

- Until you get to the last stage, at which point the problem is solved.

We note that the Stagecoach problem is a shortest path problem. Normally Dijkstra's algorithm is used to solve shortest path problems. Here the Stagecoach problem is a special case of a shortest path problem: it decomposes into layers (stages), and one can only move from a node in one layer to a node in the next layer. DP uses a general principle that is applicable to a wide range of problem types. The purpose of the Stagecoach example is to illustrate the main characteristics of DP

1.2 General Dynamic Programming Problems

Dynamic programming is appropriate for problems that can be expressed in the same form as the example above. The characteristics are:

- An objective function must be maximised or minimised.
- Decisions are taken in sequence.
- The problem can be divided into *stages*.
- Each stage has a number of possible *states*.
- An *action* (decision) transforms the current state to a state in the next stage. i.e. It tells us how best to move from the current state to a state in the next stage.
- Rewards or penalties are associated with state transformations.

A crucial assumption is that how a state was reached is irrelevant to future decisions. In the stagecoach example, this means that the shortest path from a certain state to F is the same, irrespective of how we arrived at this state while coming from A. This is also called the *Markov property*, the key property of Markov chains.

We now introduce a mathematical formalism to systematically solve a DP. It is convenient to have a systematic way of expressing these problems.

1) label the stages and states associated with the problem.

Let n be the number of stages remaining at any time. This goes backward in time since our Backward DP algorithm goes backward in time. We assume that there are N stages and thus $n = 1, 2, \dots, N$.

Let (n, s) denote state s of stage n. For example, $C2$ is denoted as $(3, 2)$, as in Figure 1.3. The set of all states is called the state space and denoted by S. Here we take it to be $S = \{1, 2, 3\}$ although it could vary with the stage, for example if there was an additional state $D4$. Sometimes the pair (n, s) is called a state to refer to the fact that in stage n we are in state s.

This yields the following form for the Stagecoach example:

2) label the possible actions (decisions), and the consequences of those actions (i.e. the transitions).

We call the set of possible actions the action space. The way we label an action/decision depends on the problem. For the shortest path example above, it is most convenient to label the decisions as the states we want to get to at the next stage. Thus the possible actions/decisions are 1,2,3 and so the action space is $\{1, 2, 3\}$. We denote by A the action space, which is the set of actions. Sometimes this is denoted by $A(n, s)$ to indicate the set of actions when in (n, s).

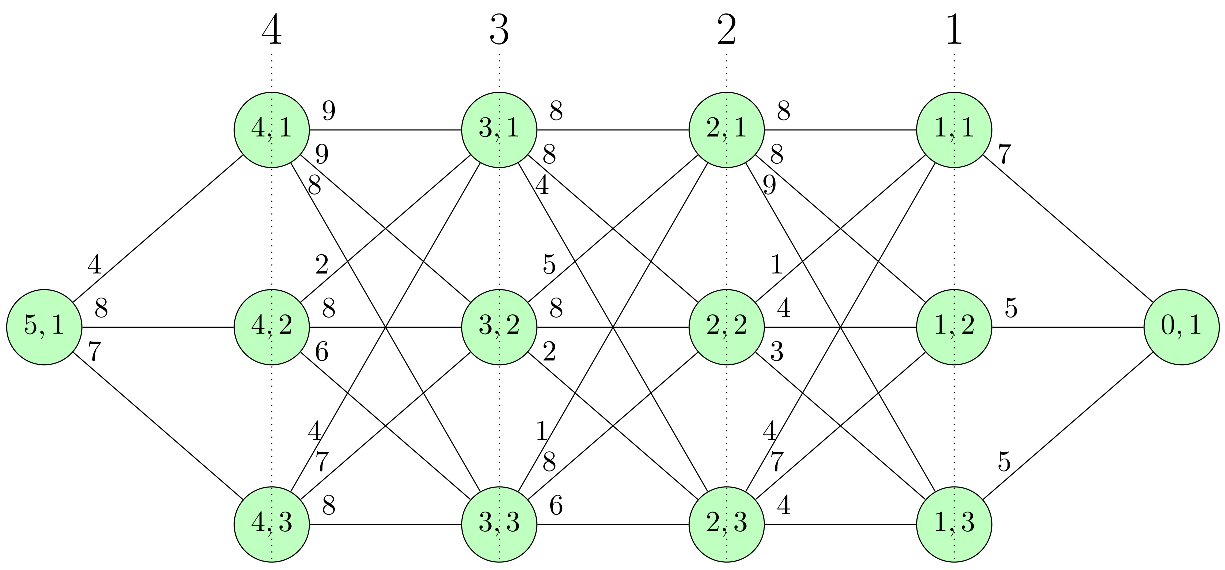

Figure 1.3: Numbering the stages and states in the Stagecoach problem

In the Stagecoach example the action space is $A = \{1, 2, 3\}$ except in stage $n = 1$ where there is only one action.

Let x be a decision variable, so that $x = 1$ means go to state 1. A *decision rule* is a rule (i.e. function) that takes as input the state we are in and returns a decision (i.e. an action). Hence $x(3, 1) = 3$ means that we are at point $(3, 1)$ and choose to go toward point $(2, 3)$. Note that we use the same notation for a decision variable and a decision rule.

Let $t(n, s, x)$ be the *transition function*. This specifies the state that we reach at the next stage $n-1$ as a consequence of our decision. In the Stagecoach problem, we just have $t(n, s, x) = x$: e.g. $t(3, 1, 3) = 3$, i.e. if we are at point $(3, 1)$ and we choose decision 3 we will end up at state 3 of the next stage, i.e. point $(2, 3)$.

This seems to be just an unnecessary formalism at this point. The "distribution effort" example of Section 1.3 will show that it might be advantageous to differentiate between actions and states. This becomes very important in stochastic dynamic programming (next lecture), when we cannot fully control where we arrive in the next stage, and hence a clear distinction between actions and states must be drawn.

3) specify the cost (or reward), c, associated with moving to the next stage.

We describe the cost function $c(n, s, x)$ as a function of the current stage/state and the action we take. That is, if we are currently in state s of stage n, and we take action x then we incur a cost $c(n, s, x)$. In the stagecoach example, $c(2, 1, 3) = 9$, as the cost of the arc from $(2, 1)$ to $(1, 3)$ is 9.

In the stagecoach example, we wish to minimise the total cost. In other applications, the function c may represent a reward incurred during this transition, in which case we wish to maximise the total reward.

4) provide a recursive formula that identifies the optimal policy for states in stage n, provided we have the optimal policies for stage $n-1$.

Let $f(n,s)$ denote the value of state s at stage n: this is the total cost (or reward) from (n,s) onwards (including stage n) to the end stage if the optimal policy is followed. Hence, for the Stagecoach example above, $f(2,1)=13$, since the shortest path from stage 2, state 1 (node (2,1)) to the final stage has cost 13.

We denote by $f(n,s,x)$ to be the overall cost (or reward) associated with a given decision, and it is found by adding the following two terms:

(1) The immediate cost (or reward) $c(n,s,x)$ of going from stage n to stage $n-1$ associated with taking decision x.

(2) The value of the state $t(n,s,x)$ that will be reached via this decision at stage $n-1$, where this value is $f(n-1,t(n,s,x))$.

Thus, we have:

$$f(n,s,x)=c(n,s,x)+f(n-1,t(n,s,x)).$$

The optimal decision is the decision that minimises (if we are working with costs) or maximises (if we are working with rewards) this sum. Thus for a minimisation problem (e.g. the shortest path example), we have

$$f(n,s)=\min_x f(n,x,s),$$

or

$$f(n,s)=\min_x \left[c(n,s,x)+f(n-1,t(n,s,x))\right],$$

whereas for a maximisation problem we have

$$f(n,s)=\max_x f(n,x,s).$$

or

$$f(n,s)=\max_x \left[c(n,s,x)+f(n-1,t(n,s,x))\right].$$

Here x ranges over all possible decisions that are available when in (n,s): $x\in A(n,s)$.

The above formulas provide *Bellman's functional recursion equations* for this scenario.

It is not always possible to decompose the cost/reward into these two terms. What matters is that provided we have all values $f(n-1,s)$, we are able to efficiently compute the values $f(n,s,x)$. Here $f(n,s,x)$ denotes the value of moving from state (n,s) to state $(n-1,t(n,s,x))$, and using the optimal policy from then onwards.

In the Stagecoach problem we had $f(n,s,x)=c(n,s,x)+f(n-1,x)$. A different type of example would be $f(n,s,x)=p(n,s,x)f(n-1,t(n,s,x))$, where $p(n,s,x)$ are given multipliers (see example 1.2).

1.2.1 How does dynamic programming save time?

The purpose of DP is to dramatically reduce the number of calculations that would be needed in complete enumeration. The total number of possibilities can grow *exponentially* with the number of stages and becomes computationally infeasible very soon.

DP can eliminate most possibilities and thus give an efficient algorithm for large instances. The number of computation steps will grow only *linearly* with the number of stages.

To illustrate this, let us compute the total number of elementary computational steps both for complete enumeration and for dynamic programming in the Stagecoach example. We consider elementary arithmetic operations (pairwise additions, subtractions, multiplications, divisions) and comparisons as the relevant computational steps.

It is easy to see that there are $3^4 = 81$ paths between A and F. For each of the 81 paths, we need to perform 4 addition operations to compute the corresponding cost (for example, for path $A \to B2 \to C1 \to D3 \to E2 \to F$ we will need to compute cost by taking the sum $8 + 2 + 4 + 7 + 5 = 26$). This means that, using complete enumeration, the total number of pair-wise additions is $81 \cdot 4 = 324$. Once computed the costs for the 81 paths, we have to compare them to establish the path of minimum cost. This can be done with 80 comparisons. Indeed, suppose for example that, in whichever order we choose the 81 paths, the first 5 path lengths are 92, 86, 87, 83 and 89. Determining that the one of minimum cost among these 5 is the fourth, with cost 83, can be done as follows

(1) Compare 92 and 86; the best so far is 86.

(2) Compare 86 and 87; the best remains 86.

(3) Compare 86 and 83; the best becomes 83; and finally

(4) Compare 83 and 89; the best (i.e. minimum) of the five distances is 83.

Thus five distances require 4 (= 5-1) pair-wise comparisons, and similarly 81 distances require 80 (=81-1) comparisons. Consequently, the total number of (pair-wise) additions and comparisons using complete enumeration is $324 + 80 = 404$.

Using DP. Let us now use DP to compute the total number of arithmetic operations and comparisons.

- First, going from level E to F in stage 1: we simply have 3 distances to record (7, 5 and 5) and no additions or comparisons.
- Next, from level D to level E in stage 2. For example, from $D1$ the shortest distance is the least of $8 + 7$ via $E1$, $8 + 5$ via $E2$, and $9 + 5$ via $E3$, so that in stage 2 we will have three lots of [3 pair-wise additions and 2 pair-wise comparisons], altogether 9 (pair-wise) additions and 6 (pair-wise) comparisons.
- The same applies for stages 3 and 4: 9 additions and 6 comparisons in both these stages.

- Finally, from A to level B in stage 5, we have 3 additions and 2 comparisons.

Consequently in this example, DP requires 30 additions and 20 comparisons, that is, 50 computational steps altogether. Compared with 404 for complete enumeration, we see that DP is approximately 8 times more efficient than complete enumeration. If the number of stages increases, this factor will become much larger. It is a matter of seconds to solve instances with hundreds of thousands of states via DP, whereas the entire lifespan of the universe would still not be enough to perform complete enumeration, even using today's fastest supercomputers.

Let us compute these with respect to N, $|S|$ and $|A|$.

For complete enumeration for each path we do $N-1$ additions to calculate the length and one comparison to keep the lowest length path, thus in total N operations. The number of paths can be calculated as follows: on a particular path in each stage the path can be at one of the $|S|$ states and this happens for stages in $n = 1, 2, \ldots, N-1$. Thus, the number of paths is $|S|^{N-1}$. So the total number of operations is $(N-1)|S|^{N-1}$.

In backward dynamic programming, for each stage $n = 1, 2, \ldots, N$ we calculate for each state $s \in S$ the value function $f(n, s)$. To calculate $f(n, s)$ we evaluate $f(n, s, x)$ for each $x \in A$. To evaluate each $x \in A$ we perform one addition and one comparison: so 2 operations for each $x \in A$. Thus, the total number of operations is $N \cdot |S| \cdot |A| \cdot 2$.

Comparing the two we can see that with complete enumeration the complexity is exponential in the number of stages N, whereas the backward DP is linear in N. This means that when N is large, complete enumeration would be impossible but backward DP would perform in reasonable times. However, note that when the state variable is a vector of a large dimension then backward DP also suffers from complexity issues and this is called the *curse of dimensionality.*

1.3 Distribution of effort

Let us now consider another common application of dynamic programming, which can be used to find the most efficient distribution of tasks over different periods, or finding the most efficient use of certain resources. This will be illustrated by two examples:

1.3.1 Distributing tasks over time

Example 1.1. *An engineering company is contracted to build nine trains during a 3-month period, so that the nine trains can be delivered to the customer at the end of the third month. For whatever reason, the cost of building x trains in month m is $w_m x^2$, where the values for w_m are given in the table below, and w_m is measured in millions of £. E.g. If they build 4 trains in month 2, the total cost then will be* $3 \cdot 16 = 48$ *(£millions).*

Month m	*1*	*2*	*3*
w_m	*6*	*3*	*2*

How many trains should be built in each month to minimise total production costs?

We formulate this as a DP problem. Let

$$\begin{aligned} n &= \text{number of months remaining at the start of any stage (i.e. month);} \\ s &= \text{total number of trains still to be produced at the start of this stage;} \\ x &= \text{number of trains to produce during the current stage;} \\ c(n,s,x) &= \text{cost of producing } x \text{ trains in state } s \text{ at stage } n. \end{aligned}$$

In stage 1 we are at the beginning of month 3: $n = 1$ corresponds to $m = 3$. Similarly, $n = 2$ corresponds to $m = 2$, and $n = 1$ to $m = 3$. Note that the relationship between m and n is given by $m = 4 - n$ for $n = 1, 2, 3$. Consequently, when computing $c(n, s, x)$, we use the multiplier w_m for $m = 4 - n$, that is,

$$c(n,s,x) = w_{4-n}x^2$$

In this example, the natural transition function to use is $t(n, s, x) = s - x$: if there are s trains still to be produced, and we produce x new ones in the current stage, then $s - x$ trains will be left to be produced in subsequent stages. Thus, the state at stage $n - 1$ will be $t(n, s, x) = s - x$.

By the end of month 3, the entire demand has to be satisfied. Henc from state $(1, s)$, the only allowed action is $x = s$: in month three, the company must produce whatever it takes to meet the full demand. For all other stages, the set of possible actions available in state s are $0 \leq x \leq s$. We impose $x \leq s$ as the company should not build more than the required number of trains.

In month 1, we have the entire demand 9 to satisfy. Hence for $n = 3$ there is only one state, namely, $(3, 9)$.

The Bellman equation for $n = 2, 3$ is:

$$f(n,s) = \min_x [w_{4-n}x^2 + f(n-1, s-x)],$$

where the set of actions is $0 \leq x \leq s$ for $n = 2, 3$. For $n = 1$ we must produce all remaining trains and thus $x = s$ which gives $f(1, s) = w_3s^2$ (we do not define $f(0, s)$). The optimal cost to the problem will be given by the variable $f(3, 9)$, since we start at stage $n = 3$ and we have $s = 9$ trains to produce.

Let $x^*(n, s)$ denote the optimal production at stage n in state s.

Stage 1 $(n = 1)$, beginning of month 3

This is month 3, with $x = s$ being the only possible action. Hence $x^* = s$, and $f(1, s) = w_3s^2 = 2s^2$. Thus we obtain the table

s	$x^*(1,s)$	$f(1,s)$
0	0	0
1	1	2
2	2	8
3	3	18
4	4	32
5	5	50
6	6	72
7	7	98
8	8	128
9	9	162

Stage 2 ($n = 2$)**, beginning of month 2**

Now $w_2 = 3$, and hence $f(2, s) = \min\{3x^2 + f(1, s - x) : 0 \le x \le s\}$.

Working through all the options and choosing the optimum x for each s yields:

$s \backslash x$	0	1	2	3	4	5	6	7	8	9	$x^*(2,s)$	$f(2,s)$
0	**0**										0	0
1	**2**	3									0	2
2	8	**5**	12								1	5
3	18	**11**	14	27							1	11
4	32	21	**20**	29	48						2	20
5	50	35	**30**	35	50	75					2	30
6	72	53	**44**	45	56	77	108				2	44
7	98	75	62	**59**	66	83	110	147			3	59
8	128	101	84	**77**	80	93	116	149	192		3	77
9	162	131	110	99	**98**	107	126	155	194	243	4	98

E.g. when $s = 3$, the formula becomes $f(2, 3) = \min\{3x^2 + f(1, 3 - x)\}$, and x can be 0, 1, 2 or 3. Substituting these values, we get:

$$\begin{aligned} f(2,3) &= \min\{3 \cdot 0^2 + f(1,3),\ 3 \cdot 1^2 + f(1,2),\ 3 \cdot 2^2 + f(1,1),\ 3 \cdot 3^2 + f(1,0)\} \\ &= \min\{18, 11, 14, 27\} = 11. \end{aligned}$$

These are the values in row 3. In each row, the optimal value is **bold**; these provide the x^* and $f(2, s)$ values.

Stage 3 ($n = 3$)**, beginning of month 3**

This is the start of the production process, and we only have a single state $s = 9$. We use $w_3 = 6$, and thus we have to compute $f(3, 9) = \min\{6x^2 + f(2, 9 - x)\}$. Working through all the possible values of x from 0 to 9 and choosing the optimum yields $x^*(3, 9) = 1$ or 2. These two choices both give $f(3, 9) = 83$.

$s\backslash x$	0	1	2	3	4	5	6	7	8	9	x^*	$f(2,s)$
9	98	**83**	**83**	98	126	170	227	299	386	486	1,2	83

Evaluating the optimal policy

We now work forward to evaluate the outcome of the optimal policy. We obtain two possible solutions for $x^*(3,9) = 1$ and for $x^*(3,9) = 2$.

- $x_3 = x^*(3,9) = 1$ moves to state $(2,8)$, giving $x_2 = x^*(2,8) = 3$. Then we move to state $(1,5)$, with optimal policy $x_1^* = 5$. Hence we produce 1 train in month 1, 3 trains in month 2, and 5 trains in month 3.
- $x_3 = x^*(3,9) = 2$ moves to state $(2,7)$, giving $x_2 = x^*(2,7) = 3$. Then we move to state $(1,4)$, with optimal policy $x_1^* = 4$. Hence we produce 2 trains in month 1, 3 trains in month 2, and 4 trains in month 3.

Possibly a secondary objective might be applied in order to choose between the two possible production schedules, and a sensible one would be to minimise the variability from month to month, so that the second schedule above would then be chosen.

1.3.2 Distributing resources to groups

We now give another example of distribution of effort is given. This example is from Sec 11.3 of the Hillier-Lieberman book (ed. 7).

Example 1.2. *The EU space agency is conducting research to solve a certain engineering problem needed for a Mars mission. Three research teams (1,2, and 3) are currently working on the project, using different approaches. The problem is solved if at least one of them succeed. It is estimated that under the current circumstances, the three teams will fail with probabilities 0.4, 0.6 and 0.8, respectively. Hence the current probability of failure is* $0.4 \cdot 0.6 \cdot 0.8 = 0.192$.

The agency has therefore mandated two more top scientists to these projects. The table shows the probabilities that each of the teams fail if 0, 1 or 2 more scientists are added. (Columns correspond to the teams, and rows to the number of new scientists.) Determine how the two scientists should be allocated to minimize the probability that all three teams will fail.

no of		teams	
scientists	1	2	3
0	0.4	0.6	0.8
1	0.2	0.4	0.5
2	0.15	0.2	0.3

Although the question involves probabilities, it is still a deterministic dynamic programming question. Our aim is to assign integer number of scientists to each project, x_1, x_2, and x_3, under the constraint $x_1 + x_2 + x_3 = 2$, such that the product

$$p_1(x_1)p_2(x_2)p_3(x_3)$$

is minimized, where $p_i(x_i)$ is the failure probability of team i if x_i additional researchers are added.

Let us formulate this as a dynamic programming problem. Let us first assign scientists to team 3, then to team 2, then to team 1 (this could be done in an arbitrary order). Hence stage $n = 1, 2, 3$ corresponds to assigning scientists to team n. The state at stage n, (n, s) represents the number of scientists s remaining at this stage, and the action x in state (n, s) is the number of scientists assigned to team n. Clearly, in state (n, s) the only actions available are $x \leq s$ because we cannot assign more scientists than available.

The natural transition function is $t(n, s, x) = s - x$: if we allocate x scientists at stage n, then $s - x$ remain for the further stages.

The cost function $f(n, s)$ in state (n, s) is the minimum probability that all teams $1, \ldots, n$ fail, if at stage n we have s scientists available to assign to them. For stage $n = 1$,

$$f(1, s) = \min_{x \leq s} p_1(x)$$

for $s = 0, 1, 2$. Bellman's equation for stages $n = 2, 3$ gives

$$f(n, s) = \min_{x \leq s} p_n(x) f(n - 1, s - x)$$

for $s = 0, 1, 2$. This is remarkably different from the previous examples: we do not have an additive cost $c(n, s, x)$, but we multiply the probability $p_n(x)$ of team n failing if we assign x researchers to it, with the optimum value $f(n - 1, s - x)$ from the previous stage. Still, this is an appropriate functional recursion equation, as it enables to compute $f(n, s)$ from the values at stage $n - 1$.

Let $x^*(n, s)$ denote the optimal action at stage n in state s.

Stage 1 $(n = 1)$

For team 1, we could in theory allocate $x < s$ scientists. This would make sense if adding more scientists could increase the failure rate. This is not the case according to the table: the failure rate decreases as we add more scientists and thus the optimal policy is to assign the maximum number available. So, $x^*(1, s) = s$.

s	$x^*(1, s)$	$f(1, s)$
0	0	0.4
1	1	0.2
2	2	0.15

Stage 2 $(n = 2)$

We now have $f(2, s) = \min\{p_2(x) f(1, s - x) : 0 \leq x \leq s\}$. We have all possible options listed below:

$s\backslash x$	0	1	2	$x^*(2,s)$	$f(2,s)$
0	0.6(0.4)=**0.24**			0	0.24
1	0.6(0.2)=**0.12**	0.4(0.4)=0.16		0	0.12
2	0.6(0.15)=0.09	0.4(0.2)=**0.08**	0.2(0.4)=**0.08**	1,2	0.08

Stage 3 ($n = 3$)

As we start the assignment with Team 3, we have a single state $s = 2$: all scientists are still available. The formula is $f(3, 2) = \min\{p_3(x)f(2, 2 - x)$.

$s\backslash x$	0	1	2	x^*	$f(3,s)$
2	0.8(0.08)=0.064	0.5(0.12)=**0.06**	0.3(0.24)=0.072	1	0.06

Evaluating the optimal policy

The value $f(3, s) = 0.06 = 6\%$ gives the answer: this is the lowest joint failure probability that could be achieved by assigning additional scientists. To evaluate the optimal policy, we first see that $x^*(3, 2) = 1$: we should assign one scientists to team 3. This moves to state $(2, 1)$. For state $(2, 1)$, the optimal policy is $x^*(2, 1) = 0$. This moves to state $(1, 1)$, with $x^*(1, 1) = 1$.

Hence we have a single optimal solution: one scientist should be assigned to Team 1, and one to Team 3.

1.4 Stochastic Dynamic Programming

We now demonstrate how dynamic programming can be applied to settings involving uncertainty. Let us revisit the stagecoach example with the network in Figure 1.3. We want to get to the point labelled (0,1) from the point labelled (5,1), minimising the distance travelled.

But we will now assume that the coach sometimes takes the wrong turns, due to poor maps of the wild west. More precisely, at all the nodes of stages $2, 3, 4, 5$, the probability that we traverse the link that we decided to take will be 0.5, whereas we will traverse the two wrong links with probability of 0.25 each.

For example if you were at $(2, 3)$ and your preferred link was to $(1, 3)$, there would be a 50% chance that you would take the link to $(1, 3)$, a 25% chance that you would do a mistake and take the link to $(1, 2)$, and a 25% chance that you would do a mistake and take the link to $(1, 1)$. Of course, from $(1, 2)$ you are certain to get to $(0, 1)$.

Hence the expected distance to be covered from $(2, 3)$ to $(0, 1)$ will not be 9 via $(1, 3)$ as in the original deterministic problem, but $0.5 \cdot (4 + 5) + 0.25 \cdot (4 + 7) + 0.25 \cdot (7 + 5) = 10.25$.

As consequences of the randomness in outcomes, there are two aspects in which this problem differs from the deterministic shortest path.

(i) We can no longer think in terms of simply maximising or minimising an objective function. Instead, with stochastic dynamic programming (SDP) we are normally concerned with maximising or minimising the *expected value* of an objective function.

(ii) The solution is no longer a single optimal route. Rather, the optimal solution is a policy, which to each possible state assigns an action (namely, the preferred link to cross), whose outcome is uncertain due to the intrinsic randomness of the process. However, in expectation the optimal value of the objective is achieved.

Assume that we want to minimise the expected distance through the network. What is the optimal policy at each node? What is the expected distance from $(5, 1)$ to $(0, 1)$?

Begin with the states (nodes) at the last stage of the problem. There is no randomness involved. Thus: from $(1, 1)$ the distance is 7; from $(1, 2)$ the distance is 5; and from $(1, 3)$ the distance is 5.

Now consider the nodes at stage 2. What are the optimal policies from $(2, 1)$, $(2, 2)$ and $(2, 3)$?

From $(2, 1)$ we first compute the distances if we were able to follow the 3 possible links (just as in the deterministic setting):

- we can go to $(1, 1)$. This involves a distance of 8, and the distance following the optimal policy from $(1, 1)$ to $(0, 1)$ is 7. Hence the distance from $(2, 1)$ via $(1, 1)$ to $(0, 1)$ is $8 + 7 = 15$.
- we can go to $(1, 2)$. This involves a distance of 8, and the distance following the optimal policy from $(1, 2)$ to $(0, 1)$ is 5. Hence the distance from $(2, 1)$ via $(1, 2)$ to $(0, 1)$ is $8 + 5 = 13$.
- we can go to $(1, 3)$. This involves a distance of 9, and the minimum distance following the optimal policy from $(1, 3)$ to $(0, 1)$ is 5. Hence the distance from $(2, 1)$ via $(1, 3)$ to $(0, 1)$ is $9 + 5 = 14$.

Recall that the different actions correspond to intended directions; however, we only get there with probability 50% and end up using the other links with probability 25%

- *Intended action* $(1, 1)$: the expected distance to $(0, 1)$ is $(0.5 \cdot 15) + (0.25 \cdot 13) + (0.25 \cdot 14) = 14.25$
- *Intended action* $(1, 2)$*:* the expected distance to $(0, 1)$ is $(0.25 \cdot 15) + (0.5 \cdot 13) + (0.25 \cdot 14) = 13.75$
- *Intended action* $(1, 2)$*:* the expected distance to $(0, 1)$ is $(0.25 \cdot 15) + (0.25 \cdot 13) + (0.5 \cdot 14) = 14$

Hence the optimal policy at $(2, 1)$ is to attempt to go to $(1, 2)$, and the expected distance to $(0, 1)$ if the optimal policy is followed is 13.75.

So we can similarly find the optimal policy and expected distance for the other states at stage 2. We then work backwards to find the optimal policy and expected distance for all possible states in earlier stages. Etc., etc.

In general, note that the optimal policy for a stochastic shortest route problem may look quite different from that for the deterministic case.

1.4.1 Mathematical Formulation of Stochastic DPs

In the above example actions corresponded to the states in the next stage. The driver had an intended direction, but with a certain probability he was not able to get there. The model is more general than that: actions can correspond to arbitrary probability distributions on states. For example, another possible action from state $(2, 1)$ could be to follow every link with equal probability.

Consider state s at stage n. Every possible action x corresponds to a probability distribution on the states of stage $n-1$. Formally, given an action x available at a state (s, n), the probability that action x will lead to state j in stage $n - 1$ is denoted by $p(n, s, x, j)$. We require for each state (n, s) and each action x available from s that

$$\sum_j p(n, s, x, j) = 1$$

holds, where summation is over all states j at stage $(n-1)$. Note that in the deterministic case the transition function $t(n, s, x)$ determined the state at stage $n - 1$. Here this is determined by the transition probabilities $p(n, s, x, j)$.

We define $f(n, s)$ as the *expected cost (or reward)* from state (n, s) onwards, until the final state is reached.

Assume we have a cost function $c(n, s, x, j)$ giving the cost of taking action x from (n, s), and arriving at state $(n - 1, j)$. Then the functional recursion equation for a cost minimisation problem is given as

$$f(n, s) = \min_x \sum_j p(n, s, x, j) \left(c(n, s, x, j) + f(n - 1, j) \right).$$

In the stagecoach model, the cost $c(n, s, x, j)$ did not depend on the action x (i.e. where we intended to go), but only on (n, s) and j, that is, where we really ended up. In other models, the cost might be independent on j, and only depend on the action taken.

1.4.2 Stock option example

We have the option of buying a certain amount of stock of a company for £99 during the next 3 days, regardless of the market price of the stock on the day. We have no obligation to exercise this option, but if we do, we can do so in any of the next 3 days. We can exercise the option at most once.

The price on day 1 is £100, and the stock price changes randomly according to the following stochastic model: every day, the price can go up by £1 with 40% probability, can stay the same with 20% probability, and can decrease by £1 with 40% probability. Our aim is to devise an optimal policy for exercising the option: every day, depending on the current price, we need to decide whether to exercise the option to buy the stock, or wait to see what happens the next day.

If the price on any date is $s > 99$, then our revenue is $s - 99$ if we use the option. On day 1 the price is 100, and we can gain 1 if we exercise the option but we might be better off if we wait. This is what the DP will help us decide.

We use stochastic dynamic programming to model this problem. Our stages correspond to the days, numbered backwards: stage $n = 1$ is the beginning of day 3, stage $n = 2$ is the beginning of day 2, and $n = 3$ is the beginning of day 1.

The states correspond to the current price: (n, s) means that in stage n, the stock price is s, and we have not yet used the option; there will be a further special state $(n, \star)$. For $n = 1$ (day 3), the price can be between 98 and 102, as it could have increased or decreased by at most 2. Hence we have 5 possible states: $98 \leq s \leq 102$. For $n = 2$ (day 2), the possible states are $99 \leq s \leq 101$. For $n = 3$, we have only one state, $s = 100$: this is the price of the stock on day 1.

The special state $(n, \star)$ means that we have already used the option before. That is, $(2, \star)$ means that we used the option in stage $n = 3$ (day 1).

The value $f(n, s)$ will be the expected maximum revenue we can achieve from state (n, s). The value of the special state is always $f(n, \star) = 0$. There are two possible actions in state (n, s): either we buy the stock using the option, or we wait.

- **x = buy.** In this case, our revenue is $s - 99$; this can be negative if the current price s is lower than £99. We move from state (n, s) to the special state $(n - 1, \star)$.
- **x = wait.** In this case, we will move to one of the states $(n - 1, s - 1)$, $(n - 1, s)$, and $(n - 1, s + 1)$, with respective probabilities 40%, 20%, and 40%. Our expected revenue will be
$$0.4f(n-1, s-1) + 0.2f(n-1, s) + 0.4f(n-1, s+1).$$
This applies for $n = 2, 3 > 1$; for $n = 1$, if we do not use the option on the last day, the revenue will be 0.

The Bellman equations are obtained by taking the maximum of the two values. For $n > 1$, it gives

$$f(n, s) = \max\{s - 99, 0.4f(n-1, s-1) + 0.2f(n-1, s) + 0.4f(n-1, s+1)\},$$

and for $n = 1$, we have
$$f(1, s) = \max\{s - 99, 0\}.$$

n=1 The stock price can be any of $98, 99, 100, 101, 102$. From bellman recursive equations we have: $f(1, s) = maxs - 99, 0$, which means we exercise the option only if we gain some profit. Also, $f(1, *) = 0$.

s	x*(1,s)	f(1,s)
98	wait	0
99	wait/buy	0
100	buy	1
101	buy	2
102	buy	3
*	--	0

n=2 The stock price can be any of $99, 100, 101$. The bellman recursive equations are:

$$f(2,s) = max\{s-99, 0.4f(1,s-1) + 0.2f(1,s) + 0.4f(1,s+1)\}$$

with $f(2,*) = 0$.

s/x	x = buy	x = wait	x*(2,s)	f(2,s)
99	0	0.4(0) + 0.2(0) + 0.4(1) = 0.4	wait	0.4
100	1	0.4(0) + 0.2(1) + 0.4(2) = 1	buy/wait	1
101	2	0.4(1) + 0.2(2) + 0.4(3) = 2	buy/wait	2
*	--	--	--	0

n=3 The stock price is 100. The bellman recursive equations are:

$$f(3,s) = max\{s-99, 0.4f(2,s-1) + 0.2f(2,s) + 0.4f(2,s+1)\}$$

and there is no $(3,*)$ state since we have not yet had a chance to exercise the option.

s/x	x = buy	x = wait	x*(3,s)	f(3,s)
100	1	0.4(0.4) + 0.2(1) + 0.4(2) = 1.16	wait	1.16

Thus, the optimal policy is:

- On day 1
- On day 2 wait if the price is 99 but either buy or wait if the price is higher
- On day 3 buy only if the price is 99 or more i.e. buy only if we make a profit

Large stock option example

Now suppose that the option is to buy the stock at the price of 103 and instead of 3 days we have 10 days to exercise the option. Everything else remains the same. In this case $n = 1, \dots, 10$ and the possible states are $91, \dots, 109$.

The next table shows the values of $f(n, s)$ rounded to the second decimal place.

$f(n,s)$	n=1	n=2	n=3	n=4	n=5	n=6	n=7	n=8	n=9	n=10
s=91	0									
s=92	0	0								
s=93	0	0	0							
s=94	0	0	0	0						
s=95	0	0	0	0	0					
s=96	0	0	0	0	0	0				
s=97	0	0	0	0	0	0	0			
s=98	0	0	0	0	0	0	0.01	0.02		
s=99	0	0	0	0	0.01	0.02	0.04	0.05	0.07	
s=100	0	0	0	0.03	0.05	0.08	0.11	0.14	0.17	0.2
s=101	0	0	0.06	0.1	0.16	0.2	0.26	0.3	0.35	
s=102	0	0.16	0.22	0.32	0.39	0.46	0.52	0.58		
s=103	0.4	0.48	0.62	0.7	0.8	0.87	0.94			
s=104	1	1.16	1.22	1.32	1.39	1.46				
s=105	2	2	2.06	2.1	2.16					
s=106	3	3	3	3.03						
s=107	4	4	4							
s=108	5	5								
s=109	6									

The value of the final state $f(10, 100) = 0.202418$ shows that the expected revenue we can get from this option is 20 pence.

To decide whether to buy or wait at a certain state (n, s), we need to compare $s - 103$ with $f(n, s)$. If $f(n, s) > s - 103$ then we wait, because the expected revenue is higher than what we would make by selling on this day; otherwise we must have $f(n, s) = s - 103$, in which case we buy because we earn exactly the optimal expected revenue for state (n, s).

For example, $f(4, 106) = 3.0256$. This means that if on day 6 ($n = 4$) the price is 106, the optimal policy is to wait, as the expected revenue is higher than the 3 we would gain by buying the stock today.