2.2 Shortest path problems

To consider a shortest path problem in general, we need to introduce some new concepts.

Definition: A **graph**, or **network**, is defined by two sets of symbols: *nodes* and *arcs*. An **arc** consists of a pair of nodes, and represents a possible direction of motion between the two nodes. The *length* of an arc from node i to node j is denoted by d_{ij}. In general, d_{ij} could be negative. Note that the length of an arc does not necessarily mean the physical length, it could stand for some very general quantity associated with the arc, say for example, cost.

Definition: An arc is said to be **directed** if it is only allowed in one direction. Usually the direction is indicated by an arrowhead at the end of the arc. An arc is said to be **undirected** if both directions are allowed. In this case, there is no arrowhead on the arc. An undirected arc can be equivalently represented by two directed arcs.

Definition: A **path** is a sequence of arcs such that the terminal node of each arc is identical to the initial node of the next arc.

Below we will consider the shortest path problem for a network. Suppose the nodes of a network are denoted by $\{1, 2, \cdots, N\}$, and one wants to find a path from node 1 to node N with the shortest total length.

2.2.1 Shortest path for simple networks

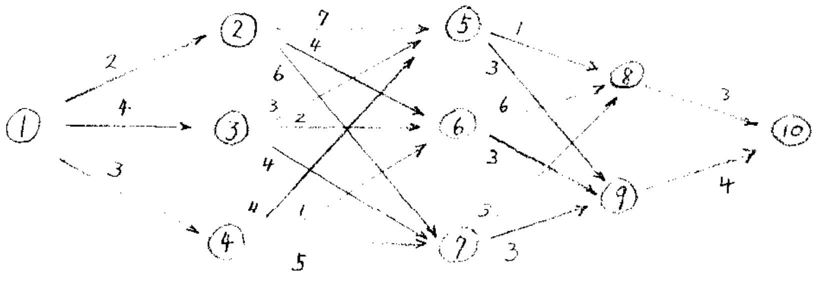
As we have seen in Section 1.2, we can use dynamic programming to solve the shortest path problem if the nodes can be divided into groups, which we call "stage", and one always travels from a node in one stage to a node in the next stage.

Assume node 1 is in stage 0 and node N is in stage K. The minimal distance from node 1 to node N can be recursively determined by the DPE that for any node x in stage k,

$$V_k(x) = \min\{d_{xy} + V_{k+1}(y) : \ y \in \text{Stage}(k+1)\}; \quad k = K-1, K-2, \cdots, 1.$$

and $V_K(N) = 0$. With convention $d_{xy} \doteq \infty$ if there is no arc from x to y.

Exercise: Consider the following network. Find the shortest path from node 1 to node 10. Also Find the shortest path from node 3 to node 10.



Solution: There are three shortest paths: $1 \to 3 \to 5 \to 8 \to 10$, $1 \to 4 \to 6 \to 9 \to 10$, and $1 \to 4 \to 5 \to 8 \to 10$. Each of these path has total length 11. The shortest path from node 3 to node 10 is $3 \to 5 \to 8 \to 10$ with total length 7.

2.2.2 Shortest path by Dijkstra's algorithm

An assumption here is the length of each arc is non-negative. Even though the Dijkstra's algorithm is implicitly defined by a DPE, it is not necessary to specify the stages.

The idea is as follows. Instead of searching the shortest path from node 1 to node N, we search the shortest path from node 1 to node n for any node $n \in \{1, 2, \cdots, N\}$. Define

$$v(j) = \text{the shortest distance from node 1 to the } j\text{-th nearest node},$$

and with convention, the 1st nearest node is node 1 itself. Suppose N is the k-th nearest node to node 1, then the shortest path from node 1 to node N is $v(k)$. The key observation is that

> *The shortest path from node 1 to the $(j+1)$-th nearest node only passes through nodes contained in the j nearest nodes to node 1. In other words, there exists an $1 \le i \le j$ such that the shortest path from node 1 to the $(j+1)$-th nearest node will consist of a shortest path to the i-th nearest node and an arc connecting the i-th nearest node and the $(j+1)$-th nearest node.*

Let m_i denote the i-th nearest node to node 1 and $A_j = \{m_1, \cdots, m_j\}$. The equation to determine the $(j+1)$-th nearest node and $v(j+1)$ is

$$v(j+1) = \min_{1\le i\le j} \min_{n \notin A_j} [v(i) + d_{m_i n}] = \min_{1 \le i \le j} \left[v(i) + \min_{n \notin A_j} d_{m_i n} \right],$$

with convention $d_{mn} = \infty$ if no arc from node m to node n. The minimizing node n^* in the RHS will be the $(j+1)$-the nearest node to node 1. The shortest path to from node 1 to node n^* is the shortest path to m_{i^*} (the minimizing node m_i) and then the arc from m_{i^*} to n^*.

We will illustrate the algorithm by the following examples. Figure 1 is a undirected network, i.e. every arc is undirected. The arcs in Figure 2 is mixed. In both network, the goal is to determine the shortest path from node 1 to node 8.

Figure 1.

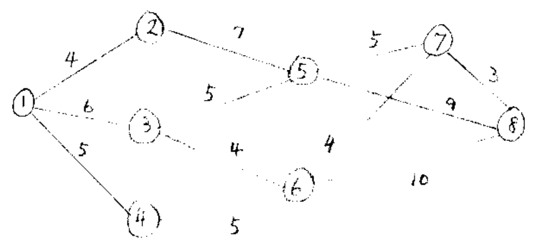
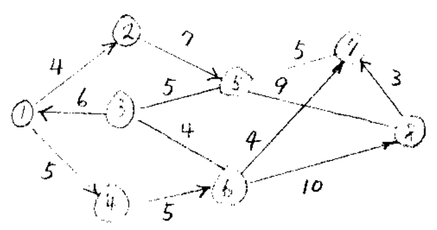


Figure 2.



For Figure 1, there are two shortest path $1 \to 3 \to 6 \to 7 \to 8$ and $1 \to 4 \to 6 \to 7 \to 8$. Each has total length 17. For Figure 2, there are also two shortest path $1 \to 4 \to 6 \to 8$ and $1 \to 2 \to 5 \to 8$, each has total length 20.

Figure 1.

n	Solved Nodes	Closest Connected Unsolved Nodes	Total Distance	n-th nearest Node	Minimum Distance	Last Arc
1		1	0	1	0	$1 \to 1$
2	1	2	4	2	4	$1 \to 2$
3	1	4	5	4	5	$1 \to 4$
	2	5	11			
4	1	3	6	3	6	$1 \to 3$
	2	5	11			
	4	6	10			
5	2	5	11			
	4	6	10	6	10	$4 \to 6$
	3	6	10			$3 \to 6$
6	2	5	11	5	11	$2 \to 5$
	3	5	11			$3 \to 5$
	6	7	14			
7	5	7	16			
	6	7	14	7	14	$6 \to 7$
8	6	8	20			
	7	8	17	8	17	$7 \to 8$

Figure 2.

n	Solved Nodes	Closest Connected Unsolved Nodes	Total Distance	n-th nearest Node	Minimum Distance	Last Arc
1		1	0	1	0	$1 \to 1$
2	1	2	4	2	4	$1 \to 2$
3	1	4	5	4	5	$1 \to 4$
	2	5	11			
4	2	5	11			
	4	6	10	6	10	$4 \to 6$
5	2	5	11	5	11	$2 \to 5$
	6	3	14			
	6	7	14			
6	6	3	14	3	14	$6 \to 3$
	6	7	14	7	14	$6 \to 7$
	5	3	16			
	5	7	16			
7	6	8	20	8	20	$6 \to 8$
	5	8	20			$5 \to 8$

Exercise: In Figure 2, what is the shortest path from node 1 to node 3?

2.2.3 Shortest path in general network

Below we present another algorithm which is explicitly defined by a recursive DPE. This algorithm is very general and does not require the length of each arc to be non-negative. Define function

$$v_j(n) \doteq \text{the length of the shortest path from node 1 to node } n \text{ when } j \text{ or fewer arcs must be used.}$$

When there is no path from node 1 to node n with j or fewer arcs, then $v_j(n) \doteq \infty$.

With this formulation, one get the shortest path from node 1 to any other nodes n; indeed, the shortest path from node 1 to node n has a total length $v_j(n)$ when $j = N-1$ since the path can contain at most $N-1$ arcs.

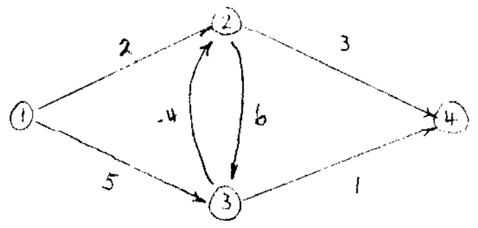
The DPE of v_j is given by

$$v_{j+1}(n) = \min_{m \neq n} [v_j(m) + d_{mn}], \quad \forall\, j = 0, 1, \cdots, N-1, \;\; n = 1, 2, \cdots, N$$

(except for node $n = 1$, include $m = n = 1$ in the minimization). Note in this formulation, j represents the stage, and m, n represent the nodes. The initial conditions are given by

$$v_0(n) = \left\{ \begin{array}{lll} 0 & ; & \text{if } n = 1 \\ \infty & ; & \text{if } n = 2, 3, \cdots, N \end{array} \right\}.$$

Consider the following example.



The calculation are as follows.

$$\begin{aligned} v_0(1) &= 0 \\ v_0(2) &= \infty \\ v_0(3) &= \infty \\ v_0(4) &= \infty. \end{aligned}$$

$$\begin{aligned} v_1(1) &= \min[0+0, \infty+\infty, \infty+\infty, \infty+\infty] = 0, \quad (m^* = 1) \\ v_1(2) &= \min[0+2, \infty-4, \infty+\infty] = 2, \quad (m^* = 1) \\ v_1(3) &= \min[0+5, \infty+6, \infty+\infty] = 5, \quad (m^* = 1) \\ v_1(4) &= \min[0+\infty, \infty+3, \infty+1] = \infty, \quad (m^* = 1, 2, 3). \end{aligned}$$

$$\begin{aligned} v_2(1) &= \min[0+0, 2+\infty, 5+\infty, \infty+\infty] = 0, \quad (m^* = 1) \\ v_2(2) &= \min[0+2, 5-4, \infty+\infty] = 1, \quad (m^* = 3) \\ v_2(3) &= \min[0+5, 2+6, \infty+\infty] = 5, \quad (m^* = 1) \\ v_2(4) &= \min[0+\infty, 2+3, 5+1] = 5, \quad (m^* = 2). \end{aligned}$$

$$\begin{aligned}
v_3(1) &= \min[0+0, 1+\infty, 5+\infty, 5+\infty] = 0, \quad (m^* = 1)\\
v_3(2) &= \min[0+2, 5-4, 5+\infty] = 1, \quad (m^* = 3)\\
v_3(3) &= \min[0+5, 1+6, 5+\infty] = 5, \quad (m^* = 1)\\
v_3(4) &= \min[0+\infty, 1+3, 5+1] = 4, \quad (m^* = 2).
\end{aligned}$$

Exercise: Compute v_4 for each node, and verify that $v_4 = v_3$.

Therefore, the shortest path from node 1 to node 4 is $1 \to 3 \to 2 \to 4$, which has total length 4.

2.3 Resource allocation

A special case of resource allocation is the so called *knapsack problem.* Consider the following example.

Example: Suppose a 10-lb knapsack is to be filled with the items listed in the table below. Assume there are unlimited number of items of each type. To maximize the total benefit, how should the knapsack be filled?

	Weight (lb)	benefit
Item 1	4	11
Item 2	3	7
Item 3	5	12

Solution: This optimization can indeed be represented by an LP.

$$\text{Maxmize} \quad Z = 11x_1 + 7x_2 + 12x_3$$

such that

$$4x_1 + 3x_2 + 5x_3 \leq 10,$$

and $x_1, x_2, x_3 \geq 0$ are non-negative integers.

The LP is different from before is that the x_i is required to be an integer.

There are several ways to solve this problem using DP. In the following, we set

$$g(w) = \text{the maximum benifit that can be gained from a } w\text{-lb knapsack.}$$

The recursive equation to solve $g(w)$ is

$$g(w) = \max_j\{b_j + g(w - w_j)\};$$

here j denote the j-th item, with w_j its weight and b_j its benefit, and j must satisfy $w_j \leq w$. The intuition behind this equation is self-obvious enough.

By definition, clearly $g(0) = g(1) = g(2) = 0$. Furthermore,

$$\begin{aligned}
g(3) &= \max_j\{b_j + g(w - w_j)\} = 7 + g(0) = 7, \qquad (j^* = 2)\\
g(4) &= \max\{11 + g(0), 7 + g(1)\} = 11, \qquad (j^* = 1)\\
g(5) &= \max\{11 + g(1), 7 + g(2), 12 + g(0)\} = 12, \qquad (j^* = 3)\\
g(6) &= \max\{11 + g(2), 7 + g(3), 12 + g(1)\} = 14, \qquad (j^* = 2)\\
g(7) &= \max\{11 + g(3), 7 + g(4), 12 + g(2)\} = 18, \qquad (j^* = 1, 2)\\
g(8) &= \max\{11 + g(4), 7 + g(5), 12 + g(3)\} = 22, \qquad (j^* = 1)\\
g(9) &= \max\{11 + g(5), 7 + g(6), 12 + g(4)\} = 23, \qquad (j^* = 1, 3)\\
g(10) &= \max\{11 + g(6), 7 + g(7), 12 + g(5)\} = 25, \qquad (j^* = 1, 2)
\end{aligned}$$

Therefore the maximal benefit is 25, and the way to fill in the knapsack is that one item of type 1, and two items of type 2.

2.4 A Turnpike theorem

A drawback of the above dynamic programming is that when w is large, too many iterations are involved to solve for $g(w)$. We will introduce a *Turnpike theorem* to reduce the computational effort. Observe that the "best item" is the item with the largest value of b_j/w_j (benefit per unit weight). Assume that n type of items have been ordered, so that

$$\frac{b_1}{w_1} \geq \frac{b_2}{w_2} \geq \cdots \geq \frac{b_n}{w_n}.$$

The ordering could be different from the original labeling.

Turnpike Theorem: Consider a knapsack problem for which

$$\frac{b_1}{w_1} > \frac{b_2}{w_2}.$$

Set

$$w^* \doteq b_1 \bigg/ \left(\frac{b_1}{w_1} - \frac{b_2}{w_2}\right).$$

Suppose the knapsack can hold w pounds, with $w \geq w^*$. Then the optimal solution to the knapsack problem must use at least one item of type 1.

Proof: Consider the same knapsack problem except that item of type 1 are not going to be used. The corresponding LP becomes

$$\text{Maxmize} \quad Z = b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

such that

$$w_2x_2 + w_3x_3 + \cdots + w_nx_n \leq w,$$

and $x_2, \cdots, x_n \geq 0$ are non-negative integers.

But clearly, the value of this knapsack problem is at most $b_2 w/w_2$ (why?).

Now consider the following solution to the original knapsack problem: put as many items of type 1 as possible into the knapsack. This way, we can put in

$$\lfloor \frac{w}{w_1} \rfloor$$

items of type 1. Here $\lfloor \cdot \rfloor$ denotes the integer part of a real number. Therefore, the benefit from this solution is at least

$$b_1 \cdot \lfloor \frac{w}{w_1} \rfloor > b_1 \cdot \left(\frac{w}{w_1} - 1 \right).$$

In case $w \geq w^*$, we have

$$b_1 \cdot \lfloor \frac{w}{w_1} \rfloor > b_1 \cdot \left(\frac{w}{w_1} - 1 \right) = w \frac{b_1}{w_1} - b_1 \geq w \frac{b_2}{w_2}.$$

Therefore, it turns out that an solution with no items of type 1 is never optimal. □

Corollary: The optimal solution of the knapsack problem will indeed include at least $1 + \lfloor (w - w^*)/w_1 \rfloor$ items of type 1.

Example: Reconsider the preceding example but with $w = 4000$ lb.

Solution: We have

$$\frac{11}{4} > \frac{12}{5} > \frac{7}{3}.$$

Therefore

$$w^* = 11 \Big/ \left(\frac{11}{4} - \frac{12}{5} \right) = \frac{220}{7}.$$

Therefore, the optimal solution will contain at least

$$1 + \lfloor \frac{w - w^*}{w_1} \rfloor = 1 + \lfloor \frac{4000 - 220/7}{4} \rfloor = 993$$

items of type 1. Therefore, all we need to solve is a knapsack problem with $w = 4000 - 993 \cdot 4 = 28$-lb knapsack, which greatly reduces the computational effort.

Exercise: Complete the above example by solving the knapsack problem with $w = 28$-lb.

2.5 General resource allocation problem, formulation, DPE

A general resource allocation problem can be expressed as follows. Suppose we have w units of resource available, and N activities to which the resource can be allocated. If the activity n is implemented at the level x_n units (assumed to be a non-negative integer), then $g_n(x_n)$ units of the resources are used and a benefit $r_n(x_n)$ is obtained. The problem to maximize the total benefit subject to the limited resource availability may be written as

$$\text{Maxmize} \quad \sum_{n=1}^{N} r_n(x_n)$$

such that

$$\sum_{n=1}^{N} g_n(x_n) \le w$$

and $x_1, \cdots, x_N$ are non-negative integers.

In the knapsack problem, the total unit of resource is the weight the knapsack can hold; x_n stand for the number of type n items put in, and g_n and r_n are the weight and benefit of type n items, respectively.

One approach to formulate this problem as dynamic programming is as follows: Consider the functions

$$v_j(w) = \max_{\{x_n\}} \sum_{n=j}^{N} r_n(x_n), \quad \text{such that } \sum_{n=j}^{N} g_n(x_n) \le w,$$

with $v_{N+1}(w) = 0$. In this case $v_j(w)$ can be interpreted as the maximal benefit one can receive if there are w unit of resources available to be allocated to resources $j, \cdots, N$.

The DPE is

$$v_j(w) = \max_x \left[r_j(x) + v_{j+1}(w - g_j(x)) \right], \quad \forall\, w,$$

where x must be a non-negative integer satisfying $g_j(x) \le w$.

Exercise: Solve the previous knapsack example using the dynamic programming equation above.

Exercise: The number of crimes on each of a city's three police precincts depends on the number of patrol cars assigned to each precinct. Three patrol cars are available. Use dynamic programming to determine how many patrol cars should be assigned to each precinct so as to minimize the total number of crimes.

	No. of patrol cars			
	0	1	2	3
Precinct 1	14	10	7	4
Precinct 2	25	19	16	14
Precinct 3	20	14	11	8

Solution: Let x_i = number of partrol cars assigned to Precinct i. Then the optimization problem is

$$\text{Minimize} \quad \sum_{n=1}^{3} r_n(x_n)$$

such that

$$\sum_{n=1}^{3} x_n \le 5$$

and x_1, x_2, x_3 are non-negative integers.

Define

$$v_j(w) = \sum_{n=j}^{3} r_n(x_n)$$

such that

$$\sum_{n=j}^{3} x_n \le w,$$

and x_n are all non-negative integers.

- $v_4(w) \equiv 0$.
- $v_3(w) = \min_x [r_3(x) + v_4(w-x)] = \min_x r_3(x)$ over all non-negative integer x such that $x \le w$. We have

$$\begin{aligned} v_3(3) &= 8, & (x^* = 3) \\ v_3(2) &= 11, & (x^* = 2) \\ v_3(1) &= 14, & (x^* = 1) \\ v_3(0) &= 20, & (x^* = 0). \end{aligned}$$

- $v_2(w) = \min_x [r_2(x) + v_3(w-x)]$ over all non-negative integer x such that $x \le w$. We have

$$\begin{aligned} v_2(3) &= \min[14+20, 16+14, 19+11, 25+8] = 30, & (x^* = 1, 2) \\ v_2(2) &= \min[16+20, 19+14, 25+11] = 33, & (x^* = 1) \\ v_2(1) &= \min[19+20, 25+14] = 39, & (x^* = 1, 0) \\ v_2(0) &= \min[25+20] = 45, & (x^* = 0). \end{aligned}$$

- $v_1(w) = \min_x [r_1(x) + v_2(w-x)]$ over all non-negative integer x such that $x \le w$. We have

$$v_1(3) = \min[4+45, 7+39, 10+33, 14+30] = 43, \qquad (x^* = 1)$$

Therefore, the optimal solution is to assign one patrol car to each precinct, which will yield a minimal number of crimes 43. □

Exercise: What if there are only two patrol cars available?

3 Preliminary probabilistic dynamic programming

The deterministic DPE (for minimization of an additive cost criteria) can be loosely written as

$$V_j(\text{current state}) = \min_{\text{all feasible decisions}} \{\text{cost during the current stage} + V_{j+1}(\text{new state})\}.$$

The idea for probabilistic dynamic programming is the same, the only difference is that now the new state is a *random* outcome and the goal is to minimize the *average* cost. The DPE in this case is very similar, with "V_{j+1}(new state)" replaced by "Average of V_{j+1}(new state)".

Example: A gambler has \$2. She is allowed to play a game of chance two times, and her goal is to maximize her probability of ending up with at least \$4. If she gambles b dollars on a play of the game, with probability 0.4 she wins the game and increases her capital by b dollars; with probability 0.6 she loses the game and decrease her capital by b dollars. On any play of the game, the gambler cannot bet more than she has available. Design a betting strategy for the gambler.

Solution: For $j = 1, 2, 3$, define

$$\begin{aligned} V_j(x) \quad = \quad & \text{Maximal probability of having at least \$4 at the end of game two} \\ & \text{given that at the beginning of } j\text{-th play the gambler has capital } x \text{ dollars.} \end{aligned}$$

Also define $V_3(x) = 1$ if $x \geq 4$ and $V_3(x) = 0$ if $x < 4$. We are interested in $V_1(2)$.

One can solve the problem recursively. Indeed, we have the DPE

$$V_j(x) = \max_{b \in \{0,1,\cdots,x\}} \text{Average of } V_{j+1}(\text{new state}) = \max_{b \in \{0,1,\cdots,x\}} [0.4V_{j+1}(x+b) + 0.6V_{j+1}(x-b)]$$

for $j = 1, 2$ (why?). From V_3 we can recursively determine all v_j.

- $V_2(x) = \max_{b \in \{0,1,\cdots,x\}} [0.4V_3(x+b) + 0.6V_3(x-b)]$. Clearly, we have

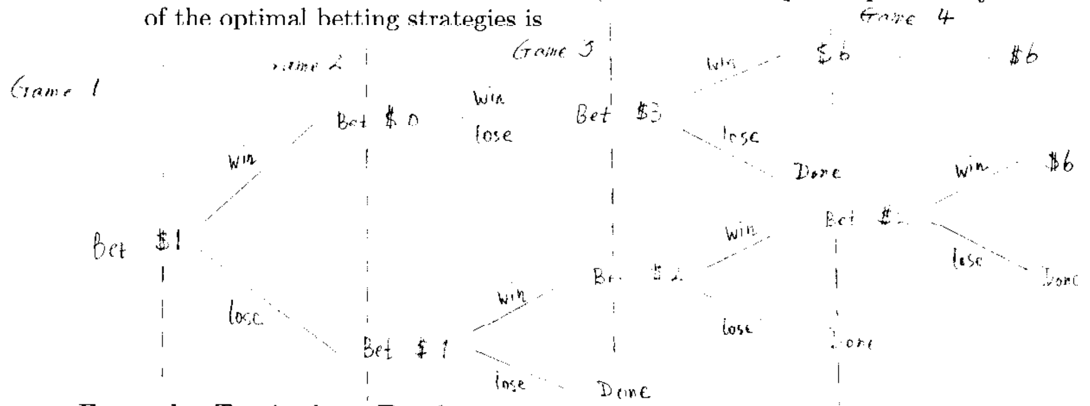
$$V_2(x) = \begin{cases} 1 \quad ; \quad \text{if } x = 4, 5, \cdots & b^* = \{0, 1, \cdots, x-4\} \\ 0.4 \quad ; \quad \text{if } x = 2, 3 & b^* = \{4-x, \cdots, x\} \\ 0 \quad ; \quad \text{if } x = 0, 1 & b^* = \{0, 1, \cdots, x\} \end{cases}$$

- $V_1(x) = \max_{b \in \{0,1,\cdots,x\}} [0.4V_2(x+b) + 0.6V_2(x-b)]$. In particular,

$$V_1(2) = \max_{b \in \{0,1,2\}} [0.4V_2(2+b) + 0.6V_2(2-b)] = \begin{cases} 0.4 \quad ; \quad b^* = 0 \\ 0.16 \quad ; \quad b = 1 \\ 0.4 \quad ; \quad b^* = 2 \end{cases}$$

Therefore, the maximal probability is 0.4 and one best policy is to bet 2 in the first play; if it is lost, so be it; if the gambler wins the first play, she should sit out in the second play. Another best strategy is to sit out in the first play, and bet 2 in the second play. □

Exercise: Suppose the gambler wants to maximize her probability of having at least \$6 at the end of the 4th play. How should she play? (*Solution:* The optimal probability is 0.1984, and one of the optimal betting strategies is



Example: Tennis player Tom has two types of serves: a hard serve (H) and a soft serve (S). The probability that Tom's hard serve will land in bounds is p_H, and the probability of his soft serve will land in bounds is p_S. If Tom's hard serve lands in bounds, there is a probability w_H that Tom will win the point. If Tom's soft serve lands in bounds, there is a probability w_S that Tom will win the point. We assume $p_H < p_S$ and $w_H > w_S$. Tom's goal is to maximize the probability of winning a point on which he serves. Remember that if both serves are out of bounds, Tom loses the point.

Solution: Define f_i, $i = 1, 2$ as the probability that Tom wins a point if he plays optimally and is about to take his i-th serve. To determine the optimal strategy, we will work backward. What is f_2? If Tom serves hard on his second serve, he has a probability $p_H w_H$ to win the point, while he has probability $p_S w_S$ to win the point if he serves soft. Therefore, we have

$$f_2 = \max\{p_H w_H,\ p_S w_S\}.$$

To determine f_1, observe the following equation:

$$f_1 = \max\{p_H w_H + (1 - p_H) f_2,\ p_S w_S + (1 - p_S) f_2\}$$

There are three possibilities,

1. $p_H w_H \geq p_S w_S$: In this case, $f_2 = p_H w_H$ and

   $$f_1 = \max\{p_H w_H + (1 - p_H) p_H w_H,\ p_S w_S + (1 - p_S) p_H w_H\} = p_H w_H + (1 - p_H) p_H w_H,$$

   since

   $$\begin{aligned}&[p_H w_H + (1 - p_H) p_H w_H] - [p_S w_S + (1 - p_S) p_H w_H]\\ &= p_H w_H (1 + p_S - p_H) - p_S w_S\\ &\geq p_H w_H - p_S w_S \geq 0.\end{aligned}$$

   Therefore, Tom should serve hard on both serves.

2. $p_S w_S(1+p_H-p_S) \le p_H w_H < p_S w_S$: In this case, $f_2 = p_S w_S$, and

$$f_1 = \max\{p_H w_H + (1-p_H)p_S w_S,\ p_S w_S + (1-p_S)p_S w_S\} = p_H w_H + (1-p_H)p_S w_S.$$

So Tom should serve hard on the first serve, and soft on the second.

3. $p_H w_H < p_S w_S(1+p_H-p_S)$: In this case, $f_2 = p_S w_S$, and

$$f_1 = \max\{p_H w_H + (1-p_H)p_S w_S,\ p_S w_S + (1-p_S)p_S w_S\} = p_S w_S + (1-p_S)p_S w_S,$$

and Tom should serve soft on both serves.

Example: We will return to the bass problem: Every year the owner of a lake must decide how many bass to capture and sell. During year n the price is p_n for the bass. If the lake contains b bass in the beginning of the year n, the cost of catching x bass is $c_n(x|b)$. But now the bass population grows by a random factor D, where $\text{Prob}(D=d) = q(d)$. Can you formulate a dynamic programming recursion if the owner wants to maximize the average net profit over the next five years?

Solution: Define for $n = 1, 2, \cdots, 5$,

$$v_n(b) = \text{the maximal average net profit during the years } n, n+1, \cdots, 5 \text{ if the lake contains } b \text{ bass at the beginning of year } n.$$

and $v_6(b) \equiv 0$. The recursive DPE is

$$v_n(b) = \max_{0 \le x \le b} \left[x p_n - c_n(x|b) + \sum_d q(d) v_{n+1}\big(d(b-x)\big) \right].$$