Dynamic Programming Summary

The University of Queensland

MATH3202

Matthew Rea
44304533
June 18, 2020

Summary

Notes

• Dynamic programming problems are constructed in the following way:
S, discrete state space
A, discrete action space
Stages, indexed by t, often time
Transition Function: $s_{t+1} = s^M(s_t, a_t)$
Contribution Function: $C_t(s_t, a_t)$
Value Function: $V_t(s_t) = \min_{a_t} \Big\{ C_t(s_t, a_t) + V_{t+1}(s_{t+1}) \Big\}$ (or max)

• There can also be problems with a transition matrix:
$\sum_{s_{t+1}} P_t(s_{t+1}|s_t, a_t) = 1$

The below are the Bellman equations:
$V_t(s_t) = \min_{a_t} \Big\{ C_t(s_t, a_t) + \sum_{s' \in s} P_t(s'|s_t, a_t) V_{t+1}(s') \Big\}$ (or max)
$V_t(s_t) = \min_{a_t} \Big\{ C_t(s_t, a_t) + \mathbb{E}\big[V_{t+1}(s_{t+1}|s_t, a_t) \big] \Big\}$ (or max)
$V_t(s_t) = \min_{a_t} \mathbb{E}\Big\{ C_t(s_t, a_t) + V_{t+1}(s_{t+1}) \Big\}$ (or max)

• Often when working with the value function, it is simpler to work backwards if you know what the final bound should be

• **Dijkstra's Algorithm**

1. Set $s_i = \infty,\ \forall i \in O,\ s_0 = \emptyset,\ done_i = False,\ \forall i \in N$.
2. Choose i' such that $s_{i'} = \min\{s_i | done_i = False\}$.
3. If $i' = D$ stop.
4. Set $done_{i'} = True$.
5. For all $j \in A_{i'}$, if $s_{i'} + d_{i'j} < s_j$, set $s_j = s_{i'} + d_{i'j}$ and $p_j = i'$.
6. Go to step (2).

Note that $s_i + e_{iD}$ is the admissable heuristic.

Inventory Problem

Problem

A company knows that the demand for its product during each of the next four months will be as follows:

Month	1	2	3	4
Demand	2	3	2	4

At the beginning of each month, the company must determine how many units should be produced during the current month. During a month in which units are produced, a setup cost of \$3 is incurred. In addition, there is a variable cost of \$1 for every unit produced. At the end of each month, a holding cost of 50 cents per unit on hand is incurred. Capacity limitations allow a maximum of 5 units to be produced during each month. The size of the company's warehouse restricts the ending inventory of each month to at most 4 units.

How many units should the company produce each month to satisfy demand and minimise total cost?

Solution

Stages
t, months $\{1, \ldots, 4\}$

State Space
s_t, amount in stock

Action Space
a_t, how much we make

Data
dem_t, demand at month t

Note that other data may exist here, such as the production cost for each unit, but for simplicity it will be left out.

Transition Function
$s_{t+1} = s_t + a_t - dem_t$

Contribution Function
Setup and Production Cost:

$$F(x) = \begin{cases} 0, & \text{if } x = 0 \\ 3 + x, & \text{otherwise} \end{cases}$$

Total Cost (incl. Holding Cost):

$$C_t(s_t, a_t) = \underbrace{F(a_t)}_{\text{setup/production}} + \underbrace{0.5(s_t + a_t - dem_t)}_{\text{holding}}$$

Value Function

We know what the contribution function will be in the final stage (month = 4). If we have x in stock, then we must produce (or "action") the demand of stage 4 minus x.

$$V_4(x) = C_4(x,\ dem_t - x)$$

We develop the value function, which is always a function of the transition and contribution functions. We want to minimise the total cost, where the amount we produce is between 0 and 5 always and so that the ending inventory of each month is at most 4 units. Also, we must produce the minimum required to satisfy the month's demand, hence the constraints on a_t in the below value function.

$$V_t(s_t) = \min_{\max\{0,\ dem_t - s_t\} \le a_t \le \min\{5,\ 4+dem_t - s_t\}} \{ \underbrace{C_t(s_t, a_t)}_{\text{contribution}} + V_{t+1} \quad \underbrace{s_{t+1}}_{\text{transition}} \ \}$$

Code

```
#Demand
dem = [2,3,2,4]

#Function for cost of setup and production
def F(x):
    if x == 0:
        return 0
    else:
        return 3+x

#Contribution function
def C(t,s,a):
    return F(a) + 0.5*(s+a-dem[t])

#Value function
def V(t,s):
    if t == 4:
        return (0,0)
    return min(
        (C(t,s,a) + V(t+1, s+a-dem[t])[0],a)
        for a in range(max(0, dem[t] - s), min(5, 4+dem[t]-s)))
```

Nonlinear Objective

Problem

Maximise

$$(x_1 + 5)(x_2 + 1)(x_3 + 2)$$

subject to

$$3x_1 + 2x_2 + x_3 \leq 6$$

with $x_1, x_2, x_3 \in \mathbb{Z}^+ \cup \{0\}$.

Solution

Stages

t, Variables

State Space

s_t, amount of total available

Action Space

No actions for this problem.

Data

$d = \{5, 1, 2\}$
$w = \{3, 2, 1\}$

Value Function

$$V_t(s_t) = \max\Big[(x_t + d_t)(x_{t+1} + d_{t+1}) \dots (x_3 + d_3)\Big]$$

Which is interpreted as:

$$\begin{aligned} V_3(s_3) &= \max(x_3 + 2), \text{ subject to } x_3 \leq s_3 \\ V_2 &= \max(x_2 + 1)(x_3 + 2) \\ V_1 &= \max(x_1 + 5)(x_2 + 1)(x_3 + 2) \end{aligned}$$

So we get

$$V_t(s_t) = \max_{0 \leq x_t \leq \left\lfloor \frac{s_t}{w_t} \right\rfloor} \Big\{(x_t + d_t) \times V_{t+1}(s_t - w_t x_t)\Big\}$$

and

$$V_3(s_3) = s_3 + 2$$

We want $V_1(6)$.

Minimal Studying

Problem

In order to graduate from State University, Angie Warner needs to pass at least one of the three subjects she is taking this semester. She is now enrolled in Algebra, Calculus, and Statistics. Angie's busy schedule of extra-curricular activities allows her to spend only 4 hours per week on studying. Angie's probability of passing each course depends on the number of hours she spends studying for the course, as follows:

	Probability of Passing		
Study per week (hrs)	**Algebra**	**Calculus**	**Statistics**
0	0.20	0.25	0.10
1	0.30	0.30	0.30
2	0.35	0.33	0.40
3	0.38	0.35	0.45
4	0.40	0.38	0.50

How many hours per week Angie should spend studying each subject?

Solution

Stages
j, subjects $\{0, 1, 2\}$

State Space
s, hours remaining

Action Space
a_j, hours of study for subject j

Data
P_{aj}, probability of passing subject j with a hours of studying per week

Value Function
$V_j(s_j)$ is the minimum probability of failing subjects $j, \ldots, 2$ with s_j hours available.

We want $V_0(4)$. We know that $V_2(s_2) = 1 - P_{s_2,2}$. Thus, we get:

$$V_j(s_j) = \min_{0 \le a_j \le s_j} \left\{ (1 - P_{a_j,j}) \times V_{j+1}(s_j - a_j) \right\}$$

where a_j is the hours for subject j and $(s_j - a_j)$ is s_{j+1}.

Note that even though there are probabilities involved, this is still a deterministic problem.

Code

```
# MinFail(j,s) is the minimum probability of failing subjects
# j,..., 2 with s hours of study available.
# We want 1 - MinFail(0,4)

P = [[0.20, 0.30, 0.35, 0.38, 0.40],
     [0.25, 0.30, 0.33, 0.35, 0.38],
     [0.10, 0.30, 0.40, 0.45, 0.50]]

def MinFail(j,s):
    if j == 2:
        return (1 - P[j][s], s, 0)
    else:
        return min(((1 - P[j][a]) * MinFail(j+1, s-a)[0], a, s-a)
                   for a in range(0,s+1))

#Print results
def MinFailSolution():
    s = 4
    for j in [0,1,2]:
        v = MinFail(j,s)
        print(v[1], "hours for subject", j)
        s = v[2]
```

This gave the optimal solution of passing at least one subject with 0.711 probability if they studied 1 hour of algebra, 0 hours of calculus, and 3 hours of statistics.

Chess Strategy

Problem

Vladimir is playing Keith in a two-game chess match. Winning a game scores one match point and drawing a game scores a half match point. After the two games are played, the player with more match points is declared the champion. If the two players are tied after two games, they continue playing until somebody wins a game (the winner of that game will be the champion).

During each game, Vladimir can play one of two ways: boldly or conservatively. If he plays boldly, he has a 45% chance of winning the game and a 55% chance of losing the game. If he plays conservatively, he has a 90% chance of drawing the game and a 10% chance of losing the game.

What strategy should Vladimir follow to maximise his probability of winning the match?

Solution

Stages
t, games

State Space
s_t, points at the state of game t

Action Space
a_t, whether to play boldly or conservatively for game t

Data
Data may be present for this problem, but has not been explicitly listed, for simplicity.

Value Function
$V_t(s_t)$ is the maximised probability of winning the match if we start game t with s_t points.
We want $V_1(0)$ and we know:

- $V_3(0) = 0 = V_3\left(\frac{1}{2}\right)$
- $V_3(2) = 1 = V_3\left(\frac{3}{2}\right)$
- $V_3(1) = 0.45$, if you play boldly

Thus, the value function is dependent on whether Vladimir plays boldly or conservatively.

$$V_t(s_t) = \max \begin{cases} 0.45V_{t+1}(s_t + 1) + 0.55V_{t+1}(s_t), & \text{playing boldly} \\ 0.90V_{t+1}(s_t + 1/2) + 0.10V_{t+1}(s_t), & \text{playing conservatively} \end{cases}$$

Note that there does not actually exist V_3 as beginning the third game on 0 or 0.5 points means that you have already lost. It is simply shown to display the probability of winning if you end the second game with these points. A similar case is observed for having 2 points at the end of game 2, as you would have already won.

Also, note that Vladimir has a probability of 0 of winning the game if he plays conservatively,

as he can only either draw or lose the game. Therefore, he must play boldly to have a chance of winning.

Code

```
# Chess(t,s) is the maximum probability of winning the match
# if we start game t with s points

def Chess(t,s):
    if t == 3:
        if s < 1:
            return(0, 'Lost')
        elif s > 1:
            return(1, 'Won')
        else:
            return(0.45, 'Bold')
    else:
        bold = 0.45*Chess(t+1, s+1)[0] + 0.55*Chess(t+1, s+0)[0]
        conservative = 0.9*Chess(t+1, s+0.5)[0] + 0.1*Chess(t+1, s+0)[0]
        return max((bold, 'Bold)'), (conservative, 'Conservative'))

# Play boldly on first game
# If you win, play conservatively on the second game
# If you lose, play boldly on the second game
# If tied on second game, play boldly
```

Knapsack Problem

Problem

We have a container of size 20 units and want to pack it with the following valuable items:

Item j	**Size** v_j	**Value** t_j
1	7	25
2	4	12
3	3	8

How many of each item should we pack in order to maximise the total value?

Solution

Stages
j, items $\{1, 2, 3\}$

State Space
s_j, size remaining in the knapsack

Action Space
a_j, number of item j to pack in the knapsack

Data
$maxsize$, the maximum size of the knapsack
v_j, the size of item j
t_j, the value of item j

Value Function
$V_j(s_j)$ is the maximum value of packing s_j size-worth of items $j, \ldots, 3$.
We want $V_0(maxsize)$.
We know $V_0(z) = 0,\ z \in \{0, 1, 2\}$.
We get

$$V_j(s_j) = \max_{0 \leq a_j \leq s_j/v_j} \Big\{ t_j + V_{j+1}(s_j - v_j) \Big\}$$

Code

```
#Stages
J = range(3)

#Data
maxsize = 20
v = [7, 4, 3] #sizes
t = [25, 12, 8] #values

#Use memoization to improve runtime
_V = {}

#Value function (V(s) is the max value from packing a
#                knapsack of size s)
def V(s):
    if s < 3:
        return (0, 'Full')
    else:
        if not s in _V:
            _V[s] = max((t[j] + V(s - v[j])[0],
                         'Item ' + str(j+1), s - v[j])
                        for j in J if v[j] <= s)
        return _V[s]

#Print results
print("The maximum value from a knpasack of size", maxsize,
      "is", V(maxsize)[0])
```

This gave a maximum value of 66.

Fibonacci Sequence

Problem

Define a function in Python to calculate the n^{th} number in the Fibonacci sequence.

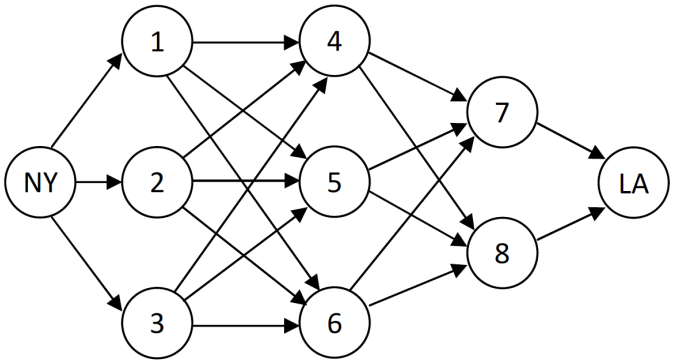
Code

```
#Original Function
def Fib(n):
    if n <= 2:
        return 1
    else:
        return Fib(n-1) + Fib(n-2)


#Memoization to store values and reduce runtime
_Fib = {}

def Fib(n):
    if not n in _Fib:
        if n <= 2:
            _Fib[n] = 1
        else:
            _Fib[n] = Fib(n-1) + Fib(n-2)
    return _Fib[n]
```

A Transcontinental Drive

Problem

An up and coming actor decides to drive from New York to Los Angeles to seek fame and fortune. Because he is more or less penniless, he needs to stay with friends along the way. Joe only has a limited number of cities in which he has friends and he categorises each of these cities as one, two, or three days driving from New York. With the help of a friend who does some operations research, he comes up with the following graph and distance table:



NY to: (1, 550), (2, 900), (3, 770)
1 to: (4, 680), (5, 790), (6, 105)
2 to: (4, 580), (5, 760), (6, 700)
3 to: (4, 510), (5, 700), (6, 830)
4 to: (7, 610), (8, 790)
5 to: (7, 540), (8, 940)
6 to: (7, 790), (8, 270)
7 to: (LA, 1030)
8 to: (LA, 1390)

What is the shortest path from NY to LA?

Solution

We use the information in the right table to work backwards and determine the shortest distances, or lengths, of each arc on the network diagram.

Node From	Node To	Shortest Distance	Min. Route
7	LA	1030	7 to LA
8	LA	1390	8 to LA
6	7 or 8	$\min\{790+1030,\ 270+1390\} = 1660$	6 to 7
5	7 or 8	$\min\{540+1030,\ 940+1390\} = 1570$	5 to 7
4	7 or 8	$\min\{610+1030,\ 790+1390\} = 1640$	4 to 7

This is a network problem that can be solved using dynamic programming. We use the following information and implement Dijkstra's algorithm.

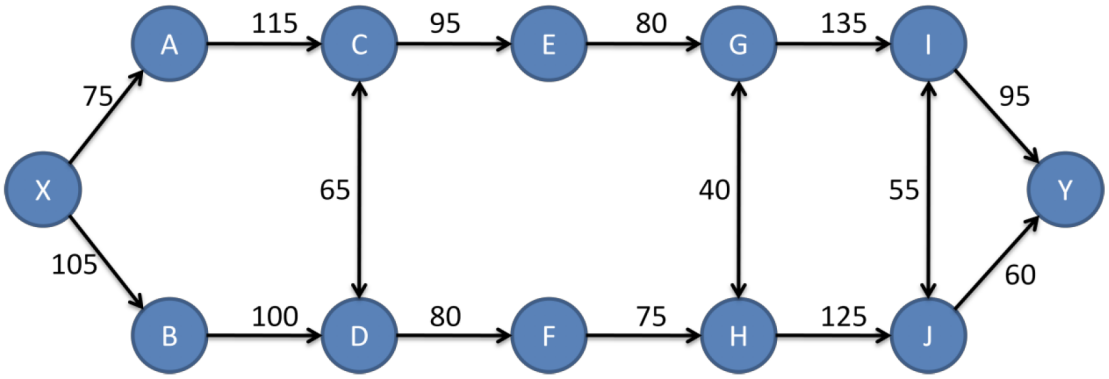
Let N be the set of nodes. For each node we have the connected nodes in a set A_i. We also have $d_{ij},\ \forall j \in A_i$. Let origin O, to the destination D

We then implement *Dijkstra's algorithm*.

Optimising the Family Drive

Problem

A family is setting out to drive between two cities, X and Y. These cities are joined by two main highways, with some options to cross over between the highways. The highways can be drawn as follows, with distances marked in kilometres.



Along the way the family will need to stop for fuel and food. The costs of these vary as follows:

City	Cost of family meal ($)	Fuel cost (cents/L)
A	80	152
B	43	152
C	40	186
D	40	153
E	74	123
F	72	143
G	78	186
H	45	124
I	73	191
J	55	126

Due to holiday traffic, the car travels at an average speed of 60 km/hr. The family has just fed before setting off and can go at most 4 hours between meals (end of one meal to start of the next). They will eat with friends on arrival at Y. The car uses 10 litres per 100 km and can safely go at most 400 km on a tank. Whenever the family stops to refuel it completely fills the tank. The tank is full on setting off. When the car arrives at the destination it will need to be refuelled at a cost of $1.20 per litre.

(a) Write down a general-purpose formulation for calculating the cheapest way to get from X to Y.
(b) Perhaps the family wants to have the fewest stops possible (where a stop for both food and petrol counts as only one stop). Modify your formulation to calculate the cheapest way to travel, amongst journeys that also have the smallest number of stops.

Solution

Data

R, set of resources
N, set of nodes
A_i, set of nodes connected to i
d_{ij}, distance
v_{ij}, vector of resource usage

Labels

L_i, set of labels for node i
$(i, s, V, done, pred)$, node, shortest path, vector of remaining resource, whether label processed or not, predecessor.

We then implement *Dijkstra's algorithm*.

1. Set $L_i = \emptyset,\ \forall i \neq 0,\ L_0 = \{(0, \emptyset, V_0, false)\}$.
2. Choose label l such that $s_l = \min_{l \in L}\{s_l, done_l = false\}$.
3. If $i_l = D$, stop.
4. Set $done_l = True$.
5. For all $j \in A_{i_l}$, if $V_l - V_{i_l j} \geq 0$, add label $(j, s_j + d_{i_l j}, V_l - V_{i_l j}, false)$.
6. Go to step (2).

Label l_1 dominates label l_2 if $i_{l_1} = i_{l_2},\ s_{l_1} \leq s_{l_2},\ V_{l_1} \geq V_{l_2}$

Code

```
import math

Arcs = [
        ('X', 'A', 75),
        ('X', 'B', 105),
        ('A', 'C', 115),
        ('B', 'D', 100),
        ('C', 'D', 65),
        ('C', 'E', 95),
        ('D', 'C', 65),
        ('D', 'F', 80),
        ('E', 'G', 80),
        ('F', 'H', 75),
        ('G', 'I', 135),
        ('G', 'H', 40),
        ('H', 'G', 40),
        ('H', 'J', 125),
        ('I', 'J', 55),
        ('I', 'Y', 95),
        ('J', 'I', 55),
        ('J', 'Y', 60)
]

Costs = {
    'A': (80, 152),
    'B': (43, 152),
    'C': (40, 186),
    'D': (40, 153),
    'E': (74, 123),
    'F': (72, 143),
    'G': (78, 186),
    'H': (45, 124),
    'I': (73, 191),
    'J': (55, 126),
    'Y': (0, 120)
}

FullFood = 240
FullFuel = 400
Orig = 'X'
Dest = 'Y'

UndoneLabels = set([(0, 'X', FullFood, FullFuel, None)])
AllLabels = set(UndoneLabels)

def AddLabel(cost, where, food, fuel, pred):
    global UndoneLabels
    for l in AllLabels:
        if l[1] == where and l[0] <= cost and l[2] >= food \
        and l[3] >= fuel:
            return
    #Check for and remove dominated labels in UndoneLabels
```

```
53      Dominated = set([l for l in UndoneLabels if
54                       l[1] == where and l[0] >= cost and
55                       l[2] <= food and l[3] <= fuel])
56      UndoneLabels -= Dominated
57
58      UndoneLabels.add((cost, where, food, fuel, pred))
59      AllLabels.add((cost, where, food, fuel, pred))
60
61  while True:
62      L = min(l for l in UndoneLabels)
63      if L[1] == 'Y':
64          break
65      UndoneLabels.remove(L)
66
67      #Process all arcs that leave the node at L
68      #Check they have enough food & fuel to make it
69      for a in Arcs:
70          if a[0] == L[1] and L[2] >= a[2] and L[3] >= a[2]:
71              #Add 4 possible arcs
72              if a[1] != 'Y':
73                  #Do nothing
74                  AddLabel(L[0], a[1], L[2] - a[2], L[3] - a[2], L)
75
76                  #Food
77                  AddLabel(L[0] + Costs[a[1]][0], a[1], FullFood,
78                           L[3] - a[2], L)
79
80              #Fuel
81              FuelCost = Costs[a[1]][1]/100 * (FullFuel - L[3] + a[2])/10
82              AddLabel(L[0] + FuelCost, a[1], L[2] - a[2], FullFuel, L)
83
84              #Food + Fuel
85              AddLabel(L[0] + Costs[a[1]][0] + FuelCost, a[1], FullFood,
86                       FullFuel, L)
```

Travelling Artist Problem

Problem

An artist has the possibility to visit a number of different exhibitions in different cities over the next four days. Based on her experience, she has estimated the probabilities of sales at each exhibition, given that she attends the exhibition for a day. Each of her paintings sells for $500. She also knows how much it costs to travel between exhibitions and from her home to each of the exhibitions. She can only attend each exhibition once. The data for sales and travel costs are given below.

Exhibition	Probability of Paintings Sold		
	0	1	2
A	0.3	0.4	0.3
B	0.2	0.5	0.3
C	0.2	0.7	0.1
D	0.3	0.5	0.2
E	0.3	0.6	0.1
F	0.4	0.3	0.3
G	0.0	0.3	0.7
H	0.1	0.1	0.8

	A	B	C	D	E	F	G	H
Home	143	108	118	121	88	121	57	92
A		35	63	108	228	182	73	162
B			45	86	193	165	42	129
C				46	190	203	73	105
D					172	224	98	71
E						174	160	108
F							129	212
G								117

(a) The artist wants to maximise her expected profit from a tour of four exhibitions. What path should she take?

(b) Suppose now that the artist only has 5 paintings to sell. What is her optimal strategy for a tour of at most four exhibitions assuming that she will return home once all paintings are sold?

Solution (a)

Stages
t, days

State Space
s, cities we have already visited
i, where we are

Action Space
j, next city we go to

Data
c_{ij}, cost from i to j
$sales_{ik}$, probability of selling k paintings in city i
$ESales_i = \sum_{k\in\{0,1,2\}} k \times sales_{ik}$, expected sales

Value Function
$V_t(s,i)$ is the maximum expected profit from the remaining tour if we have been gone for t days and are in city i having visited cities s.

We know $V_4(s,i) = -c_{i,Home}$.

We want $V_0(\emptyset, Home)$.

We get:

$$V_t(s,i) = \max_{j\in Cities\setminus s} \Big\{500 \times ESales_j - c_{ij} + V_{t+1}\big(s\cup\{j\},\ j\big)\Big\}$$

Code (a)

```
c = [[0, 143, 108, 118, 121, 88, 121, 57, 92],     # Home
     [143, 0, 35, 63, 108, 228, 182, 73, 162],     # A
     [108, 35, 0, 45, 86, 193, 165, 42, 129],      # B
     [118, 63, 45, 0, 46, 190, 203, 73, 105],      # C
     [121, 108, 86, 46, 0, 172, 224, 98, 71],      # D
     [88, 228, 193, 190, 172, 0, 174, 160, 108],   # E
     [121, 182, 165, 203, 224, 174, 0, 129, 212], # F
     [57, 73, 42, 73, 98, 160, 129, 0, 117],       # G
     [92, 162, 129, 105, 71, 108, 212, 117, 0]]    # H

sales = [[0.0, 0.0, 0.0], # Home
         [0.3, 0.4, 0.3], # A
         [0.2, 0.5, 0.3], # B
         [0.2, 0.7, 0.1], # C
         [0.3, 0.5, 0.2], # D
         [0.3, 0.6, 0.1], # E
         [0.4, 0.3, 0.3], # F
         [0.0, 0.3, 0.7], # G
         [0.1, 0.1, 0.8]] # H

Cities = range(len(c))

ESales = [sum(k * sales[i][k] for k in [0,1,2]) for i in Cities]

Home = 0

def V(s,i):
    if len(s) == 4:
        return (-c[i][Home], Home)
    return max((500 * ESales[j] - c[i][j] + V(s+[j],j)[0], j)
               for j in Cities if j not in s)

#V([], Home) = 2364 following Home -> 8 -> 2 -> 1 -> 7 -> Home
```

Solution (b)

Stages
t, days

State Space
s, cities we have already visited
i, where we are

Action Space
j, next city we go to

Data
c_{ij}, cost from i to j
$sales_{ik}$, probability of selling k paintings in city i
$ESales_i = \sum_{k\in\{0,1,2\}} k \times sales_{ik}$, expected sales

Value Function
$V_t(s,i,p)$ is the maximum expected profit from the remaining tour if we have been gone for t days and are in city i having visited cities s with p paintings remaining.

We know $V_4(s,i,p) = -c_{i,Home}$ and $V_4(s,i,0) = -c_{i,Home}$.

We want $V_0(\emptyset, Home)$.

We get:

$$V_t(s,i) = \max_{j\in Cities\setminus s} \left\{ -c_{ij} + \sum_{k\in\{0,1,2\}} sales_{jk} \times \Big(500 \times \min\{k,p\} - c_{ij} + V_{t+1}\Big(s\cup\{j\},\ j,\ p-\min\{k,p\}\Big)\Big) \right\}$$

Code (b)

```
c = [[0, 143, 108, 118, 121, 88, 121, 57, 92],     # Home
     [143, 0, 35, 63, 108, 228, 182, 73, 162],     # A
     [108, 35, 0, 45, 86, 193, 165, 42, 129],      # B
     [118, 63, 45, 0, 46, 190, 203, 73, 105],      # C
     [121, 108, 86, 46, 0, 172, 224, 98, 71],      # D
     [88, 228, 193, 190, 172, 0, 174, 160, 108],   # E
     [121, 182, 165, 203, 224, 174, 0, 129, 212], # F
     [57, 73, 42, 73, 98, 160, 129, 0, 117],       # G
     [92, 162, 129, 105, 71, 108, 212, 117, 0]]    # H

sales = [[0.0, 0.0, 0.0], # Home
         [0.3, 0.4, 0.3], # A
         [0.2, 0.5, 0.3], # B
         [0.2, 0.7, 0.1], # C
         [0.3, 0.5, 0.2], # D
         [0.3, 0.6, 0.1], # E
         [0.4, 0.3, 0.3], # F
         [0.0, 0.3, 0.7], # G
         [0.1, 0.1, 0.8]] # H

Cities = range(len(c))

ESales = [sum(k * sales[i][k] for k in [0,1,2]) for i in Cities]

Home = 0

_V = {}
def V(s,i,p):
    if (len(s) == 4) or (p == 0):
        return (-c[i][Home], Home)

    if (tuple(s),i,p) not in _V:
        _V[tuple(s),i,p] = max((-c[i][j] +
                           sum(sales[j][k] * (500 * min(k,p) +
                               V(s+[j], j, p-min(k,p))[0])
                                     for k in [0,1,2]), j)
                for j in Cities if j not in s)

    return _V[tuple(s),i,p]

# V([], Home, 5) = $2033.63
# Home -> G
# If 3 or 4 paintings -> B
# If 4 paintings after B -> A
# If 2,3,4 paintings after A -> H

# If 1,2,3 paintings after B -> C
# After C -> H

# After H -> Home
```

Optimal Stopping

Problem

Jenny drives along a straight road towards a particular shop, looking for a vacant parking space. Vacancies occur at random, on average once in every 10 places. Once past the shop, Jenny will take the next available space. But at what point in approaching the shop should she accept a vacant space?

Solution

There is a probability of 0.1 of finding an empty parking spot. We let V_j be the expected walking distance in cars when at spot j.

To find V_0, we note that it is denoted by the following infinite sum:

$$V_0 = 0.1 \sum_{i=1}^{\infty} 0.9^j j = 9$$

We get that $V_j = 0.1 \times \min\{j, V_{j-1}) + 0.9 V_{j-1}$.

We can solve this from the beginning like so:

$$\begin{aligned} V_1 &= 0.1 \times \min\{1, 9\} + 0.9 \times 9 \\ &= 0.1 + 8.1 \\ &= 8.2 \\ V_2 &= 0.1 \times \min\{2, 8.2\} + 0.9 \times 8.2 \\ &= 0.2 + 7.38 \\ &= 7.58 \\ V_3 &= 0.1 \times \min\{3, 7.58\} + 0.9 \times 7.58 \\ &\vdots \end{aligned}$$

Web Building

Problem

Female orb-weaving spiders need to gain energy to grow in weight from 35 mg, at their sexual maturity, to 80 mg, where they can lay a first batch of eggs. These spiders are sit-and-wait predators, building a web to catch prey and gain energy. They replace their web every day and so can change the size of web they use. Larger webs are more likely to catch prey but require more energy to build and more time in the open, making them more vulnerable to their own predators.

Suppose the spiders have a choice of building webs where the total length of the sticky spiral is 4 m, 8 m or 12 m. Building a web of size w requires an energy cost of

$$a_w(s) = -0.125w + 0.005ws$$

where s is the current weight of the spider in mg. Each day a spider also has a basal metabolic expenditure, a_m, of 0.4 mg.

The table below shows the daily probability, λ_w, of catching a prey and the daily probability, β_w, that the spider will be eaten for the different web sizes.

Web Size, w	**4**	**8**	**12**
λ_w	0.66	0.77	0.82
β_w	0.01	0.02	0.03

The energy value of a prey item is 6 mg. We assume that at most one prey can be caught each day. If the weight of a spider drops below 25 mg then it dies from starvation.

At the end of each day, let s be the weight of the spider in mg with $p = s - \lfloor s \rfloor$. Then the weight of the spider at the start of the next day will be $\lfloor s \rfloor + 1$ with probability p and $\lfloor s \rfloor$ with probability $1 - p$ (this keeps the possible states more manageable).

Suppose a spider starts a day with weight s. What is the maximum probability that she will reach the egg-laying weight of 80 mg within the next 10 days? What strategy should she pursue to achieve this?

Solution

Stages
t, days $\{0, \ldots, 10\}$

State Space
., weight

Action Space

Data

Value Function
$V_0(s)$ is the probability of reaching 80 mg by day 10

$$V_{10}(s) = \begin{cases} 1, & \text{if } s \geq 80 \\ 0, & \text{otherwise} \end{cases}$$

$$V_t(s) = \begin{cases} 1, & \text{if } s \geq 80 \\ 0, & \text{if } s < 25 \end{cases}$$

We get that the value function is

$$\begin{aligned} \max_{w \in \{4,8,12\}} & \lambda_w(1-\beta_w) \times V_{t+1}\Big(s + 6 - \alpha_w(s)\Big) \\ & + \beta_w \times 0 \\ & + (1-\lambda_w)(1-\beta_w) \times V_{t+1}\Big(s - \alpha_w(s)\Big) \end{aligned}$$

If we let $p = s - \lfloor s \rfloor$, then we get:

$$V_t'(s) = pV_t(\lfloor s \rfloor + 1) + (1-p)V_t(\lfloor s \rfloor)$$

Code

```
lam = {4: 0.66, 8: 0.77, 12: 0.82}
beta = {4: 0.01, 8: 0.02, 12: 0.03}

def alpha(s,w):
    return -0.125*w + 0.005*w*s + 0.4

def Vd(t,s):
    p = s - int(s)
    return p * V(t, int(s)+1)[0] + (1-p) * V(t, int(s))[0]

_V = {}

def V(t,s):
    if t == 10:
        return (s >= 80, 'End')
    if s < 25:
        return (0, 'Starve')
    if s >= 80:
        return (1, 'Eggs')
    if (t,s) not in _V:
        _V[t,s] = max((lam[w] * (1-beta[w]) * Vd(t+1, s+6-alpha(s,w)) +
                       (1-lam[w]) * (1-beta[w]) * Vd(t+1, s-alpha(s,w)), w)
                       for w in lam)
    return _V[t,s]
```

Strawberries

Problem

The owner of a chain of three grocery stores has purchased five crates of fresh strawberries. The estimated potential sales of the strawberries before spoilage differs among the three stores. Therefore, the owner wishes to know how she should allocate the five crates to the three stores to maximize expected profit.

The owner does not wish to split crates between stores but she is willing to distribute zero crates to any of her stores. The following table gives the estimated expected profit at each store when it is allocated various numbers of crates:

Crates	*Store 1*	*Store 2*	*Store 3*
0	\$0	\$0	\$0
1	\$3	\$5	\$4
2	\$7	\$10	\$6
3	\$9	\$11	\$11
4	\$12	\$11	\$12
5	\$13	\$11	\$12

How many of the five crates should be assigned to each of the three stores to maximize the total expected profit?

Solution

Stages
j, stores $\{0, 1, 2\}$

State Space
s_j, crates available

Action Space
a_j, number of crates to allocate to store j

Data
P_{ja}, profit from assigning a crates to store j

Value Function
$V_j(s_j)$ is the maximum profit from allocating s_j crates to stores $j, \ldots, 2$
We want $V_0(5)$.
We have $V_2(s_2) = P_{2,s_2}$
We get

$$V_j(s_j) = \max_{0 \le a_j \le s_j} \left\{ P_{j,a_j} + V_{j+1}(s_j - a_j) \right\}$$

Code

```
#Data
P = [[0,3,7,9,12,13],
     [0,5,10,11,11,11],
     [0,4,6,11,12,12]]

#Recursive Function for Profit
def strawberries(j,s):
    if j == 2:
        return P[j][s] #base case
    else:
        return max(P[j][a] + strawberries(j+1, s-a)
                   for a in range(0,s+1))

#Find optimal profit
strawberries(0, 5)

#This tells us the optimal solution, but not how we get there

#Recursive Function for Profit and Actions
def strawberries(j,s):
    if j == 2:
        return (P[j][s], s) #base case
    else:
        return max((P[j][a] + strawberries(j+1, s-a)[0], a, s-a)
                   for a in range(0,s+1))

#We start at store 0 with 5 crates
strawberries(0,5)
#Returns (21,2,3) which means we can get $21 expected profit
#from allocating this

#We allocate 2 crates (to 0) and have 3 left over

strawberries(1,3)
#Returns (14,2,1) which means we can get $14 expected profit
#from allocating 2 crates to store 1

#We allocate 2 crates (to 1) and have 1 left over

strawberries(2,1)
#Returns (4,1,0) which means we can get $4 expected profit
#from allocating 1 crate to store 2

#We allocate 1 crate (to 2) and have 0 left over
```

Democracy

Problem

A State consists of three cities with populations 1.2 million people, 1.4 million people, and 400,000 people. The House of Representatives consists of three representatives. Given proportional representation, City 1 should have $d_1 = 3(1.2/3) = 1.2$ representatives; City 2 should have $d_2 = 1.4$ representatives; and City 3 should have $d_3 = 0.4$ representatives. Since each city must receive an integral number of representatives, this is impossible.

The State has therefore decided to allocate x_i representatives to city i, where the allocation should minimise the maximum discrepancy between the desired and actual number of representatives received by a city. How many representatives should each city receive?

Solution

Stages
j, cities $\{0, 1, 2\}$

State Space
s_j, representatives left to allocate

Action Space
a_j, number to allocate to city j

Data
d_j, desirable reps for city j

Value Function
$V_j(s_j)$ is the minimum of the maximum discrepancy between the desired and allowed with s_j reps for cities $j, \dots, 2$.
We want $V_0(3)$.
We have $V_2(s_2) = |d_2 - s_2|$ or $\min_{0 \le a \le s_2} |d_2 - a|$. We get

$$V_j(s_j) = \min_{0 \le a_j \le s_j} \Big\{ \max \big\{ |d_j - a_j| \big\} V_{j+1}(s_j - a_j) \Big\}$$

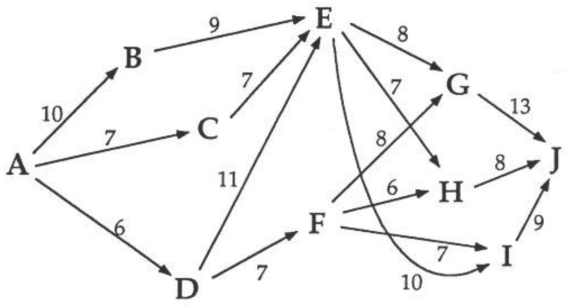
Code

```
#Data
d = [1.2, 1.4, 0.4]

#Recursive function
def cities(j,s):
    if j == 2:
        return abs(d[j] - s) #base case
    else:
        return min( max(abs(d[j] - a), cities(j+1,s-a))
                    for a in range(0,s+1))

cities(0,3)

#This returns the value but doesn't include the actions

#Recursive function with actions
def cities(j,s):
    if j == 2:
        return (abs(d[j] - s), s) #base case
    else:
        return min( (max(abs(d[j] - a), cities(j+1,s-a)[0]), a, s-a)
                    for a in range(0,s+1))
```

Altitude Sickness

Problem

Joe Cougar needs to drive from A to J. Due to a medical condition he wants to avoid high altitude. The following figure gives the maximum altitude of roads between intermediate cities on the way to J.



What route minimises the maximum altitude of Joe's journey?

Solution

Stages

i, cities $\{A, B, \ldots, J\}$

State Space

There is no state space for this problem

Action Space

a, the next city to visit

Data

h_{ij}, the maximum altitude from i to j, where $j \in D(i)$ where $D(i)$ are the destinations

Value Function

V_i is the minimum of the maximum altitude of path from i to J
We want V_A.
We know $V_J = 0$. We get

$$V_i = \min_{j \in D(i)} \Big\{ \max\{h_{ij}, V_j\} \Big\}$$

Code

```
#Data
edges = {
    ('A','B'): 10,
    ('A','C'): 7,
    ('A','D'): 6,
    ('B','E'): 9,
    ('C','E'): 7,
    ('D','E'): 11,
    ('D','F'): 7,
    ('E','G'): 8,
    ('E','H'): 7,
    ('E','I'): 10,
    ('F','G'): 8,
    ('F','H'): 6,
    ('F','I'): 7,
    ('G','J'): 13,
    ('H','J'): 8,
    ('I','J'): 9
}

#Recursive function
def joe(i):
    if i == 'J':
        return 0
    else:
        return min( max(edges[road], joe(road[1]))
                    for road in edges if road[0] == i)

#This gives the value but not the actions

#Recursive function with actions
def joe(i):
    if i == 'J':
        return (0, 'Done')
    else:
        return min( max(edges[road], joe(road[1])[0])
                    for road in edges if road[0] == i)
```

Betting Strategy

Problem

Suppose Michelle currently has \$2 and is allowed to play a game of chance three times. If she bets b dollars on a play of the game then with probability 0.4 she wins b dollars while with probability 0.6 she loses b dollars. (Each bet must be in a whole number of dollars. She can choose to bet \$0 on a game.) Suppose Michelle wants to maximise her probability of having at least \$5 after the three games. What strategy of bets should she use to achieve this?

Solution

Stages

j, games $\{0, 1, 2\}$

State Space

s_j, money at start of game j

Action Space

b_j, money to bet on game j

Data

p, probability of winning

Value Function

$V_j(s_j)$ is the maximum probability of at least \$5 after three games if we start game j with s_j dollars.
We want $V_0(2)$ (the max. probability if we start game 0 with \$2).
We know

$$V_3(s_j) = \begin{cases} 1, & \text{if } s_j \geq 5 \\ 0, & \text{if} s_j < 5 \end{cases}$$

We get

$$V_j(s_j) = \max_{0 \leq b_j \leq s_j} \Big\{ p \times V_{j+1}(s_j + b_j) + (1-p) \times V_{j+1}(s_j - b_j) \Big\}$$

Code

```
#Data
p = 0.4

#Recursive function
def bets(j,s):
    if j == 3:
        if s >= 5:
            return 1
        else:
            return 0
    else:
        return max(p * bets(j+1, s+b) + (1-p) * bets(j+1, s-b)
                   for b in range(0,s+1))

#This returns the value but not the actions

#Recursive function with actions
def bets(j,s):
    if j == 3:
        if s >= 5:
            return (1,'Yay')
        else:
            return (0,'Sigh')
    else:
        return max((p * bets(j+1, s+b)[0] + (1-p) * bets(j+1, s-b)[0], b)
                   for b in range(0,s+1))

#We want bets(0,2) = (0.256,2)
#So bet $2 in first game
#If you win, bet $1 in second game
#If you get to the third game (with $4), bet $2 or $3
```

Advertising Strategy

Problem

A firm is planning its advertising strategy for a period of four weeks. In each week the sales level will be either High or Low and the firm will receive profits on sales of \$800 or \$600, respectively.

If the sales were High in the previous week then there is a 60% chance that sales will be High again in the current week if they do not advertise in the current week or 80% if they do advertise. If the sales were Low in the previous week then there is a 20% chance that sales will be High in the current week if they do not advertise in the current week or 60% if they do advertise.

The cost of advertising in one week is \$70. An extra cost of \$80 is incurred if the level of sales (and thus production) is changed from one week to the next.

What advertising strategy should the firm pursue?

Solution

Stages

t, weeks $\{0, 1, 2, 3\}$

State Space

s_t, sales level in previous week $\{H, L\}$

Action Space

a_t, whether to advertise or not $a_t \in \{Y, N\}$

Data

p_{sa}, probability of high sales if sales were s and we take action a
r_s, revenue (\$) if sales are s
c, cost of changing production (\$)
d, cost of advertising (\$)

Value Function

$V_t(s_t)$ is the maximum expected profit from weeks $t, \ldots, 4$ if we had sales s_t in the previous week.
We want $V_0(s_0)$ for $s_0 \in \{H, L\}$.
We know $V_4(s_4) = 0$.
We get

$$V_t(H) = \max \begin{cases} p_{HY}\big(r_H - d + V_{t+1}(H)\big) + (1 - p_{HY})\big(r_L - d - c + V_{t+1}(L)\big), \\ \text{if we advertise (Y)} \\ \\ p_{HN}\big(r_H + V_{t+1}(H)\big) + (1 - p_{HN})\big(r_L - c + V_{t+1}(L)\big), \\ \text{if we do not advertise (N)} \end{cases}$$

$$V_t(L) = \max \begin{cases} p_{LY}\big(r_H - d - c + V_{t+1}(H)\big) + (1 - p_{LY})\big(r_L - d + V_{t+1}(L)\big), \\ \text{if we advertise (Y)} \\ \\ p_{LN}\big(r_H - c + V_{t+1}(H)\big) + (1 - p_{LY})\big(r_L + V_{t+1}(L)\big), \\ \text{if we do not advertise (N)} \end{cases}$$

Code

```
revHigh = 800
revLow = 600

def advertise(t,s):
    if t == 5:
        return(0,'Done')

    else:
        if s == 'High':
            yes = 0.8*(revHigh - 70 + advertise(t+1, 'High')[0]) + \
                  0.2*(revLow - 70 - 80 + advertise(t+1, 'Low')[0])
            no = 0.6*(revHigh + advertise(t+1, 'High')[0]) + \
                 0.4*(revLow - 80 + advertise(t+1, 'Low')[0])

        else: #s == 'Low'
            yes = 0.6*(revHigh - 70 - 80 + advertise(t+1, 'High')[0]) + \
                  0.4*(revLow - 70 + advertise(t+1, 'Low')[0])
            no = 0.2*(revHigh - 80 + advertise(t+1, 'High')[0]) + \
                 0.8*(revLow + advertise(t+1, 'Low')[0])

        return max((yes, 'Yes'), (no, 'No'))
```

Bird Song

Problem

A male bird needs to sing in order to find a mate but also needs to spend time foraging to survive. How should he split his time between these two tasks?

We will split a time horizon of one day into 150 time segments (just under 10 minutes each) where the bird is able to decide his behaviour in each segment. Time 0 is dawn, time 75 is dusk and time 150 is dawn of the following day. During each day segment (0– 74) he can sing, forage or rest. During each night segment (75–149) he can only rest.

We will denote the bird's food reserves by i. If the reserves reach 0 then the bird dies of starvation.

Singing
If the bird reserves i and spends a time segment singing then he will use D food reserves where

$$D = 12 + 0.002i + B$$

where B is -6.4, 0, 6.4 with probabilities 0.25, 0.50, 0.25, respectively.

In each time segment that the bird is singing he has a probability of 0.004 of pairing with a mate.

Foraging
If the bird instead spends a time segment foraging then he will use D food reserves where

$$D = 8 + 0.007i + B$$

where B is as for singing.

In each time segment that the bird is foraging he has a probability of 0.6 of finding a food patch that gives him $E = 32$ food reserves.

Resting
If the bird spends a time segment resting then he uses $D = 3.6$ food reserves.

At the end of each time segment, let $x = i + E - D$ and $p = x - \lfloor x \rfloor$. Then the food reserves at the start of the next time segment will be $\lfloor x \rfloor + 1$ with probability p and $\lfloor x \rfloor$ with probability $1 - p$.

At the end of the whole time horizon (time 150) the bird receives 2 points if he has a mate, 1 point if he is alive but has not found a mate, and 0 points if he is dead. What is the optimal strategy that the male bird should pursue?

Solution

Stages
t, time segments ($t \in [0, 150]$)

State Space
i, food reserves
m, found mate (1, if yes; 0, if no)

Action Space
Sing, Forage, Rest

Data
No data is listed for this question, for simplicity.

Value Function
$V_t(i, m)$ is the maximised expected points at each end of day if we start time segment t with i reserves and mate m.

We want $V_0(i, 0)$.

We know:

$$V_t(i, m) = 0, \ i \leq 0$$

$$V_{150}(i, m) = \begin{cases} 2, & \text{if } m = 1 \text{ and } i > 0 \\ 1, & \text{if } m = 0 \text{ and } i > 0 \\ 0, & \text{if } i \leq 0 \end{cases}$$

$$V_t(i, m) = V_{t+1}(i - 3.6, m), \text{ if } t \geq 75$$

We get (for $t \leq 75$):

$$V_t(i, m) = \max \begin{cases} V_{t+1}(i - 3.6, m), & \text{Rest} \\ 0.004V_{t+1}(i - (12 + 0.002i), 0) + 0.996V_{t+1}(i - (12 + 0.002i), m), & \text{Sing} \\ 0.6V_{t+1}(i - (8 + 0.007i) + 32, m) + 0.4V_{t+1}(i - (8 + 0.007i), m), & \text{Forage} \end{cases}$$

Code

```
from pylab import *

psing = 0.004
pforage = 0.6
restfood = 3.6
foodpatch = 32

def singfood(i):
    return 12 + 0.002*i

def foragefood(i):
    return 8 + 0.007*i

def SongDash(t,x,m):
    i = int(x) #floor of x
    p = x - i #probability of being i+1
    return p*Song(t,i+1,m)[0] + (1-p)*Song(t,i,m)[0]

def SongBlur(t,i,m):
    return 0.25*SongDash(t,i-6.4,m) + 0.5*SongDash(t,i,m) + \
           0.25*SongDash(t,i+6.4,m)

#Memoization for runtime
_Song = {}

def Song(t,i,m):
    #Base cases
    if i <= 0:
        return (0,'Dead')
    elif t == 150:
        if m == 1:
            return (2,'Mate')
        else:
            return (1,'Lonely')

    #Main function
    else:
        if (t,i,m) not in _Song:
            if t >= 75:
                _Song[t,i,m] = (SongDash(t+1,i-restfood,m), 'Rest')
            else:
                rest = SongDash(t+1,i-restfood,m)
                sing = psing * SongBlur(t+1,i-singfood(i),1) + \
                       (1-psing) * SongBlur(t+1,i-singfood(i),m)
                forage = pforage * \
                         SongBlur(t+1,i-foragefood(i)+foodpatch,m) + \
                         (1-pforage) * SongBlur(t+1,i-foragefood(i),m)

                _Song[t,i,m] = max((rest,'Rest'), (sing,'Sing'), \
                                   (forage, 'Forage'))
        return _Song[t,i,m]

```

```
53 def SingThreshold(t):
54     i = 1
55     while Song(t,i,0)[1] != 'Sing':
56         i += 1
57     return i
58 
59 thresholds = [SingThreshold(t) for t in range(75)]
60 plot(range(75), thresholds)
61 xlabel('Time period')
62 ylabel('Food reserve required to sing')
63 show()
64 
65 #By changing "Rest" to "ZZZZ" the graph changes due to the max
66 #function which returns in alphabetical order as the values
67 #become equal
```