2018

Semester 2

Assignment One

Connor McDowall
cmcd398
530913386

April 3, 2019

Listings

Contents

List of Figures

List of Tables

1 Joint Distributions

1.1 Event Definition

$$x \epsilon X$$
$$y \epsilon Y$$
$$X = DiceOne$$
$$Y = DiceTwo$$
$$A : x + y = 7$$
$$B : x = 3$$
$$C : y = 4$$
$$X \epsilon 1, 2, 3, 4, 5, 6$$
$$Y \epsilon 1, 2, 3, 4, 5, 6$$

1.2 Pairwise Independence

Event A, B and C are pairwise independent as any combination of two events is possible.
Both dice two an equal four and the sum of both dice can equal 7
Both dice one can equal three and the sum of dice 1 and 2 can still equal seven
Both dice two can equal four and dice three equal one
Therefore, the pairwise independence is shown through the following

$$P(A) = \frac{1}{6}$$
$$P(B) = \frac{1}{6}$$
$$P(C) = \frac{1}{6}$$

$$P(A \cap B) = P(A|B) \times P(B) = P(A)P(B) = \frac{1}{6} \times \frac{1}{6} = \frac{1}{36}$$
$$P(A \cap C) = P(A|C) \times P(C) = P(A)P(C) = \frac{1}{6} \times \frac{1}{6} = \frac{1}{36}$$
$$P(B \cap C) = P(B|C) \times P(C) = P(B)P(C) = \frac{1}{6} \times \frac{1}{6} = \frac{1}{36}$$

1.3 Mutual Independence

Events A, B and C are not mutually independent as satisfies pairwise independence (above) but not the following condition

$$P(A \cap B \cap C) = P((A \cap B)|C) \times P(C) = \frac{1}{6} \times \frac{1}{6} \neq P(A) \times P(B) \times P(C) = \frac{1}{6} \times \frac{1}{6} \times \frac{1}{6}$$

2 Question Two

2.1 Transition Probabilities Matrix

$$
\begin{aligned}
U &= Unfinished \\
P &= PoorCondition \\
G &= GoodCondition \\
S &= Scrapped \\
A &= AverageCondition
\end{aligned}
$$

	U	P	G	A	S
U	0	0.2	0.4	0.3	0.1
P	0	0.2	0.4	0.3	0.1
G	0	0	1	0	0
A	0	0	0	1	0
S	0	0	0	0	1

2.2 Limiting Distribution

You need the probabilities in reaching the absorbing states for the limiting distribution. We are not concerned with U or P.

$$
\begin{aligned}
Pr(G) &= P_{UG} + P_{UG} \times P_{UP} \times P_{PP} \\
&= 0.4 + 0.4 \times 0.2 \times \sum_{n=0}^{\infty} 0.2^n \\
&= 0.4 + 0.4 \times 0.2 \times \frac{1}{1-0.2} \\
&= 0.5 \\
Pr(s) &= P_{Us} + P_{US} \times P_{UP} \times P_{PP} \\
&= 0.1 + 0.1 \times 0.2 \times \sum_{n=0}^{\infty} 0.2^n \\
&= 0.1 + 0.1 \times 0.2 \times \frac{1}{1-0.2} \\
&= 0.125 \\
Pr(A) &= P_{UA} + P_{UA} \times P_{UP} \times P_{PP} \\
&= 0.3 + 0.3 \times 0.2 \times \sum_{n=0}^{\infty} 0.2^n \\
&= 0.3 + 0.3 \times 0.2 \times \frac{1}{1-0.2} \\
&= 0.375
\end{aligned}
$$

Therefore, the limiting distribution for unfinished goods is

U	P	G	S	A
0	0	0.5	0.125	0.375

2.3 Expected Profit

You need to calculate the mean hitting times from going from unfinished to a finished states (G,S or A). Form simultaneous equations to solve the problem.

$$M_{ij} = 1 + \sum_{k=1}^{n} P_{ik} \times M_{kj} \text{ where } i \neq j$$
$$M_{PG} = 1 + P_{PP}M_{PG} + P_{PG}M_{GG} + P_{PA}M_{AG} + P_{PS}M_{SG} + P_{PU}M_{UG}$$
$$M_{PS} = 1 + P_{PP}M_{PS} + P_{PG}M_{GS} + P_{PA}M_{AS} + P_{PS}M_{SS} + P_{PU}M_{US}$$
$$M_{PA} = 1 + P_{PP}M_{PA} + P_{PG}M_{GA} + P_{PA}M_{AA} + P_{PS}M_{SA} + P_{PU}M_{UA}$$

You can simplify the equations using the transition probability matrix

$$M_{PG} = 1 + P_{PP}M_{PG}$$
$$M_{PS} = 1 + P_{PP}M_{PS}$$
$$M_{PA} = 1 + P_{PP}M_{PA}$$

Rearranging these equations, you get

$$M_{PG} = 1.25$$
$$M_{PS} = 1.25$$
$$M_{PA} = 1.25$$

Use the same process for the other equations

$$M_{UG} = 1 + P_{UP}M_{PG} + P_{UG}M_{GG} + P_{UA}M_{AG} + P_{US}M_{SG} + P_{UU}M_{UG}$$
$$M_{US} = 1 + P_{UP}M_{PS} + P_{UG}M_{GS} + P_{UA}M_{AS} + P_{US}M_{SS} + P_{UU}M_{US}$$
$$M_{UA} = 1 + P_{UP}M_{PA} + P_{UG}M_{GA} + P_{UA}M_{AA} + P_{US}M_{SA} + P_{UU}M_{UA}$$

Eliminate terms and substitute equations from above

$$\begin{aligned} M_{UG} &= 1 + P_{UP}M_{PG} \\ &= 1 + 0.2 * 1.25 \\ &= 1.25 \\ M_{US} &= 1 + P_{UP}M_{PS} \\ &= 1 + 0.2 * 1.25 \\ &= 1.25 \\ M_{UA} &= 1 + P_{UP}M_{PA} \\ &= 1 + 0.2 * 1.25 \\ &= 1.25 \end{aligned}$$

Since the mean hitting time 1.25 for all possible routes through the system

$$\begin{aligned} \text{Expected Profit} &= 50 \times 0.5 + 40 \times 0.375 - 20 - 10 \times 1.25 \\ &= \$7.50 \end{aligned}$$

2.4 Updated Transition Probilities Matrix

	U	P_1	P_2	G	A	S
U	0	0.2	0	0.4	0.3	0.1
P_1	0	0	0.2	0.4	0.3	0.1
P_2	0	0	0	0.4	0.3	0.3
G	0	0	0	1	0	0
A	0	0	0	0	1	0
S	0	0	0	0	0	1

3 Question Three

3.1 Message

could you order a peri peri chicken sandwich for me and fries for alice
we want you to go to the closest restaurant
are you driving there in your car
where are you
how far are you from here
we are starving
what time will you show up here

See code in the appendix 5.1

4 Oil Testing with Bayesian Networks and Decision Trees

4.1 Bayesian Diagram and Decision Tree

Node G defines the outcome of the geological structure at the well
Node O defines the outcome of oil level given the geological structure of the well
Node T defines the outcome of the test drill given the oil level in the well
Node S defines the outcome of the seismic test given the geological structure of the well

a).

$$\begin{tikzcd}
 & \cdot \arrow[d, dashed, "f_G(g)"] & \\
 & G \arrow[dl, "f_{O|G} = g(o)"'] \arrow[dr, "f_{S|G} = g(s)"] & \\
O \arrow[d, "f_{T|O} = o(t)"'] & & S \\
T & &
\end{tikzcd}$$

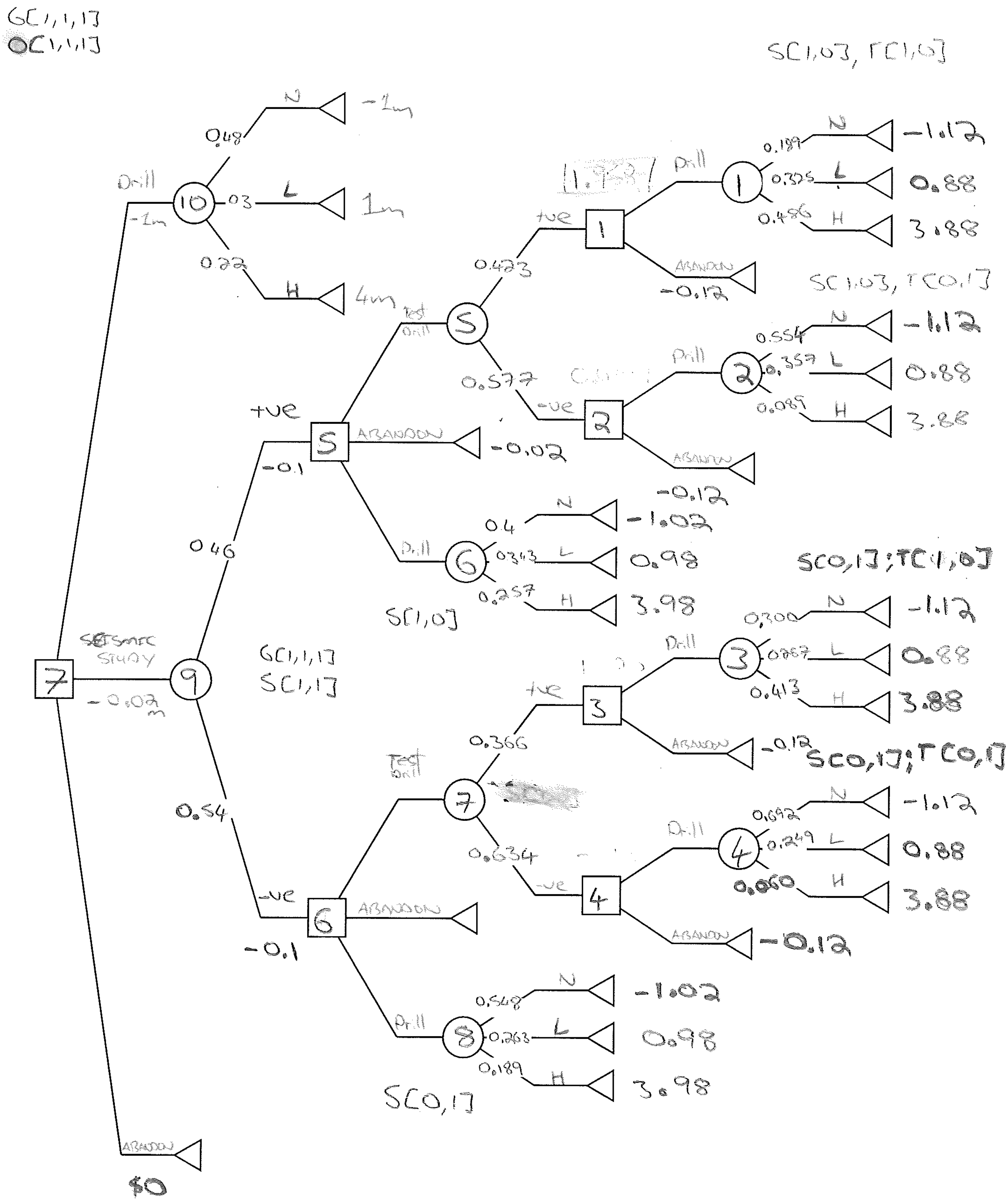
$$\begin{array}{ccc} A & B & C \end{array}$$
$$\begin{bmatrix} 0.2 & 0.4 & 0.4 \end{bmatrix}$$

$$\begin{array}{c|ccc} & N & L & H \\ \hline A & 0.8 & 0.1 & 0.1 \\ B & 0.5 & 0.3 & 0.2 \\ C & 0.3 & 0.4 & 0.3 \end{array}$$

$$\begin{array}{c|cc} & + & - \\ \hline A & 0.3 & 0.7 \\ B & 0.2 & 0.8 \\ C & 0.8 & 0.2 \end{array}$$

$$\begin{array}{c|cc} & + & - \\ \hline N & 0.2 & 0.8 \\ L & 0.4 & 0.6 \\ H & 0.8 & 0.2 \end{array}$$

b).



All Probabilities are rounded to 3DP for Diagram purposes. Unrounded values are used for Calculation.

See 5.2 for the setup of the belief propagation.

4.2 Intepretation

Using the probabilities and costs, the expected values at each chance and decision node (as per the labelled decision tree) are

Chance Node	Amount ($m)
1	1.95818931
2	0.03813252
3	1.51967613
4	-0.324439251
5	0.849565224
6	0.94956522
7	0.480000013
8	0.45037037
9	0.696000008
10	0.7

Figure 1: Chance Node Expected Values

Decision Node	Amount ($m)
1	1.95818931
2	0.03813252
3	1.51967613
4	-0.12
5	0.94956522
6	0.48000013
7	0.7

Figure 2: Decision Node Expected Values

If Anadarko is risk neutral, the optimal strategy is to drill for oil straight away with an expected profit of $700k. This is more than the expected value of seismic testing and test drilling pathway ($696k) and abandoning the well ($0).

If Anadarko is risk adverse, it is likely they will choose to undergo seismic testing and test drilling. The expected value is only $4000 less and they can choose to abandon the project if they are not comfortable with the test results along the way, minimising their potential losses.

5 Appendix

5.1 HMM code

```
def constructEmissions(pr_correct, adj):
    ## This function takes in a matrix detailing the adjacent letters on a
    # keyboard, and the probability of hitting the correct key and outputs
    # a matrix of emission probabilities
    #
    ## INPUT
    # pr_correct - the probability of correctly hitting the intended key;
    # adj - a 26 x 26 matrix with adj[i][j] = 1 if the i'th letter in the
    # alphabet is adjacent
    # to the j'th letter.
    #
```

```
    # OUTPUT
    # b - a 26 x 26 matrix with b[i][j] being the probability of hitting
    #  key j if you intended
    # to hit key i (the probabilities of hitting all adjacent keys are identical).

        # Import numpy in the function
    import numpy as np

    # Get the dimensions of the adj matrix to work out the number of letters
    letters = len(adj)

    # Create an array of ones to multiply the keyboard adjacency matrix to
    # find the number of adjacent letters by letter.
    oneArray = np.ones(letters)

    # Multiply the keyboard adjacency matrix with the ones array to get
    # the sum of each row, therefore the number of adjacent letters
    sums = np.matmul(adj,oneArray)

    # Use the sums array to calculate probability of hitting an adjacent key
    for i in range(0,letters):
        adj[i][:] = (adj[i][:]/sums[i])*(1-pr_correct)

    # Assign pr_correct down the diagonal
    for i in range(0,letters):
        adj[i][i] = pr_correct

    return adj
    # Assign the probability of hitting the correct key
def constructTransitions(filename):
    # This function constructs tranisition matricies for lowercase characters.
    # It is assumed that the file 'filename' only contains lowercase characters
    # and whitespace.
    ## INPUT
    #  filename is the file containing the text from which we wish to develop a
    #  Markov process.
    #
    ## OUTPUT
    #  p is a 26 x 26 matrix containing the probabilities of transition from a
    #  state to another state, based on the frequencies observed in the text.
    #  prior is a vector of prior probabilities based on how often each character
    #  appears in the text

    ## Read the file into a sting called text
    with open(filename,'r') as myfile:
        text = myfile.read()

    # Import numpy to use matrices
    import numpy as np

    # Create a list of the unique characters in a string
    uniqueChar2 = ['a','b','c','d','e','f','g','h','i','j',
    'k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z']

    # Create an array for the prior
    prior = np.zeros(len(uniqueChar2))

    # Count the total number of characters in a string excluding spaces and new lines
```

```
    count = 0
    for char in uniqueChar2:
        prior[count] = text.count(char)
        count = count + 1

    total = sum(prior)

    # Convert p to the prior probabilities
    for i in range(0,len(prior)):
        prior[i] = prior[i]/total

    # Initialise the transition probabilities matrix
    p = np.zeros((len(prior),len(prior)),dtype = float)
    # Use a for loop to assign values to the transiition probabilities matrix
    for i in range(0,len(text)-1):
        # Only assign values to the transition probability matrix if the
        # characters exist  in the unique character array
        if text[i] in uniqueChar2 and text[i+1] in uniqueChar2:
            # Use the character array to control indexing therefore array assignment
            p[uniqueChar2.index(text[i])][uniqueChar2.index(text[i+1])] =\
             p[uniqueChar2.index(text[i])][uniqueChar2.index(text[i+1])] + 1

    # Create a vector of ones to perform matrix multiplication
    oneArray = np.ones(len(uniqueChar2))

    # Multiply the transition probabilities matrix with the ones array to get
    # the sum of each row
    sums = np.matmul(p,oneArray)

    # Use the sums array to calculate transition probabilities
    for i in range(0,len(uniqueChar2)):
        p[i][:] = p[i][:]/sums[i]

    return (p,prior)

def HMM(p,pi,b,y):
    ## This function implements the Viterbi algorithm, to find the most likely
    # sequence of states given some set of observations.
    #
    ## INPUT
    #  p is a matrix of transition probabilies for states x;
    #  pi is a vector of prior distributions for states x;
    #  b is a matrix of emission probabilities;
    #  y is a vector of observations.
    #
    ## OUTPUT
    # x is the most likely sequence of states, given the inputs.

    # Import numpy
    import numpy as np

    # Set up the lengths for for loops
    n=len(y) # Number of oservations.
    m=len(pi) # Number of prior distributions.

    # Matrices, each row is a letter (for a given state)
    # for a given observation
    gamma = np.zeros((m,n))
```

```
    phi = np.zeros((m,n))

    # Create a character

    ## You must complete the code below
    for i in range(m):
        # Initialise the algorithm while converting the observation to
        # a number for indexing.
        gamma[i][0] = (b[i][y[0]])*pi[i]

    for t in range(1,n):
        for k in range(26):
            gamma[k,t]=0
            phi[k,t] = 0
            g=[]
            for j in range(26):
                # Calculate the transition probabilities and joint
                # probabilities for the previous state to work out the
                # gamma of this state, appending to a list
                g.append(p[j][k]*gamma[j][t-1])

            # Find the max argmax of the joint probability mulitplied
            # by the transmission probability
            gamma[k,t] = (b[k][y[t]])*np.max(g)
            phi[k,t] = np.argmax(g)

    best=0
    x=[]
    for t in range(n):
        x.append(0)

    # Find the final state in the most likely sequence x(n).
    for k in range(26):
        if best<=gamma[k,n-1]:
            best=gamma[k,n-1]
            x[n-1]=k

    for i in range(n-2,-1,-1):
        # Back track through everything until you get to the end
        x[i] = int(phi[int(x[i+1])][int(i+1)])

    return x

def main():
    # The text messages you have received.
    msgs=[]
    msgs.append('cljlx ypi ktxwf a pwfi psti vgicien aabdwucg vpd me and vtiex voe zoicw
    msgs.append('qe qzby yii tl gp tp yhr cpozwdt fwstqurzby')
    msgs.append('qee ypi xfjvkjv ygetw ib ulur vae')
    msgs.append('wgrrr zrw uiu')
    msgs.append('hpq fzr qee ypi vrpm grfw')
    msgs.append('qe zfr xtztvkmh')
    msgs.append('wgzf tjmr will uiu xjoq jp ywfw')

    print(msgs)

    #The probability of hitting the intended key.
    pr_correct= 0.5
```

```
# An adjacency matrix, adj(i,j) set to 1 if the i'th letter in the alphabet is next
# to the j'th letter in the alphabet on the keyboard.
adj=[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,1,0,0,1],
    [0,0,0,0,0,0,1,1,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0],
    [0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0],
    [0,0,1,0,1,1,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,0,0],
    [0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,0,0],
    [0,0,1,1,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,0,0,0],
    [0,1,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,1,0],
    [0,1,0,0,0,0,1,0,0,1,0,0,0,1,0,0,0,0,0,0,1,0,0,0,1,0],
    [0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0],
    [0,0,0,0,0,0,0,1,1,0,1,0,1,1,0,0,0,0,0,0,1,0,0,0,0,0],
    [0,0,0,0,0,0,0,0,1,1,0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0],
    [0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0],
    [0,0,0,0,0,0,0,0,0,1,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0],
    [0,1,0,0,0,0,0,1,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0],
    [0,0,0,0,0,0,0,0,1,0,1,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0],
    [0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0],
    [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0],
    [0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0],
    [1,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,1],
    [0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0],
    [0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0],
    [0,1,1,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
    [1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0],
    [0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1],
    [0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0],
    [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0]]

# Call a function to construct the emission probabilities of hitting a key
# given you tried to hit a (potentially) different key.
b=constructEmissions(pr_correct,adj)

# Call a function to construct transmission probabilities and
# a prior distribution
# from the King James Bible.
[p, prior]=constructTransitions('bible.txt')

# Run the Viterbi algorithm on each word of the messages
# to determine the
# most likely sequence of characters.
for msg in msgs:
    s_in = msg.split(' ') #divide each message into a list of words
    output=''

    for i in range(len(s_in)):
        y=[]

        for j in range(len(s_in[i])):
            y.append(ord(s_in[i][j])-97) #convert the letters
            # to numbers 0-25

        x=HMM(p,prior,b,y) #perform the Viterbi algorithm

        for j in range(len(x)):
            output=output+chr(x[j]+97) #convert the states x back to letters
```

```
            # Add each word to the output list
            if i!=len(s_in)-1:
                output= output + ' ' #recreate the message

        print(msg) #display received message
        print(output) #display decoded message
        print('')

if __name__ == "__main__":
    main()
```

5.2 Belief Propagation

```
import numpy as np

def main():
######################### USER INPUT STARTS HERE #############################

    # Specify the names of the nodes in the Bayesian network
    nodes=['G','O','T','S']

    # Defining arcs which join pairs of nodes (nodes are indexed 1...N)
    B=[]

    B.append(['G','O'])
    B.append(['G','S'])
    B.append(['O','T'])

    # Set up information struction
    info={}

    # Set up conditional distribution structure
    M={}

    # Specify any given information for each event (a vector of 1s means
    # there is no information given for that event.
    # If information is given for an event, place a 0 corresponding
    # to any outcome that is impossible.
    info['G']=np.array([1,1,1])
    info['O']=np.array([1,1,1])
    info['T']=np.array([1,0])
    info['S']=np.array([0,1])

    # Specify conditional distributions
    M['G']=np.array([0.2,0.4,0.4])
    M['O']=np.array([[0.8,0.1,0.1],[0.5,0.3,0.2],[0.3,0.4,0.3]])
    M['T']=np.array([[0.2,0.8],[0.4,0.6],[0.8,0.2]])
    M['S']=np.array([[0.3,0.7],[0.2,0.8],[0.8,0.2]])

    #Specify the root node and a list of leaf nodes
    root_node='G'
    leaf_nodes=['T','S']

######################### USER INPUT ENDS HERE #############################

    # Set up structures to store parent and child information for each node
    parent={}
    children={}
```

```python
count={}
# Define A to be the number of arcs in the Bayesian network
#A=len(B)

# Go through arcs, and define parents and children
for i in range(len(B)):
    if B[i][1] not in parent:
        parent[B[i][1]]=B[i][0]
    else:
        print("Multiple parent nodes dectected for node " + str(B[i][1]))

    if B[i][0] not in children:
        children[B[i][0]]=[]
        count[B[i][0]]=0

    count[B[i][0]]+=1
    children[B[i][0]].append(B[i][1])


# Set up structures for belief propagation algorithm
lambda_={}
lambda_sent={}
pi={}
BEL={}
pi_received={}

# First pass, from the leaf nodes to the root node
Q=leaf_nodes
while len(Q)!=0:
    i=Q.pop(0)

    lambda_[i]=info[i]
    if i in children:
        for j in children[i]:
            lambda_[i]=lambda_[i]*lambda_sent[j]

    if i in parent: # if the node is not the root node, send information to its pare
        lambda_sent[i]=M[i].dot(lambda_[i])
        count[parent[i]]-=1
        if count[parent[i]]==0:
            Q.append(parent[i])

# Second pass, from the root node to the leaf nodes
Q=[root_node]
while len(Q)!=0:
    i=Q.pop(0)
    if i not in parent: # if the node is the root node, pi is set to be the
        #prior distribution at the node
        pi[i]=M[i].T;
    else: # otherwise, pi is the matrix product of the message from the
        #parent and the conditional probability at the node
        pi[i]=M[i].T.dot(pi_received[i]);

    # compute a normalised belief vector
    BEL[i]=pi[i]*lambda_[i]
    BEL[i]=BEL[i]/sum(BEL[i])

    # send adjusted and normalised messages to each child
```

```
        if i in children:
            for j in children[i]:
                pi_received[j]=BEL[i]/lambda_sent[j]
                pi_received[j]=pi_received[j]/sum(pi_received[j]);
                Q.append(j)

    # Display the updated distributions, given the information.
    for i in nodes:
        print(str(i) +": "+str(BEL[i]))



if __name__ == "__main__":
    main()
```

2019

Semester 1

ENGSCI 760 Assignment 2

Connor McDowall
cmcd398
530913386

May 13, 2019

Contents

List of Figures

Listings

1 Sworn Statement

I swear on the almighty Thanos that I guarantee this assignment is my own work and in particular that all the code I handed in was written and keyed in by myself without undue assistance by others.

2 Question One

2.1 The Nighbourhood Rule

You swap any two pots that aren't in the same crucible.

2.2 Formal Definition

$$N(x) = \{y(\mathbf{x}, p_1, p_2, c_1, c_2, p_1 = 1..3, p_2 = 1..3, c_1 = 1..16, c_2 = 2...17, c_1 < c_2\} \text{ where}$$
$$y(\mathbf{x}, p_1.p_2, c_1, c_2) = \begin{bmatrix} y_{1,1} & ,..., & y_{1,3} \\ ... & ,..., & ... \\ y_{17,1} & ,..., & y_{17,3} \end{bmatrix}, y_{i,j} = \begin{cases} x_{c_1,p_1} \text{ if } (i,j) = (c_2, p_2) \\ x_{c_2,p_2} \text{ if } (i,j) = (c_1, p_1) \\ x_{i,j} \text{ otherwise} \end{cases}, \forall\ i \epsilon \{1..17\}, j \epsilon \{1..3\}$$

$$\begin{aligned} p_1 &= \text{Pot 1} \\ p_2 &= \text{Pot 2} \\ c_1 &= \text{Crucible 1} \\ c_2 &= \text{Crucible 2} \\ x &= \text{Current solution (Pots in crucibles)} \\ y &= \text{Potential solution in the neighbourhood after swapping (Pots in crucibles)} \\ N(x) &= \text{The neighbourhood of solutions} \end{aligned}$$

3 Question Two

3.1 Definition of Intermediate Values

The intermediate values are the hypothetical profits of the cruicible after swapping out one pot and replacing it with another from another crucible. These values are only for the two crucibles partaking in the swap.

3.2 Sweep Algorithm

Pseudocode
intialization;
Let **x** be a random starting solution.
Calculate the starting value of each crucible
foreach $i \epsilon \{1, .., 17\ \}$ **do**
  v_i = g(Al[Crucible i],Fe[Crucible i])
end
foreach y ϵ *N(x)* **do**
  Calculate the values of the swapped crucibles
  δ_1 = g(Al[Crucible c_1],Fe[Crucible c_1])
  δ_2 = g(Al[Crucible c_2],Fe[Crucible c_2])
  Calculate the change in objective function
  $\Delta = \delta_1 + \delta_2 - v_{c_1} - v_{c_1}$
  if $\Delta > 0$ **then**
    Make changes if the change in objective function is above 0
    x = y
    $v_{c_1} = \delta_1$
    $v_{c_2} = \delta_2$
  end
end

$$\text{Note: } g(Al[Cruciblei], Fe[Cruciblei]) = g(\frac{Al_{i,1} + Al_{i,2} + Al_{i,3}}{3}, \frac{Fe_{i,1} + Fe_{i,2} + Fe_{i,3}}{3})$$
$$g(Al[Cruciblec1], Fe[Cruciblec1]) = g(\frac{Al_{c1,1} + Al_{c1,2} + Al_{c1,3}}{3}, \frac{Fe_{c1,1} + Fe_{c1,2} + Fe_{c1,3}}{3})$$
$$g(Al[Cruciblec2], Fe[Cruciblec2]) = g(\frac{Al_{c2,1} + Al_{c2,2} + Al_{c2,3}}{3}, \frac{Fe_{c2,1} + Fe_{c2,2} + Fe_{c2,3}}{3})$$

4 Question Three

4.1 Simple Function and Task 3A

Both code listings are in 7

4.2 Task 3B

The code listing is in 7

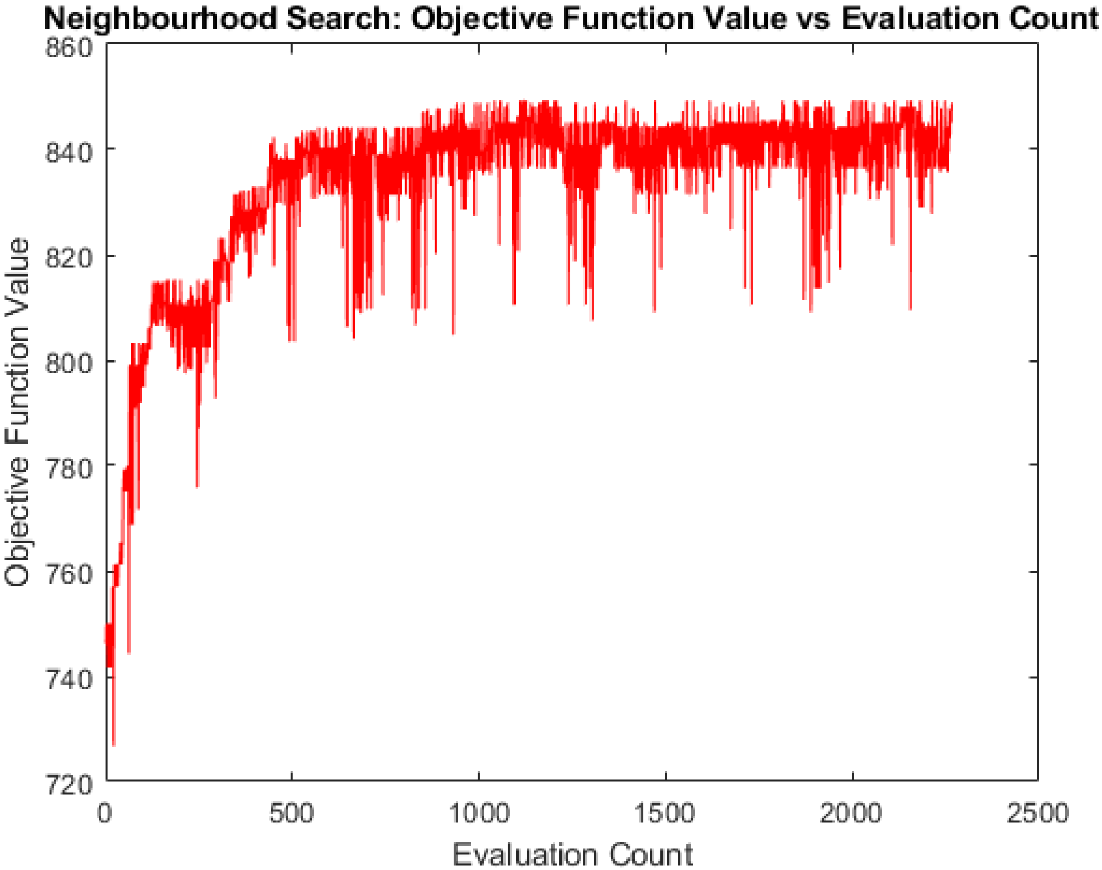


Figure 1: Task 3B plot with 2266 iterations, the sum of the crucible values at $849.28, and a max spread of 36.

```
>> TestAscendToLocalMax(11,0)
 1 [ 51  21  34] 99.51Al  0.50Fe  48.71 30
 2 [ 24   5   6] 99.76Al  0.44Fe  57.35 19
 3 [ 35  41  40] 99.77Al  0.43Fe  57.35  6
 4 [ 47  17  12] 99.50Al  0.49Fe  48.71 35
 5 [  7  27  15] 99.65Al  0.50Fe  52.44 20
 6 [ 16  11  13] 99.26Al  0.61Fe  41.53  5
 7 [ 18   1   3] 99.77Al  0.34Fe  57.35 17
 8 [  8   9  22] 99.51Al  0.52Fe  48.71 14
 9 [ 25  39  14] 99.25Al  0.66Fe  41.53 25
10 [ 28   4  30] 99.78Al  0.44Fe  57.35 26
11 [ 26  32  33] 99.51Al  0.52Fe  48.71  7
12 [ 29  20  36] 99.40Al  0.65Fe  44.53 16
13 [ 37  38   2] 99.54Al  0.48Fe  48.71 36
14 [ 19  10  42] 99.53Al  0.47Fe  48.71 32
15 [ 23  44  45] 99.53Al  0.51Fe  48.71 22
16 [ 46  43  48] 99.75Al  0.41Fe  57.35  5
17 [ 31  50  49] 99.25Al  0.50Fe  41.53 19
                           Sum,Max= 849.28,36
```

Figure 2: Task 3B solutions with 2266 iterations, the sum of the crucible values at $849.28, and a max spread of 36.

4.3 Task 3C

The code listing is in 7

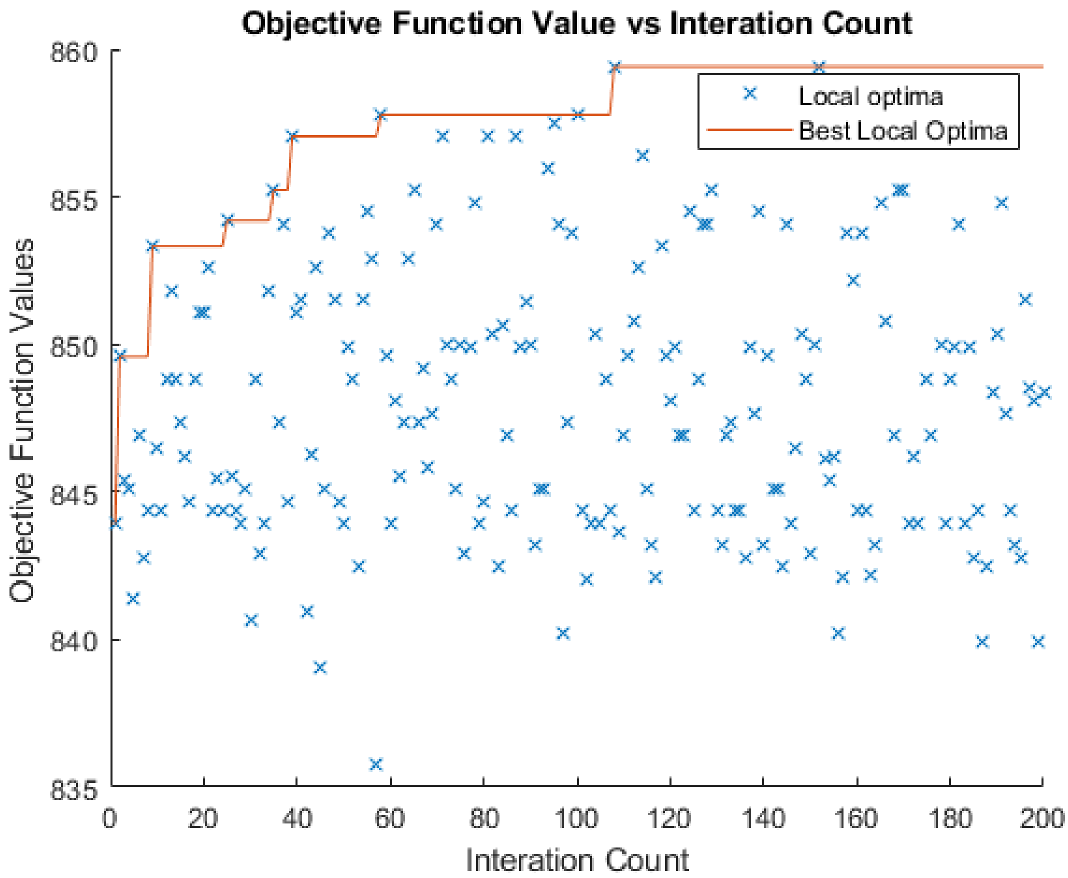


Figure 3: Task 3C plot investigating 200 local optima.

```
>> DoRepeatedAscents(200,11,0)
 1 [  6   43   3] 99.75Al  0.44Fe  57.35 40
 2 [ 24    1  40] 99.87Al  0.32Fe  68.21 39
 3 [  8   16   4] 99.52Al  0.51Fe  48.71 12
 4 [  2   47  35] 99.38Al  0.62Fe  44.53 45
 5 [ 18   30  42] 99.75Al  0.44Fe  57.35 24
 6 [ 21   37  41] 99.75Al  0.37Fe  57.35 20
 7 [  7   34  14] 99.53Al  0.49Fe  48.71 27
 8 [  5   12  31] 99.51Al  0.40Fe  48.71 26
 9 [ 27   45  25] 99.51Al  0.52Fe  48.71 20
10 [ 38   13  50] 99.26Al  0.76Fe  41.53 37
11 [ 46   17  10] 99.75Al  0.38Fe  57.35 36
12 [ 48   26  19] 99.53Al  0.50Fe  48.71 29
13 [ 49   39  33] 99.26Al  0.68Fe  41.53 16
14 [  9   32  51] 99.51Al  0.48Fe  48.71 42
15 [ 20   29  23] 99.37Al  0.69Fe  44.53  9
16 [ 15   11  36] 99.51Al  0.47Fe  48.71 25
17 [ 44   28  22] 99.52Al  0.40Fe  48.71 22
                          Sum,Max= 859.41,45
```

Figure 4: Task 3C showing the best local optima with a total crucible value of $859.41 and a max spread of 45.

5 Question Four

5.1 Plateaus

We expect the objective function to have lots of plateaus. The objective function is driven by the sum of all value functions which are driven by the average purity of aluminium and impurity of iron in the crucible. The composition of one element can cap the value where there can be a range of solutions with the same value. This is due to the discrete nature of the value thresholds. For example, if the aluminium purity is 99.1% in a crucible, the iron impurity in the same crucible may range between 0.08% and 0.089% eventhough a crucible with the same aluminium purity but a lower iron impurity should be worth more. All these combinations have a value of $21, therefore forming a plateau.

5.2 New Crucible Value Function

5.2.1 Mathematical Definition

$g(\bar{AL}, \bar{Fe}) = g(\bar{AL}, \bar{Fe}) + \epsilon((\bar{Al} - \bar{Al_{min}}) - (\bar{Fe_{max}} - \bar{Fe})$
$\epsilon =$ Small gradient.
$\bar{Al_{min}} =$ Minimun aluminium quality for the value threshold.
$\bar{Al} =$ Aluminium quality.
$\bar{Fe_{min}} =$ Minimun iron quality for the value threshold
$\bar{Al} =$ Iron quality.

5.2.2 Explanation

The new crucible value function adds gradients between the discrete value thresholds by considering both the Aluminium and Iron content of the crucible. Both Aluminium and Iron drive the value of the crucible. A maximum amount of iron decreases the value to a threshold while the minimum amount of aluminium increases the value to a threshold. By having the term $(\bar{Al} - \bar{Al_{min}}) - (\bar{Fe_{max}} - \bar{Fe})$, more value is given to the crucible if there is more aluminium than the minimum required for the threshold or there is less iron than the maximum allowed for the threshold. The ϵ is the step size to drive additional value. In this formulation, it must be small and positive. This will improve the search as will push solutions towards ones with better aluminium and iron composition as slopes the plateaus.

5.2.3 Example

With the original function g(), g($\bar{Al} = 99.23, \bar{Fe} = 0.77$) and g($\bar{Al} = 99.20, \bar{Fe} = 0.79$) give the same value of 36.25. However, g($\bar{Al} = 99.23, \bar{Fe} = 0.77$) is better as has more aluminium and less iron. Using an ϵ value of 0.5, g'($\bar{Al} = 99.23, \bar{Fe} = 0.77$) = 36.25 + 0.5((99.23 - 99.20)-(0.79 - 0.77)) = 36.255 when g'($\bar{Al} = 99.20, \bar{Fe} = 0.79$) = 36.25. Since 36.255 is slighly greater than 36.25, the new crucible value function will help drive the objective function towards better solutions as will add slopes to the plateaus.

5.3 The Additive Problem

A negative or net zero contribution to the objective function from the two affected crucible values would reject a swap in a previous iteration. This means the swap did not improve the objective function.
To improve the speed of the algorithm, keep track of all the rejected swaps in the sweep and do not compute them in the current sweep of the neighbourhood. If a swap did not improve the objective value in the last run, it won't in the current run, leading to another rejection. Avoiding repeat computations will improve run time.

6 Question Five

6.1 Mathematical Function

$$g''(\bar{Al}, \bar{Fe}, x_{c1}, x_{c2}, x_{c3}, s) = g(\bar{Al}, \bar{Fe}) - \lambda \times max(0, max(x_{c1}, x_{c2}, x_{c3}) - min(x_{c1}, x_{c2}, x_{c3}) - s)$$

$\lambda =$ The magnitude of the penalty, an arbitrary value set based on the users need. This is a cost per excess spread wher
$max(x_{c1}, x_{c2}, x_{c3}) - min(x_{c1}, x_{c2}, x_{c3}) =$ The spread of pots in the crucibles
$s =$ The max spread allowed in the crucible
$\bar{Al} =$ The aluminium content in the crucible
$\bar{Fe} =$ The iron content in the crucible

The revised value function reduces the value of the crucible if the spread exceeds the max spread. If the spread exceeds the max spread , a cost of λ per unit spread over the max spread is applied to that crucible. Foe example, if the spread is 11 and the max spread is 8, a penalty of $\lambda \times (11 - 8)$ is applied as there is 3 units of excess spread above the max spread. If the spread does not exceed the max spread, no penalty is applied through use of the max function. This penalty is subtracted from the original value function and λ can be set to any positive value to penalise excess spread.

6.2 Modified Code

The modified code can be found in 7.

6.3 Plots and Solutions for spreads 6, 8 and 11.

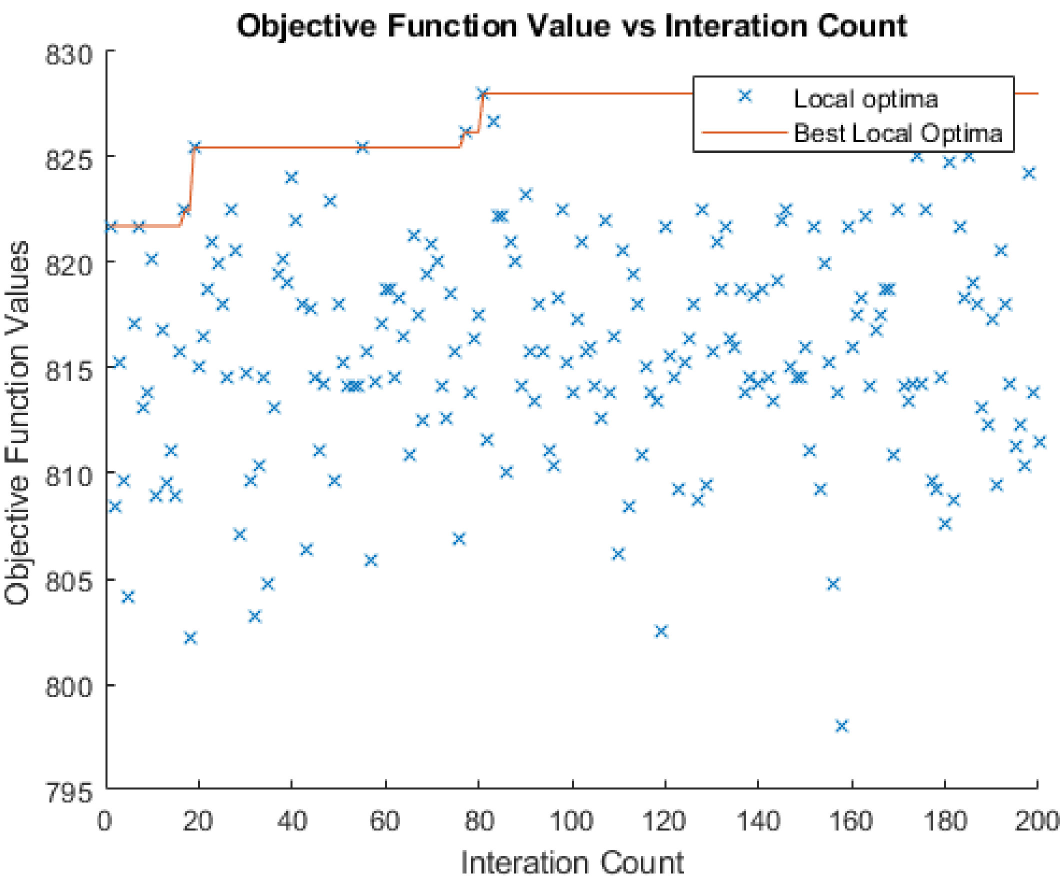


Figure 5: Task 5C plot investigating 200 local optima with a max spread of 6.

```
>> DoRepeatedAscents(200,6,1)
 1 [  9   8   7] 99.50Al  0.44Fe  48.71  2
 2 [ 48  47  51] 99.35Al  0.27Fe  44.53  4
 3 [ 40  37  38] 99.78Al  0.43Fe  57.35  3
 4 [ 27  25  21] 99.52Al  0.69Fe  44.53  6
 5 [ 36  31  34] 99.50Al  0.50Fe  48.71  5
 6 [ 49  50  45] 99.37Al  0.58Fe  44.53  5
 7 [ 15  12  11] 99.52Al  0.50Fe  48.71  4
 8 [ 20  23  26] 99.53Al  0.72Fe  44.53  6
 9 [  6   5   2] 99.55Al  0.53Fe  48.71  4
10 [ 30  28  32] 99.67Al  0.50Fe  52.44  4
11 [ 44  39  42] 99.35Al  0.40Fe  44.53  5
12 [  4   3   1] 99.77Al  0.29Fe  57.35  3
13 [ 17  14  18] 99.66Al  0.42Fe  52.44  4
14 [ 13  10  16] 99.41Al  0.71Fe  44.53  6
15 [ 46  43  41] 99.81Al  0.37Fe  57.35  5
16 [ 33  35  29] 99.36Al  0.45Fe  44.53  6
17 [ 24  22  19] 99.60Al  0.68Fe  44.53  5
                          Sum,Max= 828.01,  6
```

Figure 6: Task 5C showing the best local optima found with a max spread of 6.

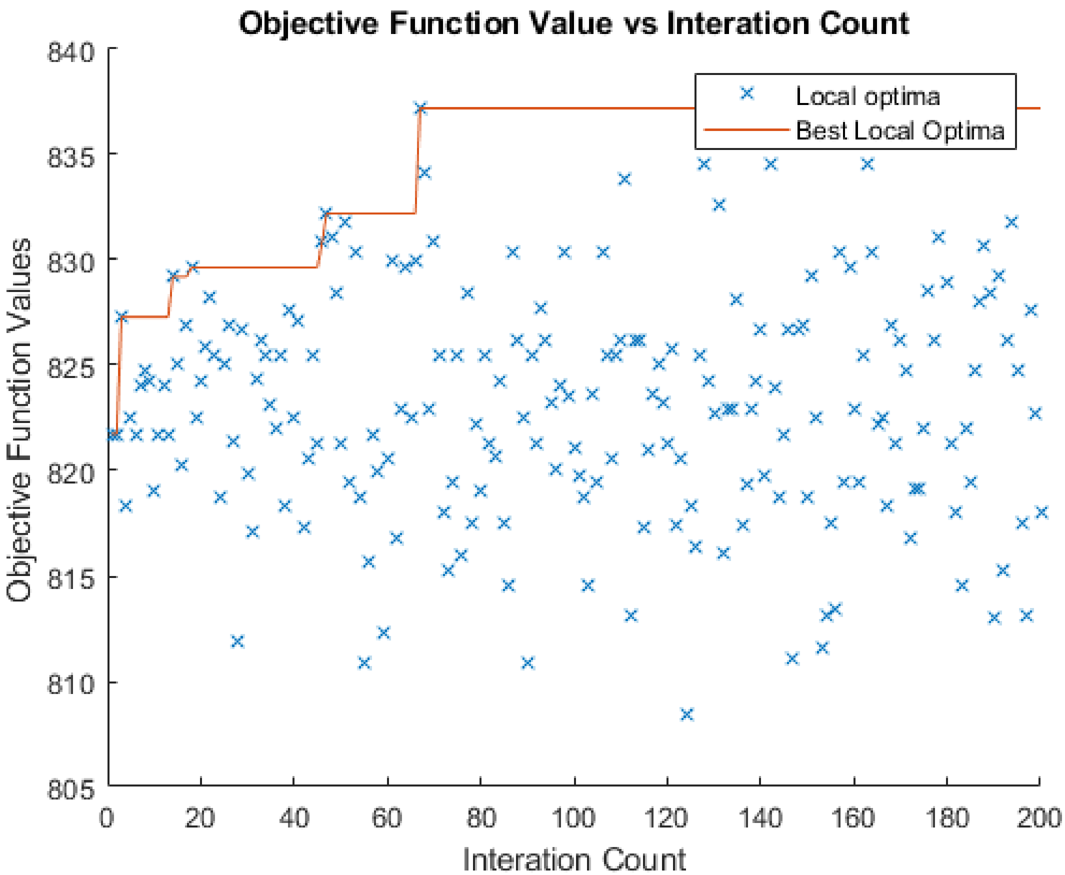


Figure 7: Task 5C plot investigating 200 local optima with a max spread of 8.

```
>> DoRepeatedAscents(200,8,1)
 1 [ 50  49  42] 99.29Al  0.46Fe  41.53  8
 2 [ 12  11  15] 99.52Al  0.50Fe  48.71  4
 3 [ 38  30  37] 99.75Al  0.41Fe  57.35  8
 4 [ 33  27  35] 99.55Al  0.52Fe  48.71  8
 5 [  4   2   5] 99.55Al  0.47Fe  48.71  3
 6 [ 45  51  47] 99.41Al  0.34Fe  44.53  6
 7 [  6   1   3] 99.77Al  0.34Fe  57.35  5
 8 [ 40  34  41] 99.75Al  0.40Fe  57.35  7
 9 [ 32  31  26] 99.53Al  0.45Fe  48.71  6
10 [ 20  14  19] 99.45Al  0.75Fe  41.53  6
11 [ 28  21  29] 99.50Al  0.51Fe  48.71  8
12 [ 36  39  44] 99.38Al  0.62Fe  44.53  8
13 [  8   9   7] 99.50Al  0.44Fe  48.71  2
14 [ 46  43  48] 99.75Al  0.41Fe  57.35  5
15 [ 18  24  17] 99.78Al  0.41Fe  57.35  7
16 [ 25  22  23] 99.38Al  0.74Fe  41.53  3
17 [ 10  13  16] 99.41Al  0.71Fe  44.53  6
                             Sum,Max= 837.19,  8
```

Figure 8: Task 5C showing the best local optima found with a max spread of 8.

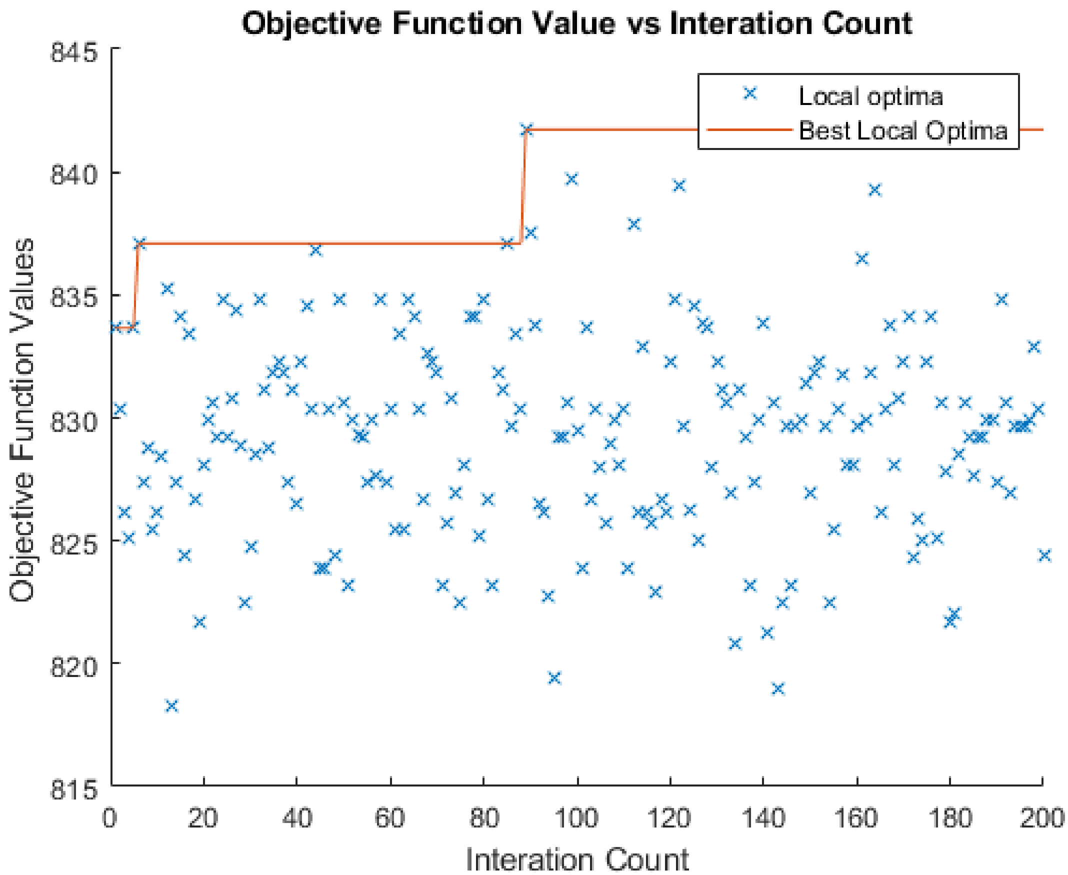


Figure 9: Task 5C plot investigating 200 local optima with a max spread of 11.

```
>> DoRepeatedAscents(200,11,1)
 1 [ 43  48  51] 99.50Al  0.19Fe  48.71  8
 2 [ 31  22  27] 99.55Al  0.48Fe  48.71  9
 3 [ 49  50  45] 99.37Al  0.58Fe  44.53  5
 4 [ 10  16  13] 99.41Al  0.71Fe  44.53  6
 5 [ 14  25  20] 99.35Al  0.66Fe  44.53 11
 6 [ 15  12  11] 99.52Al  0.50Fe  48.71  4
 7 [ 46  40  42] 99.80Al  0.43Fe  57.35  6
 8 [  8   7   9] 99.50Al  0.44Fe  48.71  2
 9 [ 39  33  32] 99.35Al  0.70Fe  44.53  7
10 [ 24  18  17] 99.78Al  0.41Fe  57.35  7
11 [ 41  47  36] 99.54Al  0.49Fe  48.71 11
12 [ 26  23  28] 99.57Al  0.50Fe  48.71  5
13 [  5   4   2] 99.55Al  0.47Fe  48.71  3
14 [ 34  35  44] 99.52Al  0.45Fe  48.71 10
15 [ 38  30  37] 99.75Al  0.41Fe  57.35  8
16 [ 21  19  29] 99.43Al  0.72Fe  44.53 10
17 [  1   6   3] 99.77Al  0.34Fe  57.35  5
                            Sum,Max= 841.73,11
```

Figure 10: Task 5C showing the best local optima found with a max spread of 11.

7 Code Listings

Listing 1: CalcSolutionValue

```
function [SolutionValue] = CalcSolutionValue(x,MaxSpread,penalty, PotAl, PotFe, PotsPerCrucible, NoCrucibles, NoQualities, QualityMinAl, QualityMaxFe, QualityValue)
%Calculates the total value of the current solution x from scratch, and also calculates values for all
% intermediate data; this function uses

% Initialise solution value
SolutionValue = 0;

% Use a for loop to find add all the solution values together
for i = 1:NoCrucibles
    % Find the Aluminium and Iron averages for the crucible in question
    CrucibleAl = (PotAl(x(i,1))+ PotAl(x(i,2)) + PotAl(x(i,3)))/PotsPerCrucible;
    CrucibleFe = (PotFe(x(i,1))+ PotFe(x(i,2)) + PotFe(x(i,3)))/PotsPerCrucible;

    % Put in a conditional statement to calculate the value if the spread
    % penalty of no spread penalty
    if penalty == 0
        SolutionValue = SolutionValue + CalcCrucibleValue(CrucibleAl, CrucibleFe, NoQualities, QualityMinAl, QualityMaxFe, QualityValue);
    else
        % This is the crucible value if the penalty applies for task 5b.
        SolutionValue = SolutionValue + CalcCrucibleValueWithSpreadPenalty(MaxSpread,x(i,:), CrucibleAl, CrucibleFe, NoQualities, QualityMinAl, QualityMaxFe, QualityValue);
    end
end
```

Listing 2: AscendToLocalMax

```
function [SolutionValue, x, CrucibleValues] = AscendToLocalMax(x, MaxSpread,penalty, PotAl , PotFe, PotsPerCrucible, NoCrucibles, NoQualities, QualityMinAl, QualityMaxFe, QualityValue)
% Given a starting solution x, test (and accept where better) all neighbouring solutions in a next ascent
% approach, ie repeatedly sweep the complete neighbourhood, accepting all improvements that are
% found.

% Calculate the starting values for all our crucibles
for i = 1:NoCrucibles
    % Calculates the intial starting values of the crucibles if the spread
    % penalty applies (For task 5B).
    if penalty ==1
        CrucibleValues(i) = CalcCrucibleValueWithSpreadPenalty(MaxSpread,x(i,:),sum(PotAl(x(i,:)))/PotsPerCrucible, sum(
```

```
            PotFe(x(i,:)))/PotsPerCrucible, NoQualities, QualityMinAl, QualityMaxFe, QualityValue);
    % Calculates the intial starting values of the crucibles if the spread
    % penalty does not apply.
    else
        CrucibleValues(i) = CalcCrucibleValue(sum(PotAl(x(i,:)))/PotsPerCrucible, sum(PotFe(x(i,:)))/PotsPerCrucible, NoQualities, QualityMinAl, QualityMaxFe, QualityValue);
    end
end

% Calculates the solution values with the spread penalty adjusted (Task 5b)
SolutionValue = CalcSolutionValue(x,MaxSpread,penalty, PotAl , PotFe, PotsPerCrucible, NoCrucibles, NoQualities, QualityMinAl, QualityMaxFe, QualityValue);

% Initialise solution value array
PlotObjValues = SolutionValue;
count = 1;
evaluationCount = 1;

% Initialise the last optimal solution and looping condition
KeepLooping = 1;
changes = 0;
last= [inf,inf,inf,inf];
% Use a while loop to control the looping criterion
while KeepLooping
    KeepLooping = 0;
    % Use a quadratic for loop to search the neighbourhood for a better
    % solution
    for c1 = 1:NoCrucibles-1
        for p1 = 1:PotsPerCrucible
            for c2 = c1+1:NoCrucibles
                for p2 = 1:PotsPerCrucible

                    % Swap the crucibles pots
                    y = x;
                    y(c1,p1) = x(c2,p2);
                    y(c2,p2) = x(c1,p1);

                    if penalty == 0
                        % Calculate the change in objective function with
                        % no spread penalty to be applied
                        New1 = CalcCrucibleValue(sum(PotAl(y(c1,:)))/PotsPerCrucible, sum(PotFe(y(c1,:)))/PotsPerCrucible, NoQualities, QualityMinAl, QualityMaxFe, QualityValue);
                        New2 = CalcCrucibleValue(sum(PotAl(y(c2,:)))/PotsPerCrucible, sum(PotFe(y(c2,:)))/PotsPerCrucible, NoQualities, QualityMinAl, QualityMaxFe, QualityValue);
                    else
                        % Calculate the change in objective function if spread
                        % penalty to be applied (Task 5B).
```

```
                        New1 = CalcCrucibleValueWithSpreadPenalty(MaxSpread,y(c1,:),sum(PotAl(y(c1,:)))/PotsPerCrucible, sum(PotFe(y(c1,:)))/PotsPerCrucible, NoQualities, QualityMinAl, QualityMaxFe, QualityValue);
                        New2 = CalcCrucibleValueWithSpreadPenalty(MaxSpread,y(c2,:),sum(PotAl(y(c2,:)))/PotsPerCrucible, sum(PotFe(y(c2,:)))/PotsPerCrucible, NoQualities, QualityMinAl, QualityMaxFe, QualityValue);
                    end

                    %Find the change in objective function
                    change = New1 + New2 - CrucibleValues(c1) - CrucibleValues(c2);

                    % Record the solution value regardless of change
                    PlotObjValues = [PlotObjValues,SolutionValue + change];
                    count = count + 1;
                    evaluationCount = [evaluationCount,count];


                    % Make changes if a positive change
                    if change > 0.01
                        x=y;
                        CrucibleValues(c1) = New1;
                        CrucibleValues(c2) = New2;
                        SolutionValue = SolutionValue + change;
                        last = [c1,c2,p1,p2];
                        KeepLooping = 1;
                    end

                    % Check if the swap we are doing is the last swap we
                    % made
                    if  (last(1) == c1 && last(2) == c2 && last(3) == p1 && last(4) == p2 && ~KeepLooping)
                        % Do the plotting in here
                        %figure;
                        %plot(evaluationCount,PlotObjValues,'r')
                        %ylabel('Objective Function Value')
                        %xlabel('Evaluation Count')
                        %title('Neighbourhood Search: Objective Function Value vs Evaluation Count')
                        return
                    end
                end
            end
        end
    end
end
end
```

Listing 3: TestAscendToLocalMax

```
function TestAscendToLocalMax(MaxSpread,penalty)
% Initialise the data
  [NoCrucibles,NoPots,PotsPerCrucible,NoQualities, ...
```

```
            QualityMinAl, QualityMaxFe, QualityValue] = InitQual;
  [PotAl, PotFe] = InitProb;
  cost = 6;

% Generate a boring starting solution
  x = GenStart(NoPots, NoCrucibles, PotsPerCrucible);

% Do the local search here adjusting for task 5b if the penalty boolean is
% applied.
[~, x, ~] = AscendToLocalMax(x,MaxSpread,penalty, PotAl , PotFe, PotsPerCrucible, NoCrucibles, NoQualities, QualityMinAl, QualityMaxFe, QualityValue);

% View the solution (double checking its objective function)
  ViewSoln(x, PotAl, PotFe, NoCrucibles, NoQualities, QualityMinAl, QualityMaxFe, QualityValue);

end
```

Listing 4: DoRepeatedAscents

```
function DoRepeatedAscents(n,MaxSpread,penalty)
%DoRepeatedAscents() that uses AscendToLocalMax() to do n
% repeated ascents from random starting solutions.

% Initialise storage arrays and variables
objectiveValues = [];
iterationCount =[];
count = 0;
InitialSolutionFound = 0;
BestSolutionValue = 0;

% Initialise the data
  [NoCrucibles,NoPots,PotsPerCrucible,NoQualities, ...
            QualityMinAl, QualityMaxFe, QualityValue] = InitQual;
  [PotAl, PotFe] = InitProb;

for i = 1:n
    % Generates random starting solutions
    x = randperm(51);
    % Rehapes x
    x = reshape(x,[17,3]);

    % Use AscendToLocalMax for the iterations. This has been adjusted for
    % the penalty function for 5b
    [SolutionValue, x, ~] = AscendToLocalMax(x,MaxSpread,penalty, PotAl, PotFe, PotsPerCrucible, NoCrucibles, NoQualities, QualityMinAl, QualityMaxFe, QualityValue);

    % Conditional which sets the first best solution as the first.
    if InitialSolutionFound == 0
       InitialSolutionFound = 1;
       BestSolutionValue = SolutionValue;
    end

    % Use a condition to save the best x solution
    if SolutionValue >= max(objectiveValues)
```

```
        BestSolutionValue = SolutionValue;
        x_save = x;
    end

    % Add solution value to arrays for plotting
    bestobjectValues(i) = max(BestSolutionValue,SolutionValue);
    objectiveValues = [objectiveValues,SolutionValue];
    count = count + 1;
    iterationCount =[iterationCount,count];

end

% Plot all and the best objective values vs interation count.
figure;
hold on
plot(objectiveValues,'x')
plot(bestobjectValues)
ylabel('Objective Function Values')
xlabel('Interation Count')
title('Objective Function Value vs Interation Count')
legend('Local optima','Best Local Optima')

% Show best solution found
ViewSoln(x_save, PotAl, PotFe, NoCrucibles, NoQualities, QualityMinAl,
    QualityMaxFe, QualityValue);
end
```

Listing 5: CalcCrucibleValueWithSpreadPenalty

```
function Value = CalcCrucibleValueWithSpreadPenalty(MaxSpread,
    CruciblePots , CrucibleAl, ...
     CrucibleFe, NoQualities, QualityMinAl, QualityMaxFe, QualityValue)
% This functions calculates the value of the crucible including the
    spread
% penalty.

% Set an overshooting penalty when the spread exceeds the max spread
% The penalty is the per unit cost of the spread exceeding the max
    spread.
% This cost is arbitrary and can be set to any unit.
cost = 8;

% Set the penalty component of the function
penalty = cost*max(0,(max(CruciblePots)-min(CruciblePots)-MaxSpread));

% Find the value of the crucible given the pots aluminium and iron
    purities
% and subtracting the penalties
Value = CalcCrucibleValue(CrucibleAl, CrucibleFe, NoQualities, ...
    QualityMinAl, QualityMaxFe, QualityValue) - penalty;

end
```

2019

Semester 1

Assignment 3: Dynamic Programming

Connor McDowall
cmcd398
530913386

June 7, 2019

Listings

Contents

List of Figures

List of Tables

1 Coin Counting (30 Marks)

1.1 Stages, States, Actions, Costs

Stages : The denominations of coins, $D_n = \{D_1, D_2, ..., D_N\}$
States : Current change owed (x), assume $x \geq 0$
Actions : The number of coins used in the transaction as per the denominations, $a \epsilon A_n(x) = \{0, 1, 2, ..., \dfrac{x}{D_n}\}$
Costs : The number of coins exchanged, $c_n(a) = a$ The costs are independent of x

1.2 Value Function

$$V_N(x) = \begin{cases} \frac{x}{D_N}, & \text{if } \frac{x}{D_N} \epsilon \mathbb{Z}. \\ \infty, & \text{otherwise.} \end{cases} \tag{1}$$

The problem is infeasible if $\frac{x}{D_N}$ is not integer. Change must be given as an integer number of coins Since D_N is the smallest denomination, if the output of $\frac{x}{D_N}$ doesnt give an integer number of coins, then there is there is no combination of coins that can be given as change to form a feasible solution for the value of x in the transaction. This is model by giving this non integer combination a value function of infinity since we are minimising. This is so they are heavily penalised.

1.3 Dynamic Programming Recursion

1.3.1 Equation

$$V_n(x) = \begin{cases} \min\limits_{a \epsilon A_n(x)} \{c_n(a) + V_{n+1}(x - D_n c_n(a))\}, & \text{if x} > 0. \\ 0, & \text{if x} = 0. \end{cases} \tag{2}$$

1.3.2 Explaination

$V_n(x)$: The minimum number of coins with denomination given at stage n as change.
$A_n(x)$: The set of actions, the denominations of coin available for change at stage n.
$c_n(a)$: The cost of given change at stage n.
$V_{n+1}(x - D_n c_n(a))$: The cost to go based on subsequent decisions.

1.4 Optimal substructure and overlapping subproblems explanations

1.4.1 Optimal substructure

Optimal substructure means a combination of locally optimal subproblems find a globally optimal solution. This is the case for most recursive problems. In the context of coin coin tossing, take the following example. If the optimal solution to give change given the set of denominations for x money is c coins, you can break up the problem into two subproblems where the optimal solutions to give change given the same set of coin denominations for y and z money is d and e coins respectively. The combination of the optimal solutions to the two subproblems form the global optimal solution as x money = y + z money and c coins = d + e coins.

1.4.2 Overlapping subproblems

Overlapping subproblems means the optimal solution to subproblem(s) is reused in constructing the optimal solution to the main problem. In the context of this problem, the minimum number of coins to give change for x will be repeatedly used to find the minimum number of coins to give change for y where x < y.

1.5 Natural Ordering

$$\text{New Action Set : The denominations of coins, } A(x) = \{D_1, D_2, ..., D_N\}$$

1.5.1 New recursion

$$V(x) = \begin{cases} \min\limits_{a \epsilon A(x), 0<a\leq x} \{1 + V(x-a)\}, & \text{if otherwise.} \\ 0, & \text{if x = 0.} \end{cases} \tag{3}$$

1.5.2 Natural ordering

A bottom up ordering is the natural ordering of the subproblems. Firstly, solve for n = 0. Secondly, solve upwards to from n = 0 to n = x. No explicit recurson is required as all solutions to the subproblems are solved and saved by the time you reach n = 0, creating a more efficient formulation.

1.6 Algorithm

Listing 1: Bottom Up Optimal Coin Change

```
function numCoins = optimalCoinChange(x, denoms)
% Function finds the minimum number of coins required to change a monetary
% amount.
% Inputs:
% x = amount of money to be given in coins, given as an INTEGER, in cents.
%           e.g. $1.35 is input as 135
% denoms = denominations of coins available, in INTEGER cents,
%           given as a ROW VECTOR.
%
% Output:
% numCoins = optimal number of coins used to find x
%
% Connor McDowall, cmcd398, 530913386

% This code implements a bottom up approach. This approach was adapted from
% https://github.com/bephrem1/backtobackswe/blob/master/Dynamic...
%20Programming,%20Recursion,%20&%20Backtracking/changeMakingProblem.java
% as this is a very common problem.

% Set a parameter to create the appropriately sized subproblem storage
% array and set the values of the matrix to the largest number of coins
% possible plus one (The value of change required). The array includes one
% additional element for the base case.
```

```
dpsols = ones(1,x+1)*(x+1);

% Set the base case for the problem
dpsols(1) = 0;

% Sets the number of coin denominations passed into the function
n_denoms = size(denoms,2);

% Solves all the subproblems
% Iterates through all subproblems
for donIdx = 2:x+1

    %Iterates through all coin denominations
    for coinIdx = 1:n_denoms

        % Compares the coins values to the subproblem value
        if denoms(coinIdx) <= donIdx - 1
            % Performs a form of recursion test for an improved
                solution
            % and sets the new sub problem.
            dpsols(donIdx) = min(dpsols(donIdx), ...
                dpsols(donIdx-denoms(coinIdx)) + 1);
        end
    end
end
% Returns the minimum number of coins to use as change. The minimum
    value
% will be the last value.
 numCoins = dpsols(end);
end
```

2 Conducting Interviews (10 Marks)

2.1 Mathematical expression

$$\hat{V}_N = \mathbb{E}(R) = 1 \tag{4}$$

The derivation is show below

$$\begin{aligned}
\hat{V}_N &= \mathbb{E}(R) \\
&= \int_0^\infty f(r)dr \\
&= \int_0^\infty e^{-r}dr \\
&= [-e^{-r}]_0^\infty \\
&= [-e^{-\infty}] - [-e^0] \\
&= 0 - (-1) \\
&= 1
\end{aligned}$$

2.2 Policy

$$\begin{aligned}
\hat{V}_{N-1} &= \max\{r, \hat{V}_N\} \\
&= \max\{r, 1\}
\end{aligned}$$

Therefore, hire the applicant and stop interviewing if $r \geq 1$. Otherwise, reject the candidate. Therefore, the policy is:

$$\pi \geq 1 \tag{5}$$

2.3 Proof

$$\hat{V}_{N-1} = \int_0^\infty \max\{\hat{V}_N, r\} f(r) dr \tag{6}$$

Use the accept or reject policy criteria to split it up then solve

$$\begin{aligned}
\hat{V}_{N-1} &= \int_0^\pi \hat{V}_N f(r) dr + \int_\pi^\infty r f(r) dr \\
&= \int_0^1 e^{-r} dr + \int_1^\infty r e^{-r} dr \\
&= \int_0^1 e^{-r} dr - [re^{-r}]_1^\infty - \int_1^\infty -e^{-r} dr \\
&= [-e^{-r}]_0^1 - [re^{-r}]_1^\infty - [e^{-r}]_1^\infty \\
&= -e^{-1} + 1 - 0 + e^{-1} - 0 + e^{-1} \\
&= e^{-1} + 1 \text{ as required}
\end{aligned}$$

2.4 Dynamic programming recursion

Similar working as the previous question, just replace the policy in the integrals with $\hat{V}_{N+1}$

$$\begin{aligned}
\hat{V}_N &= \int_0^{\hat{V}_{N+1}} \hat{V}_{N+1} f(r) dr + \int_{\hat{V}_{N+1}}^\infty r f(r) dr \\
&= \hat{V}_{N+1} \int_0^{\hat{V}_{N+1}} e^{-r} dr + \int_{\hat{V}_{N+1}}^\infty r e^{-r} dr \\
&= \hat{V}_{N+1} \int_0^{\hat{V}_{N+1}} e^{-r} dr - [re^{-r}]_{\hat{V}_{N+1}}^\infty - \int_{\hat{V}_{N+1}}^\infty -e^{-r} dr \\
&= \hat{V}_{N+1} [-e^{-r}]_0^{\hat{V}_{N+1}} - [re^{-r}]_{\hat{V}_{N+1}}^\infty - [e^{-r}]_{\hat{V}_{N+1}}^\infty \\
&= -\hat{V}_{N+1}(e^{-\hat{V}_{N+1}}) + \hat{V}_{N+1} - 0 + \hat{V}_{N+1}(e^{-\hat{V}_{N+1}}) - 0 + e^{-\hat{V}_{N+1}} \\
&= e^{-\hat{V}_{N+1}} + \hat{V}_{N+1}
\end{aligned}$$

2019

Semester 1

Assignment 3: Set Partitioning

Connor McDowall
cmcd398
530913386

June 10, 2019

Listings

Contents

List of Figures

List of Tables

1 Question 1

1.1 Part 1

1.1.1 A Matrix Representation

Carts		0	0	0	0	...	0	0	0	0	...	3	4	5	6	...	0	0	0	1	...	8	9	10	11	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	1	2			
Jobs	1	1	1	1	1	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		-	1
	2	0	0	0	0	...	1	1	1	1	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		-	1
	3	0	0	0	0	...	0	0	0	0	...	1	1	1	1	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		-	1
	4	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	1	1	1	1	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		-	1
	5	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	1	1	1	1	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		-	1
	6	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	1	1	1	1	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		-	1
	7	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	1	1	1	1	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		-	1
	8	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	1	1	1	1	...	0	0	0	0	...	0	0	0	0		-	1
	9	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	1	1	1	1	...	0	0	0	0		-	1
	10	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	1	1	1	1		-	1
Time	1	1	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	2	1	1	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	3	1	1	1	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	4	0	1	1	1	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	5	0	0	1	1	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	1	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	6	0	0	0	1	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	1	1	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	7	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	1	1	1	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	8	0	0	0	0	...	1	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	1	1	1	1	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	9	0	0	0	0	...	1	1	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	1	0	0	0	...	1	1	1	1	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	10	0	0	0	0	...	1	1	1	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	1	1	0	0	...	1	1	1	1	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	11	0	0	0	0	...	1	1	1	1	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	1	1	1	0	...	1	1	1	1	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	12	0	0	0	0	...	1	1	1	1	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	1	1	1	1	...	1	1	1	1	...	0	0	0	0	...	1	0	0	0	...	0	0	0	0		<-	1
	13	0	0	0	0	...	0	1	1	1	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	1	1	1	1	...	0	1	1	1	...	1	0	0	0	...	1	1	0	0	...	0	0	0	0		<-	1
	14	0	0	0	0	...	0	0	1	1	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	1	1	1	1	...	0	0	1	1	...	1	1	0	0	...	1	1	1	0	...	0	0	0	0		<-	1
	15	0	0	0	0	...	0	0	0	1	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	1	1	1	1	...	0	0	0	1	...	0	1	1	0	...	1	1	1	1	...	0	0	0	0		<-	1
	16	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	1	1	1	...	0	0	0	0	...	0	0	1	1	...	1	1	1	1	...	0	0	0	0		<-	1
	17	0	0	0	0	...	0	0	0	0	...	1	0	0	0	...	1	0	0	0	...	0	0	0	0	...	0	0	1	1	...	0	0	0	0	...	0	0	0	1	...	1	1	1	1	...	0	0	0	0		<-	1
	18	0	0	0	0	...	0	0	0	0	...	1	1	0	0	...	1	1	0	0	...	0	0	0	0	...	0	0	0	1	...	0	0	0	0	...	0	0	0	0	...	1	1	1	1	...	1	0	0	0		<-	1
	19	0	0	0	0	...	0	0	0	0	...	1	1	1	0	...	0	1	1	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	1	1	1	1	...	1	1	0	0		<-	1
	20	0	0	0	0	...	0	0	0	0	...	1	1	1	1	...	0	0	1	1	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	1	1	1	...	1	1	1	0		<-	1
	21	0	0	0	0	...	0	0	0	0	...	1	1	1	1	...	0	0	0	1	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	1	1	...	1	1	1	1		<-	1
	22	0	0	0	0	...	0	0	0	0	...	1	1	1	1	...	0	0	0	0	...	1	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	1	...	0	1	1	1		<-	1
	23	0	0	0	0	...	0	0	0	0	...	1	1	1	1	...	0	0	0	0	...	1	1	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	1	1		<-	1
	24	0	0	0	0	...	0	0	0	0	...	0	1	1	1	...	0	0	0	0	...	1	1	1	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	1		<-	1
	25	0	0	0	0	...	0	0	0	0	...	0	0	1	1	...	0	0	0	0	...	1	1	1	1	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	26	0	0	0	0	...	0	0	0	0	...	0	0	0	1	...	0	0	0	0	...	1	1	1	1	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	27	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	1	1	1	1	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	28	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	1	1	1	1	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	29	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	1	1	1	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	30	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	1	1	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	31	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	1	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	32	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	33	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	34	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	35	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	36	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	37	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	38	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	39	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	40	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	41	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	42	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	43	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	44	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	45	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	46	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	47	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	48	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	49	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	50	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	51	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	52	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	53	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	54	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	55	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	56	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	57	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	58	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	59	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1
	60	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0	...	0	0	0	0		<-	1

Figure 1: Condensed A matrix Representation

The entire $\min c^T X$, $Ax \leq b$ formulation can be found in cmcd398 762 Assignment 3 Worksheet.xlxs. The time intervals index the end of the period. For example t = 1 is the end of the first interval.

1.1.2 Formulation: Matlab Implementation

The following script was used to formulate the problem and solve both the LP relaxation and IP.

Listing 1: LP Relaxation and IP Implementation

```
% Connor McDowall, cmcd398, 530913386
% This script conducts Question One for the problem
% load in the problem data

jobi = [1,2,3,4,5,6,7,8,9,10];
pi = [3,5,7,2,7,7,8,2,8,4];
di = [12,16,20,20,20,20,29,19,25,22];
ri = [0,7,16,16,21,8,4,12,11,17];

% Initialise the A matrix as an array for 10 jobs and 60 times
    intervals
A = [];
c = [];
A_add = zeros(70,1);
count = 1;
rollcol = 0;


% Use many for loops for the function
for i = 1:length(jobi)
    % Populate the A matrix with all possible time intervals
    for j = 1:(60 - pi(i)-ri(i)+1)
        % Add a new column to the A matrix
        A = [A,A_add];
        A(length(jobi) + count + (ri(i)):length(jobi)+ count +(ri(i)) +
            ...
            pi(i)-1,end) = 1;
        % Determine the lateness, therefore the tardiness for the cost
        % function
        c = [c,max(0, count +(ri(i)) + pi(i)-1-di(i))];
        count = count + 1;
     end
    % Add all the ones at the top of the Matrix for this job
    A(jobi(i),rollcol +1:rollcol +(60 - pi(i)-ri(i)+1)) = 1;
    % Reset the count and increment
    count = 1;
    % Increment the rolling number of columns to help add new columns
    rollcol = rollcol + (60 - pi(i)-ri(i)+1);
end

% Create the b matrix
b = ones(70,1);

% Save A matrix
% save A;

% Write the A matrix and Cost matrix into an excel file to check the
% Correct structure and show in assignment.
xlswrite('Amatrix.xlsx',A)
xlswrite('cmatrix.xlsx',c)
```

```

% Set up the bounds properly to get the correct mix of equality and
% inequality constraints
Aeq =A(1:10,1:end);
beq =b(1:10,1);
Aineq = A(11:end,1:end);
bineq = b(11:end,1);
lb = zeros(445,1);
ub = ones(445,1);
intcon = ones(445,1);

% Uses linprog to calculate the solution to the linear relaxation
X = linprog(c,Aineq,bineq,Aeq,beq,lb,ub);
% Use the sumproduct to work out what the minimum cost is
obj_LP_Relaxation = c*X;

% Uses intlinprog to calculate the solution to the integer programme
Xint = intlinprog(c,intcon,Aineq,bineq,Aeq,beq,lb,ub);

% Calculate the integer solutions objective value
obj_Int = c*Xint;

% Find out the start time, end time and tardiness of each job for the
% integer solution(Write to an excel file and work out manually).
xlswrite('Xint.xlsx',transpose(Xint))
```

1.2 Part 2

Job i	1	2	3	4	5	6	7	8	9	10
pi	3	5	7	2	7	7	8	2	8	4
di	12	16	20	20	20	20	29	19	25	22
ri	0	7	16	16	21	8	4	12	11	17
Start time	1	12	32	19	39	25	4	17	46	21
End time	4	17	39	21	46	32	12	19	54	25
Ti	0	1	19	1	26	12	0	0	29	3

The objective value function ($\min c^T x$) for both LP Relaxation and IP is 91 as the formulation creates naturally integer problem, therefore a naturally integer solution.

2 Question Two

2.1 Original LP Relaxation

The solution is z = 10.333.

Variables:	0	0	0	0	0.333333	0.666667	0	0	0.833333333	0	0	0.166667	0.166667	0.5	0	0	0.333333			
Index:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17			
Cost:	7	5	12	11	1	2	14	12	2	13	6	9	9	2	13	6	9	10.33333	Objective Value	
A	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	=	1
B	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	1	=	1
C	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	=	1
Shift 1	0	1	0	1	1	0	0	0	0	0	0	1	1	0	0	0	1	1	>=	1
Shift 2	0	1	1	1	0	1	0	1	0	1	1	0	0	0	1	0	1	1	>=	1
Shift 3	0	0	1	0	0	0	1	0	1	0	0	0	1	0	0	1	0	1	>=	1
Shift 4	1	0	0	0	1	1	0	0	0	0	1	1	1	0	0	0	0	1.333333	>=	1
Shift 5	0	0	1	1	0	0	1	0	1	0	1	0	1	1	0	0	0	1.5	>=	1
Shift 6	1	0	0	0	0	1	0	1	0	0	0	1	1	0	0	1	0	1	>=	1
Shift 7	0	0	1	0	0	1	1	0	0	0	1	1	1	0	0	1	0	1	>=	1

Staff- Shift Row Coverage	**1**	**2**	**3**	**4**	**5**	**6**	**7**
A	0.333333	0.666667	0	1	0	0.666667	0.666667
B	0.166667	0	0.833333	0.166667	0.833333	0.166667	0.166667
C	0.5	0.333333	0.166667	0.166667	0.666667	0.166667	0.166667

Shift-Shift Row Coverage	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1		0.333333	0.166667	0.666667	0.166667	0.333333	0.333333
2			0	0.666667	0	0.666667	0.666667
3				0.166667	1	0.166667	0.166667
4					0.166667	1	1
5						0.166667	0.166667
6							1
7							

	UB:	**Row Coverage**	**Type**
Next Branch	B3	0.833333333	Staff - Shift
	13	0.166666667	Shift - Shift

Figure 2: The solution to the LP Relaxation with z = 10.333

2.2 Staff-Shift Constraint Branching

2.2.1 Depth = 1, Y_{B3}

Variables:	-1.5E-08	0	2.98E-08	-1E-07	0.333333	0.666667	-4.5E-08	0	1.000000066	0	0	0	0.333333	0.333333	0	0	0.333333			
Index:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17			
Cost:	7	5	12	11	1	2	14	12	2	13	6	9	9	2	13	6	9	10.33333	Objective Value	
A	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	=	1
B	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	1	=	1
C	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	=	1
Shift 1	0	1	0	1	1	0	0	0	0	0	0	1	1	0	0	0	1	1	>=	1
Shift 2	0	1	1	1	0	1	0	1	0	1	1	0	0	0	1	0	1	1	>=	1
Shift 3	0	0	1	0	0	0	1	0	1	0	0	0	1	0	0	1	0	1.333333	>=	1
Shift 4	1	0	0	0	1	1	0	0	0	0	1	1	1	0	0	0	0	1.333333	>=	1
Shift 5	0	0	1	1	0	0	1	0	1	0	1	0	1	1	0	0	0	1.666667	>=	1
Shift 6	1	0	0	0	0	1	0	1	0	0	0	1	1	0	0	1	0	1	>=	1
Shift 7	0	0	1	0	0	1	1	0	0	0	1	1	1	0	0	1	0	1	>=	1
UB: B3	1	1	1	1	1	1	1	0	1	0	0	0	1	1	1	1	1			

Staff- Shift Row Coverage	1	2	3	4	5	6	7
A	0.333333	0.666667	2.98E-08	1	-7.2E-08	0.666667	0.666667
B	0	0	1	0	1	0	-4.5E-08
C	0.666667	0.333333	0.333333	0.333333	0.666667	0.333333	0.333333

Shift-Shift Row Coverage	1	2	3	4	5	6	7
1		0.333333	0.333333	0.666667	0.333333	0.333333	0.333333
2			2.98E-08	0.666667	-7.2E-08	0.666667	0.666667
3				0.333333	1.333333	0.333333	0.333333
4					0.333333	1	1
5						0.333333	0.333333
6							1
7							

	UB:	Row Coverage
Next Branch	A2	0.666666605
Key	1	0
UB:	Keep Column	Remove Column

Figure 3: The solution to the LP Relaxation with a $Y_{B3} = 1$ contraint branch and z = 10.333

2.2.2 Depth = 2, Y_{A2}

Variables:	0	0.5	2.98E-08	0	0	0.5	0	0	1	0	0	0	0.5	0.5	0	-9.9E-09	-3.3E-07			
Index:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17			
Cost:	7	5	12	11	1	2	14	12	2	13	6	9	9	2	13	6	9	11	Objective Value	
A	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	=	1
B	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	1	=	1
C	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	=	1
Shift 1	0	1	0	1	1	0	0	0	0	0	0	1	1	0	0	0	1	1	>=	1
Shift 2	0	1	1	1	0	1	0	1	0	1	1	0	0	0	1	0	1	1	>=	1
Shift 3	0	0	1	0	0	0	1	0	1	0	0	0	1	0	0	1	0	1.5	>=	1
Shift 4	1	0	0	0	1	1	0	0	0	0	1	1	1	0	0	0	0	1	>=	1
Shift 5	0	0	1	1	0	0	1	0	1	0	1	0	1	1	0	0	0	2	>=	1
Shift 6	1	0	0	0	0	1	0	1	0	0	0	1	1	0	0	1	0	1	>=	1
Shift 7	0	0	1	0	0	1	1	0	0	0	1	1	1	0	0	1	0	1	>=	1
UB: B3	1	1	1	1	1	1	1	0	1	0	0	0	1	1	1	1	1			
UB:A2	0	1	1	1	0	1	1	0	1	0	0	0	1	1	1	1	1			

Staff- Shift Row Coverage	1	2	3	4	5	6	7
A	0.5	1	2.98E-08	0.5	2.98E-08	0.5	0.5
B	0	0	1	0	1	0	0
C	0.5	-3.3E-07	0.5	0.5	1	0.5	0.5

Shift-Shift Row Coverage	1	2	3	4	5	6	7
1		0.5	0.5	0.5	0.5	0.5	0.5
2			2.98E-08	0.5	2.98E-08	0.5	0.5
3				0.5	1.5	0.5	0.5
4					0.5	1	1
5						0.5	0.5
6							1
7							

	UB:	Row Coverage
Next Branch	A1	0.500000358
Key	1	0
UB:	Keep Column	Remove Column

Figure 4: The solution to the LP Relaxation with a $Y_{A2} = 1$ contraint branch and z = 11

2.2.3 Depth = 3, Y_{A1}

Variables:	0	1	0	0	0	0	-7.3E-07	0	1.000000732	0	0	0	1	-5E-07	-7.4E-07	1.58E-06	0			
Index:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17			
Cost:	7	5	12	11	1	2	14	12	2	13	6	9	9	2	13	6	9	15.99999	Objective Value	
A	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	=	1
B	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	1	=	1
C	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	=	1
Shift 1	0	1	0	1	1	0	0	0	0	0	0	1	1	0	0	0	1	2	>=	1
Shift 2	0	1	1	1	0	1	0	1	0	1	1	0	0	0	1	0	1	0.999999	>=	1
Shift 3	0	0	1	0	0	0	1	0	1	0	0	0	1	0	0	1	0	2.000002	>=	1
Shift 4	1	0	0	0	1	1	0	0	0	0	1	1	1	0	0	0	0	1	>=	1
Shift 5	0	0	1	1	0	0	1	0	1	0	1	0	1	1	0	0	0	1.999999	>=	1
Shift 6	1	0	0	0	0	1	0	1	0	0	0	1	1	0	0	1	0	1.000002	>=	1
Shift 7	0	0	1	0	0	1	1	0	0	0	1	1	1	0	0	1	0	1.000001	>=	1
UB: B3	1	1	1	1	1	1	1	0	1	0	0	0	1	1	1	1	1			
UB:A2	0	1	1	1	0	1	1	0	1	0	0	0	1	1	1	1	1			
UB:A1	0	1	0	1	1	0	1	0	1	0	0	0	1	1	1	1	1			

Staff- Shift Row Coverage	1	2	3	4	5	6	7
A	1	1	0	0	0	0	0
B	0	0	1	0	1	0	-7.3E-07
C	1	-7.4E-07	1.000002	1	0.999999	1.000002	1.000002

Shift-Shift Row Coverage	1	2	3	4	5	6	7
1		1	1	1	1	1	1
2			0	0	0	0	0
3				1	2	1.000002	1.000001
4					1	1	1
5						1	0.999999
6							1.000002
7							

	UB:	Row Coverage
Next Branch		Max Depth Reached
Key	1	0
UB:	Keep Column	Remove Column

Figure 5: The solution to the LP Relaxation with a $Y_{A1} = 1$ contraint branch and z = 16

2.2.4 Branch and Bound Tree

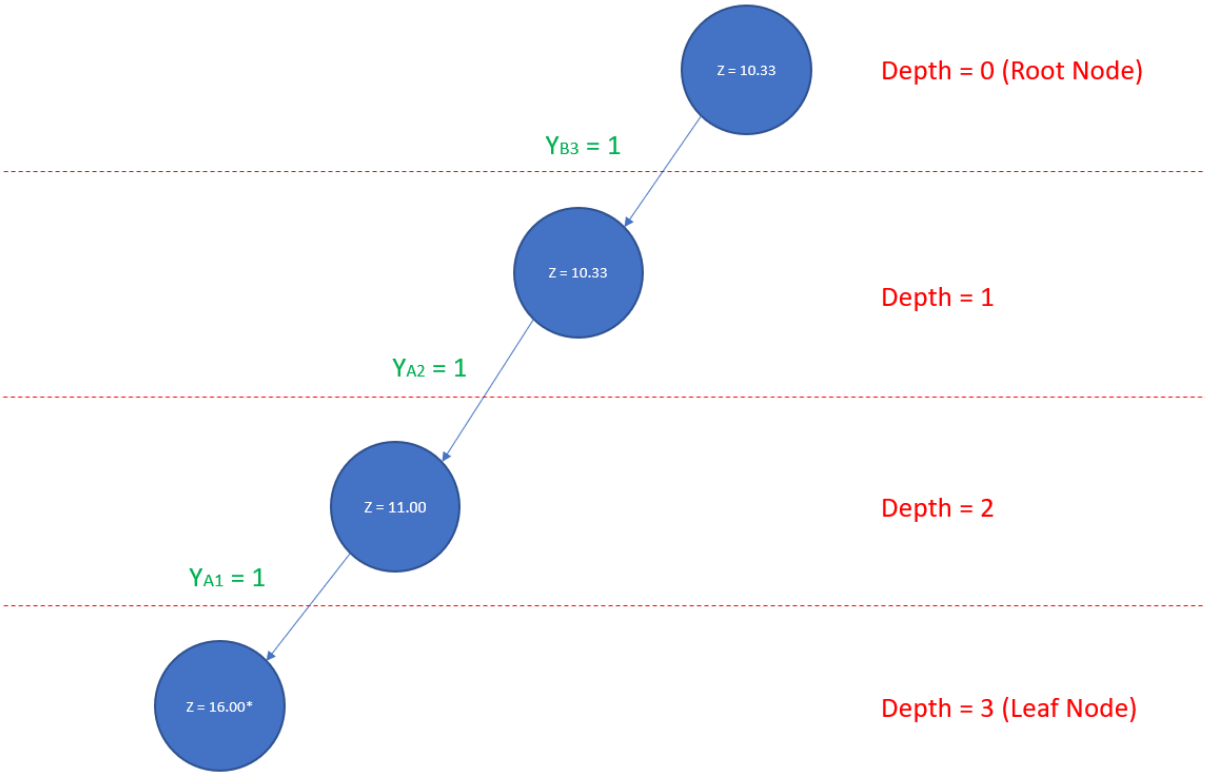


Figure 6: The constraint branch and bound tree with max depth of 3

2.3 Shift-Shift Constraint Branching

The row coverage table suggests you create contraints branches on the Y_{13} variable as per the conditions in the handout. This is the closest to integer branch with the smallest first row and smallest second row index.

2.3.1 Y_{13}

The problem wasn't resolved (objective function) as the question didn't ask for a new solution or to resolve the problem, hence z = 10.333 (it's the LP Relaxation)

Variables:	0	0	0	0	0.333333	0.666667	0	0	0.833333333	0	0	0.166667	0.166667	0.5	0	0	0.333333			
Index:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17			
Cost:	7	5	12	11	1	2	14	12	2	13	6	9	9	2	13	6	9	10.333333	Objective Value	
A	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	=	1
B	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	1	=	1
C	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	=	1
Shift 1	0	1	0	1	1	0	0	0	0	0	0	1	1	0	0	0	1	1	>=	1
Shift 2	0	1	1	1	0	1	0	1	0	1	1	0	0	0	1	0	1	1	>=	1
Shift 3	0	0	1	0	0	0	1	0	1	0	0	0	1	0	0	1	0	1	>=	1
Shift 4	1	0	0	0	1	1	0	0	0	0	1	1	1	0	0	0	0	1.3333333	>=	1
Shift 5	0	0	1	1	0	0	1	0	1	0	1	0	1	1	0	0	0	1.5	>=	1
Shift 6	1	0	0	0	0	1	0	1	0	0	0	1	1	0	0	1	0	1	>=	1
Shift 7	0	0	1	0	0	1	1	0	0	0	1	1	1	0	0	1	0	1	>=	1
13 (0) Branch	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1			
13 (1) Branch	1	0	0	0	0	1	0	1	0	1	1	0	1	1	1	0	0			

Staff- Shift Row Coverage	1	2	3	4	5	6	7
A	0.333333	0.666667	0	1	0	0.666667	0.666667
B	0.166667	0	0.833333	0.166667	0.833333	0.166667	0.166667
C	0.5	0.333333	0.166667	0.166667	0.666667	0.166667	0.166667

Shift-Shift Row Coverage	1	2	3	4	5	6	7
1		0.333333	0.166667	0.666667	0.166667	0.333333	0.333333
2			0	0.666667	0	0.666667	0.666667
3				0.166667	1	0.166667	0.166667
4					0.166667	1	1
5						0.166667	0.166667
6							1
7							

	UB:	Row Coverage
Proposed Branch	13	0.166666667
Key	1	0
(X) Branch	Keep Column	Remove Column

Figure 7: The solution to the LP Relaxation with a Y_{13} contraint branches (0 and 1) and the original LP relaxation.