Midsem Lab Report CS362

*Note:**Code Repository Link**

Jinel Patel[1]
[1]201951075@iiitvadodara.ac.in
Kapadia Tathya[2]
[2]201951078@iiitvadodara.ac.in
Patel Darsh[3]
[3]201951111@iiitvadodara.ac.in
Patel Het[4]
[4]201951112@iiitvadodara.ac.in

CONTENTS

I. INTRODUCTION

In these report we have included the observation,result and conclusion of the 4 experiments given to us in the lab.The 4 experiments we have included are:

1) Lab Assignment 1: Graph Search Agent for 8-Puzzle
2) Lab Assignment 3: TSP using Simulated Annealing
3) Lab Assignment 4: Game Playing Agent — Minimax — Alpha-Beta Pruning
4) Lab Assignment 5: Building Bayesian Networks in R

We have understood and visualised our result in the form of tables and charts and thus discussed the same in these section.

We have executed the codes in Jupyter Notebook for Python and RStudio for R.In these section we have discussed the approach to the problem.We have attached the link to the codes in these section.

Code Repository Link

II. WEEK I LAB ASSIGNMENT 1

Learning Objective: To design a graph search agent and understand the use of a hash table, queue in state space search. In this lab, we need prior knowledge of the types of agents involved and use this knowledge to solve a puzzle called 8-puzzle. 8-Puzzle consist of a 3x3 matrix with the tiles numbered as shown and there is one white space available in which the tile can move.

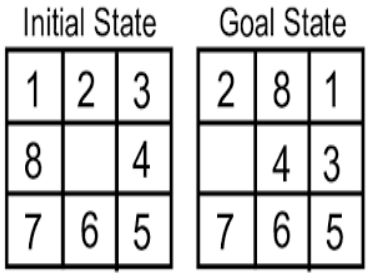


Fig. 1. Initial and Final state of 8-Puzzle [2]

A. Part A

Write a pseudocode for a graph search agent. Represent the agent in the form of a flow chart. Clearly mention all the implementation details with reasons.

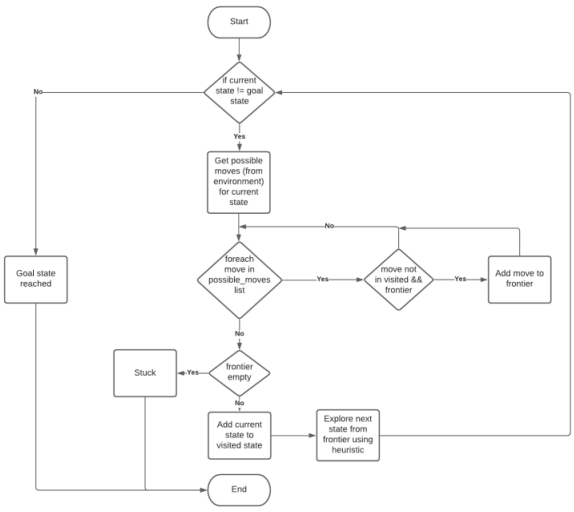


Fig. 2. Flowchart for Agent

Algorithm:

B. Part B

Write a collection of functions imitating the environment for Puzzle-8. Our code consists of following functions:

Algorithm 1 8 puzzle

```
1: procedure BOOLEAN SOLUTION::SEARCH(PRIORITYQUEUE PQ,ARRAY VISITED)
2:     if pq.isEmpty() then return false
3:     puz ← pq.extract() //all possible successors to puz
4:     if search(pq) then return true
5:     for each suc in successors do
6:         if suc not in visited then
7:             pq.insert(suc).
8:             visited.insert(suc).
9:     if search(pq.visited) then return true
10:    else return false;
```

- **Heuritic 1:** Heuristic for calculating the distance of goal state using Manhattan distance. [6]

  Parameters:
  current state(np.ndarray): A 3x3 array with each cell containing unique elements as in current state
  goal state(np.ndarray): A 3x3 array with each cell containing unique elements as in goal state

  returns:
  heuristic1(int): Heuristic value

- **Heuristic 2:** Heuristic for calculating the distance of goal state using number of misplaced tiles

  Parameters:
  current state(np.ndarray): A 3x3 array with each cell containing unique elements as in current state
  goal state(np.ndarray): A 3x3 array with each cell containing unique elements as in goal state

  returns:
  heuristic2(int): Heuristic value

- **generate instance:**

  Parameters:
  goal state(np.ndarray): A 3x3 array with each cell containing unique elements representing the goal
  depthstate(int): The depth at which the state is to be generated
  debug(bool): Get intermediate states and the heuristic values.Default value is False

  returns:
  curr state(np.ndarray): A 3x3 array with each cell containing unique elements representing the state at the given depth form, the goal state

- **get possible next state:** function to get the next possible state from the current state from the environment

  Parameters:
  current state(np.ndarray): A 3x3 array representing the current states parent(string): The path taken to reach the current state from initial Arrangement

  returns:
  possible moves(list): List of possible states from current state
  possible paths(list): List of possible paths moves from current state

- **sort:** This function sorts the state according to the heuristic values generated from one of the heuristic function as selected.
  Parameters:
  possible moves(list): List of possible states from current state goal state(np.ndarray): A 3x3 array representing the goal state heuristic(Integer): An integer indicating the heuristic function to use from 1 or 2. possible paths(list): List of possible moves from current state

  returns:
  sorted possible moves(list): List of possible states from current state, sorted according to heuristic

- **solution:** This function returns success if the goal state is found and prints failure if no goal state is found or the programme is strucked
  Parameters: current state(np.ndarray): A 3x3 array representing the current state goal state(np.ndarray): A 3x3 array representing the goal state heuristic(Integer): An integer indicating the heuristic function to use.

C. Part C

Describe what is Iterative Deepening Search.

Iterative deepening depth first search (IDDFS) Iterative Deepening search was mainly introduced to overcome the problems faced by BFS and DFS algorithms.We do a DFS search in BFS algorithm.The graph/tree is searched in DFS pattern but is allowed to go to a certain depth only.So we do a DFS in BFS algorithm.It is an approach which takes lower space and optimal time compared to DFS or BFS.This would be clearly explained from the following figure [1] Suppose b is the branching factor and depth is d then we have time complexity ans Space Complexity as
Time Complexity: $O(b^d)$
Space Complexity: O(bd)

D. Part D

Considering the cost associated with every move to be the same (uniform cost), write a function which can backtrack and produce the path taken to reach the goal state from the

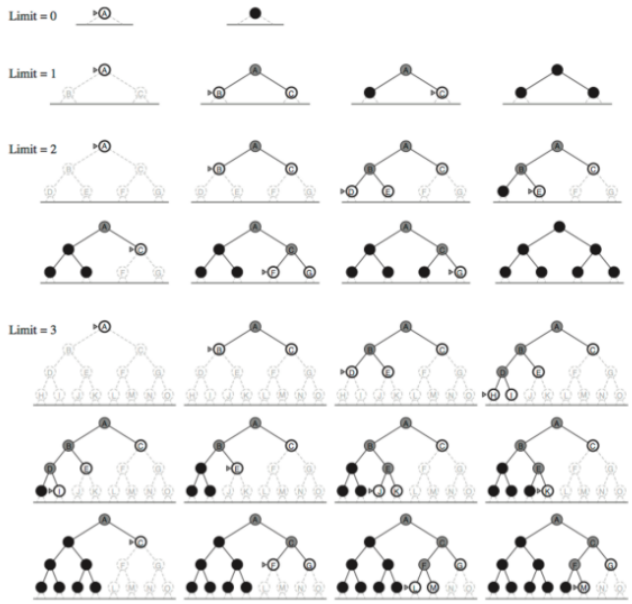


Fig. 3. Iterative deepening search example [7]

source/initial state

The Code Snippet for the function is given Under as follows:

```
def get_possible_moves(curr_state):

    row, col = np.array(np.where(curr_state == 0)).reshape(-1)
    possible_moves = []
    if(row > 0):
        next_state = curr_state.copy()
        next_state[row, col], next_state[row-1, col] = next_state[row-1, col], next_state[row, col]
        possible_moves.append(next_state)
    if(row < 2):
        next_state = curr_state.copy()
        next_state[row, col], next_state[row+1, col] = next_state[row+1, col], next_state[row, col]
        possible_moves.append(next_state)
    if(col > 0):
        next_state = curr_state.copy()
        next_state[row, col], next_state[row, col-1] = next_state[row, col-1], next_state[row, col]
        possible_moves.append(next_state)
    if(col < 2):
        next_state = curr_state.copy()
        next_state[row, col], next_state[row, col+1] = next_state[row, col+1], next_state[row, col]
        possible_moves.append(next_state)
    return possible_moves
```

Fig. 4. Function to backtrack

E. Part E

Generate Puzzle-8 instances with the goal state at depth “d”.

The Function generating these instances is ”GeneralInstances” and the snippet of output is as follows:

F. Part F

Prepare a table indicating the memory and time requirements to solve Puzzle-8 instances (depth “d”) using your graph search agent.
For tracking the time and memory there are packages available in python and we have used memory_profile.The three tables generated are given below and they are made considering the 2 different hueristic function and without any hueristic function.

```
CURR_STATE = generate_instance(GOAL_STATE, 8)
print(CURR_STATE)
```

```
[[2 8 3]
 [1 4 5]
 [0 7 6]]
```

Fig. 5. Function to backtrack

Depth	Time (sec)	Memory (KiB)
2	0.008	52668
4	0.063	52780
8	0.661	52968
16	163.2	61904
32	865.7	72660

Using Manhattan Distance Heuristic

III. Week 3 Lab Assignment 3

Learning Objective: Non-deterministic Search — Simulated Annealing For problems with large search spaces, randomized search becomes a meaningful option given partial/full-information about the domain.

Problem Statement: Travelling Salesman Problem (TSP) is a hard problem, and is simple to state. Given a graph in which the nodes are locations of cities, and edges are labelled with the cost of travelling between cities, find a cycle containing each city exactly once, such that the total cost of the tour is as low as possible.

The entire code for week 3 has been coded in Jupyter notebook and you can find the link for that code here. Code Repository Link

Here we are visiting 25 random nodes/points.

Depth	Time (sec)	Memory (KiB)
2	0.015	52756
4	0.144	52680
8	0.615	52740
16	174.1	61120
32	3586	71270

Using Misplaced tiles Heuristic

Depth	Time (sec)	Memory (KiB)
2	0.016	52936
4	0.136	52732
8	1.194	53052
16	208.6	60848

Without using any Heuristic

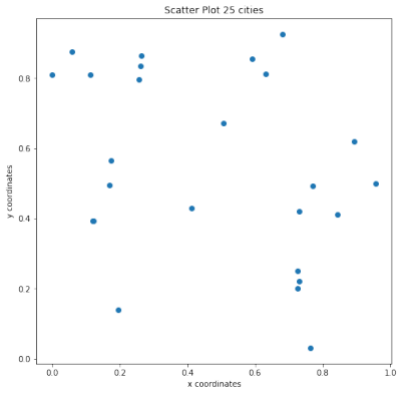


Fig. 6. Scatter plot for 25 nodes

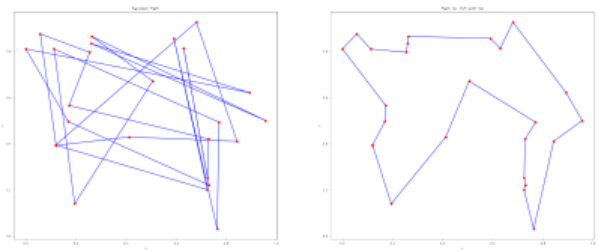


Fig. 7. Comparison between the routes of 25 nodes

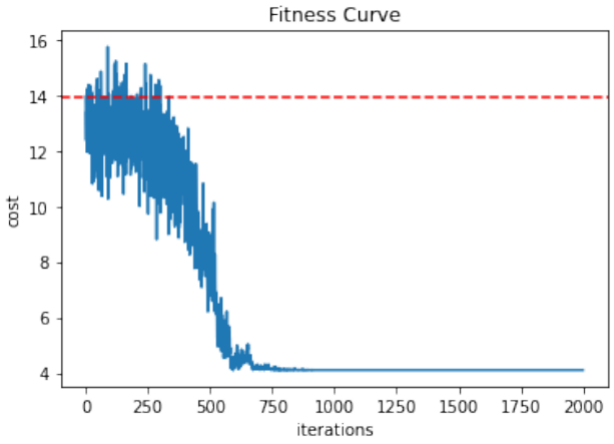


Fig. 8. Fitness curve for 25 nodes

The second plot graph in fig 7 is showing the optimal path to returning to the starting point covering all the nodes using simulated annealing to reduce the cost.

Fig 8 shows how simulated annealing reduces the cost over each iteration. The fitness curve shows the behavior of the cost w.r.t to the number of iterations we are running to obtain the optimal path. Initially the cost is higher than the random path cost but it significantly reduces over successive iterations and as soon as the temperature is low it becomes harder to accept the worst solution cost, there the cost moves towards optimal cost with the decrease in temperature, after some iterations the curve becomes stable - so we can conclude that not many changes are happening and the cost that we are getting is the optimal cost.

A. Part A

For the state of Rajasthan, find out at least twenty important tourist locations. Suppose your relatives are about to visit you next week. Use Simulated Annealing to plan a cost effective tour of Rajasthan. It is reasonable to assume that the cost of traveling between two locations is proportional to the distance between them.

We have selected 25 locations for us to visit. Now we will calculate the euclidean distance for all the pair of coordinates for all the locations. After that we will plot a random route connecting all the nodes and using simulated annealing we will find the optimal cost to cover all the locations/nodes shown in fig 10. [4]

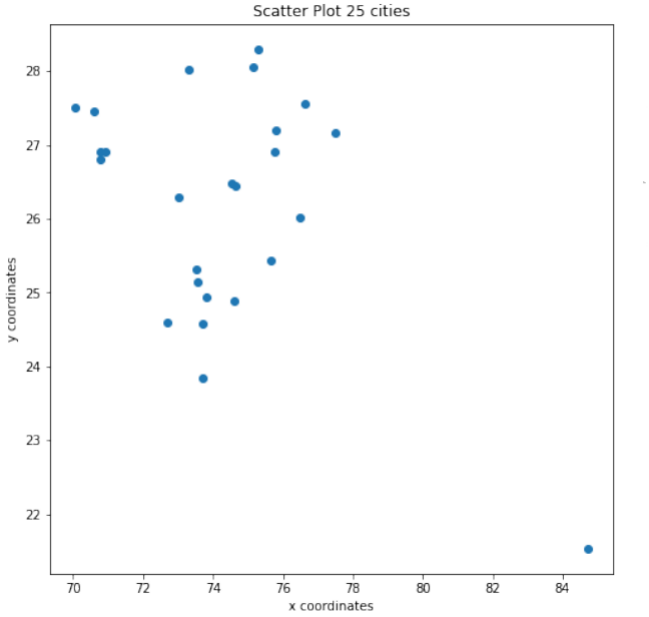


Fig. 9. Scatter plot for 25 locations in Rajasthan

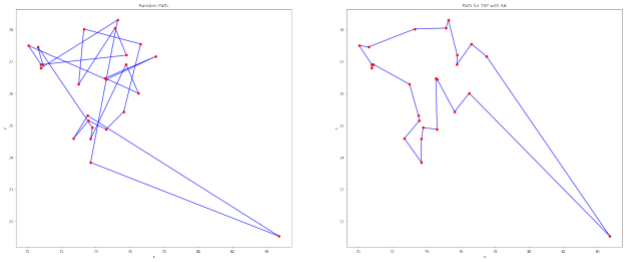


Fig. 10. Optimal path for 25 locations in Rajasthan

B. Part B

VLSI: Dataset Attempt at least five problems from the above list and compare your result

Here we will be doing the same procedure to find the optimal route as we did in the Rajasthan problem in the datasets from VLSI viz. 131 points, 237 points, 343 points, 379 points, and 380 points.

1) XQF131 - 131 points: –

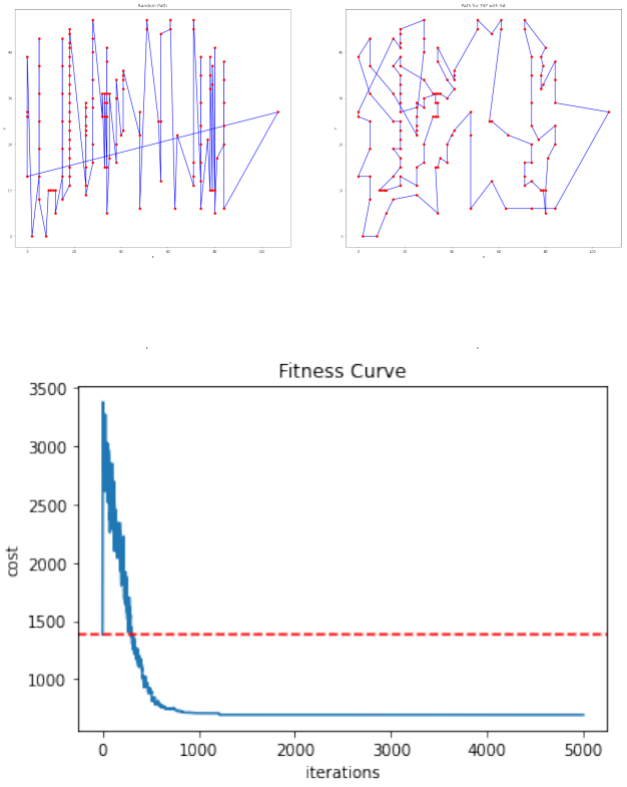
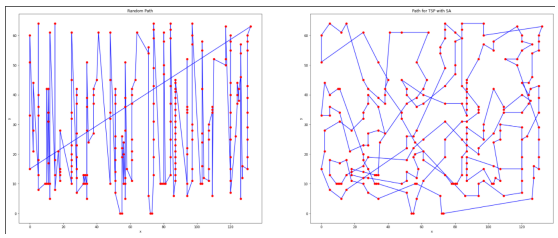


Fig. 11. Plots for 131 points

2) XQF237 - 237 points: -



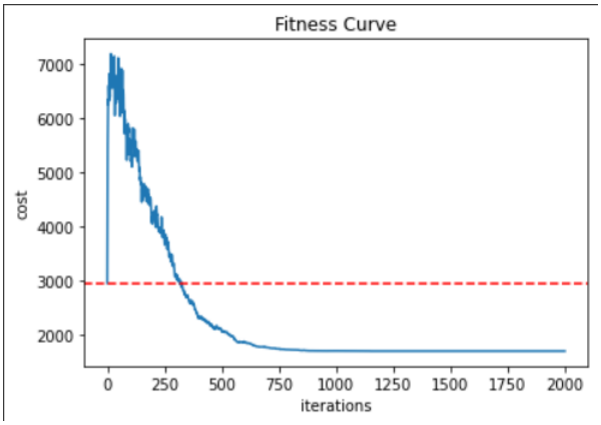


Fig. 12. Plot for 237 points

3) PMA343 - 343 points: –

4) PKA379 - 379 points: –

5) BLC380 - 380 points: –

C. Comparing Results

In fig 16 we can see that the final cost that we are getting i.e the optimal cost is smaller than the cost that we get when we use a random route. When we compare our results with the ones given in the VLSI datasets we find that our calculated final cost is comparable with the optimal cost of VLSI.

We can also further decrease the cost by increasing

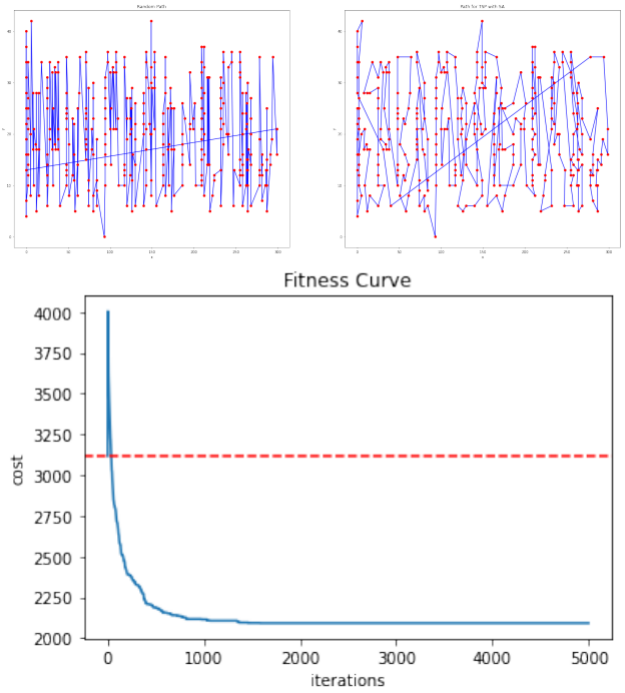


Fig. 13. Plot for 343 points

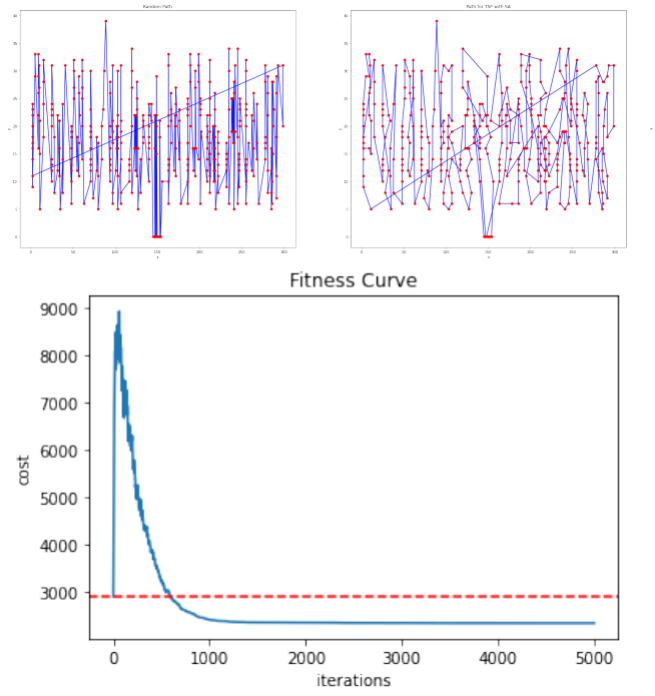


Fig. 14. Plot for 379 points

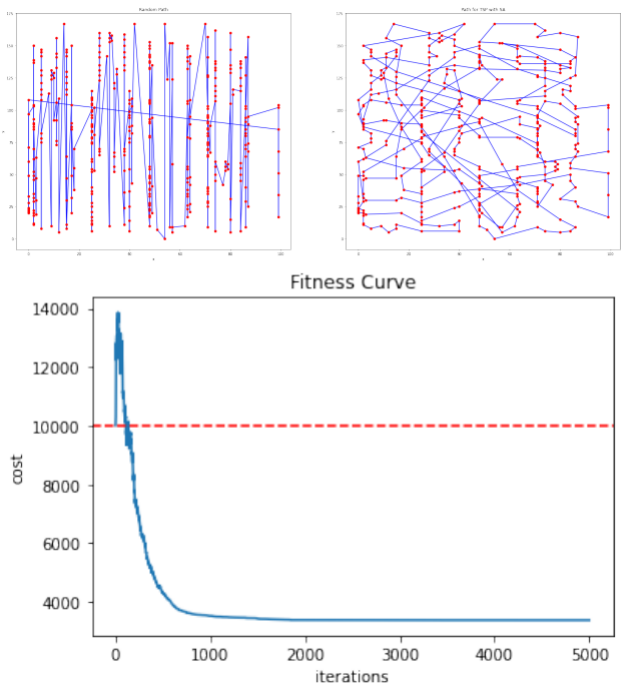


Fig. 15. Plot for 380 points

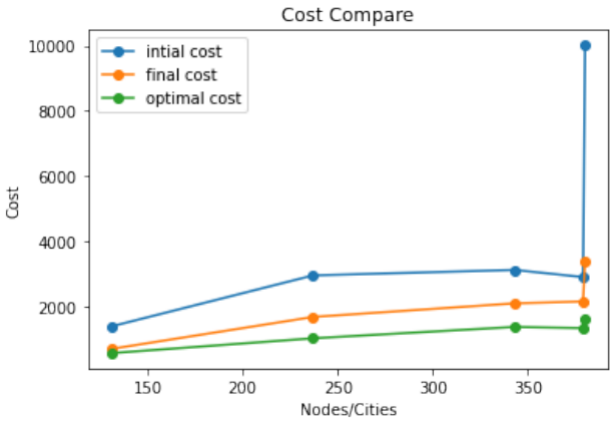


Fig. 16. Plot for 380 points

the iterations and using heuristic functions to solve the traveling salesman problem.

Points	Initial Cost	Final Cost	Optimal Cost
131	1383.916	693.312	564
237	2949.579	1673.994	1019
343	3117.179	2091.522	1368
379	2898.213	2148.960	1332
380	10013.540	3378.900	1621

IV. WEEK 5 LAB ASSIGNMENT 4

Learning Objective: Game Playing Agent — Minimax — Alpha-Beta Pruning

A. Part 1

What is the size of the game tree for Noughts and Crosses? Sketch the game tree.

Considering the Depths at every level we have
At Depth 1 = 9 Possibilities
At Depth 2 = 9*8 Possibilities
At Depth 3 = 9*8*7 Possibilities
So we have total number of states available are almost equal to 10^6
The Graph tree can be given as Follows:

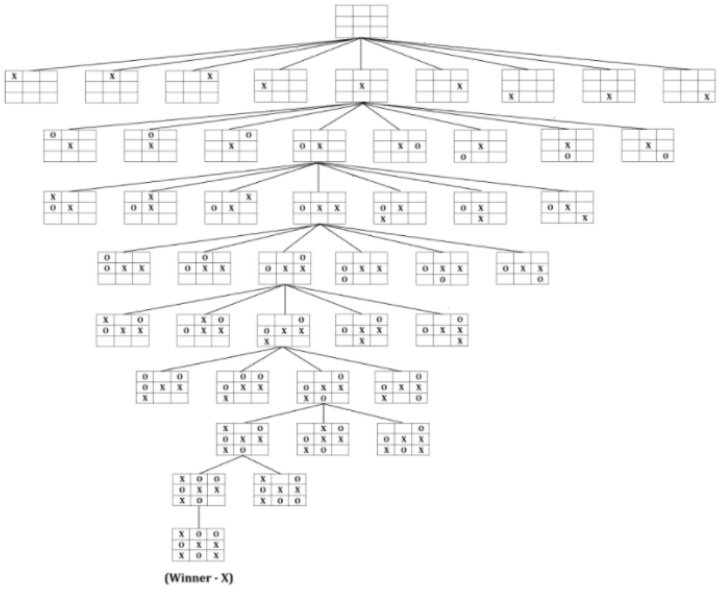


Fig. 17. Graph of the tree

B. Part 2

Read about the game of Nim (a player left with no move losing the game). For the initial configuration of the game with three piles of objects as shown in Figure, show that regardless of the strategy of player-1, player-2 will always win. Try to explain the reason with the MINIMAX value backup argument on the game tree
The initial Configuration of the nim game given to us is:
To show that Player 2 always win we use the minimax algorithm and the player is allowed to take from one column only so if the player moves such that the XOR of all the three towers become zero [9] so if the player 1 proceeds like this player 2 would always win as shown in the figure:

Fig. 18. Graph of the tree

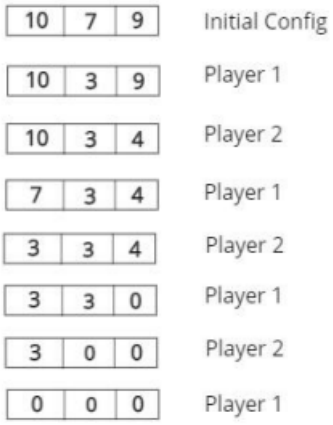


Fig. 19. Player taking turns in Nim

C. Part 3

Implement MINIMAX and alpha-beta pruning agents. Report on number of evaluated nodes for Noughts and Crosses game tree. [5]

The given code are hereCode Repository Link

The output snippet is given as shown in figure 20:

D. Part 4

Using recurrence relation show that under perfect ordering of leaf nodes, the alpha-beta pruning time complexity is O(bm/2), where b is the effective branching factor and m is the depth of the tree.
Let m be the depth of the tree and b be the effective branching factor.
T(m) be the minimum number of states to be traversed to find the exact value of the current state, and
K(m) be the minimum number of states to be traversed to find the bound on the current state.

```
print(pointABP,"\n Evaluated Nodes = ",
```

```
Minimax :
0 [['X' 'O' 'X']
 ['X' 'O' 'O']
 ['O' 'X' 'X']]
 Evaluated Nodes =  549946
Alpha Beta Pruining :
0
 Evaluated Nodes =  18297
```

Fig. 20. Output of Minimax and Alpha beta-pruning

We get the equation as:

$$T(m) = T(m-1) + (b-1)K(m-1) \tag{1}$$

T(m-1) is to traverse to child node and find the exact value, and (b-1)K(m-1) is to find min/max bound for the current depth of the tree as we have to find using recursion.

In best case, we know that

$$T(0) = K(0) = 1 \tag{2}$$

exact value of one child is

$$K(m) = T(m-1) \tag{3}$$

From the given relation we have:

$$T(m) = T(m-1) + (b-1)K(m-1) \tag{4}$$

$$T(m-1) = T(m-2) + (b-1)K(m-2) \tag{5}$$

so on we get

$$T(1) = T(0) + (b-1)K(0) = 1 + b - 1 = b(10) \tag{6}$$

using equation 4 and 5 we get

$$T(m) = T(m-2) + (b-1)K(m-2) + (b-1)K(m-1) \tag{7}$$

$$\begin{aligned} T(m) &= T(m-3) + (b-1)K(m-3) \\ &+ (b-1)K(m-2) + (b-1)K(m-1) \end{aligned} \tag{8}$$

Using 3 and 8 we get

$$\begin{aligned} T(m) &= T(m-2) + (b-1)T(m-3) + \\ (b-1)T(m-2) &= b(T(m-2)) + (b-1)T(m-3) \end{aligned} \tag{9}$$

Now we can clearly say that

$$T(m-2) > T(m-3) \tag{10}$$

So we can predict

$$T(m) < (2b-1)T(m-2) \tag{11}$$

Considering large values of b

$$T(m) < 2bT(m-2) \tag{12}$$

From these we can conclude from these that the effective branching factor is less than $\sqrt{2b}$

$TimeComplexity = O(b^{m/2})$

V. WEEK 6 LAB ASSIGNMENT 5

Learning Objective: Understand the graphical models for inference under uncertainty, build Bayesian Network in R, Learn the structure and CPTs from Data, naive Bayes classification with dependency between features.

Problem Statement: A table containing grades earned by students in respective courses is made available to you in (codes folder) 2020_bn_nb_data.txt.

A. Part A

Here let us consider the grades earned in each of the courses as random variable and learn the dependencies between the courses.
The dependencies can be learned using 'bnlearn package in R as well as using the function called hill climbing greedy search.

Hill climbing is the score-based algorithm. As our dataset is categorical we learn the scores of discrete Bayesian Network - k2 and bic - and compare both of them. [3]

1) Using k2 score: There are in total 7 arcs in fig 21 and the model string is showing this dependency: '[IT161] [IT101—IT161] [MA101—IT101] [HS101—IT101] [EC100—MA101] [PH160—HS10] [EC160—EC100] [PH100—EC100]'

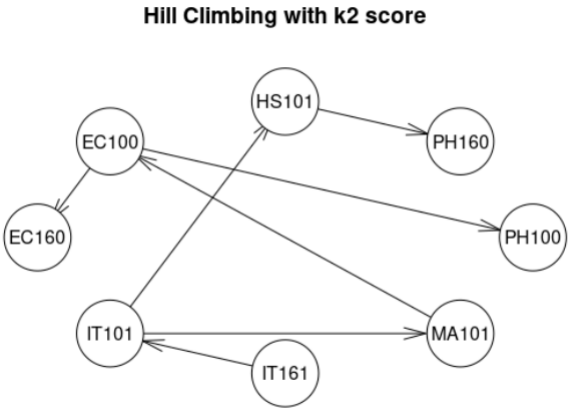


Fig. 21. Hill Climbing Bayesian Network using K2 score

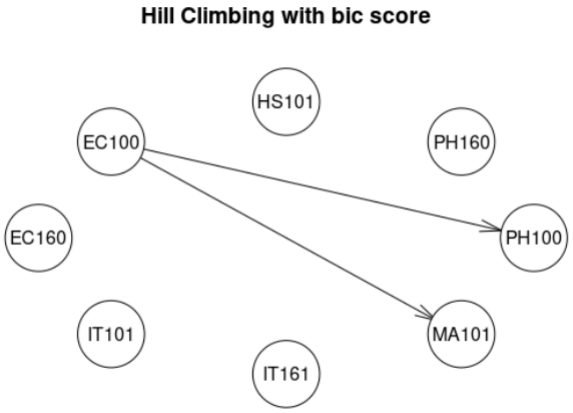


Fig. 22. Hill Climbing Bayesian Network using bic score

2) Using bic score: There are only two arcs in fig 22 and the model string showing this dependency is '[EC100] [EC160] [IT101] [IT161] [PH160] [HS101] [MA101—EC100] [PH100—EC100]'

Since we are interested to find the dependency of the different grades, we can see that the k2 score gives a better idea of how the scores are dependent on each other, furthermore k2 is generally considered a good choice for large datasets. So, for the further parts, we'll only use the network learned using the k2 score. [8]

B. Part 2

Using the data, learn the CPTs for each course node.

According to the network learned using the k2 score, we plot the conditional probabilities.

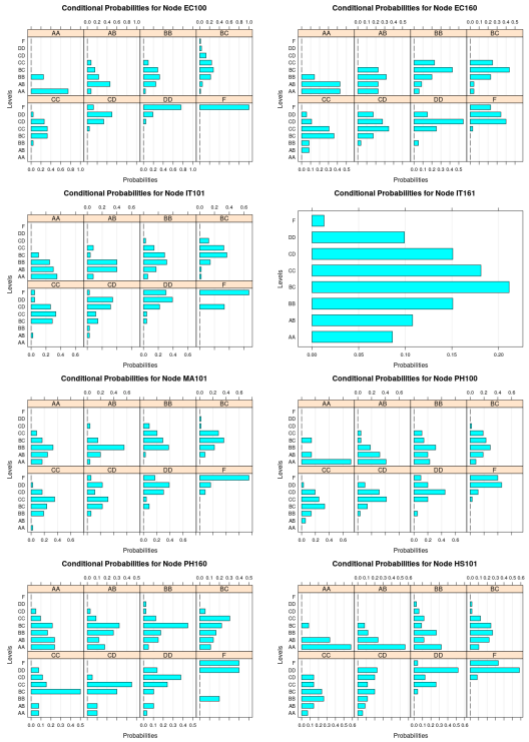


Fig. 23. Conditional probability distributions made form the CPTs of the learned data

C. Part 3

What grade will a student get in PH100 if he earns DD in EC100, CC in IT101 and CD in MA101.

We use the "cpdist" function of the "bnlearn" package in R to get distribution of grades of PH100 when the student has earned DD in EC100, CC in IT101 and CD in MA101.That distribution graph is shown in fig 23 and the distribution table shown below.

D. Part 4

The last column suggests if the student is eligible for internship or not. Now we take 70 percent for training and build a naive Bayes classifier, which will take in students performance and return if the student is eligible for internship or not. Now test the model for the remaining 30 percent. Repeat the experiment for 20 iterations.

So now we split the dataset of 231 into training of 162 and testing of 69 samples.

We use the NBC using the function 'nb' from the package 'bnclassify' in R tolearn the naive Bayes network structure and then the function 'lp' to learn the parameters. Here, we

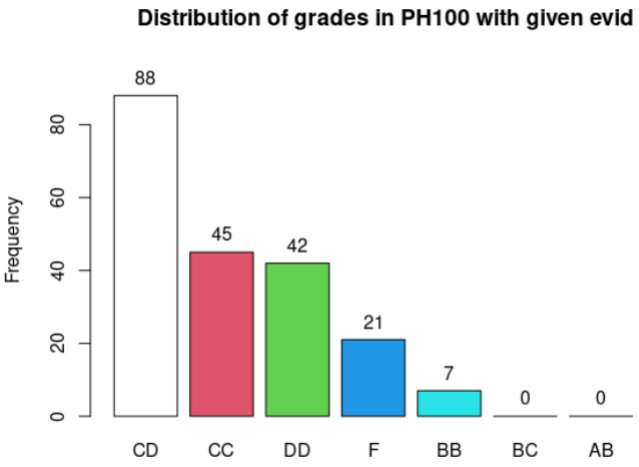


Fig. 24. Probability distribution of getting a grade in PH100 as per the given evidence

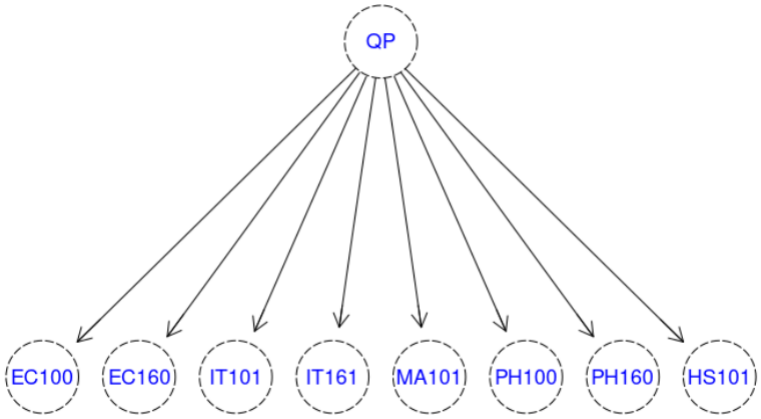


Fig. 25. Naive Bayes Classifier for independent data

Grade	Frequency	Percent	Cum. percent
CD	101	43.5	43.5
DD	49	21.1	64.7
CC	47	20.3	84.9
F	23	9.9	94.8
BB	12	5.2	100
BC	0	0	0
AB	0	0	0
AA	0	0	0
Total	232	100	100

TABLE VI: Frequency distribution table

do not assume any dependency between the grade noc therefore classifier learns assuming the data to be independe

This network model learns on the training dataset that split. The accuracy has been shown in the table below.

Experiment no.	Accuracy
1	0.9130435
2	0.942029
3	0.9565217
4	0.942029
5	0.9565217
6	0.9710145
7	0.9710145
8	0.9855072
9	0.9855072
10	0.9565217
11	0.9710145
12	0.9710145
13	0.9714286
14	0.9855072
15	0.942029
16	0.9710145
17	0.942029
18	0.942029
19	0.9857143
20	0.9857143

TABLE VII: Accuracy on test dataset on Naive Bayes classifier considering independent data

E. Part 5

Repeat part 4, just considering the grades earned dependent.

To learn the features we use 'tan-chow' function. The classifier learns on the structure as shown below.

This network learns on training dataset that we split in the earlier part.

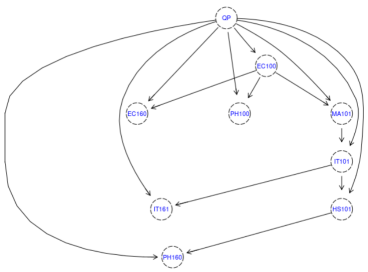


Fig. 26. Naive Bayes Classifier for dependent data

VI. WEEK 6 LAB ASSIGNMENT 5

Learning Objective: To implement Expectation Maximization routine for learning parameters of a Hidden Markov Model, to be able to use the EM framework for deriving algorithms for problems with hidden or partial information. **Code Repository:** The given code are here Code Repository Link

Experiment no.	Accuracy
1	0.9710145
2	0.942029
3	0.9565217
4	0.942029
5	0.9275362
6	0.9565217
7	0.9571429
8	0.9565217
9	0.9857143
10	0.9130435
11	0.9428571
12	0.9714286
13	0.9275362
14	0.9565217
15	0.9142857
16	0.9710145
17	0.9
18	0.9130435
19	0.9565217
20	0.9710145

TABLE VIII: Accuracy on test dataset on Naive Bayes classifier considering independent data

A. Part A

Read through the reference carefully. Implement routines for learning the parameters of HMM given in section 7. In section 8, "A not-so-simple example", an interesting exercise is carried out. Perform a similar experiment on "War and Peace" by Leo Tolstoy. From the reference "A Revealing Introduction to Hid- den Markov Models" [10], we calculated α-pass , β-pass, di-gammas for re-estimating the state transition probability matrix (A), observation probability matrix (B), initial state distribution (π) for Leo Tolstoy's book War and Peace. We re-estimated A,B and based on the observed sequence O, taking initial values for A, B and and calculating a, 3, the di-gammas and log probability for the data i.e. War and Peace. We took 50000 letters from the book after removing all the punctuation and converting the letters to lower case just as given in the example problem in section 8 [16]. We initialized each element of π and A randomly to approximately 1/2. The initial values are.

$$\pi = [0.5130.486] \tag{13}$$

$$A = \begin{bmatrix} 0.47648 & 0.52532 \\ 0.51656 & 0.48344 \end{bmatrix}$$

Each element of B was initialized to approximately 1/27. The precise values in the initial B are given in the Tab IX. After initial iteration,

$$log([P(\mathcal{O}|\lambda)]) = -142533.41283009356$$

After the 100 iterations the model converged to

$$log([P(\mathcal{O}|\lambda)]) = -138404.4971796029$$

So our model has improved significantly after 100 iteration. The model converges after 100 iteration to:

$$\pi = \begin{bmatrix} 0.00000 & 1.00000 \end{bmatrix}$$

$$A = \begin{bmatrix} 0.28438805 & 0.71561195 \\ 0.81183208 & 0.18816792 \end{bmatrix}$$

With final values of transpose of B in the Table given below. By looking at the B matrix we can see that the hidden state contains vowels and consonants and space is counted as a vowel. The first column of initial and final values in Table given below are the vowels and the second column are the consonants.

	Initial		Final	
a	0.03735	0.03909	7.74626608e-02	6.20036245e-02
b	0.03408	0.03537	9.00050395e-10	2.34361673e-02
c	0.03455	0.03537	2.85482586e-08	5.54954482e-02
d	0.03828	0.03909	1.90968101e-10	6.91132175e-02
e	0.03782	0.03583	1.70719479e-01	2.11816156e-02
f	0.03922	0.03630	2.33305198e-12	2.99248707e-02
g	0.03688	0.04048	3.57358519e-07	3.08209307e-02
h	0.03408	0.03537	6.11276932e-02	4.10475218e-02
i	0.03875	0.03816	1.04406415e-01	1.70940624e-02
j	0.04062	0.03909	6.60956491e-26	1.92099741e-03
k	0.03735	0.03490	2.53743574e-04	1.21345926e-02
l	0.03968	0.03723	1.94001259e-02	4.17688047e-02
m	0.03548	0.03537	4.65877545e-12	3.85907034e-02
n	0.03735	0.03909	4.83856571e-02	6.14790535e-02
o	0.04062	0.03397	1.05740124e-01	4.23129392e-05
p	0.03595	0.03397	2.82866053e-02	1.84540755e-02
q	0.03641	0.03816	9.92576058e-19	1.32335377e-03
r	0.03408	0.03676	8.29107989e-06	1.07993337e-01
s	0.04062	0.04048	2.54927739e-03	9.75975025e-02
t	0.03548	0.03443	3.96236489e-05	1.50347802e-01
u	0.03922	0.03537	2.94555063e-02	8.54757059e-03
v	0.04062	0.03955	2.83949667e-19	3.05652032e-02
w	0.03455	0.03816	4.70315572e-25	3.37241767e-02
x	0.03595	0.03723	2.20419809e-03	1.38065996e-02
y	0.03408	0.03769	6.42635319e-04	3.04765034e-02
z	0.03408	0.03955	4.88081324e-27	1.10990961e-03
space	0.03688	0.03397	3.49317577e-01	4.28089789e-08

B. Part B

Ten bent (biased) coins are placed in a box with unknown bias values. A coin is randomly picked from the box and tossed 100 times. A file containing results of five hundred such instances is presented in tabular form with 1 indicating head and 0 indicating tail. Find out the unknown bias values. (2020_ten_bent_coins.csv) To help you, a sample code for two bent coin problem along with data is made available in

the work folder: $two_bent_coins.csvandembentcoinsol.m$

For the above problem, we used Expectation Maximizatior algorithm from the reference material [11] already shared An expectation-maximization (EM) algorithm is an iterative method to find (local) maximum likelihood or maximum a posteriori (MAP) estimates of parameters in statistical models where the model depends on unobserved latent variables [12]

In our case, the latent variables are z, the coin types. We have no knowledge of the coins but the final outcome o 500 trials. We used EM to estimate the probabilities for each possible completion of the missing data, using the curren parameters .We took reference from Karl Rosaen's solutior for 2 bent coins problem [13].

C. Part C

A point set with real values is given in 2020_em_clustering.csv Considering that there are two clusters, use EM to group together points belonging to the same cluster.

Try and argue that k-means is an EM algorithm. We used Expectation Maximization algorithm for grouping the points belonging to the same cluster. We also used k-means for comparing with EM algorithm clustering.

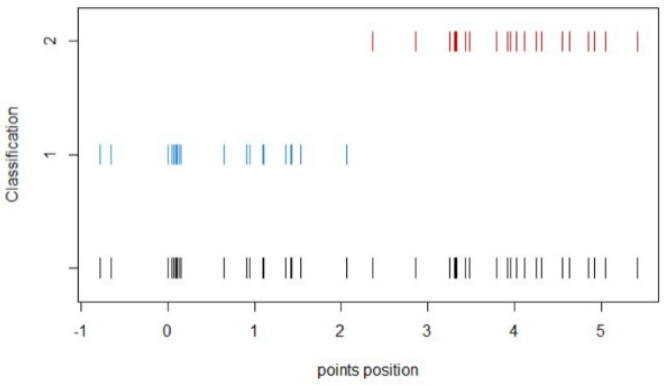


Fig. 27. EM Clustering

In above Figure, the lines depicted with black color are the points from the given dataset and the lines depicted with blue and red respectively are the two clusters formed by the EM algorithm from the dataset. The model has grouped the points belonging to the same cluster.

From the above below, we can infer that k-means and EM algorithm results are same i.e. 19 points lie in the first cluster and 21 points lie in the second cluster out 40 given points.

k - means uses hard-clustering which means that a point either belongs to the cluster or it does not belong the cluster. EM algorithm uses soft-clustering technique to assign points

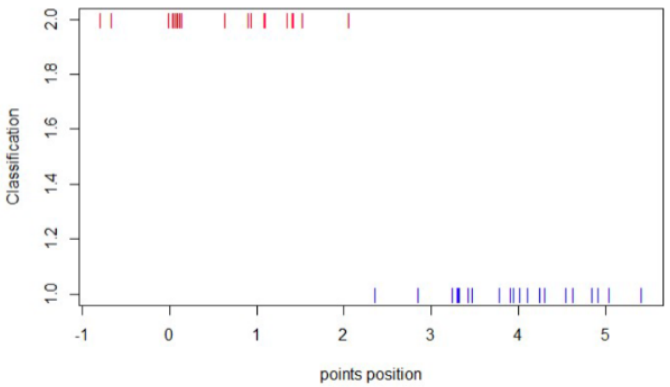


Fig. 28. k-means Clustering

to a cluster, it gives the probability that a particular point belongs to a cluster. In this assignment, from the given dataset both the methods EM algorithm and k - means yield the same result because the dataset was in such a way that the probability of a point belonging to a cluster was sufficient to assign the point to that cluster therefore the soft-clustering and hard-clustering yield the same result.

VII. Week 8 Lab Assignment 7

Learning Outcome To model the low level image processing tasks in the framework of Markov Random Field and Conditional Random Field. To understand the working of Hopfield network and use it for solving some interesting combinatorial problems

A. Part A

Many low level vision and image processing problems are posed as minimization of energy function defined over a rectangular grid of pixels. We have seen one such problem, image segmentation, in class. The objective of image denoising is to recover an original image from a given noisy image, sometimes with missing pixels also. MRF models denoising as a probabilistic inference task. Since we are conditioning the original pixel intensities with respect to the observed noisy pixel intensities, it usually is referred to as a conditional Markov random field. Refer to (3) above. It describes the energy function based on data and prior (smoothness). Use quadratic potentials for both singleton and pairwise potentials. Assume that there are no missing pixels. Cameraman is a standard test image for benchmarking denoising algorithms. Add varying amounts of Gaussian noise to the image for testing the MRF based denoising approach. Since the energy function is quadratic, it is possible to find the minima by simple gradient descent. If the image size is small (100x100) you may use any iterative method for solving the system of linear equations that you arrive at by equating the gradient to zero.

We started with first importing the image of the 'cameraman', which is a standard image for benchmarking denoising

algorithms, as it is very dynamic in the grayscale pixel range. [14].This is a 512x512 grayscale image. We normalize the pixel values to be between 0 and 1, by dividing all values by 255, and then 'binarizing' it for the Markov Random Field by converting all the normalized pixel values below 0.5 to 0 and the rest to 1, as shown in Fig 26.

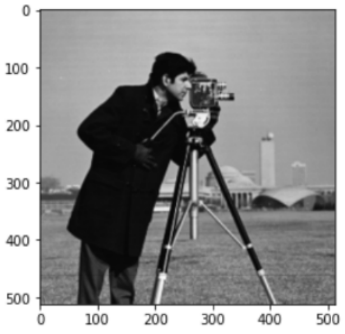
We, then introduce noise to this 'binarized' image. In order to test the capability, we add varying levels of noise, from 5% to 25% of the pixel values. The varying levels of noises are shown in the given figure

Markov random fields use a quadratic potential function to measure the energy potential of the image when changing a particular pixel, with respect to the neighbouring pixels. The quadratic potential function is given by:

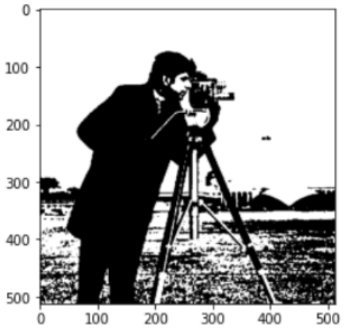
where v is the smooth ID signal, u is the IID and E is the

$$E(u) = \sum_{n=1}^{N}(u_n - v_n)^2 + \lambda \sum_{n=1}^{N-1}(u_{n+1} - u_n)^2$$

energy function. This is because for most cases, the values around a pixel are close to the pixel value. [15], [16]. We use the value of the constant lambdas as -100, while computing the quadratic potential function. We ran the algorithm for 5*512*512 = 1310720 iterations.



(a) Original Cameraman Image



(b) 'Binarized' Cameraman Image

Fig. 29. Cameraman Photographs

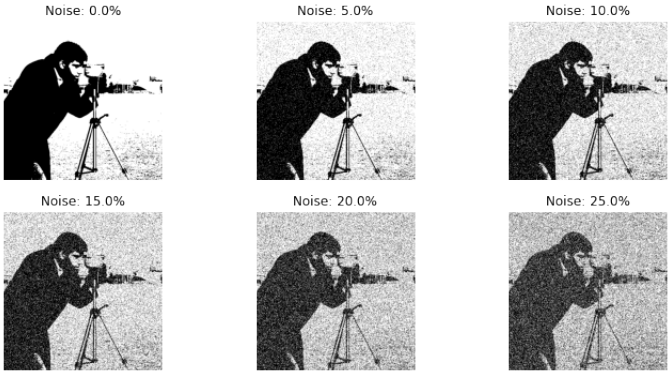


Fig. 30. Images with various level of noises

The table is as follows:

% noise Level	% pixels denoised
5	4.70
10	6.99
15	9.30
20	11.67
25	14.02

B. Part B

For the sample code hopfield.m supplied in the lab-work folder, find out the amount of error (in bits) tolerable for each of the stored patterns.

For this task, we converted the hopfield.m MATLAB codes to Python so that we could do it in the same Notebook as the other parts. The original images looked as shown in fig. 30. The network was trained using Hebb's rule, as in the file and then they were noised by changing some random pixel values. At max, 15 pixel values were noised. We can see that suprisingly, the network can correct upto max 8 errors. The results are show in fig 33.

C. Part C

Solve a TSP (travelling salesman problem) of 10 cities with a Hopfield network. How many weights do you need for the network?

This is the usual famous NP-hard problem of Travelling Salesman, that is done using the a Hop field Networks. Since in a Hop field Network, each node is connected to each other node, we needed a total of 10x10= 100 weights. We first generate 10 cities randomly, as shown in fig 31. And then let the hop field network predict an optimal least path cost. The path that we got was as shown in fig 35.

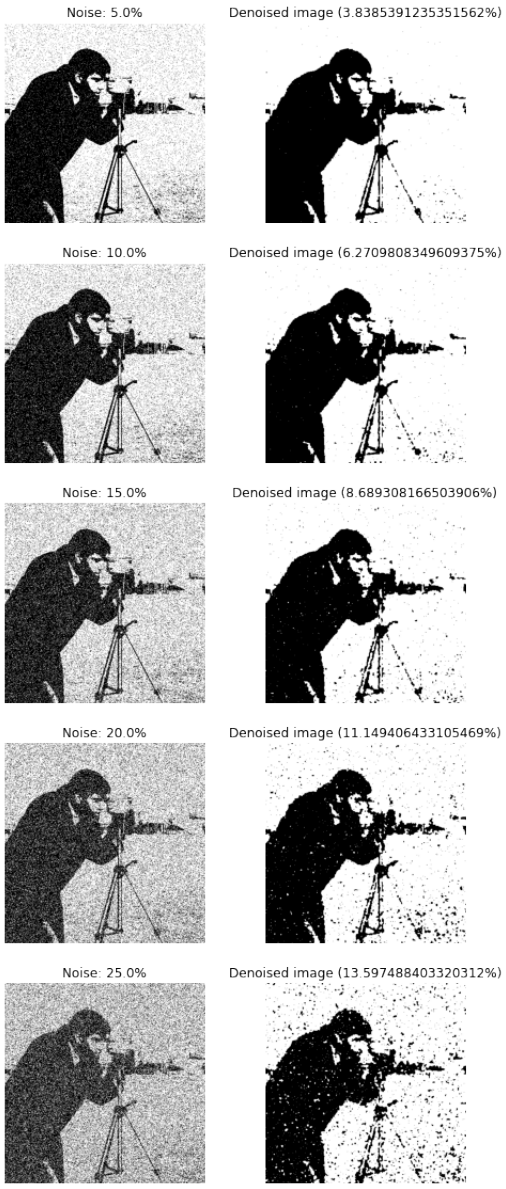


Fig. 31. Denoising Image using MRF

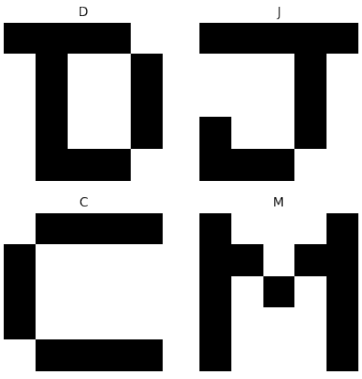


Fig. 32. Original Letters of Hopfield

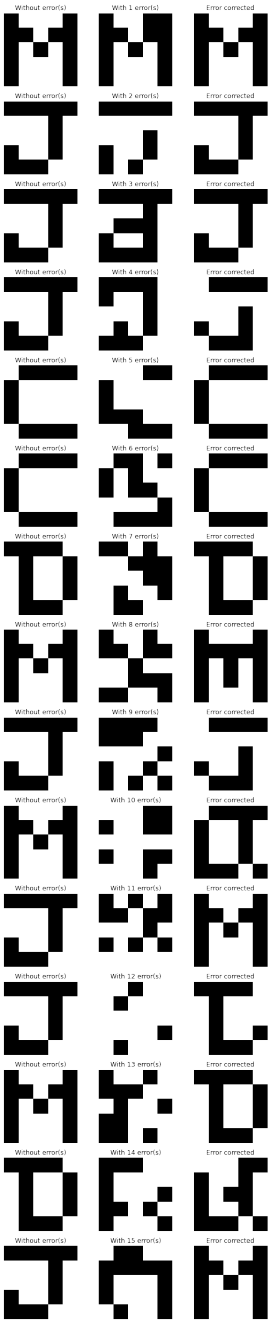


Fig. 33. Original Noised and corrected Image

VIII. Week 10 Lab Assignment 8

Learning Objective Basics of data structure needed for state-space search tasks and use of random numbers required for MDP and RL.

Problem Statement: Read the reference on MENACE by Michie and check for its implementations. Pick the one that you like the most and go through the code carefully. Highlight the parts that you feel are crucial. If possible, try to code the.MENACE in any programming language of your

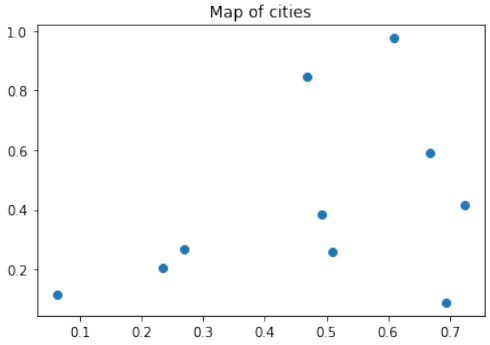


Fig. 34. Cities

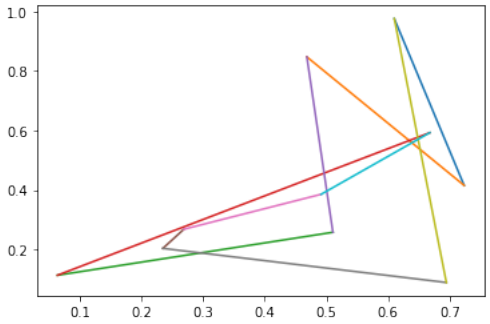


Fig. 35. Shortest Path

liking.

A. What is Menace ?

MENACE stands for Machine Educable Noughts and Crosses Engine [17]. It was originally described by Donald Michie, who used matchboxes to record each game he played against this algorithm. This provides an adequate conceptual basis for a trial-and-error learning device, provided that the total number of choice-points which can be encountered is small enough for them to be individually listed. Michie's aim was to prove that a computer could "learn" from failure and success to become good at a task.

B. Why Matchbox Machine?

Matchboxes were used because each box contained an assortment of variously coloured beads. The different colours correspond to the different unoccupied squares to which moves could be made.

C. How does Menace learn?

A loss is punished by a removing the beads that were chosen from the boxes. This means that MENACE will be less likely to pick the same colors again and has learned. A win is rewarded with three beads of the chosen color which is added to each box, reinforcing the MENACE to make the same move again. If a game is a draw, one bead is added to each box.

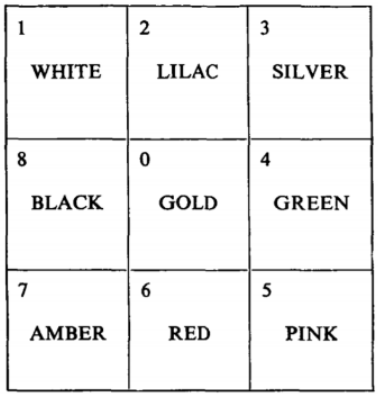


Fig. 36. Color Code of Matchbox

From the figure [37], we can see the number of beads present inside a box on any given stage. Removing one bead from each box after losing means that later these moves are less likely to be picked and this helps MENACE learn more quickly, as the later moves are more likely to have led to the loss.

It is possible that after few games some boxes may end up empty. If one of these boxes is to be used, then MENACE resigns. When playing against skilled players, it is possible that the first move box runs out of beads. In this case, MENACE should be reset with more beads in the earlier boxes to give it more time to learn before it starts resigning.[18]

STAGE OF PLAY	NUMBER OF TIMES EACH COLOUR IS REPLICATED
1	4
3	3
5	2
7	1

Fig. 37. Variation of the number of color replicates

D. Result

The result is present in the code link.

IX. Week 11 Lab Assignment 9

Learning Objective Understanding Exploitation Exploration in simple n-arm bandit reinforcement learning

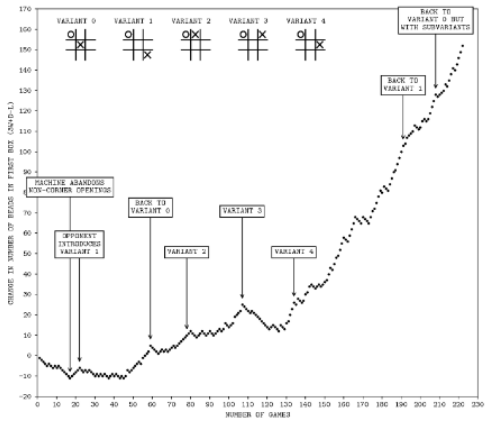


Fig. 38. The progress of MENACE'S maiden tournament against a huma opponent. The line of dots drops one level for a defeat, rises one level for draw and rises three levels for a victory.

task, epsilon-greedy algorithm

A. Part A

Consider a binary bandit with two rewards 1-success, 0 failure. The bandit returns 1 or 0 for the action that you select i.e. 1 or 2. The rewards are stochastic (but stationary). Use a epsilon-greedy algorithm discussed in class and decide upo the action to take for maximizing the expected reward. Ther are two binary bandits named binaryBanditA.m and binar BanditB.m are waiting for you.

After using both the rewards(function) binary BanditA an binary BanditB here is what we observe :

Although the probability reward was low in bandit A the expected reward was 0 now as the number of trials increases the expected reward also increase but this is not the same with bandit B.In this bandit initially the expected reward was 1 then it step down to be around 0$\dot{8}$ and become constant.

B. Part B

Develop a 10-armed bandit in which all ten mean-rewards start out equal and then take independent random walks (by adding a normally distributed increment with mean zero and standard deviation 0.01 to all mean-rewards on each time step).

This is the case of 10 arm-bandit where the mean-rewards are initially taken as a constant valued array initialised by 1. We performed exploration and exploitation based on the Epsilon Greedy Algorithm. A random number between 0 and 1 is generated and if it comes out to be greater than epsilon, we perform exploitation, which is based on the prior knowledge otherwise exploration. For every iteration an array of ten values is generated such that they are normally distributed with mean value zero and standard deviation of 0.01. This array is added to the mean array and this updated array is used every time. For every action a reward is given from this updated mean-rewards array. The rewards given in this case are non-stationary.

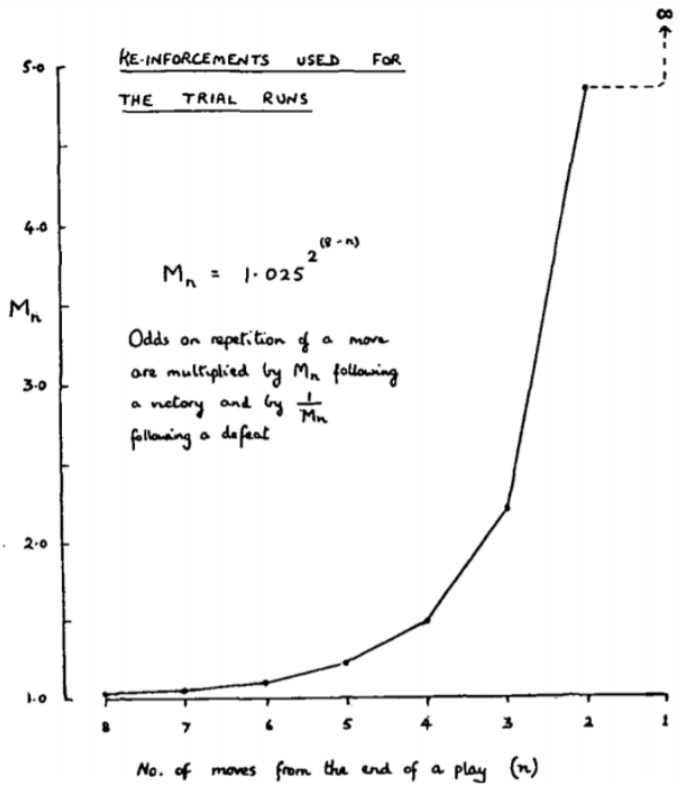


Fig. 39. Reinforcement multipliers in trial runs

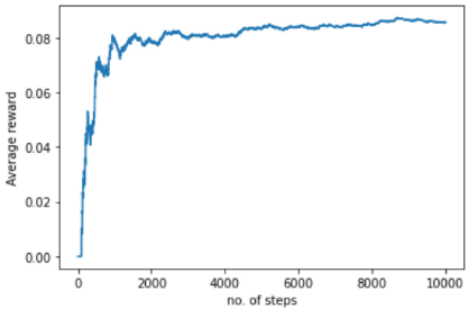


Fig. 40. Expected Reward in Bandit A

Here we observe that as initially all the rewards were equal so that we get a constant reward of say like 1 but as the steps increases, it affects the rewarding policy of every action and then starts increasing at very high non-zero rate.

C. Part C

The 10-armed bandit that you developed (banditnonstat) is difficult to crack with standard epsilon-greedy algorithm since the rewards are non-stationary. We did discuss about how to track non-stationary rewards in class. Write modified epsilon- greedy agent and show whether it is able to latch onto correct actions or not.

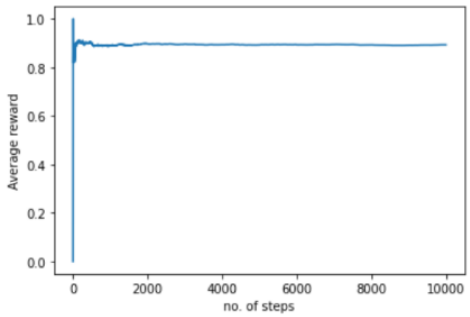


Fig. 41. Expected Reward in Bandit B

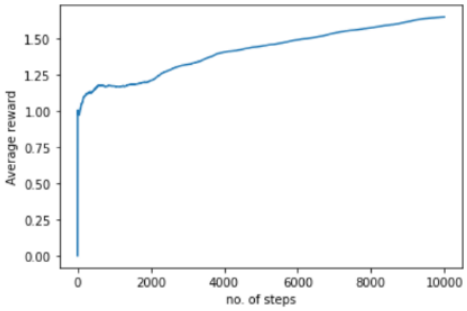


Fig. 42. Expected Rewards in Non stationary rewards(averaging)

This is same as problem 2 except the calculation of action rewards. In this case, instead of using averaging method to update estimation of action reward, we are assigning more weights to the current reward earned by using alpha parameter having value 0.7

Here we can see that as compared to problem 2, in which the expected rewards were low, we saw that when instead of averaging the profit percentage of state when we gave higher weightage to the to the current value at every step (a normally distributed array is added to the rewards array), the expected rewards we get were much higher as in the case 2 and the slope of the graph was too steeper when compared with case 2.

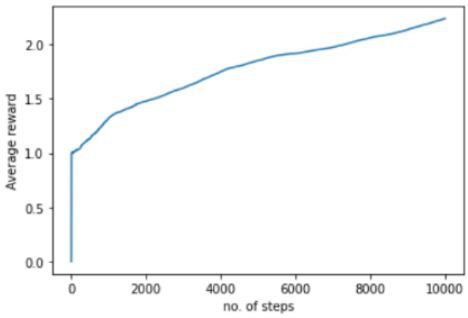


Fig. 43. Expected Rewards in Non stationary rewards(Non averaging)

CONCLUSION

As we have demonstrated in the above problems we have successfully solved 8 puzzle problem,Travelling salesman problem (tsp), Noughts and Crosses Game, Nim game and understood the Bayesian graphical Models to build network Graphs.

The importance of Heuristic plays an important role in solving the real life problems as as solving the problems in considerable amount of time as compared without Heuristic functions saving time as well as memory for big dataset problems.

In the Travelling Salesman problem we learned to use simulated annealing to find the optimal path or the sub optimal path.

From our observation in the minimax algo we find that alpha-beta pruning is not a different algorithm than mini-max it is just a speed up process which does not evaluate approximate values instead evaluates to perfectly same values.

Lastly we explored the Bayesian network and were able to model it using grades data-set, we then calculated Conditional Probability Tables (CPTs) for them and saw how they were dependent.

From the Markov Random Field and Hopfield networks, we learned how to construct learning machines using Markov Decision Processes. Hopfield Network showed how we can use a simple collection of neurons to construct a very robust network, that can resist ignore large noises and still predict a correct output and lastly, we saw how they can be used in Travelling Salesman Problem to get an optimal path.

In Menace we have learnt that how the machine learns using Reinforcement Learning but it requires various Iteration.

From the HMM and EM algorithm problem, we applied a HMM model to the 50,000 letters observations including space and all lowercase characters. After about 100 iterations

we observed that the B matrix tells us about two distinct categories of characters. Upon closely observing, we find that one set contains consonants while the other contains vowels. For the second part we apply EM algorithm to the 10 bent coin problem (similar to well known 2 bent coin example) to find the latent parameters and estimate the hidden biases for the coins. In the third part we learnt about soft and hard clustering and EM algorithm clustering for given dataset, we also learnt about the k-means and how k-means compares to EM algorithm.

For the n arm bandit we learned how the problem works and the epsillon greedy algorithm,we saw that how that algorithm can be modelled for both the stationary environment case and the non stationary case just be simply giving more weightage to the current step.

ACKNOWLEDGMENT

We would like to thank out Prof. Pratik Shah and our TA Sir Prashant Dhameja as well as my colleagues.it is due to the lectures in AI that we were able to understand the problem as well as able to solve it.
We would also like to thank the sources as well the online materials that we have used.

REFERENCES

[1] Russell, S. and Norvig, P., 2002. Artificial intelligence: a modern approach.
[2] 8 puzzle problem -benchpartner.com
[3] Shah Pratik, An Elementary Introduction to Graphical Models February 2021
[4] Zhou, Ai-Hua, Traveling-salesman-problem algorithm based on simulated annealing and gene-expression programming. Information 10.1 2019.
[5] Best-Case Analysis of Alpha-Beta Pruning. http://www.cs.utsa.edu/ bylander/cs5233/a-b-analysis.pdf.
[6] Khemani, D., 2020. Artificial Intelligence: The Big Picture. Resonance: Journal of Science Education, 25(1).
[7] iddfs - stackoverflow.
[8] Bojan Mihaljevic, ´ Bayesian networks with R November 2018.
[9] Marianne Freiberger, Play to win with Nim.July 21, 2014.
[10] Stamp, Mark, A revealing introduction to hidden Markov models, 2004.
[11] What is the expectation maximization maximization algorithm? Chuong B Do and Serafim Batzoglou, Nature Biotechnology, Vol 26, Num 8, August 2008 https://datajobs.com/data-science-repo/ Expectation-Maximization-Primer-[Do-and-Batzoglou].pdf
[12] Expectation-maximization algorithm https://en.wikipedia.org/wiki/ Expectation maximization algorithm
[13] Expectation Maximization with Coin Flips http://karlrosaen.com/ml/ notebooks/em-coin-flips/
[14] What is the reason the test image "Cameraman" is used widely to test algorithms in image processing and image encryption https:www.researchgate.net
[15] Image Denoising Benchmark https://www.cs.utoronto.ca/ strider/ Denoise/Benchmark/
[16] Markov Random Fields https://web.cs.hacettepe.edu.tr/ erkut/bil717. s12/w11a-mrf.pdf
[17] MENACE: Machine Educable Noughts And Crosses Engine https: //people.csail.mit.edu/brooks/idocs/matchbox.pdf