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.

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.
Algorithm:

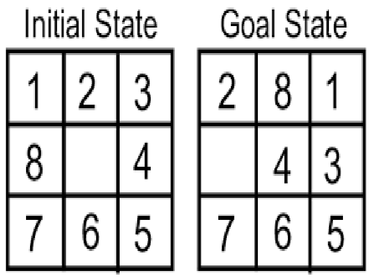


Fig. 1. Initial and Final state of 8-Puzzle [2]

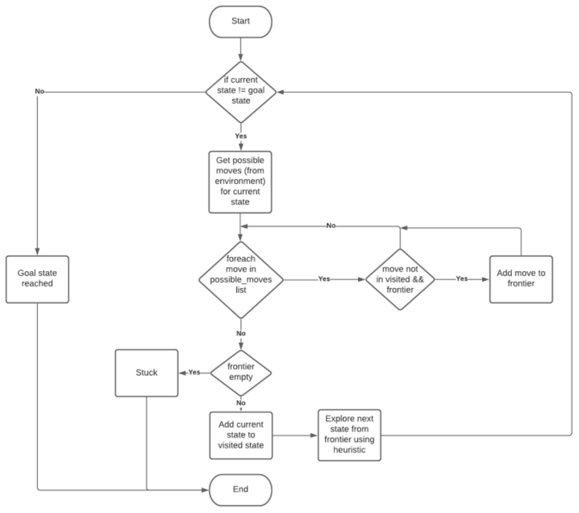

Start
if current state != goal state
No
Yes
Get possible moves (from environment) for current state
foreach move in possible_moves list
Yes
move not in visited && frontier
Yes
Add move to frontier
No
Goal state reached
No
frontier empty
Yes
Stuck
No
Add current state to visited state
Explore next state from frontier using heuristic
End

Fig. 2. Flowchart for Agent

B. Part B

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

- **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

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;
```

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 source/initial state

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

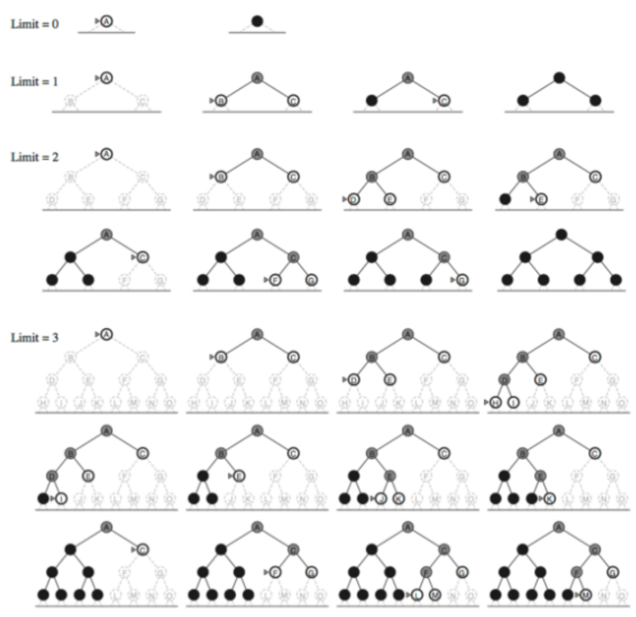
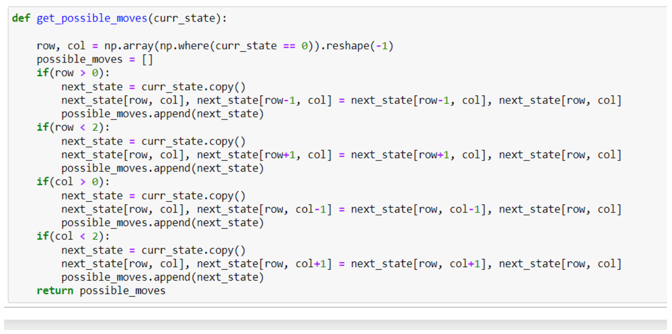


Fig. 3. Iterative deepening search example [7]



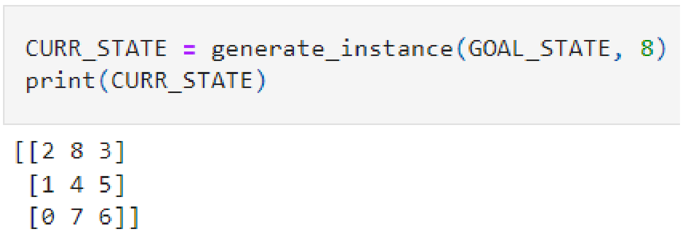
```
def get_possible_moves(curr_state):

    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:



```
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

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

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.

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	0.016	52936
4	0.136	52732
8	1.194	53052
16	208.6	60848

Without using any Heuristic

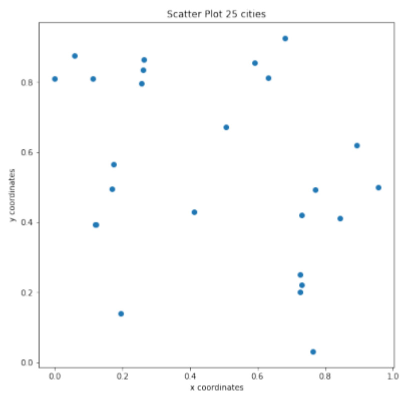


Fig. 6. Scatter plot for 25 nodes

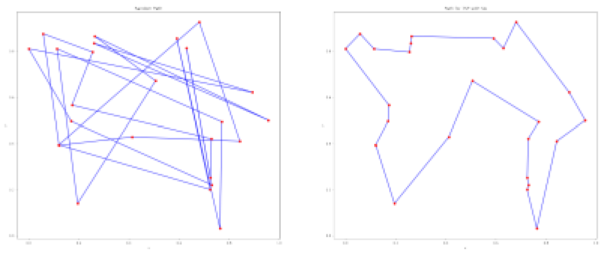


Fig. 7. Comparison between the routes of 25 nodes

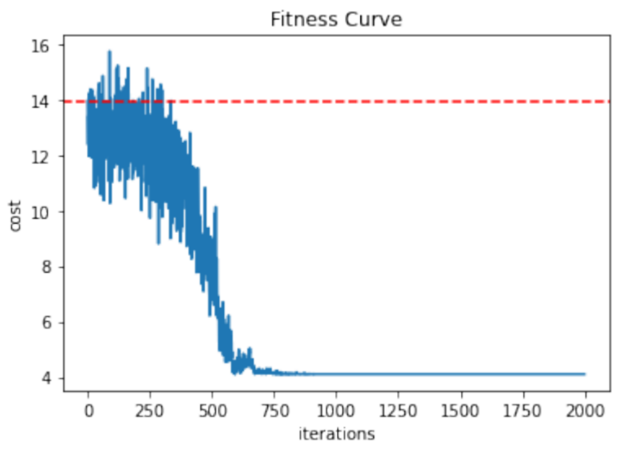


Fig. 8. Fitness curve for 25 nodes

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]

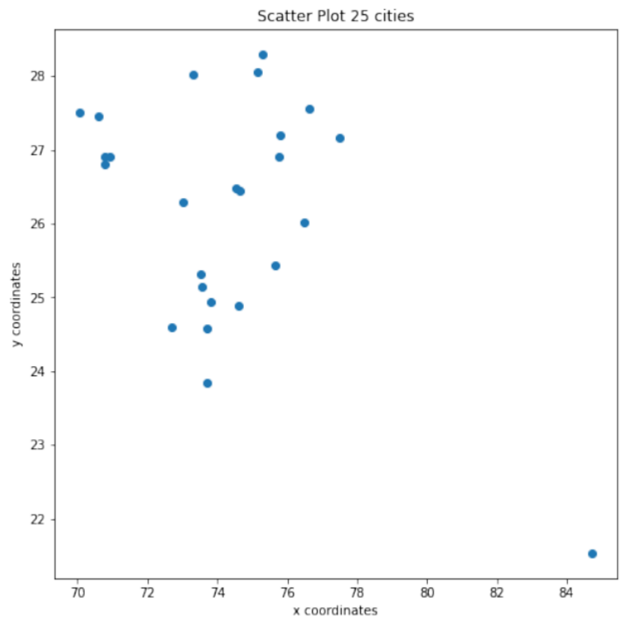


Fig. 9. Scatter plot for 25 locations in Rajasthan

B. *Part B*

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

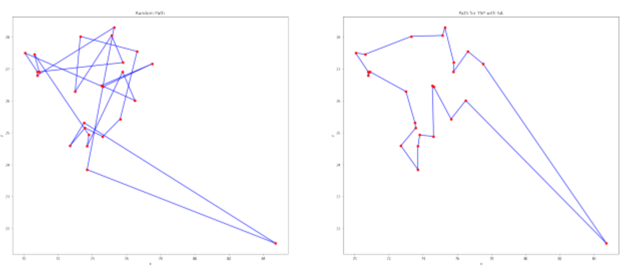


Fig. 10. Optimal path for 25 locations in Rajasthan

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: –

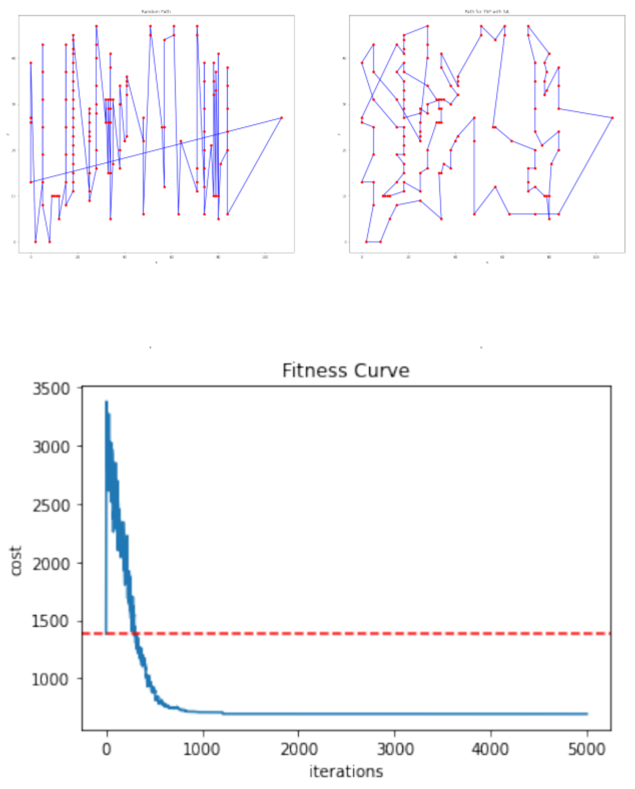


Fig. 11. Plots for 131 points

2) XQF237 - 237 points: -

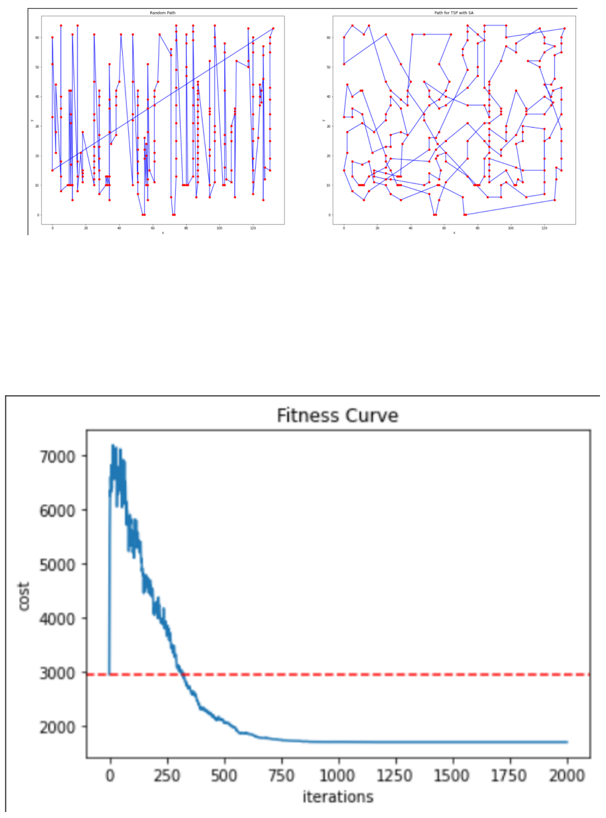


Fig. 12. Plot for 237 points

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

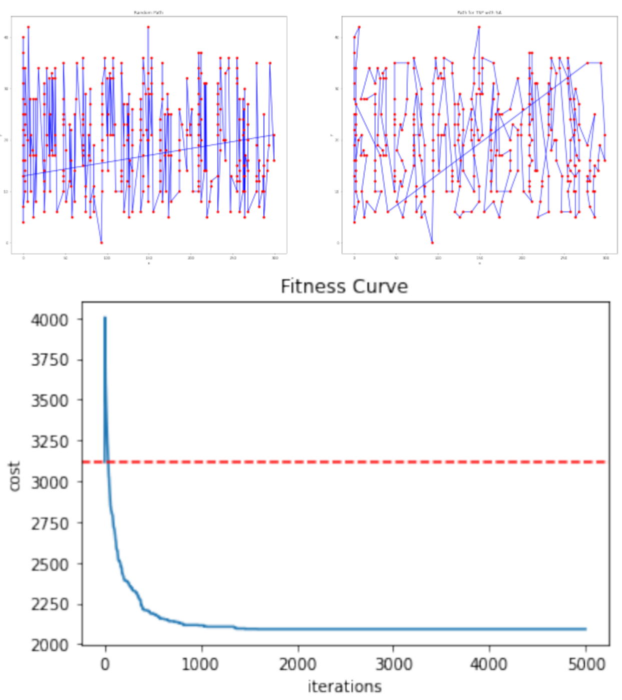


Fig. 13. Plot for 343 points

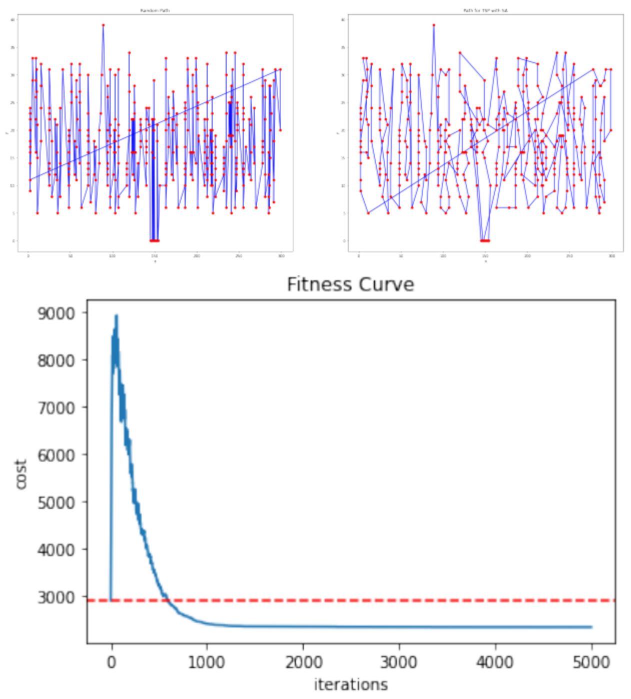


Fig. 14. Plot for 379 points

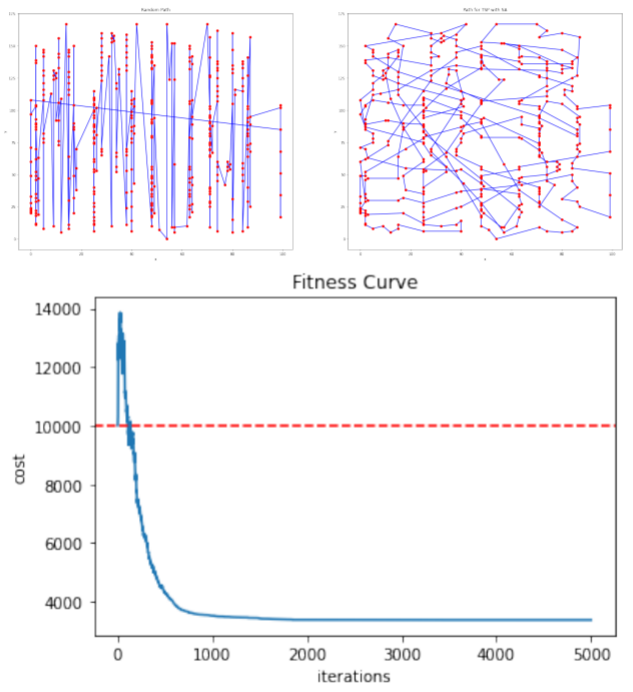


Fig. 15. Plot for 380 points

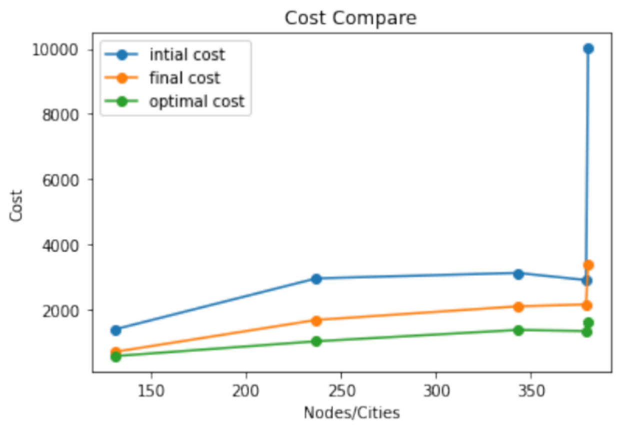


Fig. 16. Plot for 380 points

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:

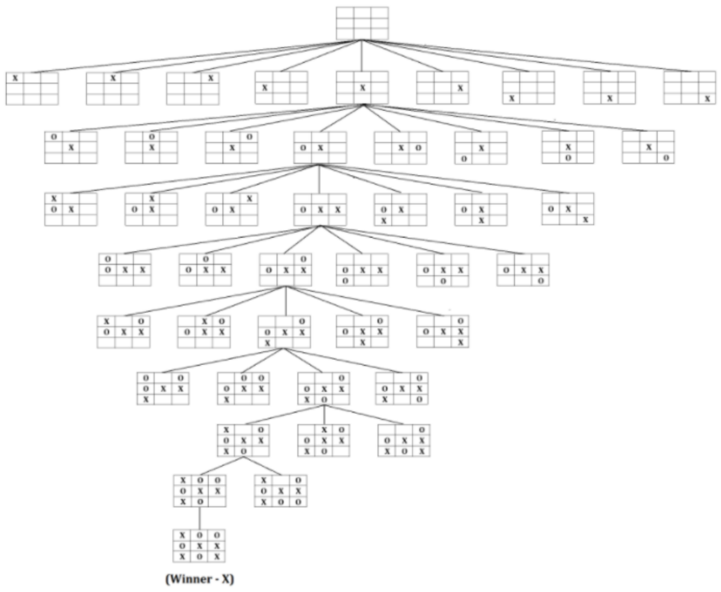


Fig. 17. Graph of the tree

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:

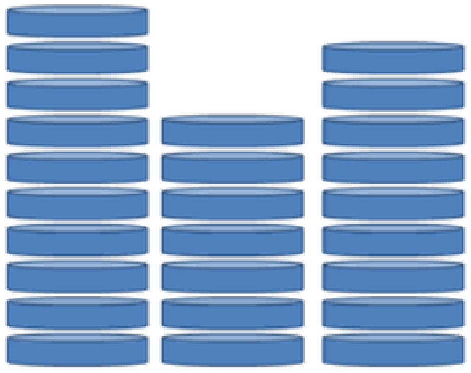
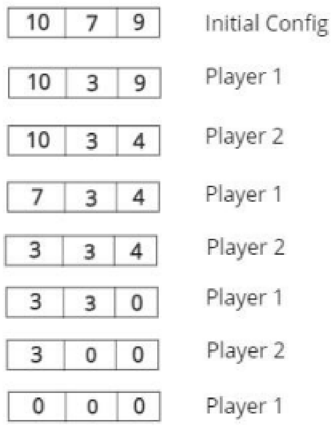


Fig. 18. Graph of the tree



10	7	9	Initial Config
10	3	9	Player 1
10	3	4	Player 2
7	3	4	Player 1
3	3	4	Player 2
3	3	0	Player 1
3	0	0	Player 2
0	0	0	Player 1

Fig. 19. Player taking turns in Nim

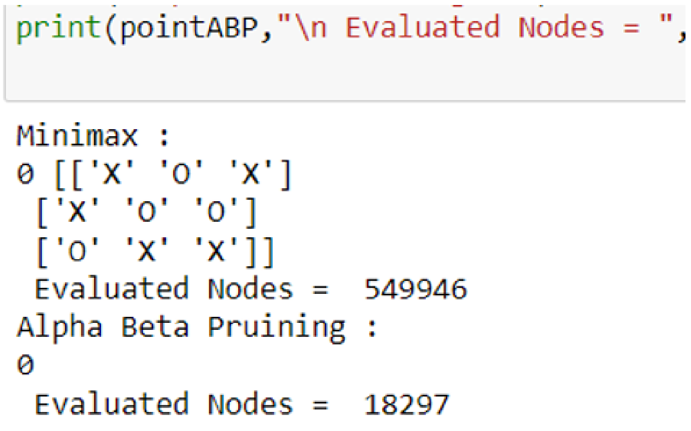
C. *Part 3*

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]'

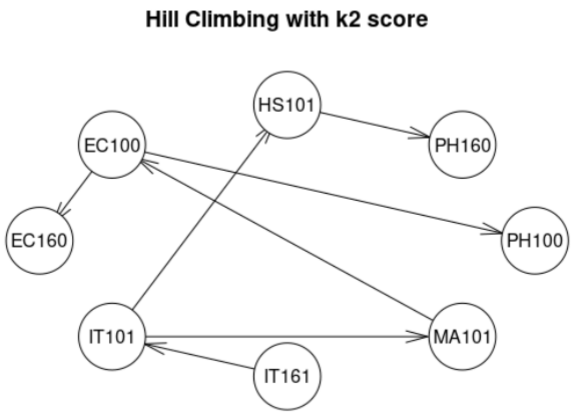


Fig. 21. Hill Climbing Bayesian Network using K2 score

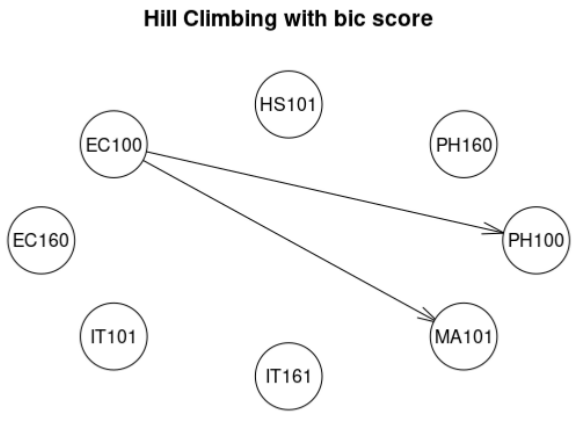


Fig. 22. Hill Climbing Bayesian Network using bic score

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.

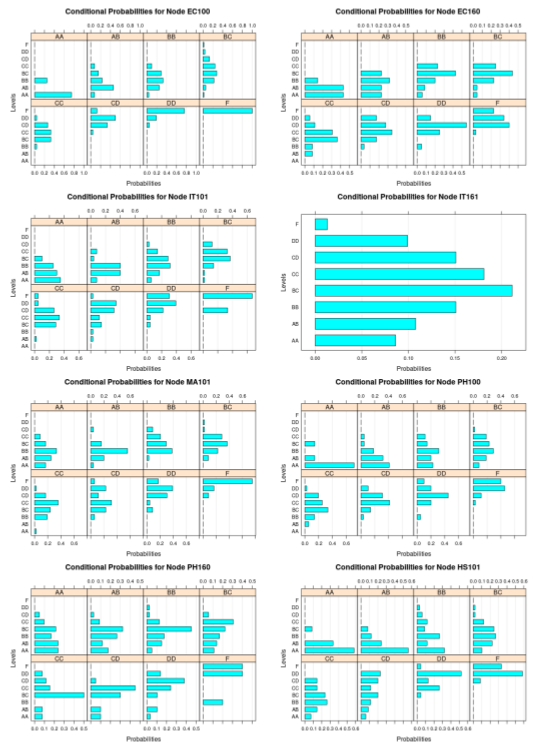


Fig. 23. Conditional probability distributions made form the CPTs of the learned data

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

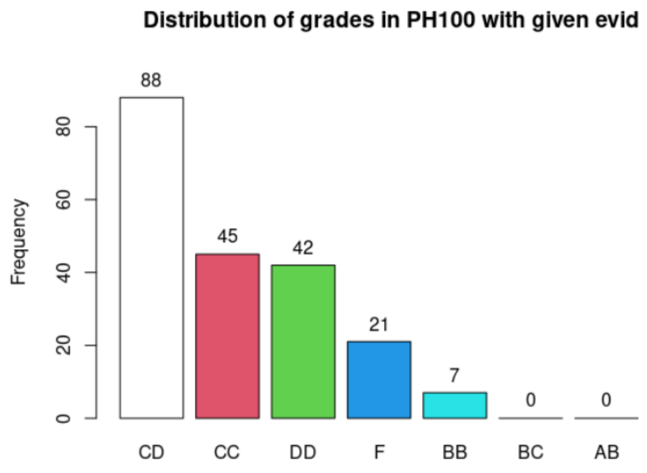


Fig. 24. Probability distribution of getting a grade in PH100 as per the given evidence

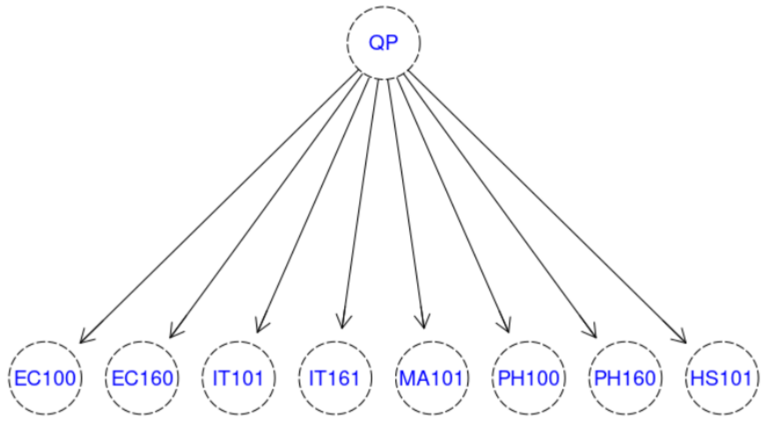


Fig. 25. Naive Bayes Classifier for independent data

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

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

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.

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.

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

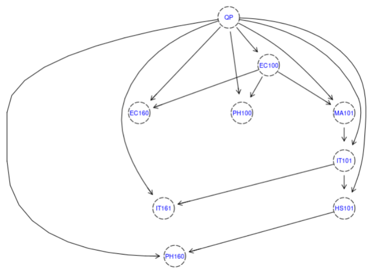


Fig. 26. Naive Bayes Classifier for dependent data

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

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 minimax 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.

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.