CS362
Artificial Intelligence
Midsem Lab Report

Raj
B.Tech. CSE
201951123

Shivansh Kumar
B.Tech. CSE
201951144

Deepanshu Singh
B.Tech. CSE
201951054

Project Reporting done by - Raj (201951123)

GitHub Code Link is: **Click here**

Week 1 Lab Assignment 1

8-puzzle solver

Learning Objective:

To design a graph search agent and understand the use of a hash table, queue in state space search. The 8 puzzle consists of eight numbered, movable tiles set in a 3x3 frame. One cell of the frame is always empty thus making it possible to move an adjacent numbered tile into the empty cell. Such a puzzle is illustrated in following diagram.

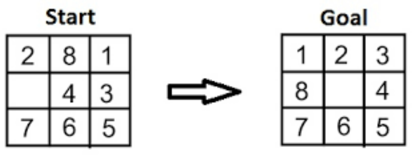


Figure : 8 Puzzle Problem

Problem Statement A

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.**

h1: Heuristic for calculating the distance of goal state using Manhattan distance.
Parameters:
curr state(np.ndarray): A 3x3 numpy array with each cell containing unique elements
goal state(np.ndarray): A 3x3 numpy array with each cell containing unique elements
returns:
h(int): Heuristic value

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;
```

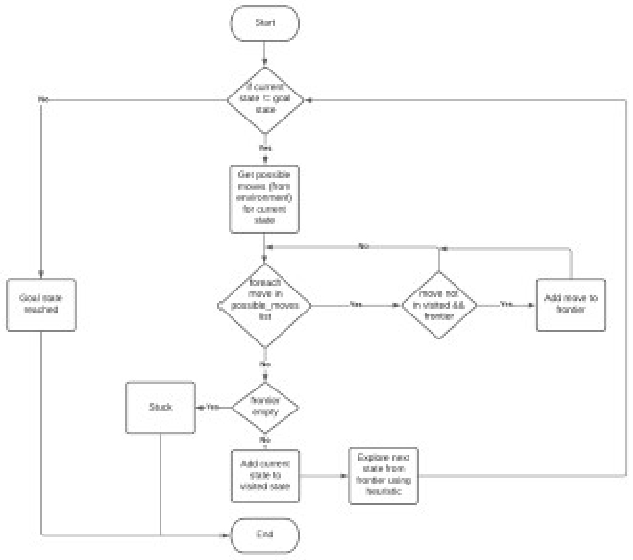

Start
if current state != goal state
No
Yes
Get possible moves (from environment) for current state
foreach move in possible_moves list
Yes
move not in visited && frontier
Yes
Add move to frontier
No
Goal state reached
No
frontier empty
Yes
Stuck
No
Add current state to visited state
Explore next state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

Fig. 2: Flowchart for Agent

Problem Statement B

B. Write a collection of functions imitating the environment for Puzzle-8.

h1: Heuristic for calculating the distance of goal state using Manhattan distance
. Parameters:
curr state(np.ndarray): A 3x3 numpy array with each cell containing unique elements
goal state(np.ndarray): A 3x3 numpy array with each cell containing unique elements
returns: h(int): Heuristic value

h2: Heuristic for calculating the distance of goal state using number of misplaced tiles
Parameters:
curr state(np.ndarray): A 3x3 numpy array with each cell containing unique elements
goal state(np.ndarray): A 3x3 numpy array with each cell containing unique elements
returns: h(int): Heuristic value

generate instance: Heuristic for calculating the distance of goal state using number of misplaced tiles
Parameters:
goal state(np.ndarray): A 3x3 numpy array with each cell containing unique elements representing the goal state depth(int): The depth at which the state is to be generated
debug(bool): To print intermediate states and the heuristic values, or not. Default: False
returns:
curr state(np.ndarray): A 3x3 numpy array with each cell containing unique elements representing the state at the given depth form, the goal state

get possible moves: Function to get a list of possible states after valid moves form current state tiles
Parameters:
curr state(np.ndarray): A 3x3 numpy array with each cell containing unique elements, 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 after valid moves form current state
possible paths(list): List of possible paths after valid moves form current state

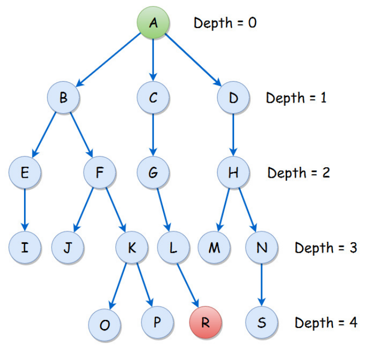
sort by heuristic: Helper function to sort the next possible states based on heuristic value passed Parameters:
possible moves(list): List of possible states after valid moves form current state
goal state(np.ndarray): A 3x3 numpy array with each cell containing unique elements representing the goal state
heuristic(Integer): An integer indicating the heuristic function to use. 1 for heuristic h1 and 2 for heuristic h2 possible paths(list): List of possible moves after valid moves form current state
returns:
sorted possible moves(list): List of possible states after valid moves form current state, sorted according to heuristic
sorted possible moves(list): List of possible paths after valid moves form current state, sorted according to heuristic

solve: Solves the puzzle by finding a path from current state to the goal state, using the heuristic provided. If no heuristic is provided, solves using normal BFS. Prints ”GOAL REACHED” if goal is reached and prints ”GOAL NOT REACHABLE” if no possible move is left
Parameters:
curr state(np.ndarray): A 3x3 numpy array with each cell containing unique elements, representing the current state
goal state(np.ndarray): A 3x3 numpy array with each cell containing unique elements representing the goal state
heuristic(Integer):
An integer indicating the heuristic function to use. 1 for heuristic h1 and 2 for heurostic h2. If not provided or passed as 0, solves using normal BFS
returns:
sorted possible moves(list): List of possible states after valid moves form current state, sorted according to heuristic
sorted possible moves(list): List of possible paths after valid moves form current state, sorted according to heuristic

Problem Statement C

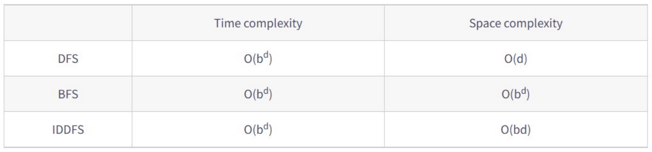
C. Describe what is Iterative Deepening Search.

This algorithm is used when you have a goal directed agent in an infinite search space (or search tree).Iterative deepening depth first search (IDDFS) is a hybrid of BFS and DFS. In IDDFS, we perform DFS up to a certain “limited depth,” and keep increasing this “limited depth” after every iteration.



DEPTH DLS traversal

0 – A
1 – A B C D
2 – A B E F C G D H
3 – A B E I F J K C G L D H M N
4 – A B E I F J K O P C G L R D H M N S



	Time complexity	Space complexity
DFS	$O(b^d)$	O(d)
BFS	$O(b^d)$	$O(b^d)$
IDDFS	$O(b^d)$	O(bd)

Problem Statement D

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.**

Watch Colab file for function code

Problem Statement E

E. **Generate Puzzle-8 instances with the goal state at depth "d".**

We created a function "generate instances" to create goal state at depth "d". The function takes the goal state and depth as input, generates a random instance of the 8-puzzle at the given depth from the goal state and return a 3x3 numpy array with each cell containing unique elements representing the state at the given depth form the goal state. Watch Colab file for function code

Week 3 Lab Assignment 3

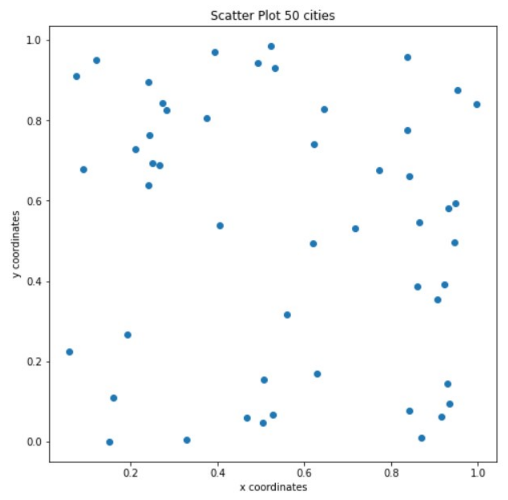
Traveling Salesman Problem

Learning Objective:

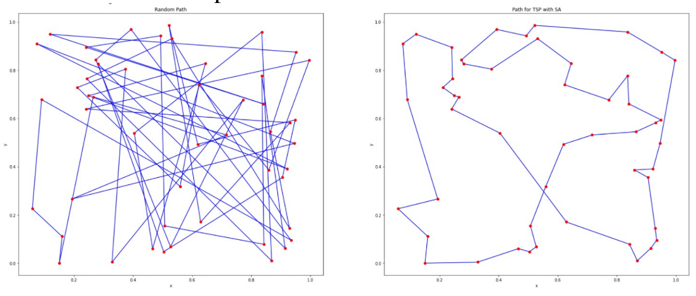
Non-deterministic Search — Simulated Annealing For problems with large search spaces, randomized search becomes a meaningful option given partial/ full-information about the domain.

Problem:
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.
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 travelling between two locations is proportional to the distance between them.

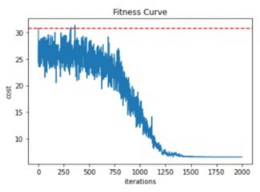
Below is the scatter plot for 50 random points



Below is the comparision between 2 routes for 50 nodes



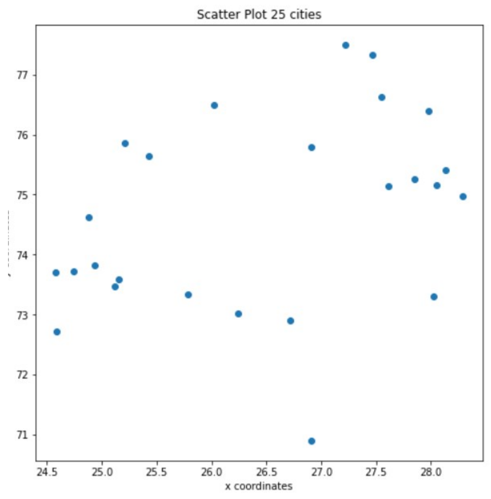
Below is the Fitness curve for 50 points



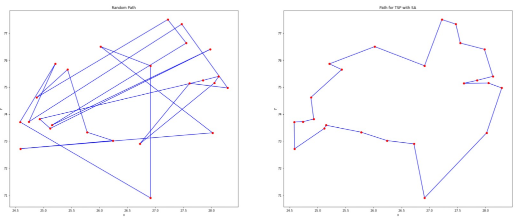
For 25 cities of Rajasthan

F. **Finding Best path to cover all nosed in shortest time.**

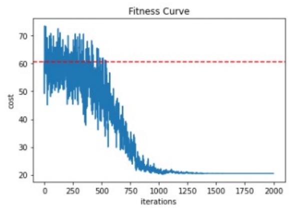
Below is the scatter plot for 50 random points



Below is the comparision between 2 routes for 50 nodes



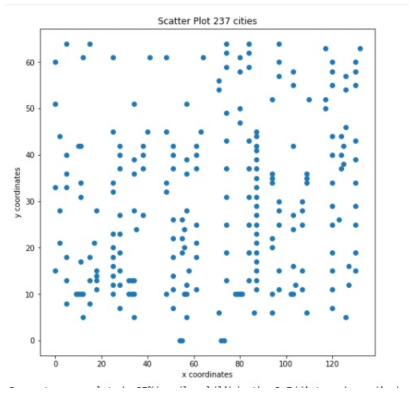
Below is the Fitness curve for 50 points



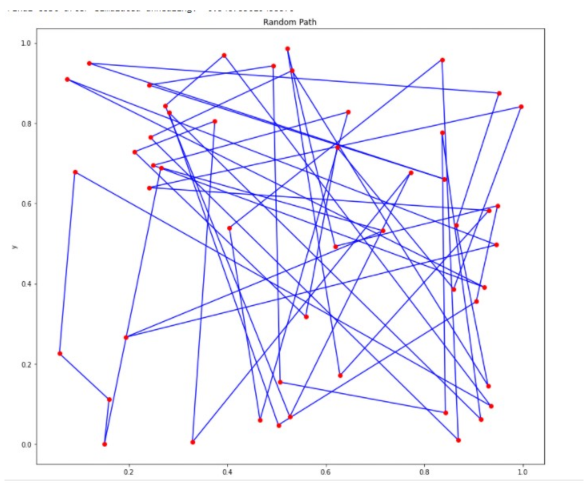
For one of the 6 VLSI datasets

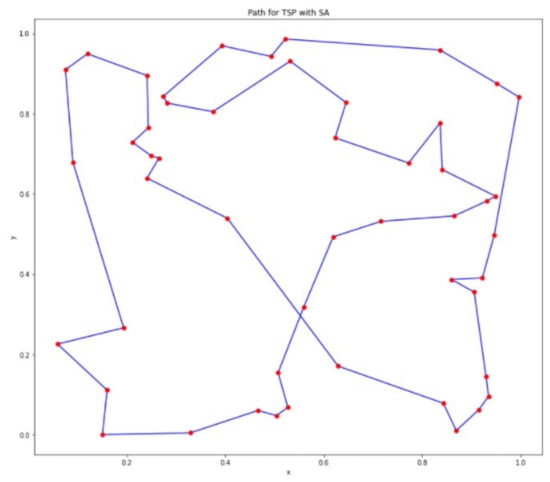
G. **XQG237 - 237 points**

Below is the scatter plot for 50 random points

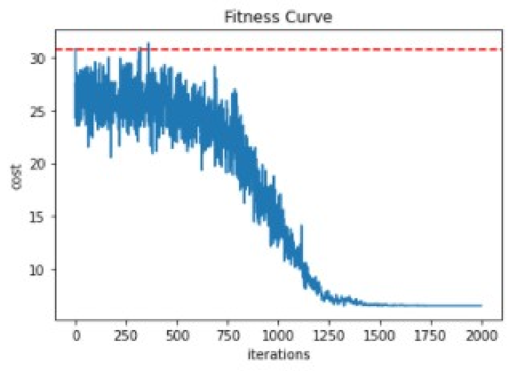


Below is the comparision between 2 routes for 50 nodes





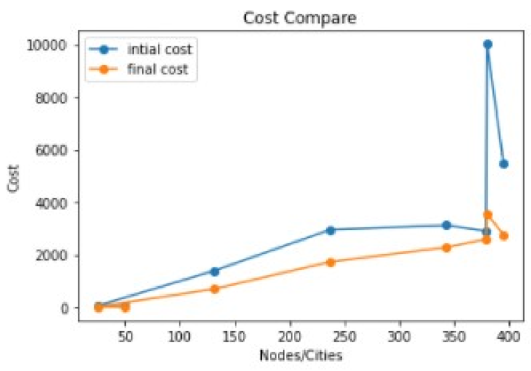
Below is the Fitness curve for 50 points



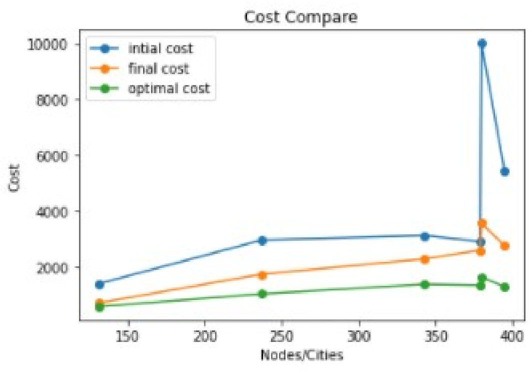
Comparision of results

H. **Creating a graph to compare cost.**

Below is the comparision between all the datasets including random dataset, VLSI dataset, 25 Rajasthan cities dataset



Below is the comparision between all 6 datasets



Week 5 Lab Assignment 4

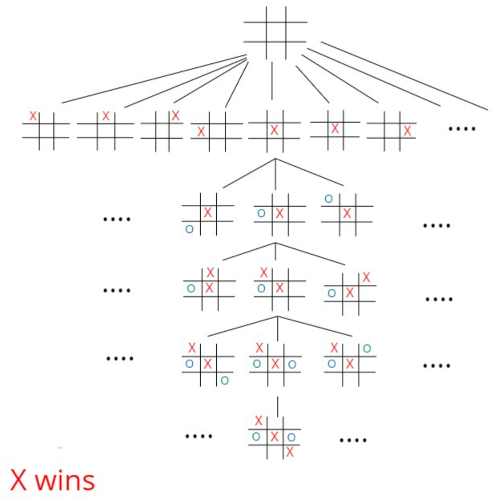
Learning Objective:

Game Playing Agent — Minimax — Alpha-Beta Pruning Systematic adversarial search can lead to savings in terms of pruning of sub-trees resulting in lesser node evaluations

Problem:

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

In the game of Noughts and Crosses, assume Player 1 is ’X’ and Player 2 (computer) is ’O’. Then if Computer goes First we have 9! possible moves i.e., 362,880 possible moves, and if computer goes second we have 8! possible moves i.e., 40,320 possible moves. At depth = 1, we can have 9 different states. At depth = 2, we can have 9*8 different states. At depth = 3, we can have 9*8*7 different states, and so on till At depth = 9, we can have 9! different states. Total different states available will be : 9 + 9 * 8 + 9 * 8 * 7 + ::: + 9! = 106

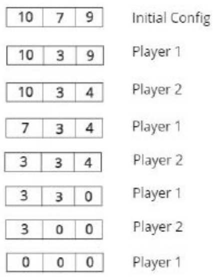


2. Read about the game of Nim (a player left with no move losing the game). For the initial configuration of the game with three piles of objects as shown in Figure, show that regardless of the strategy of player-1, player-2 will always win. Try to explain the reason with the MINIMAX value backup argument on the game tree.

With Minimax Algorithm both the players will try to maximize their chances of wining which leaves everything to the number of tower, since any number of tiles can be picked from a single tower at once, we can check

$$M(depth) = \begin{cases} 0 & \text{if current player loses} \\ -1 & \text{if tiles still remain} \\ 1 & \text{if current player wins} \end{cases} \quad (3)$$

If Player 1 always move such that XOR of three towers is equal to 0 then this will result in player 1 always winning no matter what player 2 does.[13] For example in Fig : 17



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. 17: Right section represents move played by player 1 or Player 2

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

Implementation can be found on our code.
Initial State – Minimax , Alpha-beta pruining
empty-state – 549946 , 18297

4. Use recurrence to 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.

O(bm=2) It corresponds to the best case time complexity of alpha-beta pruning. In this the branching factor of b, and a depth of m plies, the maximum number of leaf node positions will be evaluated.If the move ordering for the search is optimal, the number of leaf node positions evaluated is about O(b*1*b*1*...*b) for odd depth and O(b*1*b*1*...*1) for even depth, or O(b(m=2)): In the latter case, where the ply of a search is even, the effective branching factor is reduced to its square root, or, equivalently, the search can go twice as deep with the same amount of computation.

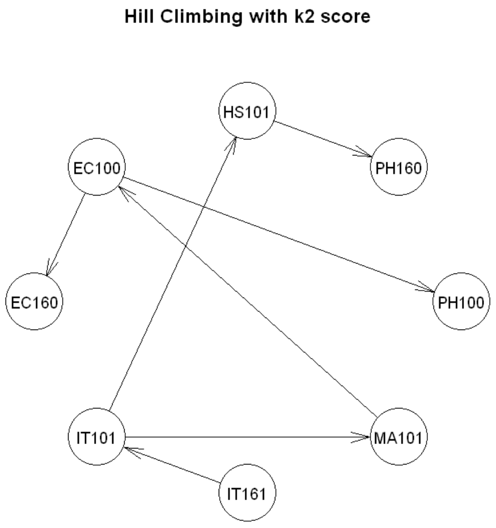
Week 6 Lab Assignment 5

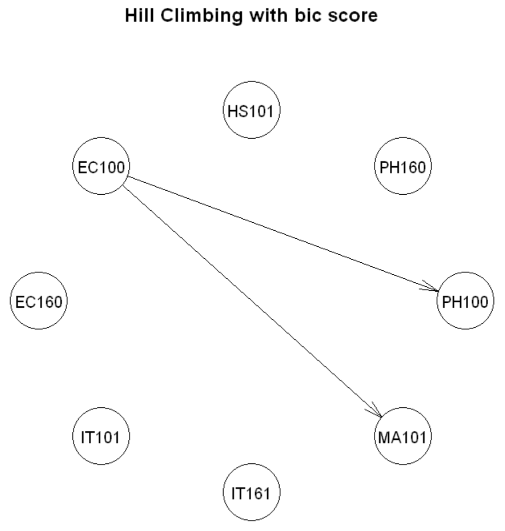
Learning Objective:

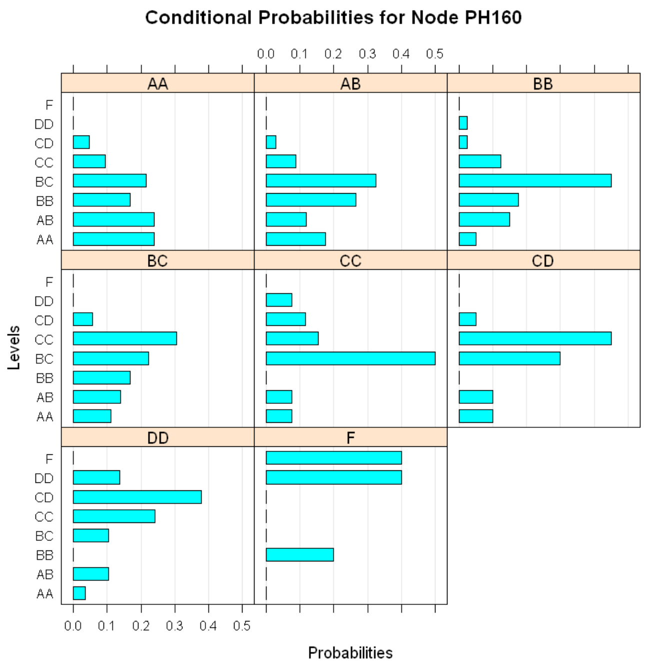
Understand the graphical models for inference under uncertainty, build Bayesian Network in R, Learn the structure and CPTs from Data, naive Bayes classification with dependency between features.

Problem Statement:
A table containing grades earned by students in respective courses is made available to you in (codes folder) $2020_{b}n_{n}b_{d}ata.txt.$

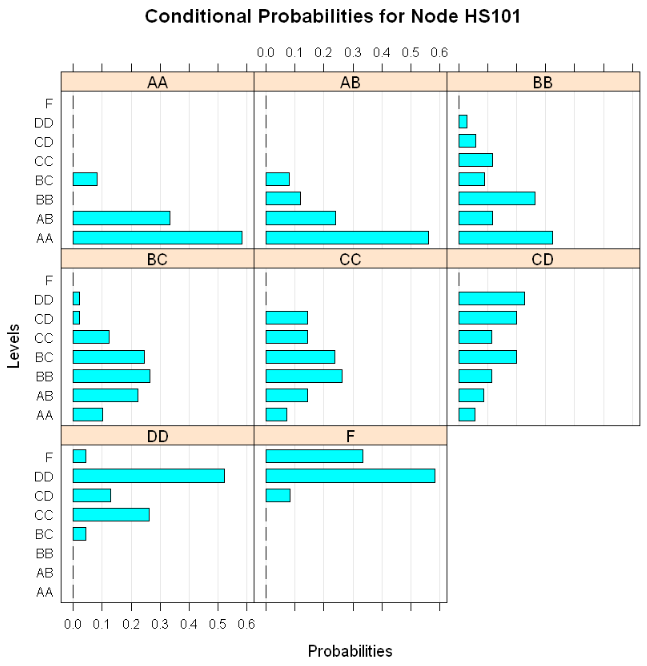
1. Consider grades earned in each of the courses as random variables and learn the dependencies between courses.

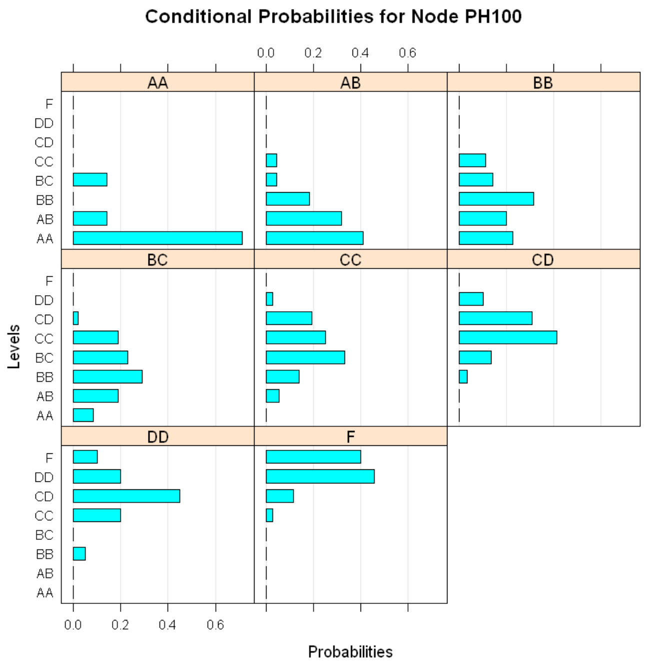


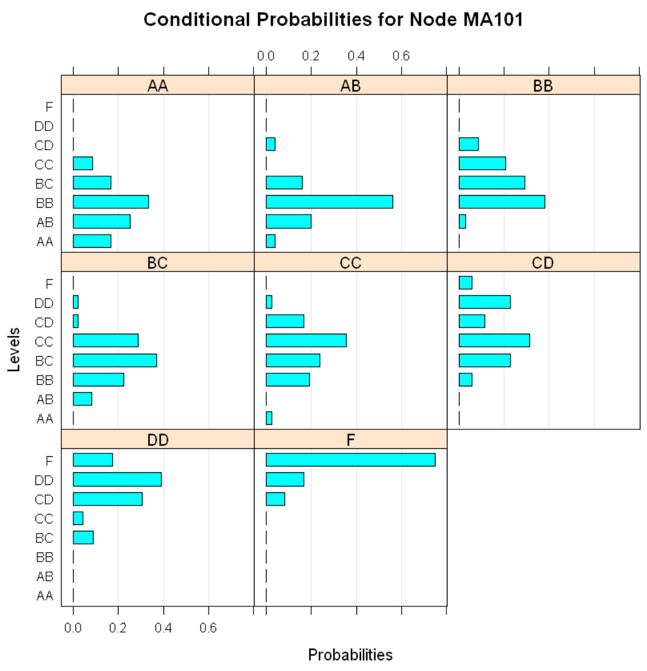


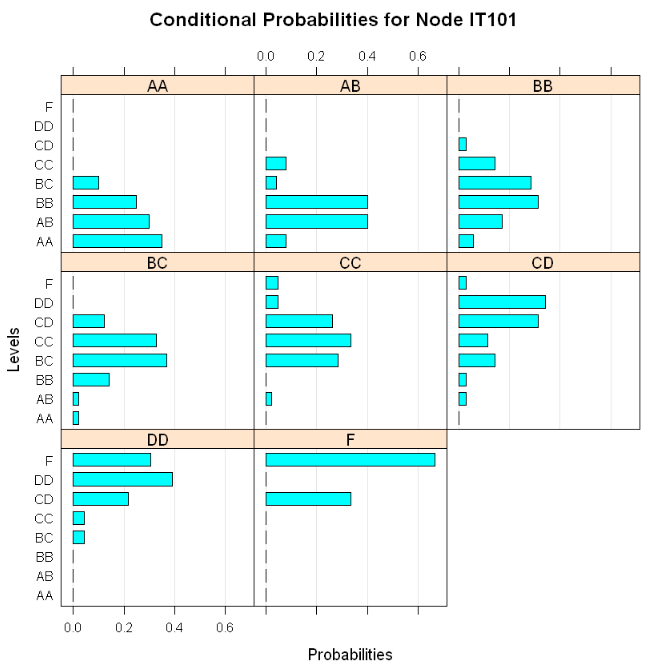


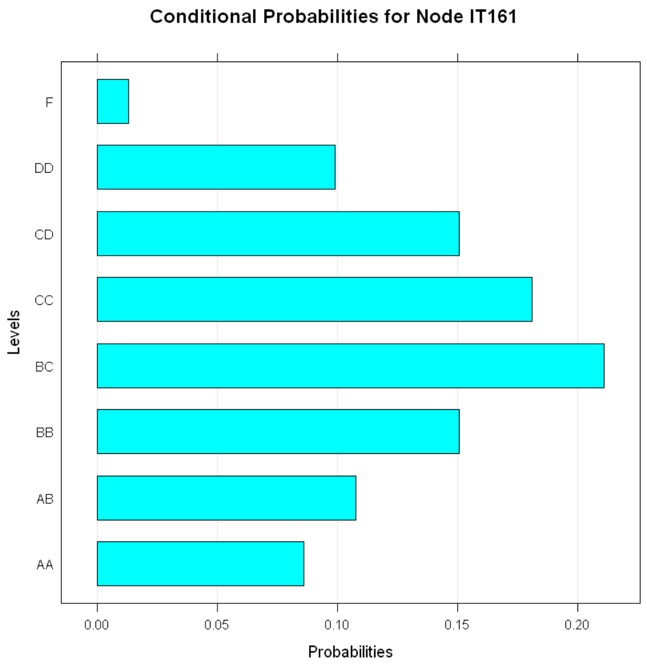
2. Using the data, learn the CPTs for each course node.

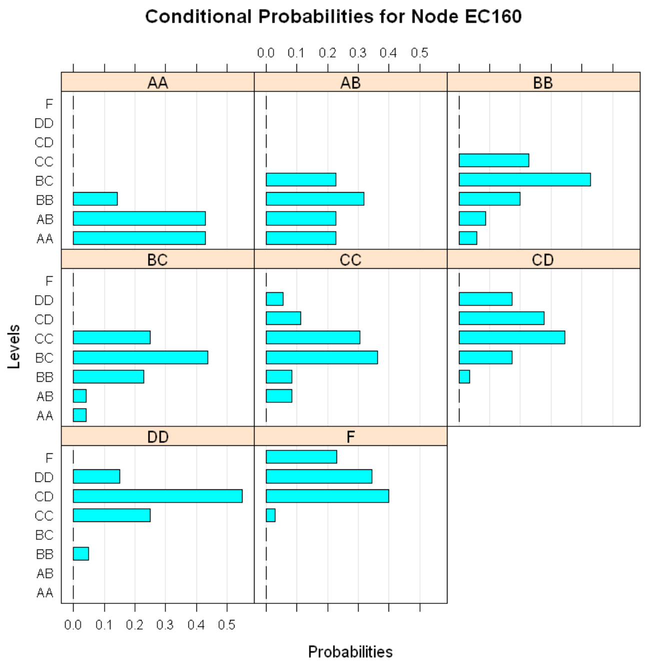






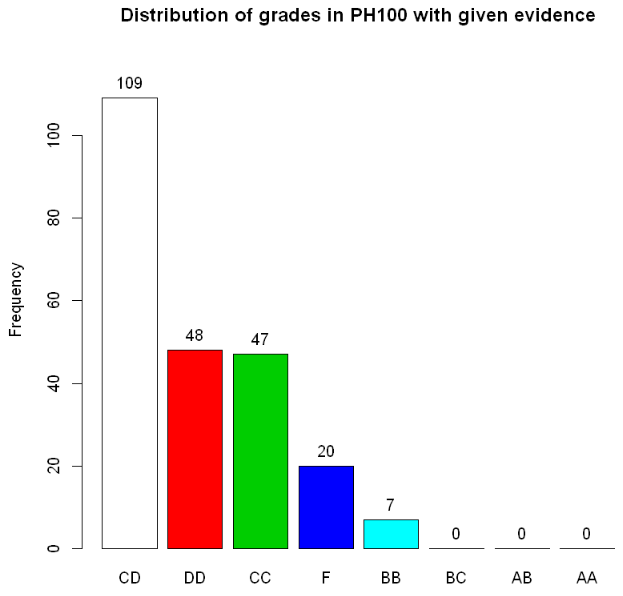


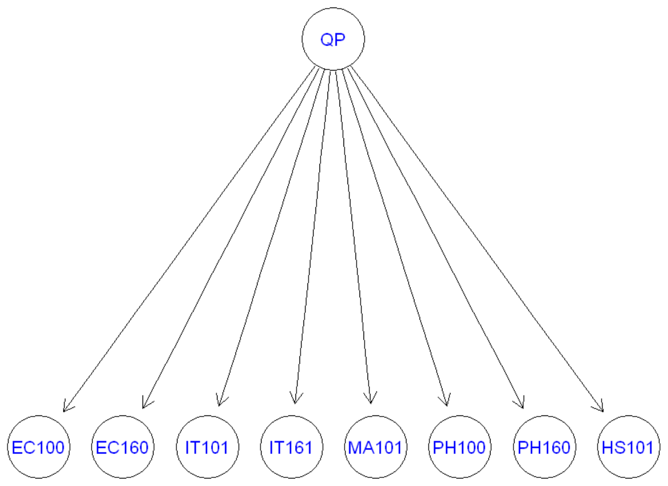




3. What grade will a student get in PH100 if he earns DD in EC100, CC in IT101 and CD in MA101.

Watch Code for solution

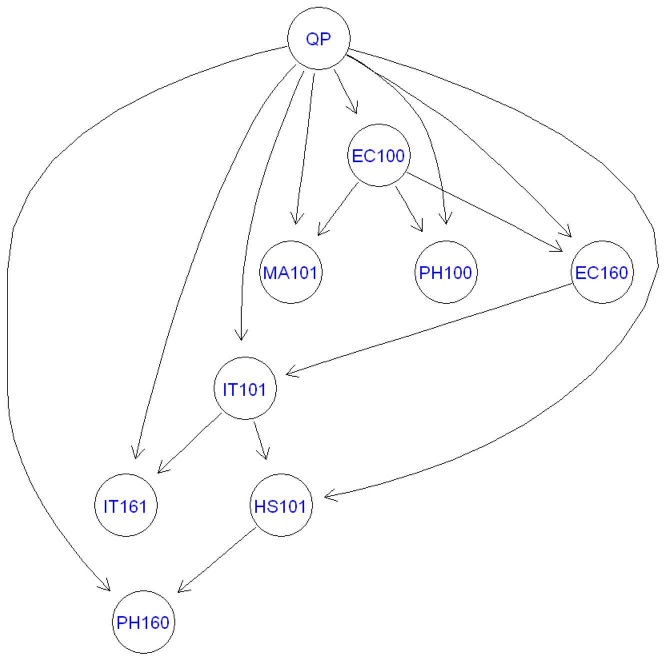




4. The last column in the data file indicates whether a student qualifies for an internship program or not. From the given data, take 70 percent data for training and build a naive Bayes classifier (considering that the grades earned in different courses are independent of each other) which takes in the student's performance and returns the qualification status with a probability. Test your classifier on the remaining 30 percent data. Repeat this experiment for 20 random selection of training and testing data. Report results about the accuracy of your classifier.

Watch code for solution

5. Repeat 4, considering that the grades earned in different courses may be dependent.
Watch code for details.



Conclusions

As demonstrated above we have solved 8 puzzle problem, Travelling Salesman Problem, Noughts and Crosses game, Nim Game and understood the Bayesian graphical models for to build network graphs.

In 8-puzzle problem we were able observe performance of different heuristic functions, where heuristic with Manhattan distance gave best results whereas when solved without heuristic performance was the worst.

For the Travelling Salesman Problem, we observed that with simulated annealing it provides an optimal or sub optimal solution which reduces the cost for given set of points/coordinates.

We also observed that how the decrease in temperature and number of iterations affects the solution.

From our observation for mini-max and alpha-beta pruning less nodes were evaluated in alpha-beta pruning than in minimax algorithm, which was the essence of the assignment. We can clearly say 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 (See Fig. 20). Naive Bayes Classifier gave very accurate results on the test data-set.

To get the main code, **Click here**

References

[1] Josh Richard, An Application Using Artificial Intelligence, January 2016
[2] dpthegrey, 8 puzzle problem June 2020.
[3] Ajinkya Sonawane, Solving 8-Puzzle using A* Algorithm September 2018
[4] javatpoint Uninformed Search Algorithms
[5] Zhou, Ai-Hua, Traveling-salesman-problem algorithm based on simulated annealing and gene-expression programming. Information 10.1 2019.
[6] Frank Liang, Optimization Techniques for Simulated Annealing. https://towardsdatascience.com/ optimization-techniques-simulated-annealing-d6a4785a1de7
[7] Traveling Salesperson Problem. https://www.youtube.com/watch?v= 35fzyblVdmAt=656s.
[8] Marco Scutar, Learning Bayesian Networks with the bnlearn R Package, Issue 3. Journal of Statistical Software, July 2010.
[9] Bojan Mihaljevi´c, Bayesian networks with R November 2018.
[10] Shah Pratik, An Elementary Introduction to Graphical Models February 2021.
[11] Shah Pratik, An Elementary Introduction to Graphical Models February 2021.