Artificial Intelligence Laboratory
Mid-sem Report
TeamLoki

Abhishek Singh 201851005
Gaurav Singh 201851044
Harish Kumar 201851046
Janmejay 201851053
Varun Gupta 201851141

Contents

I. Week 1: 11 Jan - 15 Jan 2021 Lab Assignment- 1

Learning Objective: To design a graph search agent and understand the use of a hash table, queue in state space search.

(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.
Solution:
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure

```
node<-NODE(problem,INITIAL)
if problem.IS-GOAL(node.STATE)
then return node
frontier<-a FIFO queue
reached<-{problem.INITIAL}
while not IS-EMPTY(frontier) do
node=POP(frontier)
 for  each child in EXPAND(problem,node)
  s=child.STATE
  if problem.IS-GOAL(s) then return child
  if  s is not in reached then
    add s to reached
    add child to frontier
return failure
```

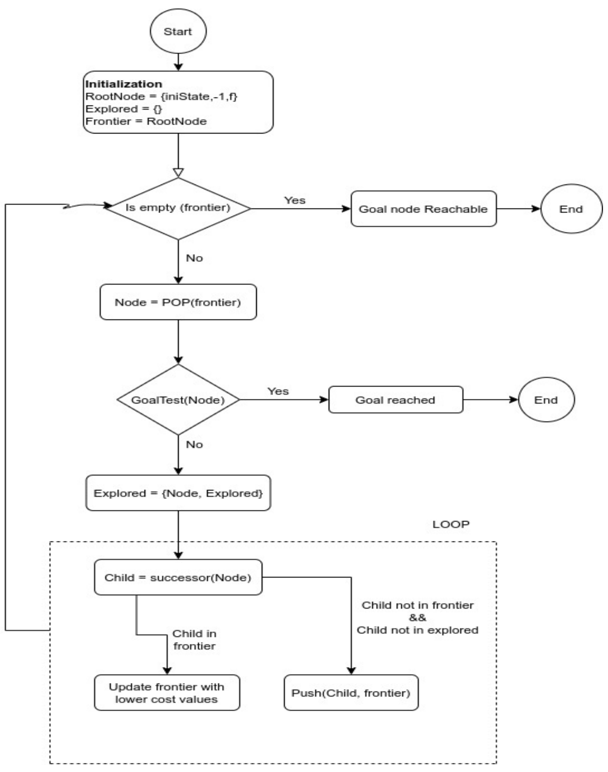


Fig. 1. Flow chart of Agent

Explanation:
Initial node is root node if the node is goal state then it will return the node,otherwise insert the element in queue in FIFO manner in the frontier and reached state is the problem.INITIAL state.if the frontier is not empty the loop will iterate until the frontier becomes empty.and expand every node to check whether the goal state has been reached or not.To make the entry of the node mark that node as 1 if visited or maintain the frontier if it is then return the node otherwise it will return failure.

(B) Write a collection of functions imitating the environment for Puzzle-8.
Solution:
we take one dimensional array and think this array as 3*3 matrix (row wise) and assume 0 as empty space.

For initial state

```
puzzle = []
print(" Input vals from 0-8 for start state ")
for i in range(0,9):
    x = int(input("enter vals :"))
    puzzle.append(x)
```

User input of goal state

```
goal = []
node=[]
print(" Input vals from 0-8 for goal state ")
for i in range(0,9):
    x = int(input("Enter vals :"))
    goal.append(x)
```

for next state of the current state

```
def sucessor(self,node=[]):
    subNode=[]
    getZeroLocation=node.index('0')+1
    subNode.extend(node)
    fronts=[]
    child=[]
  ##if 0 is not present in the last row of the
  matrix,can move down.
   if getZeroLocation+3<=9:
        temp1=subNode[node.index('0')]
        temp2=subNode[node.index('0')+3]
        subNode[node.index('0')]=temp2
        subNode[node.index('0')+3]=temp1
        child.append(subNode)
        subNode=[]
        subNode.extend(node)

## if 0 is not present in the first row of
the matrix,can move up
    if getZeroLocation-3>=1:
        temp1=subNode[node.index('0')]
        temp2=subNode[node.index('0')-3]
        subNode[node.index('0')]=temp2
        subNode[node.index('0')-3]=temp1
        child.append(subNode)
        subNode=[]
        subNode.extend(node)

##if 0 is not present in the first column
of the matrix,can move left
    if ((getZeroLocation)%3)!=1:
        temp1=subNode[node.index('0')]
        temp2=subNode[node.index('0')-1]
        subNode[node.index('0')]=temp2
        subNode[node.index('0')-1]=temp1
         child.append(subNode)
        subNode=[]
        subNode.extend(node)

##if 0 is not present in the last column
of the matrix,can move right
    if ((getZeroLocation)%3)!=0:
        temp1=subNode[node.index('0')]
        temp2=subNode[node.index('0')+1]
        subNode[node.index('0')]=temp2
        subNode[node.index('0')+1]=temp1
        child.append(subNode)
         subNode=[]
        subNode.extend(node)
```

(C) Describe what is Iterative Deepening Search.
Solution: Iterative Deepening Search is a state search strategy in which it limits the depth of Depth First Searching at some depth d and do backprop from there .It attains the Breadth First Search optimality when the branching factor is finite and also reduces the memory consumption at every iteration.

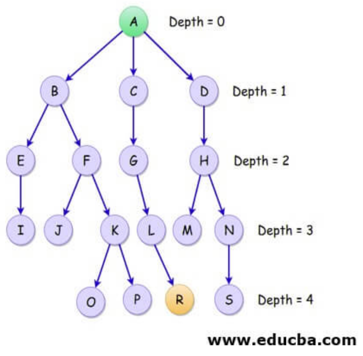


Fig. 2. Iterative Deepening Search

DEPTH = 0,1,2,3,4

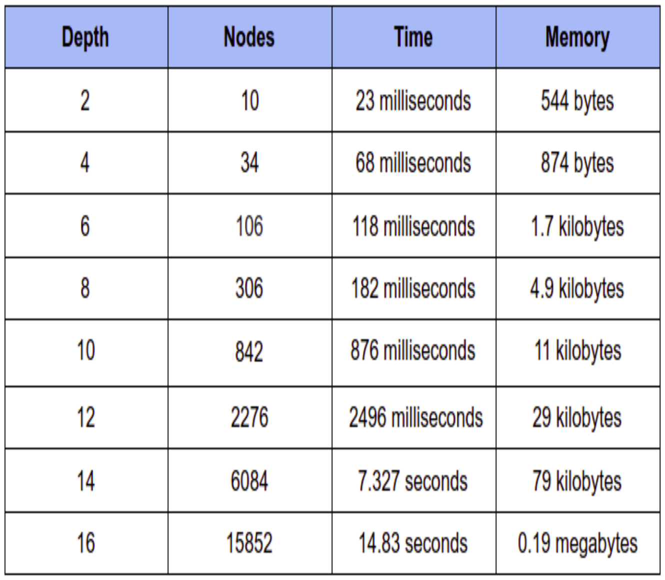
Depth-Limit	IDDFS
0	A
1	ABCD
2	ABEFCGDH
3	ABEIFJKCGLDHMN
4	ABEIFJKOPCGL(**R**)DHMNS

Time complexity: O(b^d)
Space complexity: O(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.
Code Link: For backtrack and produce the path

(E) Generate Puzzle-8 instances with the goal state at depth "d".
Code Link: Generate Puzzle-8 instance at depth d.

(F) Prepare a table indicating the memory and time requirements to solve Puzzle-8 instances (depth "d") using your graph search agent.



Depth	Nodes	Time	Memory
2	10	23 milliseconds	544 bytes
4	34	68 milliseconds	874 bytes
6	106	118 milliseconds	1.7 kilobytes
8	306	182 milliseconds	4.9 kilobytes
10	842	876 milliseconds	11 kilobytes
12	2276	2496 milliseconds	29 kilobytes
14	6084	7.327 seconds	79 kilobytes
16	15852	14.83 seconds	0.19 megabytes

Fig. 3. Memory and Time Requirement

II. Week 3: 18-22 January, 2021 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: 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.

Code Link: TSP Using Simulated Annealing for city of Rajasthan.

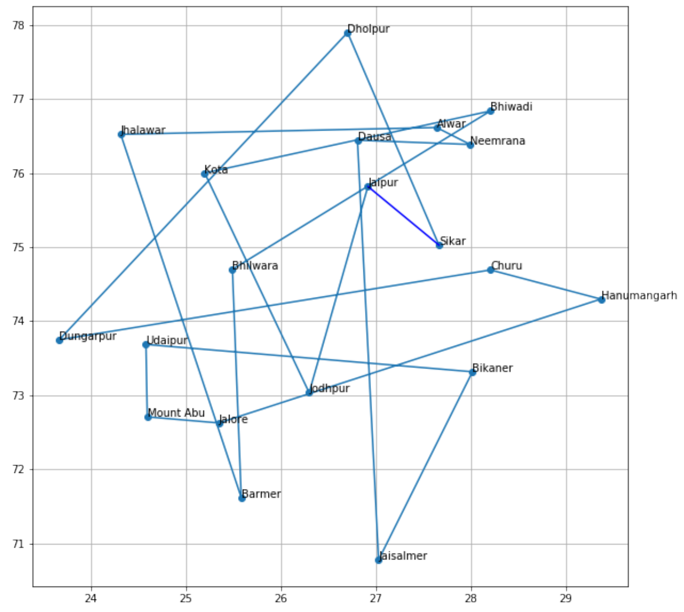


Fig. 4. Path for travelling Rajasthan before Using TSP

We have taken 20 cities from state of Rajasthan and found a path for travelling all 20 cities at random as shown in figure 4. Then we have used simulated annealing to reduce the cost and we got a path with less cost as shown in figure 5

The search is started with a randomized state. In a polling loop we will move to neighboring states always accepting the moves that decrease the path cost while only accepting bad moves accordingly to a probability distribution dependent on the "temperature" of the system.

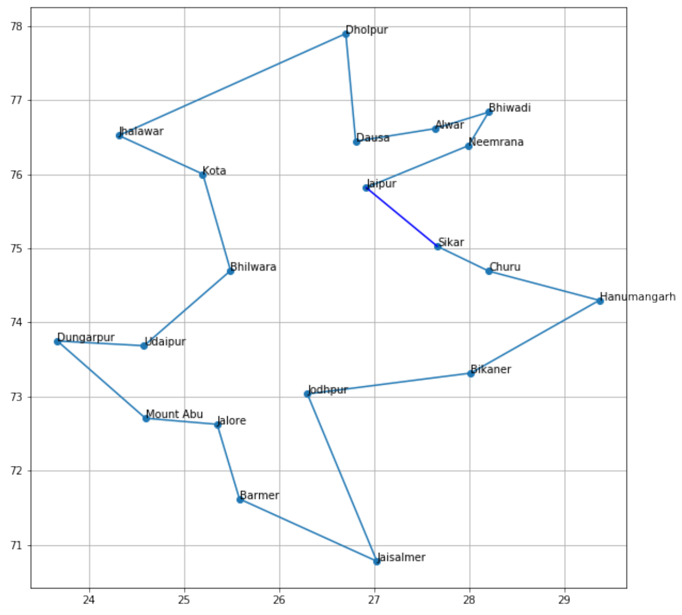


Fig. 5. Path for travelling Rajasthan after Using TSP

An interesting problem domain with TSP instances: VLSI: http://www.math.uwaterloo.ca/tsp/vlsi/index.htmlXQF131 (Attempt at least five problems from the above list and compare your results.)

We taken 5 VLSI Dataset with the size of 131, 237, 343, 395, 380 points respectively, and applied the TSP algorithm using Simulated Annealing to find the good path.

Code Link: TSP Using Simulated Annealing

The results of VLSI dataset after applying TSP using SA are also shown in the notebook file.

III. Week 5: 1-5 February, 2021 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

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

Solution : Game tree is shown in figure 6.

Size of the game Tree is :

Size: 9+9*8+9*8*7+9*8*7*6+9*8*7*6*5+9*8*7*6*5*4 +9*8*7*6*5*4*3+9*8*7*6*5*4*3*2+9*8*7*6*5*4*3*2 which is equal to

$$\sum_{n=1}^{9} \frac{9!}{9-n!}$$

As there are 9 squares and each squares have 3 choices ("X","O"and "Draw") so there can be 3^9 possible tic toe game board board which is roughly equal to 20000. But in these possible outcomes there are some irrelevant game play also has been counted like all X's and all O's so these have to be excluded. But if we take upper bound of the value we can say that the space complexity of the game tree is of the order of O(10^4)

B: Problem: 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 MINI MAX value backup argument on the game tree.

Rectified Problem: For the initial configuration of the game with three piles of objects as shown in Figure, show that regardless of the strategy of player-2, player-1 will always win.

Reason of Rectification:

Game of NIM depends on two factors:

- The player who starts first.
- The initial configuration of the piles/heaps.

As shown in figure 7 the initial configuration of stack is (10,7,9). We calculate Nimsum(The cumulative XOR value of the coins in stack at any moment in game.) to predict the winner of the game if players play optimally.

Winning Formula: "If both A and B play optimally (i.e-

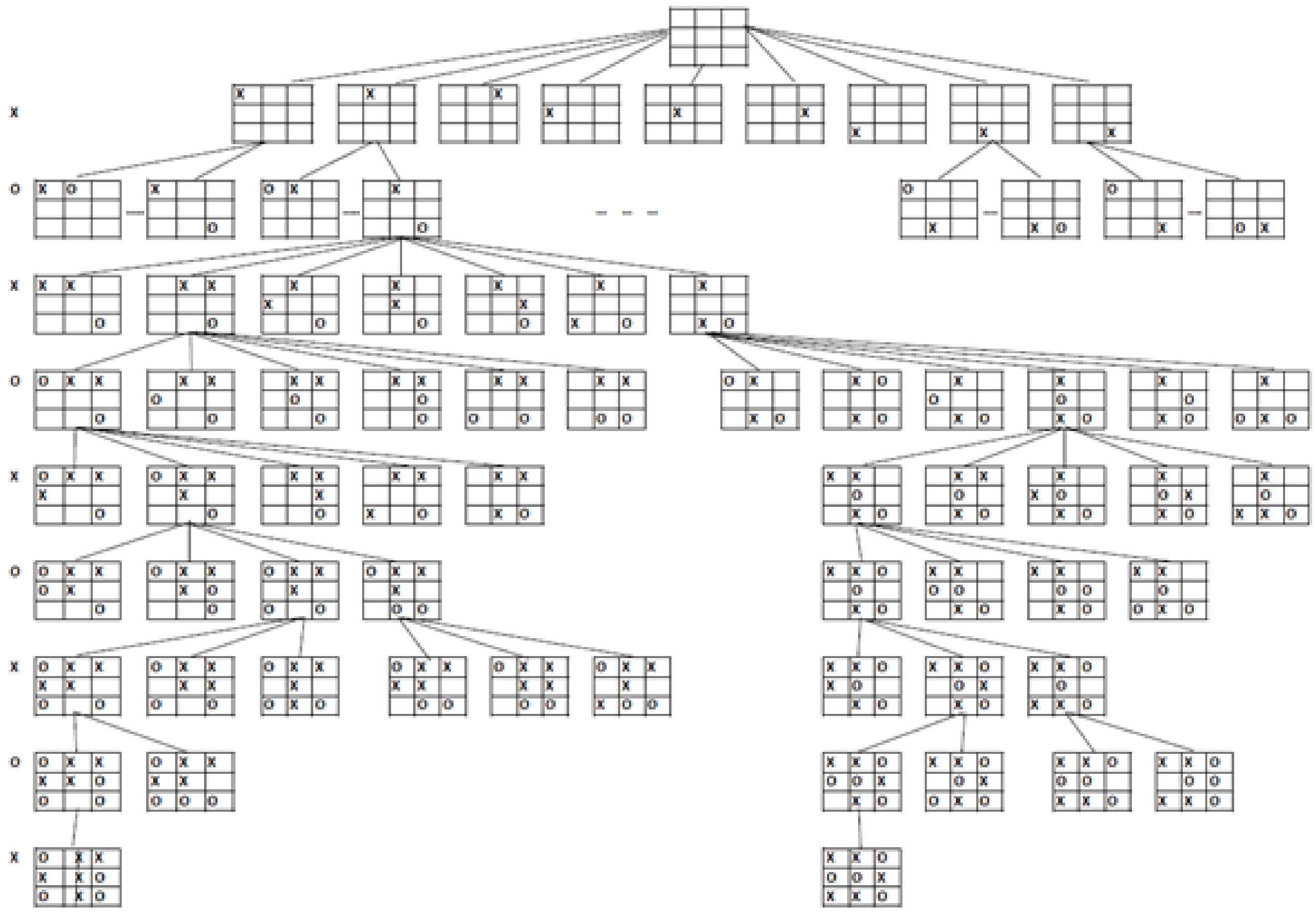


Fig. 6. Game Tree for Tic-Tac-Toe

Fig. 7. Game of NIM Stacks

they don't make any mistakes), then the player starting first is guaranteed to win if the Nim-Sum at the beginning of the game is non-zero. Otherwise, if the Nim-Sum evaluates to zero, then player A will lose definitely."

optimal Strategy: If the XOR sum of 'n' numbers is already zero then there is no possibility to make the XOR sum zero by single reduction of a number. If the XOR sum of 'n' numbers is non-zero then there is at least a single approach by which if you reduce a number, the XOR sum is zero.

initial Nimsum of 3 stacks (10,7,9)= 4

Initial Nimsum of coins in stacks is non zero so player-1 will win in all optimal moves, player-2 is going to win only if player-1 didn't move optimally.When player-1 start the game with optimal move, the player-2 will left with no optimal move so player-1 is going to win if he plays every move optimally.

Explanation For Player1 winning by Minimax

As explained above that Player1(optimally playing) will always win as initial configuration of tiles having non-zero value of XOR. And in NIM game, winning depends upon two factors:
1. First Move 2. Initial configuration of tiles
So, if we take game instance having non zero value of XOR for configuration and Player1 is making move then it will be same game play like for the Figure- 7. configuration.Suppose, our initial configuration is [1,0,2]
P1 move: Possible sets will be [0,0,2],[1,0,1],[1,0,0]
Here, optimal move is [1,0,1]
P2 move:[0,0,2]–>[0,0,1],[0,0,0] here,[0,0,0] is optimal for P2 and P2 will win for this move.
[1,0,1]–>[0,0,1],[1,0,0] here no move is optimal for P2 but optimal for P1
[1,0,0]–>[0,0,0] optimal move for P2 and P2 will win
P1 move:[0,0,1]–>[0,0,0] here P1 will win
[1,0,0]–>[0,0,0] here P1 will win

Now, we have assigned the values to nodes (1 for P1's win and 0 for P2's win) On assigning the values to nodes and on applying Minimax algorithm (backtracking the tree) we get the result as shown in below Figure- 8
As we can see that if P1 play move optimally then P1 will definitely Win no matter what the P2 took move.

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

As the game tree for Noughts and Crosses is very complex to draw so ultimately it is very difficult to explain with the help of full game tree. So, we have taken the particular instance at a particular stage for explaining minimax implementation and alpha-beta pruning. The Minimax algorithm in game tree is shown in: Figure on GitHub
Implementation of minimax
Implementation of alpha beta pruning
Results of above both implementation :Results As we can see that in the results, the number of evaluated nodes are less in-case of alpha-beta pruning compared to evaluated nodes in Minimax implementation for the same depth **Example:** At the depth of 7, for alpha-beta part evaluated nodes are 1030 while for minimax part it is 8231 for the configuration shown in :Results
D : Using recurrence relations 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.

To determine the exact value of a state, we need the exact value of one of its children, and bounds on the rest of its children. To determine a bound of a state's value, we will need the exact value of one of its children.
let S(k) be the minimum number of states to be considered k ply from a given state when we need to know the exact value of the state. Similarly, let R(k) be the minimum number of states to be considered k ply from a given state when we need to know a bound on the state's value. As usual, let b be the branching factor. Thus, we have:

```
S(k) = S(k  1) + (b  1)R(k  1)
i.e., the exact value of one child and
bounds on the rest, and
R(k) = S(k  1)
i.e., the exact value of one child.
The base case is
S(0) = R(0) = 1.
Note, for the above figure, this gives
S(3) = b2 + b  1 = 11
for b = 3.When we expand the recursive
equation, we get:
S(k) = S(k  1) + (b  1)R(k  1)
= (S(k2)+(b1)R(k2))+(b−1)S(k2)
= b S(k  2) + (b  1)R(k  2)
= b S(k  2) + (b  1) S(k  3)
It is obvious that S(k  3)<S(k  2),
```

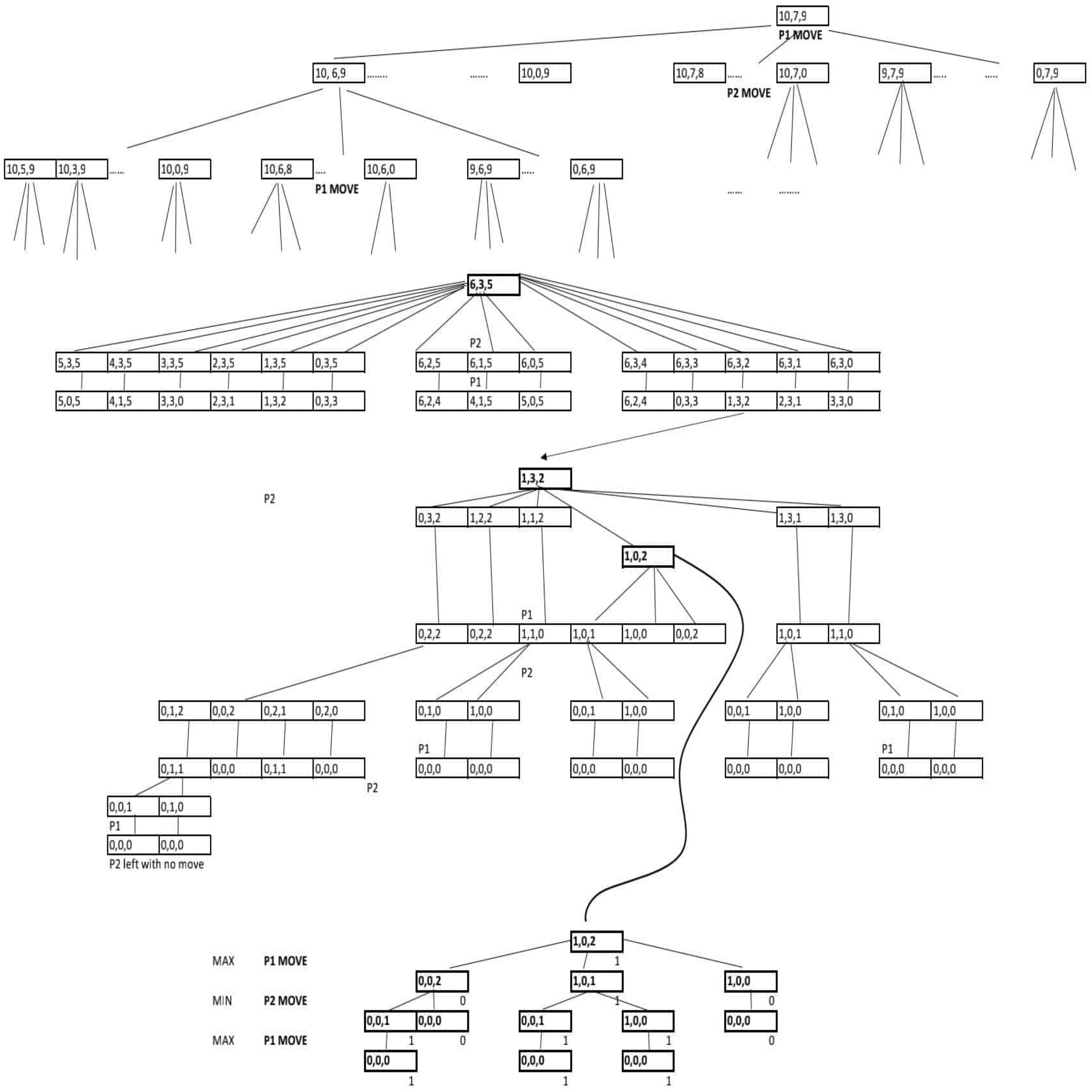


Fig. 8. Game Tree for Game Of NIM

```
so:S(k) < (2b  1) S(k  2)
S(k) < 2b S(k  2)
```

That is, the branching factor every two levels is less than 2b, which means the effective branching factor is less than $2b^{\frac{1}{2}}$. So, for even k, we derive S(k) ,which is not too far off the asymptotic upper bound of $((b^{\frac{1}{2}} + 1/2)^(k + 1))$. In effect, alpha-beta pruning can nearly double the depth that a game tree can be searched in comparison to straightforward minimax.

IV. Week 5: 8-12 February, 2021 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 given. **A:** Consider grades earned in each of the courses as random variables and learn the dependencies between courses.
We have applied hill climber and k2 score to get good structural dependency between courses among all Directed Acyclic Graphs.

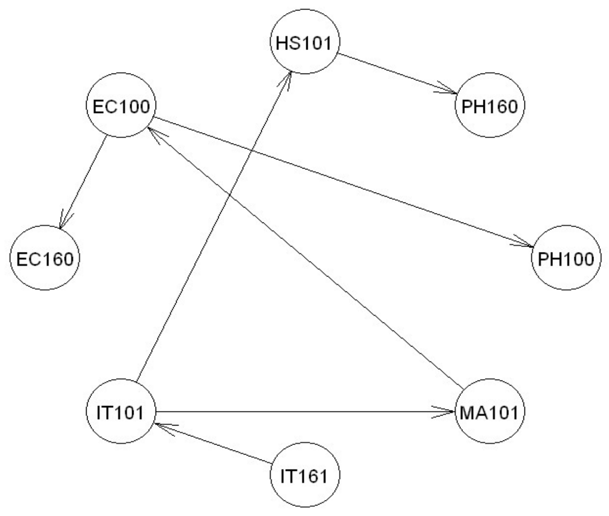


Fig. 9. Structural Dependency among Courses

B: Using the data, learn the CPTs for each course node. Link for CPTs for each course node.

C: What grade will a student get in PH100 if he earns DD in EC100, CC in IT101 and CD in MA101.
P(IT101,MA101,EC100,PH100)=P(IT101)*P(MA101|IT101) *P(EC100|MA101)*P(PH100|EC100)
The grade earn by student in PH100 on given other subject grades is "CD" and the joint probability of them is 0.00633.

D: 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.

In Naive Bayes, the courses are independent of each other, after applying it over 20 different random selection of training and test data, the average accuracy on testing data is 95.91%.

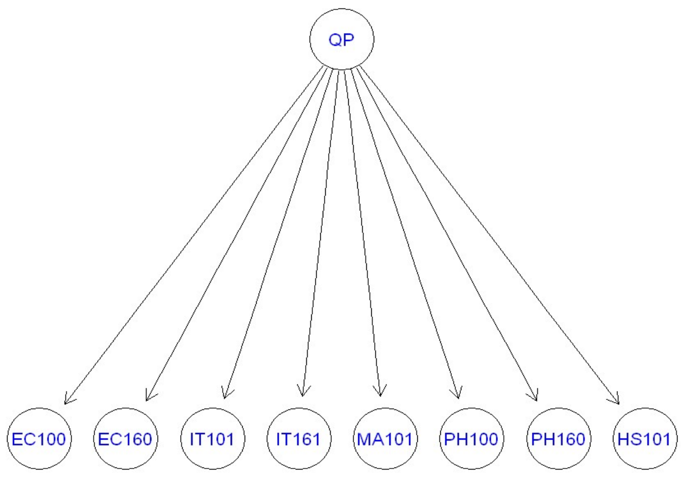


Fig. 10. Naive Bayes Structure of Courses

E: Repeat 4, considering that the grades earned in different courses may be dependent.

In this we consider dependency which means learning relationship between variables which reflected into factorization of joint probability and by fitting the model means finding out conditional probability numbers, we apply Chow-Liu's algorithm to learn structure and after over 20 different random selection of given training and test data, the average accuracy on testing data is 94.23%.

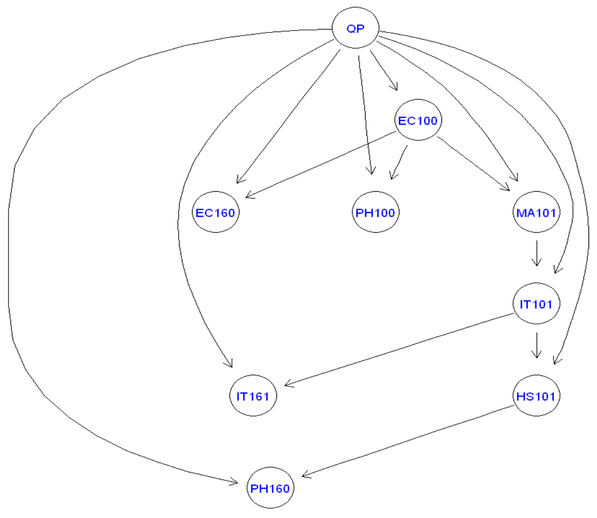


Fig. 11. Learned Dependency Structure of Courses

V. References

1) First Course in Artificial Intelligence, Khemani D., Tata McGraw Hill, 3014,Author: Khemani
2) https://en.wikipedia.org/wiki/Minimax
3) Artificial Intelligence – A Modern Approach, S. Russell, P. Norvig, Pearson Education, New Delhi, 2002.
4) http://gauss.inf.um.es/umur/xjurponencias/talleres/J3.pdf