CS362 Artificial Intelligence Midesem Lab Report

TEAM : Code_Brigrade

Bhavik Patel 202051134 | Suraj Poddar 202051186 | Tejas Gundale 202051191 | Tushar Maithani 202051194

***Abstract*—In this report, we completed lab assignment1,2,3 and 7.Solution for each lab discussed in deifferent sections, and their subproblems in respective subsevtion. Github repositories are provided at the end of the report for code review.**
Link to code repository

I. LAB ASSIGNMENT 1

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.

Pseudocode :

```
def Graph_Search(env, start, goal):
        frontier = PriorityQueue()
        explored = dict()
        start_node = agent(start.state,start.cost,none)
        frontier.push(start_node)
        while not frontier.is_empty():
            current = frontier.pop()
            if(current == goal):
                return true
            else
                add it to explored
                its childs to frontier(find all childs using env class)
        return false
```

Implementation Details :

1) frontier : This is the priority queue used to store the nodes to be explored.
2) explored : This is the dictionary which stores the explored nodes.
3) First of all we are initializing our start node with the help of agent passing some parameters.
4) Then we are running while loop till our frontier get empty.
5) In the loop we pop the element from frontier, check that is it our goal node.If true we return true
6) If poped state is not the goal state, then we will add it to explored and its childs to frontier.
7) Till getting frontier empty,if we never get goal state then return false.

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

C. Describe what is Iterative Deepening Search.

BFS takes less time but more memory. And in case of DFS it consumes more time ,less memory, but it not always able to find goal state. Also DFS can stuck into infinite loop as it never keeps record of visited node.

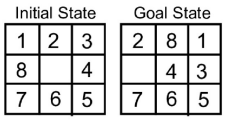

Fig. 1. 8-puzzle

-In function we have took one state ,depth as input.
-Then we are searching for blank space and storing it in tuple.

```
Space(0,0)
 for i in range(3):
   for j in range(3):
        if(state[i,j] == '_':
            space = (i,j)
```

-now on the basis of blank position we are applying all possible swapping functions as follows.
(Basically we are swapping numbers)

```
If space[0] > 0 then we can move it up:
    new_state = copy(state)
    val = new_state[space[0], space[1]]
    new_state[space[0], space[1]]  = new_state[space[0]-1, space[1]]
    new_state[space[0]-1, space[1]] = val
if space[0] < 2 then we can move it down:
    new_state = copy(state)
    val = new_state[space[0], space[1]]
    new_state[space[0], space[1]]  = new_state[space[0]+1, space[1]]
    new_state[space[0]+1, space[1]] = val
if space[1] < 2 then we can move it right:
    new_state = copy(state)
    val = new_state[space[0], space[1]]
    new_state[space[0], space[1]] = new_state[space[0], space[1]+1]
    new_state[space[0], space[1]+1] = val
if space[1] > 0 then  we can move it left:
    new state = copy(state)  val = new_state[space[0], space[1]]
    new_state[space[0], space[1]] = new_state[space[0], space[1]-1]
    new_state[space[0], space[1]-1] = val
```

In depth limited search we supply a depth limit ‘l’, and treat all nodes at depth ‘l’ as if they had no successors.

But choosing such ‘l’ such that we never miss desirable node is challenging, this problem is solved by iterative deepening search.

In Iterative deepening search, it solve this problem by trying all values for ‘l’ starting from 0,then 1,then 2, so on until either a solution is found or depth limited search returns the failure.

Thus we will get appropriate ‘l’ such that we get our goal state. First we perform DFS till ‘l’ ,then BFS at depth ‘l’ in this way it reduces space complexity a lot(.ie same as DFS) with assurity of getting solution(i.e. completeness).
Time Complexity : O(b^d) when there is solution, or O(b^m)

when there is no solution.
Space Complexity : O(bd)

It is preferred uninformed search when state space is larger than provided memory and d is unknown.

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.

```
def Path_to_goal(start,goal,graph):
        stack = []          //stack to store path(backtracking)
        set = {}            //to store visited node
        stack.push(start)
        set.add(start)
        while(set.size() ne number of node) :
                current = stack.pop()
                if(current eq goal)
                        return stack content in reverse order
                else
                        push such a child state to stack which is not in set
                        if such state not exist then pop from stack
        return goal state not found
```

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

```
def generate_start_state(self,depth,goal_state):
    past_state = goal_state
    i=0
    while i!= depth:
        new_states = self.get_next_states(past_state)
        choice = np.random.randint(low=0, high=len(new_states))

        if np.array_equal(new_states[choice], past_state):
            continue

        past_state = new_states[choice]
        i+=1

    return past_state
```

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

```
0 8.1634521484375e-05 56.0
10 0.0005667829513549805 803.04
20 0.0024642467498779295 2906.4
30 0.020881495475769042 13380.64
40 0.14380372524261475 46206.72
50 0.23339185237884522 72383.36
```

As we can see from the above output table, as the depth increases while searching by the agent , cost time taken and the memory usage also increases exponentially.
Time Complexity : O(b^d)
Space Complexity : O(b^d)
Where b = branching factor ,d = depth

II. LAB ASSIGNMENT 2

Learning Objective: To understand the use of Heuristic function for reducing the size of the search space. Explore non-classical search algorithms for large problems.

A. Read about the game of marble solitaire. Figure shows the initial board configuration. The goal is to reach the board configuration where only one marble is left at the centre. To solve marble solitaire, (1) Implement priority queue based search considering path cost, (2) suggest two different heuristic functions with justification, (3) Implement best first search algorithm, (4) Implement A, (5) Compare the results of various search algorithms.*

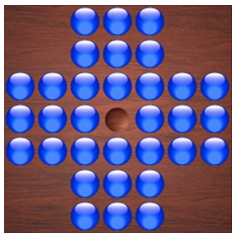

Fig. 2. Marble solitaire Game Board

Marble solitaire is a classic single-player board game. The objective of the game is to eliminate all but one marble from the board by making valid moves.

The game is played on a board with holes arranged in the shape of a plus sign (+). At the start of the game, all the holes on the board are filled with marbles except for one hole, which is left empty. The player then chooses a marble and makes a move by jumping over another marble into an empty space. The marble that was jumped over is removed from the board. The player continues to make moves, jumping over marbles until no more moves can be made. The game is won if only one marble remains on the board.

Here Marble solitaire board 33 holes, with the central hole being left empty. This variation is also known as English solitaire.

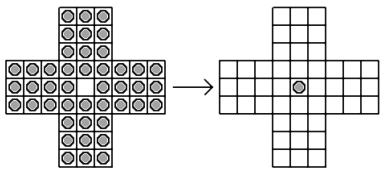

Fig. 3. Marble solitaire Game Board

B. Write a program to randomly generate k-SAT problems. The program must accept values for k, m the number of clauses in the formula, and n the number of variables. Each clause of length k must contain distinct variables or their negation. Instances generated by this algorithm belong to fixed clause length models of SAT and are known as uniform random k-SAT problems.

k-SAT (k-Satisfiability) is a type of Boolean satisfiability problem where the goal is to find a satisfying assignment for a given Boolean formula in conjunctive normal form (CNF) where each clause contains at most k literals (i.e., Boolean variables or their negations).

The k-SAT problem is a well-known NP-complete problem, meaning that it is believed to be computationally intractable to solve in general. This means that, for larger problem sizes, it can be very difficult (or even impossible) to find an algorithm that can solve the problem efficiently in all cases.

The problems are generated depending on the values of k, m and n entered by the user. A set of all possible problems is displayed in the output.

C. Write programs to solve a set of uniform random 3-SAT problems for different combinations of m and n, and compare their performance. Try the Hill-Climbing, Beam-Search with beam widths 3 and 4, Variable-Neighborhood-Descent with 3 neighborhood functions. Use two different heuristic functions and compare them with respect to penetrance.

Hill-Climbing : Hill-climbing is a simple optimization algorithm that starts with an initial solution and iteratively improves it by making small modifications in the hope of finding a better solution. The basic idea is to move in the direction of the steepest ascent or descent, depending on whether the objective is to maximize or minimize a function.

Pseudo-code:

1. Initialize current solution s
2. Initialize best solution s_best = s
3. Repeat until a stopping condition is met:
   a. Generate a set of neighboring solutions N(s)
   b. Select the best neighboring solution s' from N(s) based on the objective function
   c. If f(s') > f(s_best), set s_best = s'
   d. If f(s') > f(s), set s = s'
   e. If there are no better solutions in N(s), terminate and return s_best

III. LAB ASSIGNMENT 3

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.]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.

Simulated Annealing :

Simulated annealing is a metaheuristic optimization algorithm inspired by the annealing process in metallurgy. It is used to find an optimal solution to a problem by iteratively searching for the global minimum of a cost function.

The algorithm starts with an initial solution, and then generates a new candidate solution by making a small random perturbation to the current solution. The cost of the new solution is then calculated and compared to the cost of the current solution. If the new solution has a lower cost, it is accepted as the new current solution. If the new solution has a higher cost, it may still be accepted with a certain probability determined by a temperature parameter and the difference in cost between the two solutions.

The temperature parameter is gradually decreased over time, which means that the algorithm becomes less likely to accept worse solutions as it progresses. This gradual cooling process is what gives simulated annealing its name, as it is similar to the process of cooling a metal and allowing it to anneal and settle into a more stable state.

Algorithm 1 Simulated Annealing Function

$path \leftarrow initialPath$
$cost \leftarrow getCost(path)$
$T_m \leftarrow$ initial temperature
$coolingFactor \leftarrow$ initial value
$itr \leftarrow 0$
while $itr \neq itrMax$ **do**
    $nextPath \leftarrow getNeighbour(path)$
    $nextCost \leftarrow calculateCost(nextPath)$
    $\Delta E \leftarrow cost - nextCost$
    $T \leftarrow T_m * coolingFactor$
    if T is too low ($\leq 10^{-6}$) **then**
        break loop
    end if
    if $\Delta E > 0$ **then**
        $path \leftarrow nextpath$
    else if $random(0,1) < 1/(1+e^{\frac{-\Delta E}{T}})$ **then**
        $path \leftarrow nextPath$
    end if
    $itr \leftarrow itr + 1$
end while
return $path$

Algorithm 2 Tour Cost Function

$distance \leftarrow 0$
$itr \leftarrow 0$
while $itr \neq path.length() - 1$ **do**
    $distance \leftarrow distance+$ distance b/w node $path[i]$ and $path[j]$
    $distance \leftarrow distance+$ distance b/w last node and first node
end while
return distance

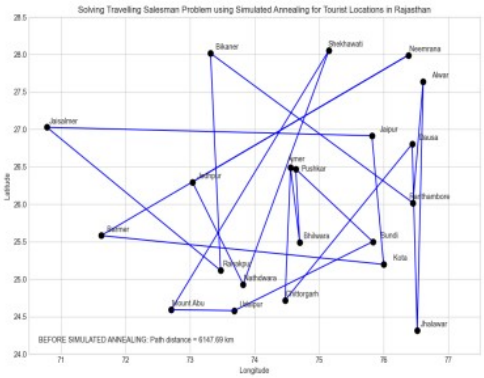

Fig. 4. Before applying simulated annealing :

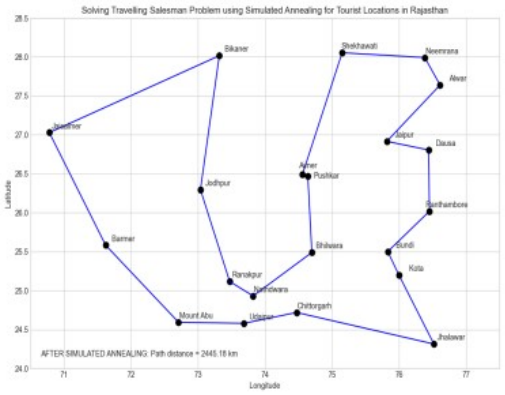

Fig. 5. After applying simulated annealing :

IV. LAB ASSIGNMENT 7

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.

MRF Image Denoising :

We began the MRF image denoising 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 [1].The image is a 512*512 grayscale image and we normalize it's pixesl values to between 0 and 1, by dividing all the 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. 4 and fig. 5. After this we add noise to the image. For the purpose of testing the capability by Markov Random Field, we add varying levels of noise, from 5The varying levels of noises are shown in fig. 6. MRF 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:

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

where v is the sum of the smooth ID signal u and E is the energy function.

This is because for most cases, the values around a pixel are close to the pixel value. [2], [3].We use the value of the constant lambdas as -100, while computing the quadratic potential function. The max denoised pixel levels are given below in table 1. The denoised images with MRF are shown in fig 7.

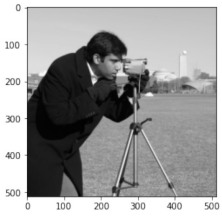

Fig. 6. Original Cameraman Image

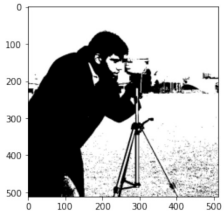

Fig. 7. Binarsed Cameraman Image

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.

Hopfield networks are a special kind of recurrent neural networks that can be used as associative memory. Associative memory is memory that is addressed through its contents. That is, if a pattern is presented to an associative memory, it returns whether this pattern coincides with a stored pattern.

That is if we add a certain amount (tolerable) of error to any of the patterns that is known to the network , then it is able to retrieve the original pattern using the Hopfield Network.

Hebbian learning :

A simple model due to Donald Hebb (1949) captures the idea of associative memory. Imagine that the weights between neurons whose activities are positively correlated are

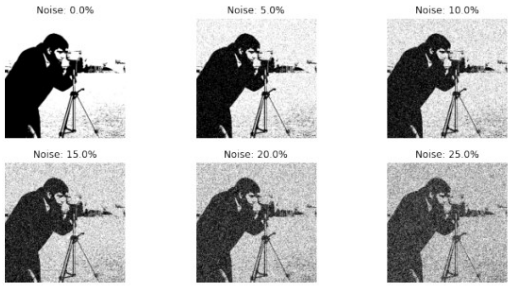

Fig. 8. Varying Image Noises

% noise Level	% pixels denoised
5	3.88
10	6.26
15	8.73
20	11.09
25	13.52

Fig. 9. Denoised images using MRF

increased:

$$\frac{\mathrm{d}w_{ij}}{\mathrm{d}t} \sim \mathrm{Correlation}(x_i, x_j).$$

Binary Hopfield network :

Convention for weights. Our convention in general will be that wij denotes the connection from neuron j to neuron i. Architecture:- A Hopfield network consists of I neurons. They are fully connected through symmetric, bidirectional connections with weights wij = wji . There are no self-connections, so wii = 0 for all i. Biases wi0 may be included (these may be viewed as weights from a neuron '0' whose activity is permanently x0 = 1). We will denote the activity of neuron i (its output) by xi

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

Travelling Salesman problem is a very famous NP-hard problem. We solve this here by using the Hopfield Network. Since in a Hopfield Network, each node is connected to each other node, we needed a total of 10x10 = 100 weights because of 10 cities. We first generate 10 cities randomly, as shown in fig. 8. And then we used the hopfield network to predict a path with optimal least path cost. The path that we got is shown in fig. 9.

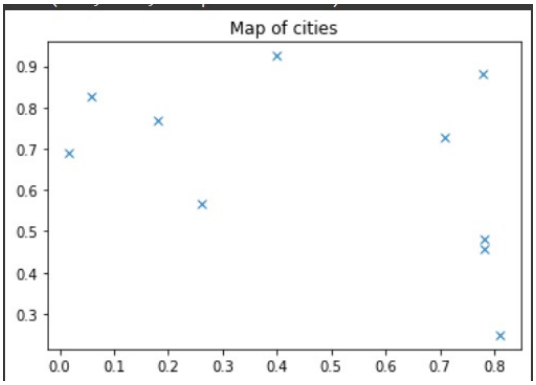

Fig. 10. Input

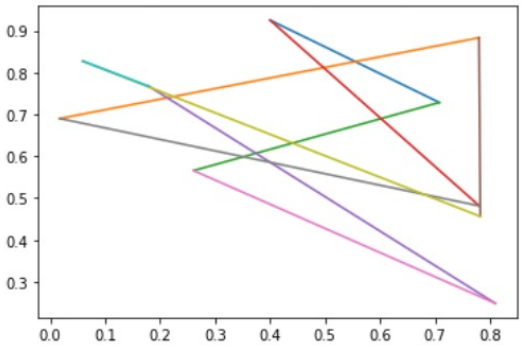

Fig. 11. Output

V. Conclusion :

In the first lab we have designed graph search agent for 8-puzzle problem.In this lab we came to know about use of different data structures required for state space search.In second lab we understood how our heuristic function will drastically decrease size of state space search and explored non-classical search algorithm. In third lab we are done with simulated annealing.In seventh lab we explored about Markov Random Field and understood working of Hopfield network.

References

[1] Artificial Intelligence: a Modern Approach, Russell and Norvig (Fourth edition)
[2] Khemani, D., 2013. A first course in artificial intelligence. McGraw-Hill Education.
[3] https://www.geeksforgeeks.org/simulated-annealing/
[4] https://web.cs.hacettepe.edu.tr/~erkut/bil717.s12/w11a-mrf.pdf
[5] http://www.inference.phy.cam.ac.uk/mackay/itila/