Artificial Intelligence Lab Report 2

1st Dipean Dasgupta
202151188
BTech CSE
IIIT,Vadodara

2nd Shobhit Gupta
202151149
BTech CSE
IIIT,Vadodara

3rd Rahul Rathore
202151126
BTech CSE
IIIT,Vadodara

4th Rohan Deshpande
202151133
BTech CSE
IIIT,Vadodara

***Abstract*—To understand the use of Heuristic function for reducing the size of the search space. Explore non-classical search algorithms for large problems.**

I. Introduction

The Problem statements of the lab are as follows:

- Problem Statement: 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.
- 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.
- 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.

II. Solution To problem Statements

A. Problem A

The goal of marble solitaire is to remove all of the marbles from the board by leaping over them with other marbles until there are no more hops left. The objective is to clear as many marbles from the board as you can until there is just one marble remaining in the center. By approaching the game as a search issue, where the objective state is a board with only one marble remaining in the center and the actions are moves that remove one marble by leaping over another marble, we can use search algorithms to solve the puzzle of marble solitaire.
Priority queue based search considering path cost:
A* or Uniform Cost Search are two priority queue-based search algorithms that we can use to find the shortest path to the goal state based on path cost. The cost of the journey can be defined as the number of moves required to reach the goal state. The search method enlarges nodes based on their path costs, starting with the nodes with the lowest travel costs.
Two different heuristic functions with justification:
The search process can be directed toward the goal state by using heuristic functions to determine how far the current state is from the desired state. For marble solitaire, two potential heuristic functions are:
Number of marbles remaining: One possible heuristic function is the quantity of marbles left on the board. The heuristic function yields a result of 1 for the target state and the number of marbles left for other states since the goal state has just one marble left. Since this heuristic never overestimates the true cost of achieving the objective state, it is acceptable.
*Manhattan distance:*Another possible heuristic function is the Manhattan distance, which measures the separation between the center of the board and the location of the last stone. This heuristic calculates the bare minimum of moves needed to get the last marble to the board's center. Since this heuristic never overestimates the true cost of achieving the objective state, it is also acceptable.
Best First Search algorithm:
Best First Search develops nodes based on the output of a heuristic function. The method selects the node with the lowest heuristic value, beginning from the initial state, as the next node to grow. The search continues until the target state is located.
A* algorithm:
The path cost and a heuristic function are combined by the A* algorithm, an educated search algorithm, to direct the search towards the desired state. Nodes are expanded by the algorithm by adding the path cost and the heuristic value. Until the desired state is attained, the quest never stops. Provided that the heuristic function is acceptable and consistent, A* will always find the best answer.
Comparison of search algorithms:
It is anticipated that A* will identify the best solution in the quickest amount of time, with Best First Search, Uniform Cost Search, and Breadth First Search following suit. It is anticipated that the Manhattan distance heuristic will yield superior outcomes to the number of marbles left heuristic. However, based on the specific issue instance and how the algorithms are implemented, the actual outcomes could differ.

B. Problem B

Pseudo-code:
$\rightarrow$ Three parameters are passed to the function: n (the total number of variables), m (the number of clauses), and k (the number of variables per clause).
$\rightarrow$ To begin, an empty list of clauses is created.
$\rightarrow$ Next, we construct each clause one at a time using a loop.
$\rightarrow$ For the present sentence, we construct an empty list inside the loop
$\rightarrow$. Until the phrase contains k distinct literals, we add literals to it using a while loop. We select a literal at random from the range -n to n (inclusive) for every while loop iteration.
$\rightarrow$ We go on to the following iteration of loop
if the chosen literal is zero or already occurs in the phrase (or its negation does).
$\rightarrow$ Otherwise, the selected literal are added to the clause
$\rightarrow$ Once a clause is generated with k distinct literals, we add it to the list of clauses.
$\rightarrow$ Finally, we return the list of generated clauses
This strategy will yield a uniformly random k-SAT problem with the given values. Each clause will have k distinct variables or their negations.

C. Problem C

Hill-Climbing:To check if any adjustments can enhance the quality of the answer, start with a random initial solution and iteratively make tiny changes to the solution. Continue until there is no more room for improvement.
Beam-Search:Keep track of k potential solutions, where k is the beam width. From the existing set of solutions, generate k new candidate solutions by slightly altering each one. Save the top k answers and keep trying until there is no more room for improvement.
Variable-Neighborhood-Descent: Apply a series of neighborhood functions of progressively larger sizes after beginning with a random initial answer on multiple occasions. Choose the neighborhood's best answer at each stage, then proceed to the next one. Continue until there is no more room for improvement.
Two different heuristic functions possible:
Minimum Remaining Values: The variable that appears in the fewest unsatisfied sentences in the current state is chosen by the MRV heuristic because it is most likely to have a greater effect on the number of satisfied clauses when flipped. To apply this heuristic, count how many unsatisfied clauses there are for each variable, then choose the variable with the lowest count.

Least Constraining Value (LCV) heuristic: The variable whose value (True or False) leaves the most alternatives for the other variables in the clause is chosen by the LCV heuristic. To apply this heuristic, count how many sentences are satisfied for each value of the variable and choose the value that fulfills the greatest number of clauses. The value that removes the fewest alternatives for the remaining variables might be chosen to further refine the heuristic in the event of ties.

REFERENCES

[1] S. Russell and P. Norvig, “Artificial Intelligence: a Modern Approach,” 4th ed.,Pearson.
[2] Deepak Khemani, A first course in Artificial Intelligence, 2nd ed., McGraw Hill.