Assignment 1: Peg Solitaire Solver

Santosh Kumar Ghosh SBU ID: 109770622

The Problem:

Peg Solitaire is a classic board game that can be modlled as a search problem. The game has many versions but we are dealing with a 7 X 7 board with 33 hole sin it. There are pegs on the board and the goal is to start from a given configuration and end up with one peg at the center hole. There are certain restrictions on the way a peg can move.

This game is solved using classic search algorithms A* and Iterative Deepening.

Solving Peg Solitaire Using A*

This is an informed search algorithm that provides a direction to the search by using a heuristic. A heuristic gives information to the search algorithm by informing it how close it is to the goal state. The quality of the heuristic depends on how well it predicts if the goal state can be reached from the given state. The algorithm starts at the board input configuration and generates all its successors. It applies the heuristic to all the generated successors and adds them to a queue. It then chooses the best among all the nodes in the queue and continues the search in the same manner till the goal state is reached or no further moves can be made.

Heuristics Used:

In this project two heuristics were used to guide the search:

1. Manhattan Distance:
2. Weighted Cost Matrix

Manhattan distance:

It is the summation of the vertical and horizontal distances between two points on a Euclidian plane. At each iteration while generating a neighbor the Manhattan distance of each peg was calculated from the center (3, 3) of the board.

Intuition behind using the heuristic:

The aim of the game is to bring the peripheral pegs near the center. So larger the Manhattan distance of a configuration from the center, farther it is from the goal. So that have a higher cost and hence a lower chance of getting selected next.

Weighted Cost Matrix:

In this approach we assign weights to the holes on the board based on how solvable they are. Specifically the following weights were used:

Intuition behind using the heuristic:

Our goal is to assign weights based on how much a peg contributes to the board being unsolvable. Since corners have the lowest degree of freedom, they have the maximum chances of making a board unsolvable and hence we assign them the highest weight. The central hole is given a weight of 1 to make sure that there is no peg on it, to pave way for another peg, towards the end. Since a peg in position with weight 3 in the board above has a 50% probability of moving towards the corner and another 50% of moving towards another 3-weighted hole, we have assigned it a high weightage of 3.
Given a choice between two states, one with the total weight lower than the other, we would naturally choose the one with lower weight, since it provides us with the higher probability of moving towards a solution.

-	-	4	0	4	-	-
-	-	0	0	0	-	-
4	0	3	0	3	0	4
0	0	0	1	0	0	0
4	0	3	0	3	0	4
-	-	0	0	0	-	-
-	-	4	0	4	-	-

Performance of A*

The following performance was observed in A Star:

Heuristic Used	Start State	Nodes Expanded	Time Consumed	Memory Needed
Manhattan Distance	--000-- --0X0-- 00XXX00 000X000 000X000 --000-- --000--	18	.00678896903992 seconds	128 Mb
	--0X0-- --X0X-- 0000000 XXX00XX 00000X0 --XX0-- --XXX--	7209	31.087334156 seconds	587Mb

Heuristic Used	Start State	Nodes Expanded	Time Consumed	Memory Needed
Weighted Cost Matrix	--000-- --0X0-- 00XXX00 000X000 000X000 --000-- --000--	14	.00764083862305 seconds	110 Mb
	--0X0-- --X0X-- 0000000 XXX00XX 00000X0 --XX0-- --XXX--	513	0.254135847092 seconds	320 Mb

Solving peg solitaire using Iterative Depth First:

Approach:

This is a uninformed search technique which is a of the classic Breadth First and Depth First search techniques. It has the best of both worlds i.e. linear space requirement from DFS and completeness of BFS.

Intuition:

Here we do a depth limited DFS with increasing depths until we find a goal. We start with a depth limit of 0 which searches the first level and then go on doing a DFS with increasing depth bounds.

Performance of IDS search:

The following statistics were calculated:

Start State	Number Of Nodes Expanded	Time Consumed	Memory Needed
--000-- --0X0-- 00XXX00 000X000 000X000 --000-- --000--	46	0.000154972076416 seconds	46 Mb

Start State	Number Of Nodes Expanded	Time Consumed	Memory Needed
--0X0-- --X0X-- 0000000 XXX00XX 00000X0 --XX0-- --XXX--	1129	0.000140190124512 seconds	340 Mb

Code Logic:

The code consists of 4 modules:

1) IDS.py – Used for doing iterative deepening search
2) aStar.py –Used for doing A Star search
3) Board.py –Used to represent the board configurations
4) PegSolitaireSolver.py –Entry point to the program

The module named IDS.py does the iterative depening search. Here two lists have been maintained an openList that keeps track of the frontier and a closedList that is used for keeping track of the nodes actually expanded. openList helps us avoid loops as each time a neighbor is generated it is checked to see if it is already in the open list. After everything has been done the closedList is iterated over to get the path.

The module aStar.py does the A Star search

Each state of the board is represented by a Board object declared in the Board.py module. This object contains the following information:

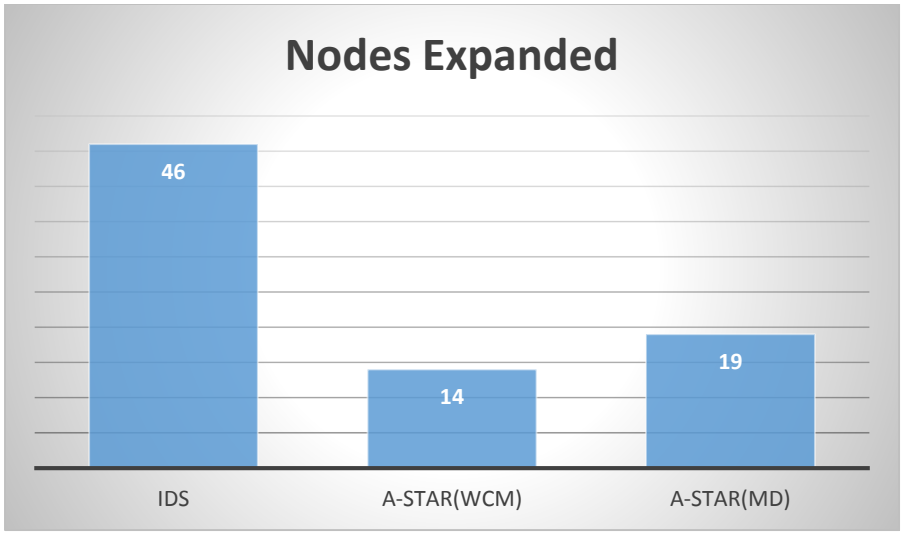
1) A string representation of the board state
2) The parent of the node
3) The path cost of this node from start state
4) The heuristic cost of this node
5) The movement info (initial Position and final position)

With this info we can easily describe each state of the world in this game.

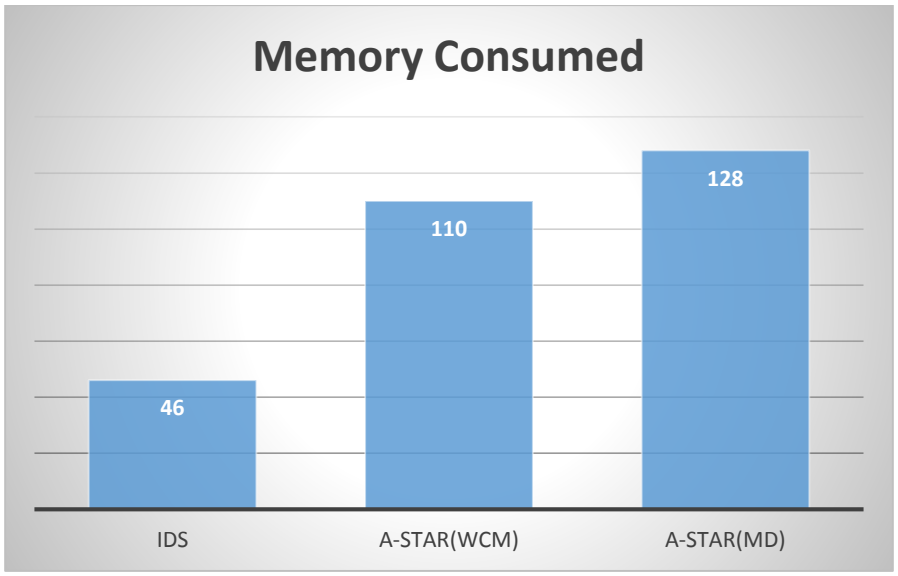
Comparative Study Observations:

From a comparative study of the 3 algorithms we can see that IDS search takes a lot less space than A Star. This is in line with the theoretical claim of linear spave required by IDS. On the other hand the number of nodes needed to be expanded and hence the memory needed to run the program is less in case of the AStar search. The following graphs were generated for the start state :

< --000----0X0--00XXX00000X000000X000—000----000-- >



Nodes Expanded Profile



Memory Footprint