CSE4082 – Artificial Intelligence

Project 1 Report

Malik Türkoğlu

Introduction

The main goal of this project is to compare given searching algorithms by running them on the same problem which is a diagonal 15-puzzle. The given algorithms are Uniform Cost Search (UCS), Iterative Lengthening Search (ILS), and A* search algorithm. For A*, two admissible heuristic functions are given in project document, which are H1 and H2.

In order to achieve testing, we came up with a random path generator which generates a path with a solution at an adjustable depth. This generator avoids producing cycles.

Test Results

Test results are provided below. Results are arithmetic average of 10 random instances.

The total number of expanded nodes

Depth	UCS	ILS	H1	H2
2	44	81	4,4	2,5
4	1628	2831	23	11,2
6	33.294,25	86.704	227,25	60,5
8	181.425,4	477.701,8	407,7	89,3
10	-	2.283.084	1151,5	146,5
12	-	-	26.862,5	804,25
16	-	-	194.573,8	1829
20	-	-	591.520	3568
24	-	-	-	33.939,43
28	-	-	-	160.887,9

The maximum number of nodes stored in memory

DEPTH	UCS	ILS	H1	H2
2	156,6	53,1	20	13,1
4	5221	1227,6	91,4	45,8
6	96.589,75	31.692	790,75	227,25
8	488.897,9	129.479,9	1364,9	284,9
10	-	1.050.667	4289	677
12	-	-	77.319	2616
16	-	-	520.593	5805,2
20	-	-	1.499.247	11.183,4
24	-	-	-	82.075,71
28	-	-	-	420.523,3

It is clear as a day some cells are empty. It is because the algorithm cannot find the optimal solution with the resources it has. For example, at depth 10, UCS filled up all 16GB RAM and found nothing as expected.

Test Results of Special Inputs

This part presents the results of special inputs which are given in project document.
The columns and their meanings:

- 1-cost: The cost of the solution found.
- 2-path: The solution path itself.
- 3-expand: The number of expanded nodes.

The meanings of the letters at the column 2-path:

- L: Left
- R: Right
- U: Up
- D: Down
- DR: Down-right
- DL: Down-left

- UR: Up-right
- UL: Up-left

	A PART			B PART			C PART		
	1-cost	2-path	3-expandd	1	2-path	3	1-cost	2-path	3-expandd
UCS	7	R-DR-DL	561	-	-	-	-	-	-
ILS	7	R-DR-DL	1238	-	-	-	-	-	-
H1	7	R-DR-DL	16	17	R-UR-UR-UL-L-DL-DR	1520	20	U-L-D-L-U-L-UR-U-DR-D-R-DL-L-U	22372
H2	7	R-DR-DL	9	17	R-UR-UR-UL-L-DL-DR	80	20	U-L-D-L-U-U-U-DR-D-R-D-L-L-UL-R	331

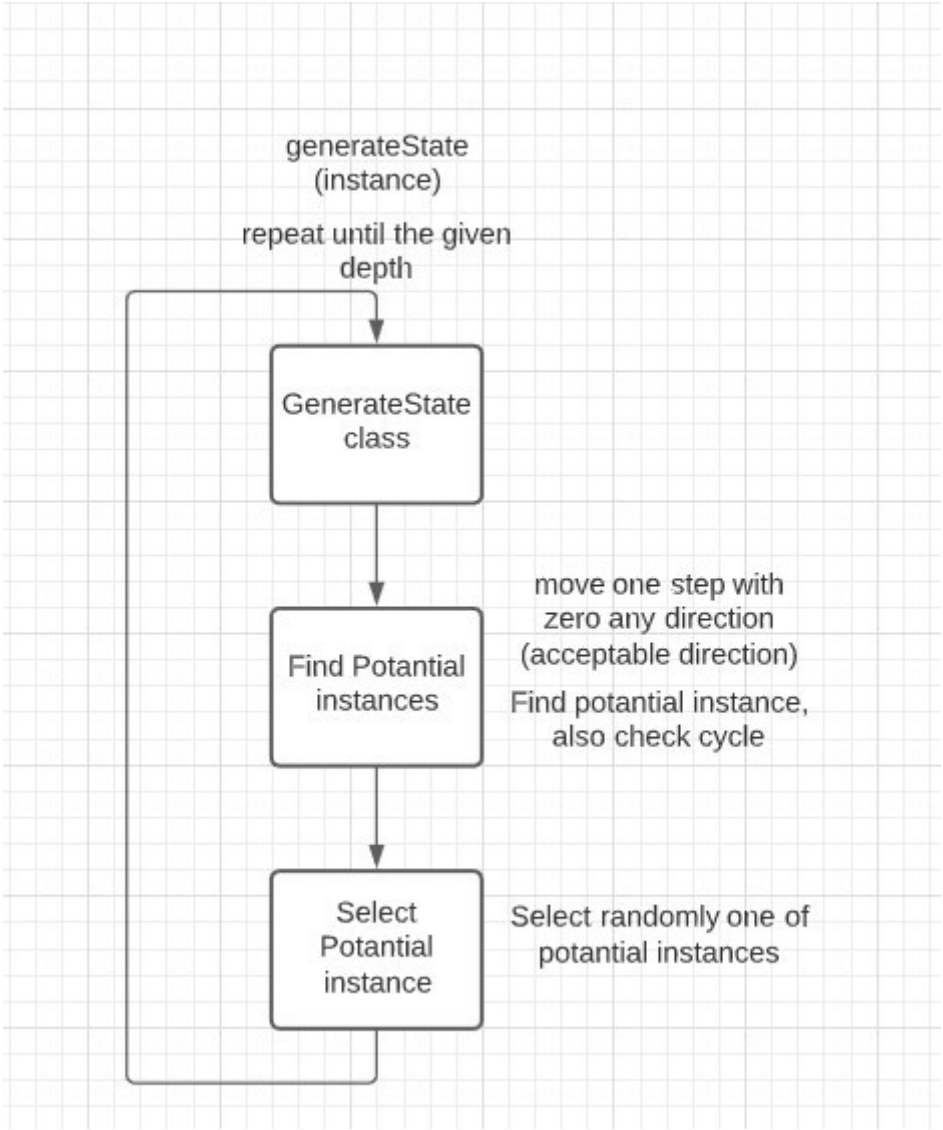
There are missing parts. It is the same reason as in the previous test results; the algorithm could not find the solution.

Classes and Explanations

The classes used in this project will be explained in short by conveying the main task that defines what they.

GenerateState Class

This class produces random instances with given depth value. The class moves the zero tile, beginning with the goal state. It checks every single move. At first, the class checks if the current move can be made by checking that the tile cannot go beyond the borders. Thereafter, it checks whether that move is done before. Hence, it prevents producing cycles. A simple flow diagram is presented below.


generateState
(instance)
repeat until the given
depth
GenerateState
class
Find Potantial
instances
move one step with
zero any direction
(acceptable direction)
Find potantial instance,
also check cycle
Select
Potantial
instance
Select randomly one of
potantial instances

Node Class

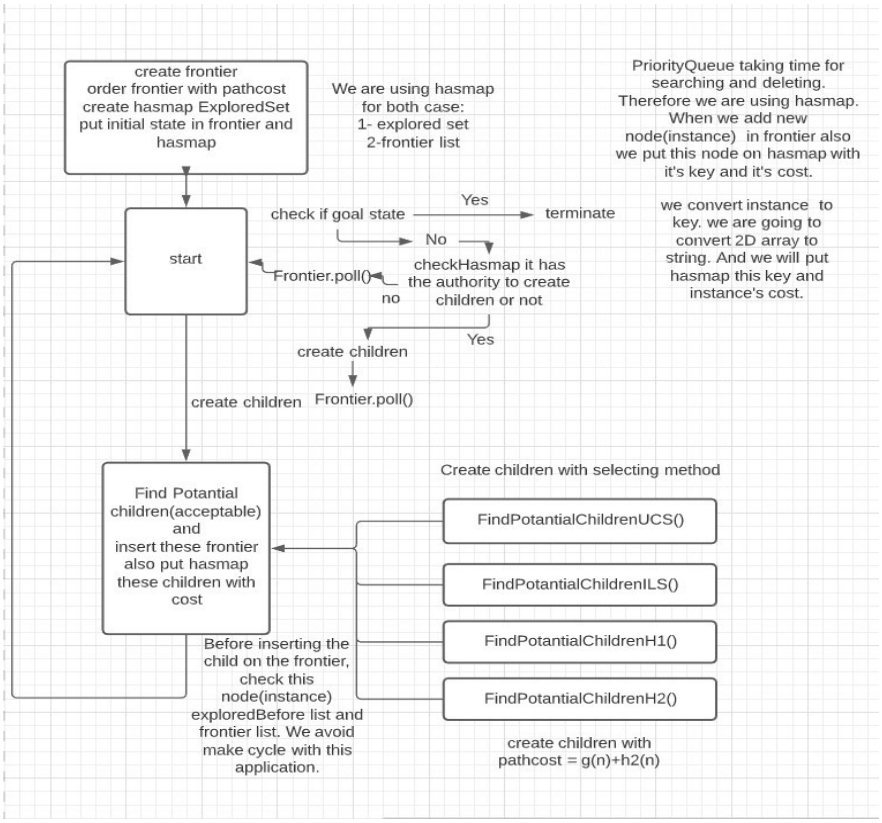
This class simply creates nodes. Nodes includes every information including depth, path cost, parent, state info etc. Calculations and defining information are done by node class. For example, it calculates which direction can the node can follow by using findPotentialChildren method. This method first checks that if the potential child can be reach from the current state. If so, it looks into the frontier and explored sets to see if the child does already count. The decision of putting the node in the frontier is based on two facts: 1) if the node is not included in neither frontier nor explored set, 2) if the node is in the frontier but the path cost is smaller. The search of checking the smallest path cost for a node is done by the help of another structure which is built as Hash map. It allows the algorithm to rapidly check for the path cost and consequently speeds up the entire work.

GraphSearch Class

As the name indicates, this class includes methods for every search algorithm. The methods for algorithms are listed below:

- UCS: startUcs
- ILS: startILS
- A*-H1: startHeuristicSearchH1
- A*-H2: startHeuristicSearchH2

Heuristics are implemented and some calculations are done in this class, as well. For example, it calculates maximum number of nodes stored in the memory. The figure below is a simple flow diagram for both GraphSearch and Node classes.


create frontier
order frontier with pathcost
create hasmap ExploredSet
put initial state in frontier and hasmap
We are using hasmap for both case:
1- explored set
2-frontier list
PriorityQueue taking time for searching and deleting. Therefore we are using hasmap. When we add new node(instance) in frontier also we put this node on hasmap with it's key and it's cost.
we convert instance to key. we are going to convert 2D array to string. And we will put hasmap this key and instance's cost.
start
check if goal state
Yes
terminate
No
Frontier.poll()
checkHasmap it has the authority to create children or not
no
Yes
create children
create children
Frontier.poll()
Find Potantial children(acceptable) and insert these frontier also put hasmap these children with cost
Create children with selecting method
FindPotantialChildrenUCS()
FindPotantialChildrenILS()
FindPotantialChildrenH1()
FindPotantialChildrenH2()
Before inserting the child on the frontier, check this node(instance) exploredBefore list and frontier list. We avoid make cycle with this application.
create children with pathcost = g(n)+h2(n)