AI Lab 2 Report

Om Patil (200010036)
Hrishikesh Pable (200010037)

December 2021

1 Implementation Overview

1.1 State Space

In our implementation of Best First Search and Hill Climbing, we dynamically create $blockWorld$ objects for each state generated via the $MoveGen()$ function. These $blockWorld$ objects stores the current state of the Block World, the parent object of the current Block World (which acts as a back pointer to the parent), and the *move* that lead to the current state from its parent.

The current state of each individual Block World object is stored in the form of a 2D array in which the outer array (The array of 1D arrays) represents the table and inner 1D arrays represent stacks. Each stack has character elements which represent blocks in order from bottom to top.

Due to the way in which states are stored, states are considered to be unique even if they are just permutations of the inner stacks.

1.2 Start Node

When the input file is parsed, we create the first $blockWorld$ object, which is the start node. It contains a 2D state array which is the initial configuration of the blocks on the table. Its parent and move attributes are set to $None$ (as it is the first node).

1.2.1 Goal Node

The second part of the input file contains the configuration of the Goal State.The goal state must be reference repeatedly in order to calculate the heuristic of each state. Hence this information is stored independently in the $BlockDict$.

In the implementation, the $BlockDict$ is generated from the second part of the input file. To do this we first create *block* objects corresponding to each block in the Goal State. These object store the goal position (ie. $goalStack$ and $goalLevel$). $BlockDict$ is a dictionary with keys as the names (character) of the blocks and the value as the corresponding *block* objects.

The actual goal node is not known since we are using uninformed search. In the process of exploration when we arrive at a node which has the same heuristic as the goal the $goalTest()$ function concludes that this node is the goal node.

1.3 Explore Function

The Explore Function is comprised of two parts $GoalTest()$ and $MoveGen()$:

1.3.1 GoalTest()

In our implementation all the heuristic functions are designed to return a measure of the difference between the state and the Goal State, ie. they are normalised to give zero when current state matches the Goal State. Hence the $GoalTest()$ function simply concludes that the state is the goal if it's heuristic is equal to zero and vice-a-versa.

1.3.2 MoveGen()

MoveGen generates the neighbours of the current state by moving the topmost block from any one of the stacks in the current state to the top of some other stack.

The heuristic value helps in reducing the difference in the current state and goal state, and ultimately reaching the goal state.

2 Exploration Methodology

2.1 Best First Search

In this methodology we push all states generated by $MoveGen()$ (which are not in the explored array) in the $frontier$ priority queue. Nodes with least heuristic are popped from the $frontier$ and then subsequently explored until either goal is found or $frontier$ is exhausted.

2.2 Hill Climb Approach

This approach is similar to Best First Search, only difference being that, at the start of every individual node exploration, the frontier is emptied, and the neighbours are pushed into the $frontier$. So, we end up exploring only the best neighbour of every node (based on heuristic).

3 Heuristics

3.1 Manhattan

In this method, the heuristic is calculated as the sum of the Manhattan distances between each block's current state position and goal state position.

The Manhattan distance is calculated as the sum of the absolute differences between each coordinate of two different positions.

3.1.1 Euclidean

In this method, the heuristic is calculated as the sum of the Euclidean distances between each block's current state position and goal state position.

The Euclidean distance between two points is the length of a line segment between the two points, which can be calculated using Pythagoras Theorem.

3.2 RelPos

In this method, if the block below the current block is the same as that in the Goal State, the heuristic value is incremented by value equal to the height of the block in the stack, and if not, then the heuristic is decremented by the same value. This is done for every block in the block world.

4 Observations and Comparisons

4.1 Observations

4.1.1 Example 1

Start State:

F
B A
E D C

Goal State:

A D B
E F C

	Manhattan	Euclidean	RelPos
Best First Search	Goal Found! States Explored: 941 Length of Path: 109 Time Elapsed: 0.61115 seconds	Goal Found! States Explored: 1325 Length of Path: 66 Time Elapsed: 1.24554 seconds	Goal Found! States Explored: 8 Length of Path: 9 Time Elapsed: 0.00397 seconds
Hill Climbing	No Solution! States Explored: 368 Length of Path: 0 Time Elapsed: 0.05196 seconds	No Solution! States Explored: 227 Length of Path: 0 Time Elapsed: 0.03595 seconds	Goal Found! States Explored: 8 Length of Path: 9 Time Elapsed: 0.00397 seconds

4.1.2 Example 2

Start State:

A
B D
C E

Goal State:

A
D
B
C E

	Manhattan	Euclidean	RelPos
Best First Search	Goal Found! States Explored: 74 Length of Path: 5 Time Elapsed: 0.01195 seconds	Goal Found! States Explored: 239 Length of Path: 20 Time Elapsed: 0.05589 seconds	Goal Found! States Explored: 5 Length of Path: 4 Time Elapsed: 0.0 seconds
Hill Climbing	No Solution! States Explored: 26 Length of Path: 0 Time Elapsed: 0.00396 seconds	No Solution! States Explored: 26 Length of Path: 0 Time Elapsed: 0.004 seconds	Goal Found! States Explored: 11 Length of Path: 12 Time Elapsed: 0.00796 seconds

4.1.3 Example 3

Start State:

F
E
D
C
B
A
_____

Goal State:

A B D
F E C
_____-

	Manhattan	Euclidean	RelPos
Best First Search	Goal Found! States Explored: 287 Length of Path: 37 Time Elapsed: 0.11582 seconds	Goal Found! States Explored: 801 Length of Path: 56 Time Elapsed: 0.64434 seconds	Goal Found! States Explored: 33 Length of Path: 18 Time Elapsed: 0.01195 seconds
Hill Climbing	Goal Found! States Explored: 306 Length of Path: 307 Time Elapsed: 0.1238 seconds	Goal Found! States Explored: 309 Length of Path: 310 Time Elapsed: 0.11982 seconds	No Solution! States Explored: 9 Length of Path: 0 Time Elapsed: 0.00396 seconds

4.2 Best First Search vs Hill Climbing

The most significant point of difference is that Best First Search is guaranteed to be complete in all test cases (i.e. if the solution exists, it will find the solution), whereas the Hill Climb approach is not complete and whether or not it finds the solution depends upon the start state, goal state and heuristic used.

However Hill Climb has a lower space complexity than Best First Search due to the fact that we do not maintain a frontier/opened list.

4.3 Comparing Heuristics

Comparison of Heuristics:

$$RelPos >> Manhattan > Euclidean$$

The main difference between $RelPos$ and the other two heuristics is that the other two are distance based while $RelPos$ looks at relative position. To solve a problem such as the one given to us, the heuristic must allow those moves which try to improve the relative position of the blocks (with respect to the goal state), even if it requires us to initially take a block away from its goal state position. The distance based heuristics (like $Euclidean$ and $Manhattan$) will penalize such moves by worsening the heuristic values; whereas the the heuristics which check the relative positioning of the blocks (like $RelPos$) will promote such moves.