**Project 2 Results**

In this project, I attempted several different heuristics in order to give an accurate estimate of how many remaining moves are left for a state to get to the goal state. I will briefly describe my attempts, and then say which one performed the best, giving an example to demonstrate.

Heuristic 1 (H1):

This heuristic used the Euclidean distance of all the blocks out of place. In other words, it took the square root of the sum of squared distances in x and y, where x is the stack number for the block out of place and y is the level of the block in the stack. This heuristic underperformed on most test cases.

H2:

This one used the Manhatten distance of all the blocks out of place. If block A is the first block in stack 1, but it needs to be the third block in stack 2, our Manhatten distance would be the difference in x (2 – 1 = 1) plus the difference in y (3 – 1 = 2). We would add such distances across all misplaced blocks. This heuristic also underperformed.

H3 and H4:

For these heuristics, I simply attempted to do the Manhatten distance, minus the consideration of distance in x or distance in y, respectively. These also did not work.

H5:

For this heuristic, add 2 for every block that’s not currently directly on top of the block on which it has to be in the goal state, or add 2 if there is such a block below it. Or, if a block is on the ground but in the wrong stack, add 2.

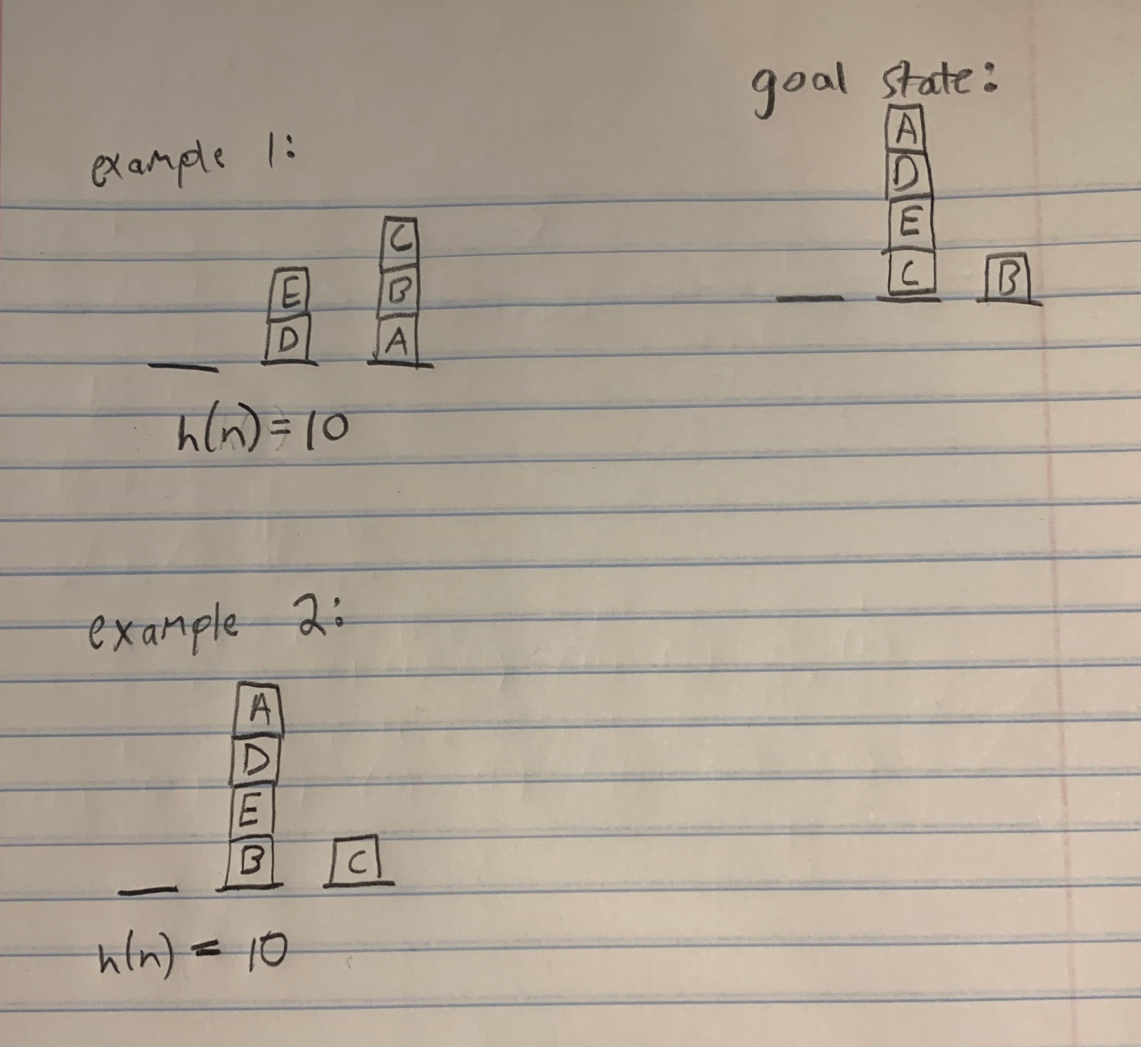
H6:

If Block A in the goal state is supposed to be on top of Block B and under Block C and in the current state it is neither on top of B or under C, then add 2 to the heuristic. This heuristic

H7:

This heuristic looks at blocks that need to be moved once and blocks that need to be moved twice. A “move once” block is a block that sits on a block different than what it sits on in the goal state, or a block that has such a block below it. A “move twice” block is a block that is on the correct block, but the block under it needs to be in a different stack, or if there is such a block under it. The heuristic is then (2 \* # move once) + (4 \* # move twice). This heuristic did not seem to follow through with its promise.

I shall now give an example of the best performing heuristic, H5. The picture below is a sample initial and final state:



In example 1, block A need to be on block D but isnt, so add 2. Block B must be on block A but isnt, so add 2. Blocks C, D, and E are also all not on their correct blocks, so add 2 for each of them. This gives us an h(n) score of 10.

In example 2, although blocks A and D are sitting on top of their correct blocks, there is a block below them (block E) which is not sitting on the correct block. Thus, we must add 2 for blocks A, D, and E. Additionally, although block B sits on the ground in both states, it is in the wrong stack, so we add 2. The same reasoning applies to block C. We get a total h(n) of 10.

Performance summary:

Heuristic 5 was able to solve 30 of the 45 Blocksworld challenge problems. It solved all problems involving 5 blocks in under 1000 iterations. It failed 3 of the problems involving 10 blocks. Interestingly, all these failures occurred when there were 3 stacks (which was the minimum amount considered). This was perhaps because the heuristic was unable to make progress in decreasing h(n), since there were only 3 stacks available for a relatively bigger number of blocks and thus the h(n) score did not vary that much. The heuristic had a few successes for problem with 20 blocks whenever the ratio of stacks to blocks reached 1:2. These successes took anywhere from 7490 iterations to 20819 iterations. Interestingly, when the configuration of the blocks seemed more spread out across the stacks in the initial and goal states, this led to fewer number of iterations, whereas if there were tall stacks/big clusters of blocks it took more iterations. In general, as solution path length increased, the max queue size also increased, though this relationship was non-linear. The degree to how non-linear it was depended on the number of blocks in the problem. For example, in bwchp.16 (5 blocks), we had a depth of 4 and max queue size of 33, and in bwchp.19 (5 blocks), we had a depth of 7 and max queue size of 427. However, in bwchp.21 (10 blocks), we had a depth of 10 and max queue size of 225, and in bwchp.22 (10 blocks), we had a depth of 18 and max queue size of 18376. The heuristic did not find sub-optimal solutions. However, this relationship did not always hold. For example, bwchp.44 (20 blocks) had a smaller depth than bwchp.43 (20 blocks) but its max queue size was much smaller.