CSCE 625: ARTIFICIAL INTELLIGENCE: PROGRAMMING ASSIGNMENT 1

- ANIKET SANJIV BONDE

Heuristics Development:

I have listed the three heuristics that I built, starting with simplest as the first one.

1) Number of blocks out of place: This heuristics measured the number of blocks that are out of their place as compared to the goal state. The main idea on this heuristic development was try to find a "score" for each state analyzing successful solution path.

Performance:

Small problems like 5 blocks and 3 stacks were solved with lot of iterations, but bigger problems involving 7 blocks and 5 stacks or further took ~3000 iterations with huge increase in queue size, hence consuming a lot of memory. The time was almost ranging from 0.01 to 0.9 seconds for small problems but for some bigger problems, solutions wasn't reached in 10000 iterations.

2) A more better heuristic designed by me is as follows:

$$\boldsymbol{f(n) = g(n) + h(n)}$$

Considering:

$$\boldsymbol{g(n)} = Depth\ cost$$

$$\boldsymbol{h(n)} = number\ of\ blocks - number\ of\ continuous\ blocks\ on\ stack\ 1\ from\ index\ 0 + number\ of\ blocks\ out\ of\ place\ on\ stack\ 1$$

This heuristic is admissible because it confers to the triangle rule of inequality.

For example consider the two configurations:
Configuration 1
Stack1: A B
Stack2: E D
Stack3: C

Configuration 2:
Stack1: A B D
Stack2: E
Stack3: C

This heuristic assigns a lower score for configuration 1 , hence it gets higher priority . For example config 1 will get a score of (5 – 2 + 0) = 3 and config 2 will get a score of (5 -2 + 1) = 4 Assuming both the configurations occur at the same depth , the algorithm choses config 1 over config 2. On average this heuristic gives me an improvement in the factor of 10 when compared to the naive heuristic .

For example if parent state is :

Stack1: ABD

Stack2: E

Stack3: C

For the best case the child will be

Stack1 : AB

Stack2: ED

Stack3: C

Hence, score(parent) <= score(child) + depth. **This heuristic is admissible because it confers to the triangle rule of inequality.**

Performance and Comparison to previous heuristic: This heuristic easily solved small problems like 5 blocks and 3 stacks, but bigger problems involving 10 blocks and 5 stacks or further took ~2000 iterations with huge increase in queue size, hence consuming a lot of memory. The main factor that decreased in this heuristic is the queue size. The queue size was a lot less than the previous heuristic. The time was almost ranging from 0.001 to 1 seconds for small problems but for some bigger problems like 12 blocks and 5 stacks, solutions wasn't reached in 10000 iterations.

3) This is the best heuristic that I have developed and it finds the solution to complex problems involving 15 blocks and 8 stacks in under 10 seconds.

$$\boldsymbol{f(n) = g(n) + h(n)}$$

Considering:

$$\boldsymbol{g(n)} = Depth\ cost$$

$$\boldsymbol{h(n)} = 5 * (number\ of\ blocks) - 5 * (length\ of\ sorted\ stack\ 1\ from\ index\ 0)$$

$$+ 2 * (no.of\ non - ordered\ blocks\ in\ stack\ 1)$$

$$+2 * (no.of\ blocks\ above\ the\ next\ best\ block) +\ number\ of\ empty\ stacks$$

Although this heuristic might seem too complicated at first but it's simple:

- **First factor:** Just to make sure that the total cost does not get negative because of the subtraction of the second factor
- **Second factor:** It's the number of blocks that are in place as compared to goal state, and this factor is multiplied by 5 as it's the most important factor in the cost function.
- **Third factor:** This factor is the number of blocks that are not in order in the first stack. This factor is multiplied by 2 and added to the cost function as we want this number to decrease.
- **Fourth factor:** Now, considering the objective, we have to make sure that the whichever is top sorted block in the first stack must be followed by its next follower, it should be ensured that this follower is not deep into the other stacks. Hence, number of blocks above the follower box multiplied by a factor of 2 is added in the cost function.
- **Fifth factor:** This was later added to the cost function as it was seen that empty stacks during iterations were left out rather than them to be filled first. A simple addition of the number of blocks to the cost function evaded this issue.

score(parent) <= score(child) + depth. **Hence, this heuristic is admissible as well.**

Performance and Comparison to previous heuristics: This is the best heuristic that I have developed and it **easily solved huge problems like 15 blocks and 7 stacks**. The queue size as well as the time taken for running this heuristic and finding the right solution was very low. Also, the number of iterations were far too reduced by this heuristic. The time was almost ranging from 0.001 to 2 seconds for almost all problems.