Programming Assignment #1

CSCE 625: Artificial Intelligence

Name: Nagaraj Thenkarai Janakiraman

UIN: 322007460

1. Compiling Instructions:

The program is written in C++ and takes the following command line arguments as inputs

```
>./blocksworld <numBlocks> <numStacks>
```

Input Parameters:

- <numBlocks> refers to the number of blocks in the blocksworld problem
- <numStacks> refers to the number of Stacks in the problem.

Output Parameters:

The program outputs the following

- **Initial state:** The initial state is generated by randomly choosing a state with the given paramaters using the “problemGenerator()” function. First the size of each stack is chosen at random such that the total sum of all the stack sizes is the numBlocks. We use multinomial distribution to sample the stack sizes. Given the stack sizes, a random permutation of the blocks are assigned to the stacks to create the initial state.
- **Goal state:** The program also prints the goal state. The program is written in general form to handle any goal state.
- **Iter, queue,f = g+h, depth:** The program prints the queue size, totalcost and current depth for each iteration.
- **Solution Path:** The program also prints the path from the intial state to the goal state in the same format as given in the handout.

2. Sample Trace of the program

```
>blocksworld 5 3

Goal State:
0|   A   B   C   D   E
1|
2|
Initial State:
0|
1|   B   A
2|   D   E   C

Starting Astar!
iter=5,  queue=11,  f=g+h=22,  depth=2
iter=10,  queue=21,  f=g+h=21,  depth=4
iter=15,  queue=28,  f=g+h=21,  depth=5
iter=20,  queue=34,  f=g+h=22,  depth=5
iter=25,  queue=34,  f=g+h=19,  depth=6
iter=30,  queue=34,  f=g+h=22,  depth=3
iter=35,  queue=39,  f=g+h=22,  depth=6
iter=40,  queue=46,  f=g+h=19,  depth=9
...
success! depth=11, total_goal_tests=42, max_queue_size=50
0|
1|   B   A
2|   D   E   C

0|   C
1|   B   A
2|   D   E

0|   C
1|   B   A   E
2|   D
```

```
0|   C   D
1|   B   A   E
2|

0|   C   D
1|   B   A
2|   E

0|   C
1|   B   A
2|   E   D

0|
1|   B   A
2|   E   D   C

0|   A
1|   B
2|   E   D   C

0|   A   B
1|
2|   E   D   C

0|   A   B   C
1|
2|   E   D

0|   A   B   C   D
1|
2|   E

0|   A   B   C   D   E
1|
2|
```

Heuristic function:

The program supports three heuristic function, which can be chosen in the Node definition.

- "simple"
- "complex"
- "combined"

1) **"simple":**
The simple heuristic is the naïve implementation where we measure the number of mismatches compared to the goal state. This heuristic is very simple and misses many important aspects such as the incorporating moves in the cost function.

2) **"complex":**
The complex heuristic function works as follows. It works differently for the goal stack (Stack 0) and other stacks (Stack 1 to Stack numStacks-1).

For Stack-0:

- **Step-1:** Increment the cost function by 2 for each block that must be moved once. These are blocks that are currently on a block different to the block upon which it rests in the goal state or a block that has such a block somewhere below it in the same pile.
- **Step-2:** Increment the cost function by 4 for each block that must be moved twice. These are blocks that are currently on the block upon which it must be placed in the goal state, but that block is a block that must be moved or if there exists a block that must be moved twice somewhere below it.

An, example cost calculation for stack-0 is shown in Figure 1. below.

For Stack>=1:

- **Step-1:** Increment the cost function by 2 for each block that must be moved once. These are blocks that are currently below (exact reverse case of stack-0) a block different to the block upon which it rests in the goal state or a block that has such a block somewhere below it in the same pile.
- **Step-2:** Increment the cost function by 4 for each block that must be moved twice. These are blocks that are currently below (again, reverse case of stack-0) the block upon which it must be placed in the goal state, but that block is a block that must be moved or if there exists a block that must be moved twice somewhere below it.

An, example cost calculation for stack-1 is shown in Figure 1. below.

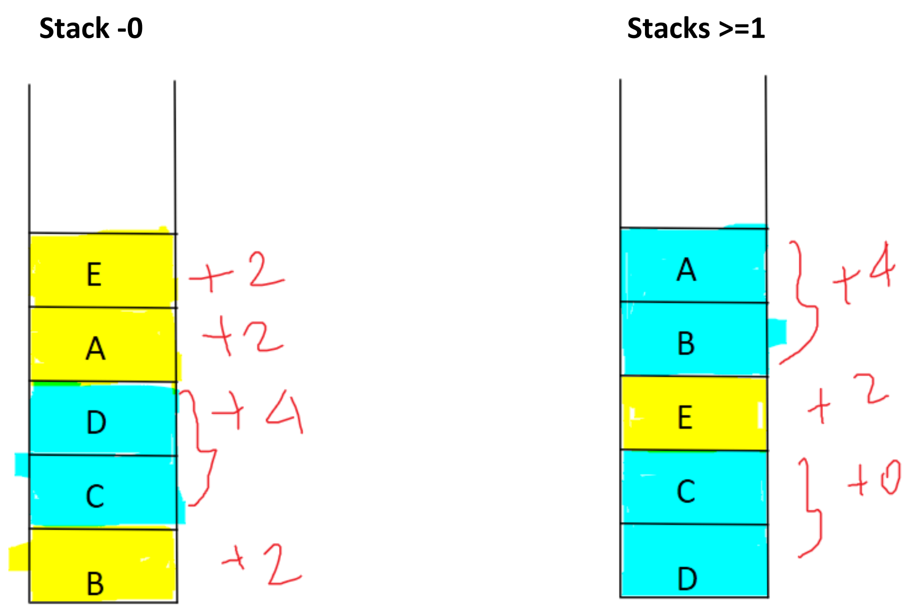


Figure 1: Applying the rules stated above, we can see that the yellow colored blocks needs to be moved once and the blue colored blocks need to be moved twice. The corresponding cost additions are also shown in the figure against each block.

3) “combined”:

The combined heuristic is a hybrid of both “simple” and “complex”. As discussed earlier, the “simple” heuristic does not capture the moves, whereas the “complex” heuristic didn’t capture the distance from the goal state directly. To get the best of both worlds, we combine both these heuristics using a linear combination.

$$combinedCost = (a \times simpleCost) + (b \times complexCost)$$

This cost captures both the features from the above two heuristics.

After testing for several runs, a = 1 and b is a free parameter chosen based on the stack and block size. In the current program, b=2.

Admissibility:

It seems admissible as it never overestimates the cost of reaching the goal, i.e. the cost it estimates to reach the goal is not higher than the lowest possible cost from the current point in the path.

Table of Performance metrics:

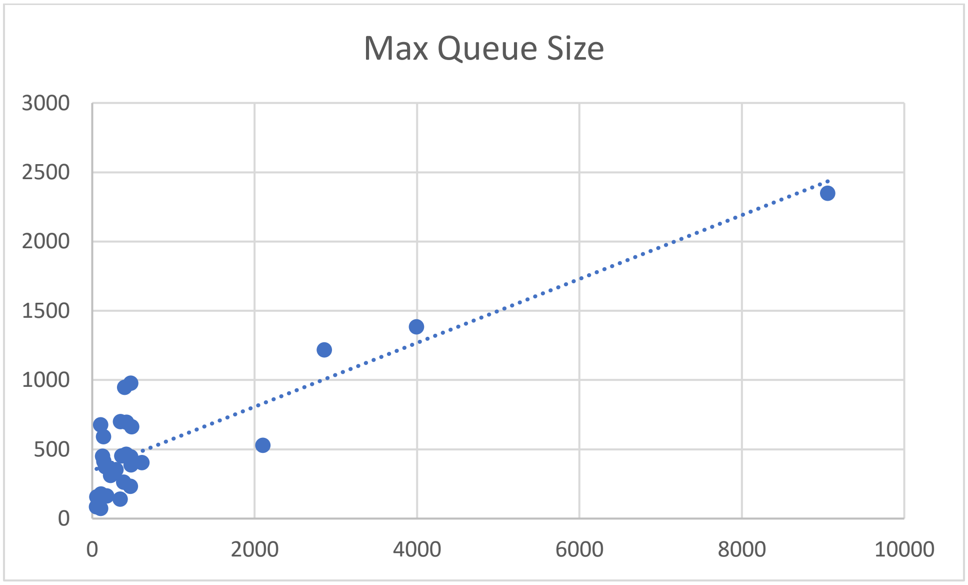
NumBlocks	NumStacks	Solution Path length	Total Goal Tests	Max Queue Size
5	3	9.3	101.6	72.85
5	4	8	52.75	84.65
5	5	8.25	71.25	160
5	6	7.75	55.4	157.7
5	7	7.45	476.1	388.95
6	3	12.35	345.15	141.9
6	4	10.4	180.15	162.95
6	5	10.45	388	263.25
6	6	10.4	292.85	354.55
6	7	10.45	141.4	414.15
7	3	14.5	469.8	233.75
7	4	13.3	107.2	178.6
7	5	12.55	224.35	310.95
7	6	12.7	163.35	373.45
7	7	12.4	423.3	696.5
8	3	18.15	2102.45	529.15
8	4	16.65	612.45	404.45
8	5	15	362.4	452.85
8	6	15.15	128.3	450.9
8	7	14.85	400.6	946.75
9	3	21.6	2856.1	1218.25
9	4	19	419.15	463.8
9	5	17.6	347.7	701.2
9	6	18.77	472.66	445
9	7	18.16	103	676.66
10	3	25.375	9058.37	2349.25
10	4	19.833	3993	1385.33
10	5	20.2	475.3	977.6
10	6	20.66	139.41	592.75
10	7	21.055	484.88	662.388

Discussion of Results:

1) The heuristic works well compared to a simple heuristic and could get to the goal state with a good number of path length and queue size.
2) I tried until numStacks = 10 and numblock = 26 and it could get to the goal state. The result is as follows.

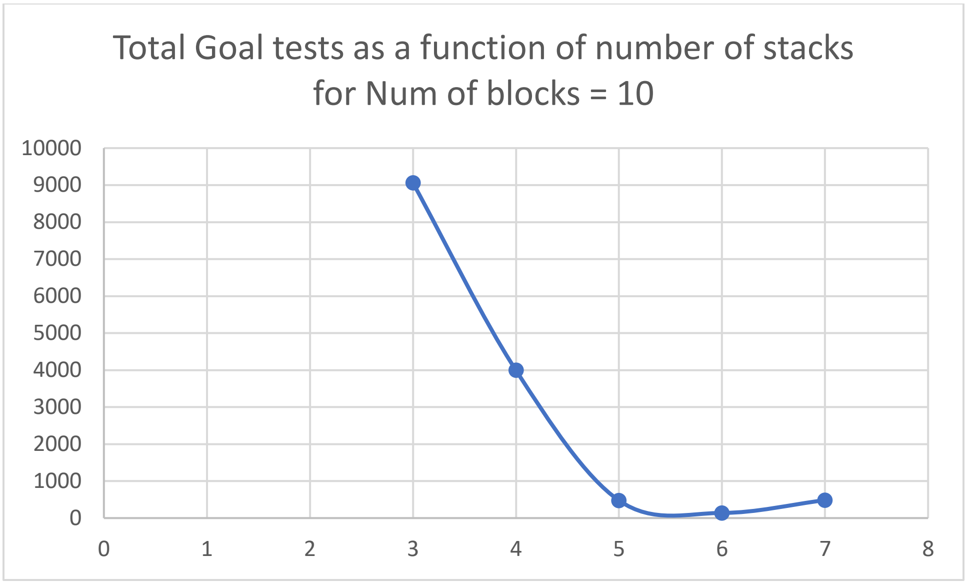
   success! depth=60, total_goal_tests=175, max_queue_size=4172

3) The queue grew linearly in the number iterations. The following graphs shows the behavior.



4) I believe my program found optimal solutions as it did finally find the goal state and the path seemed to have traced the best route.
5) Increasing the number of the stacks for a constant number of blocks made the problem easier as we have a large branching factor and hence reduces the path length and number of goal tests given the heuristic s good.

   The following graph illustrates the reduction in the number of goal tests for Number of blocks =10 and stacks varying from {3,4,5,6,7}. As can be seen from the plot, there is a exponential decrease in the number of goal tests as the number of stacks increase. This is due to the “complex” heuristic taking account of the sum of independent stack costs.



6) More complicated heuristics can be implemented by adding the information about how different stacks can be combined than the independent assumption. We could also optimize on the parameters a and b and find the relation between number of stacks, number of blocks and the weighting parameters.