**Project 1 – Blocksworld Problem**

**Project Report**

**Submitted by Adil Hamid Malla**

**UIN: 425008306**

**Steps to run my program:**

**Example Traces of my Program**

Stacks and Blocks please

3

5

Initial State

Stack 0: B

Stack 1: E A

Stack 2: D C

Iteration Number 1, Queue Size = 6, TotalCost(gCost+hCost) = 5, Depth= 0

Iteration Number 2, Queue Size = 9, TotalCost(gCost+hCost) = 8, Depth= 1

Iteration Number 3, Queue Size = 12, TotalCost(gCost+hCost) = 10, Depth= 1

Iteration Number 4, Queue Size = 13, TotalCost(gCost+hCost) = 10, Depth= 1

Iteration Number 5, Queue Size = 16, TotalCost(gCost+hCost) = 6, Depth= 2

Iteration Number 6, Queue Size = 19, TotalCost(gCost+hCost) = 6, Depth= 3

Iteration Number 7, Queue Size = 22, TotalCost(gCost+hCost) = 6, Depth= 4

Iteration Number 8, Queue Size = 23, TotalCost(gCost+hCost) = 6, Depth= 5

Iteration Number 9, Queue Size = 22, TotalCost(gCost+hCost) = 6, Depth= 6

Goal State Found

Success!! depth = 6, Total Goal Tests = 9, Maximum Queue Size= 23, Total Cost = 6

Stack 0: B

Stack 1: E A

Stack 2: D C

Stack 0:

Stack 1: E A

Stack 2: D C B

Stack 0: A

Stack 1: E

Stack 2: D C B

Stack 0: A B

Stack 1: E

Stack 2: D C

Stack 0: A B C

Stack 1: E

Stack 2: D

Stack 0: A B C D

Stack 1: E

Stack 2:

Stack 0: A B C D E

Stack 1:

Stack 2:

Done and Nailed It

**Components of the Program:**

**Heuristics Function:**

I have used composite/combination of the heuristics. I got this idea while reading the Chapter 3 of AIMA by Russel and Norvig. Since the single heuristics is not enough to handle all the cases of the program.

Since the initial intuition for the heuristics was to check how many blocks are not present in the goal state configuration, irrespective of the position. Since the goal state for our algorithm is all the blocks in the single stack (first stack), stacked over each other in ascending order.

**Heuristics 1(h1):**

The first heuristics I implemented was checking that how many correct number of blocks are in the first stack. This heuristic will make sure that rest of the elements needs to be moved from other stacks to the first stack. Using this heuristic, we can easily solve the simple problems with has less branching factor (or a number of stacks). But this heuristic doesn’t penalize or take into the consideration the internal structure of the how blocks are arranged inside each stack as well as in the first stack. This heuristic gives us the least number of steps to be used to make sure the other blocks from other stacks are transferred to the first stack. Thus this assumes the blocks in the other stacks are present in the best format and we just need to move them to the first stack, which is not true for majority cases.

**Heuristics 2:**

The second heuristic I implemented, I took into the consideration the penalty to the blocks which were not correctly in the stacks. For each block which was wrongly ordered – not in increasing order from the bottom to top in the first stack and not in decreasing order from

bottom to top in other stacks (not the first stack) were penalized by two points. The first stack is the special stack as all the nodes have to end up in this stack to achieve the goal. Moreover, I will penalize if the block present in the bottom of the first stack is not what it should be in the goal state. This heuristic even though penalizing the wrong configuration did okay for a small number of stacks, but as soon the stacks starting increasing this heuristic led the solution to distribute the blocks along the stacks in decreasing order hence to reduce the penalized value but not making sure that the elements are going to first stack. One more problem with this heuristic was that it was not handling the intermediate cases where the blocks in one stack are not continuously increasing or decreasing hence resulting in more complex situations.

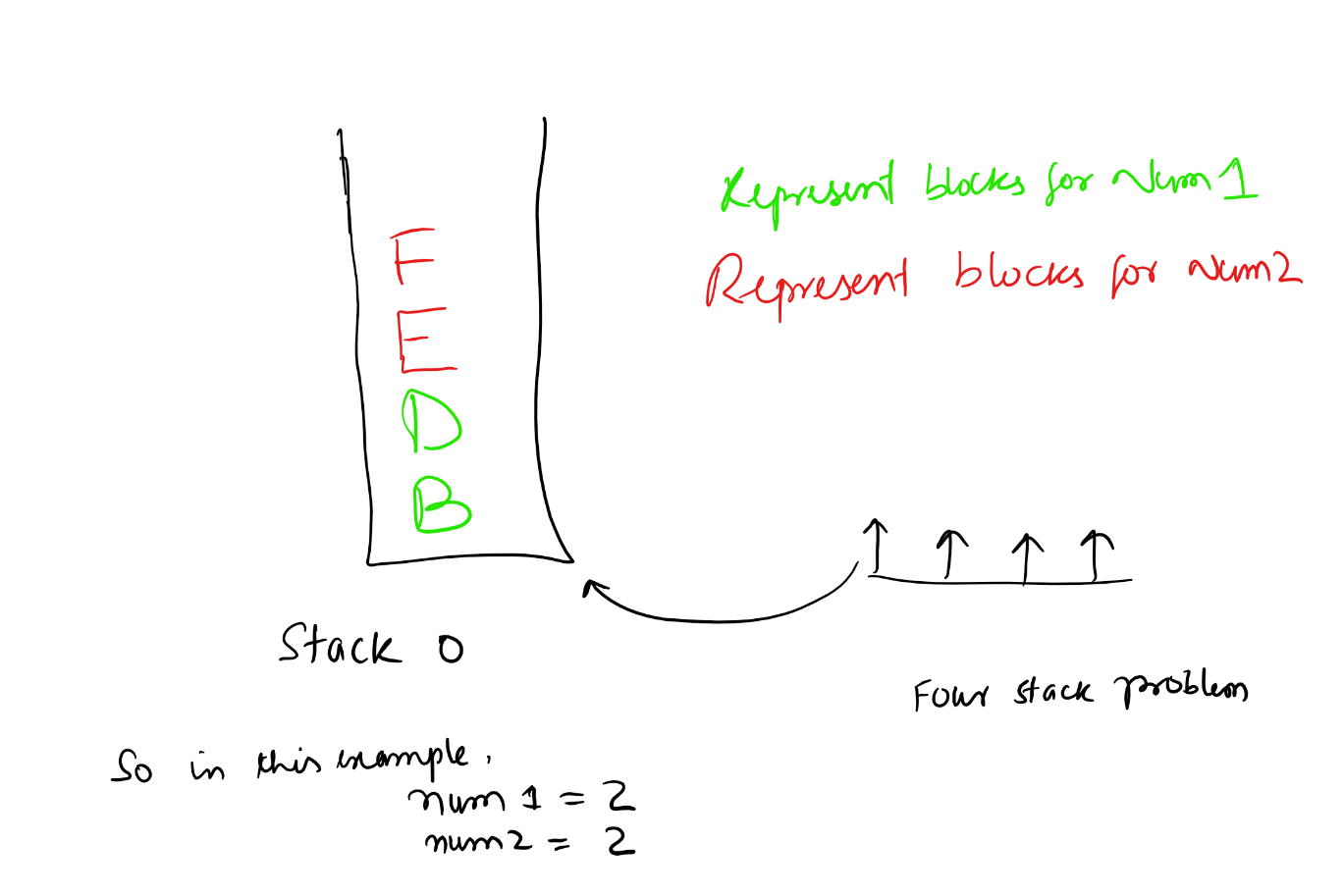
Since this heuristic was not able to converge to the solution to goal as it was settling to distribute the blocks along the stacks, it was creating a lot of nodes (around 65,000) and still not crossing the depth of 10. So the answers which were around the level 10 were easily solved.

To address the problem of complex arrangement of the blocks on the stack, I needed a different penalty for different configurations thus heuristic 3.

**Heuristics 3 (h3):**

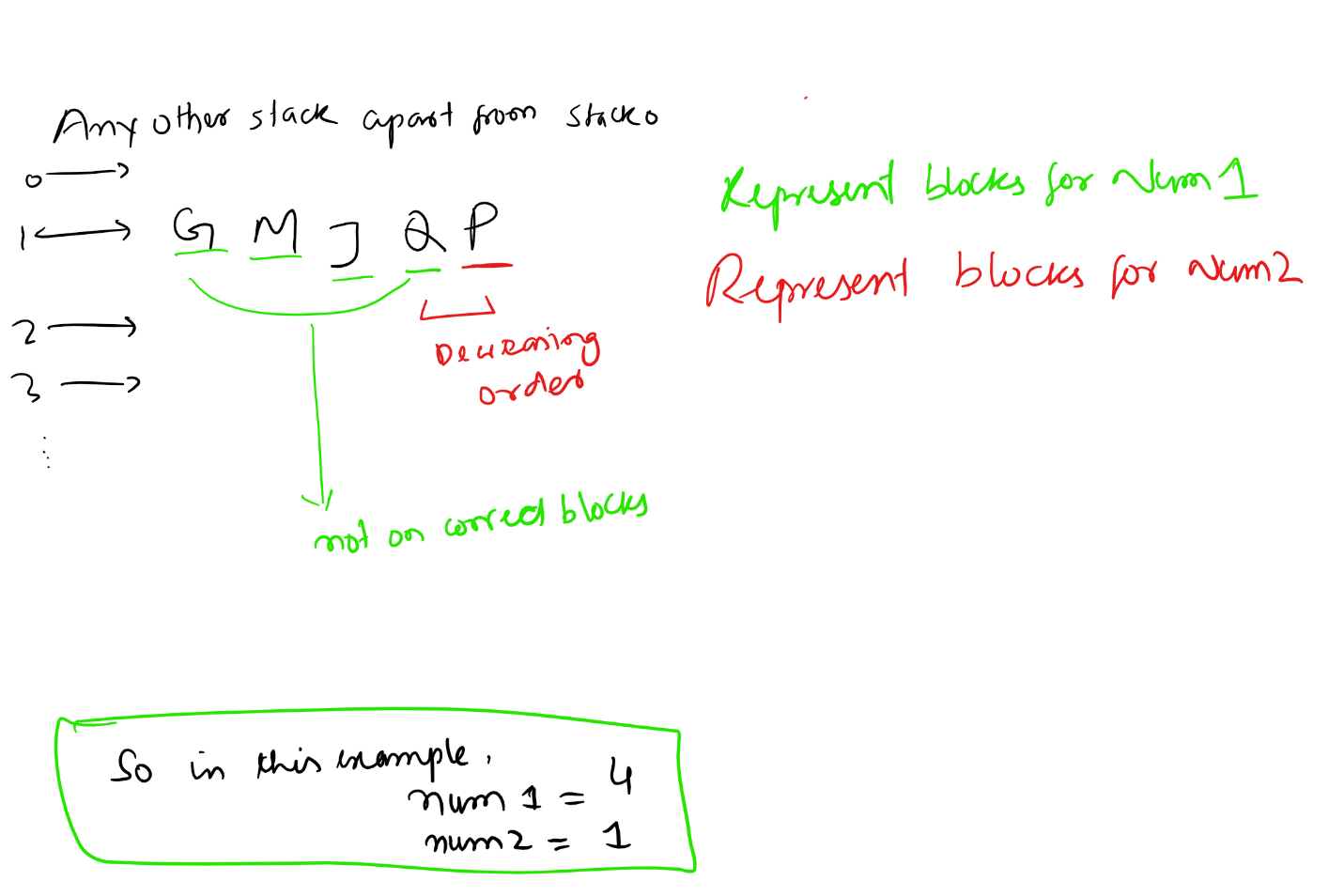
This heuristic is more complicated and takes all the configurations of the blocks in calculating the heuristics. We divide the heuristic into two parts. One is for the first stack and for other stacks. For each configuration, we will store two occurrences of two types of blocks i.e. the block which should be moved at least once and blocks which needs to be moved twice.

1. **For First Stack:** We will traverse from bottom to top and check if the block is not correctly placed on the block below it. If the block is not the next block in order of increasing order, then this block needs to be moved at least once to other stack and then back. So we increase the number of such blocks continuously till top and we call them “Num1”. If we already met such a block, we will set a flag marking that one block is wrongly placed. Now we will also check if there is a set of blocks which are in right order but placed on the wrong stack (i.e. wrong block is below them which we will know by the flag), we count these blocks as “Num1”. So to attain the correct configurations the set of correctly placed blocks on top of the wrong block needs to moved once out and then back to the goal state. After traversing this we get “Num1” and “Num2” for the first stack.   
   The special treatment is given to the first stack is because we prefer the blocks in increasing order as it is needed for goal state.



1. **For all other Stacks:**

The penalties are similar to the blocks on other stacks but the only difference from that of the first stack is the order of checking the elements, in the first stack, we checked the increasing order but in other stacks, we would prefer the decreasing order. Thus if C, D, E are in stack number 1, then we just need to move them to stack 0 in order and need not to be moved around. The example explains it more.



After calculating the number of blocks represented by num1 and num2, then we call to calculate the number of moves to be used by each type of block to be placed into the goal state.

Num1 -> Since these blocks are out of order that means they need to move out of the stack once and then back again in case of stack 0 (first stack ) and for the other stacks it needs to be moved out and then again so that the element below it can be placed correctly on the stack 0. Thus, in the best case scenario, we need move these elements twice.

Num2-> Since we have two blocks which are in order but the block below them is not in order, we need to move these blocks at least four times to get to correct stack.

Hence, the total cost generated by the heuristic is the sum of moves for “Num 1” type blocks and “Num 2” type blocks. Which is given by,

So heuristic3 = 2 \* #Num1 + 4 \*#Num2

So by the above argument and examples, the heuristic 3 is admissible for A\* Search Algorithm. As we are calculating the number of the least steps involved to move the elements from each stack without considering the inter-stack information. Thus we are only considering the lower limit of the moves needed and thus is admissible.

**The problem of using single heuristics in Blocksworld Problem:**

Although the heuristics 3 mentioned above is better in many terms for calculating the heuristic cost, it suffers a technical problem. When the number of the stacks increase for constant blocks this problem leads to a problem. The program tries to settle few blocks in each stack and thus negating the heuristic 3 cost and not moving the blocks to the first stack (i.e. goal state). So when I ran the solution on 7 stacks and 10 blocks it was branching madly as it resulted in the branches which were trying to reduce the heuristic 3 value by distributing the blocks into different stacks.

This problem led me to make sure I penalize this distributing of blocks to other stacks by counting the number of pending blocks to be moved to the stack 0.

So the total heuristic cost which I used for my A\* Blocksworld Problem is given by   
The weighted sum of h1 and h3.

Heuristics = w1 \* h1 + w2 \* h3

By empirically running the algorithm over many instances I found that the optimal number of w1 and w2 is actually a function of a number of stacks. As we increase the number of the stacks, the h3 leads tend to make the blocks distributed over all stacks. I called this state as “Low h3 State” and thus to make counter this effect I increased the weight for w1, which makes these blocks to move to the stack 0 in the correct form.

Thus using these composite heuristics, I was able to solve the problem very effectively.

**So w1 makes the blocks to get to the first stack and the w2 makes sure we put them in the correct order.**

Hence in combination, I was able to solve even the problem with parameters as high as 15 stacks and 26 blocks.

Since I generalized my code by using the ASCII values instead of using only alphabets I can even try to make it harder.

**Results:**

I ran the simulations on various configurations and the corresponding results obtained by running all of them are given in the table below:

**For w1 =1 and w2 =1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Number of Stacks** | **Number of Blocks** | **Average Goal Tests** | **Average Maximum Queue Size** | **Average Actual Path Cost (Number of Steps)** |
| 3 | 5 | 11 | 25 | 9 |
| 3 | 6 | 12 | 28 | 9 |
| 3 | 7 | 23 | 57 | 16 |
| 3 | 8 | 78.49 | 185.47 | 19.39 |
| 3 | 9 | 116.56 | 309.73 | 25.32 |
| 3 | 10 | 114.88 | 301 | 28.08 |
| 4 | 5 | 16.38 | 53.12 | 9.16 |
| 4 | 6 | 46 | 242 | 11 |
| 4 | 7 | 44.75 | 245.39 | 16.74 |
| 4 | 8 | 27 | 162 | 17 |
| 4 | 9 | 37.08 | 238.88 | 18.48 |
| 4 | 10 | 43 | 286 | 23 |
| 5 | 5 | 27 | 172.17 | 9.98 |
| 5 | 6 | 9 | 81 | 8 |
| 5 | 7 | 13 | 141 | 9 |
| 5 | 8 | 30.92 | 300.08 | 16.54 |
| 5 | 9 | 21 | 248 | 16 |
| 5 | 10 | 32.93 | 390.63 | 21.54 |
| 6 | 5 | 25.89 | 338.88 | 7.11 |
| 6 | 6 | 24.5 | 331.9 | 9 |
| 6 | 7 | 29.02 | 440.5 | 12.18 |
| 6 | 8 | 26.35 | 428.15 | 14 |
| 6 | 9 | 22.07 | 323.62 | 17.09 |
| 6 | 10 | 23 | 321 | 20 |
| 7 | 5 | 60.34 | 670.27 | 8.69 |
| 7 | 6 | 38.34 | 636.66 | 9.18 |
| 7 | 7 | 38.05 | 724.54 | 12.02 |
| 7 | 8 | 30.26 | 640.6 | 15.17 |
| 7 | 9 | 34.21 | 564.32 | 16.51 |
| 7 | 10 | 26.49 | 464.62 | 15.51 |
| 8 | 5 | 58.33 | 808.82 | 7.75 |
| 8 | 6 | 22.22 | 471.56 | 7.6 |
| 8 | 7 | 14.78 | 251.69 | 9.24 |
| 8 | 8 | 32.53 | 702.88 | 12.09 |

**For w1 =2 and w2 =1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Number of Stacks** | **Number of Blocks** | **Average Goal Tests** | **Average Maximum Queue Size** | **Average Actual Path Cost (Number of Steps)** |
| 3 | 6 | 48 | 101 | 17 |
| 3 | 7 | 29 | 59 | 16 |
| 3 | 8 | 40.4 | 106.36 | 16.6 |
| 3 | 9 | 52 | 138 | 22 |
| 3 | 10 | 96.78 | 249.79 | 25.93 |
| 4 | 5 | 13 | 65 | 10 |
| 4 | 6 | 13 | 73 | 12 |
| 4 | 7 | 15 | 80 | 13 |
| 4 | 8 | 39.8 | 223 | 19.96 |
| 4 | 9 | 30 | 187 | 19 |
| 4 | 10 | 19 | 118 | 18 |
| 5 | 5 | 11 | 94 | 9 |
| 5 | 6 | 9 | 69 | 8 |
| 5 | 7 | 15 | 127 | 14 |
| 5 | 8 | 25.03 | 253.74 | 15.27 |
| 5 | 9 | 22.9 | 255.75 | 17.94 |
| 5 | 10 | 24 | 279 | 20 |
| 6 | 5 | 18.68 | 146.93 | 9.97 |
| 6 | 6 | 11 | 140 | 10 |
| 6 | 7 | 13 | 178 | 11 |
| 6 | 8 | 12 | 135 | 11 |
| 6 | 9 | 20.9 | 339.1 | 16.3 |
| 6 | 10 | 31.54 | 538.84 | 18.45 |
| 7 | 5 | 13 | 167 | 9 |
| 7 | 6 | 8.89 | 132.67 | 7.05 |
| 7 | 7 | 14 | 222 | 11 |
| 7 | 8 | 19.12 | 382.04 | 15.16 |
| 7 | 9 | 18.55 | 340.8 | 16.55 |
| 7 | 10 | 16.4 | 329.32 | 13.28 |
| 8 | 5 | 12.6 | 189.2 | 7.8 |
| 8 | 6 | 7 | 65 | 6 |
| 8 | 7 | 19.4 | 464.16 | 13 |
| 8 | 8 | 17.72 | 368.44 | 14.36 |
| 8 | 9 | 25.18 | 713.17 | 15.53 |
| 8 | 10 | 22.45 | 564.94 | 18.79 |
| 9 | 5 | 6 | 92 | 5 |
| 9 | 6 | 32.84 | 658.29 | 10.11 |
| 9 | 7 | 18.5 | 416.25 | 12.75 |
| 9 | 8 | 13.08 | 330.46 | 11.3 |
| 9 | 9 | 22.23 | 650.13 | 16.38 |
| 9 | 10 | 21.27 | 607.13 | 16.68 |
| 10 | 5 | 16.49 | 347.03 | 9.07 |
| 10 | 6 | 19.84 | 502.4 | 10.54 |
| 10 | 7 | 19.54 | 537.78 | 12 |
| 10 | 8 | 20.77 | 666.97 | 13.88 |
| 10 | 9 | 28.45 | 937.3 | 16.42 |
| 10 | 10 | 21.6 | 633.89 | 17.77 |

**For w1 =3 and w2 =1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Number of Stacks** | **Number of Blocks** | **Average Goal Tests** | **Average Maximum Queue Size** | **Average Actual Path Cost (Number of Steps)** |
| 3 | 5 | 8 | 20 | 6 |
| 3 | 6 | 23 | 55 | 13 |
| 3 | 7 | 36.98 | 83.7 | 17.88 |
| 3 | 8 | 109.53 | 276.95 | 17.73 |
| 3 | 9 | 51 | 119 | 25 |
| 3 | 10 | 136.68 | 366.65 | 30.7 |
| 4 | 5 | 11 | 54 | 10 |
| 4 | 6 | 19.6 | 103.4 | 11.72 |
| 4 | 7 | 13 | 67 | 12 |
| 4 | 8 | 30 | 166 | 17 |
| 4 | 9 | 37 | 245 | 18 |
| 4 | 10 | 39.07 | 263.52 | 21.99 |
| 5 | 5 | 27.6 | 199.26 | 10.92 |
| 5 | 6 | 11 | 91 | 10 |
| 5 | 7 | 19 | 181 | 14 |
| 5 | 8 | 17 | 133 | 16 |
| 5 | 9 | 36.68 | 420.64 | 20.34 |
| 5 | 10 | 27.76 | 313.84 | 20.38 |
| 6 | 5 | 11 | 144 | 8 |
| 6 | 6 | 14 | 172 | 12 |
| 6 | 7 | 12.24 | 142.56 | 11.24 |
| 6 | 8 | 18 | 261 | 14 |
| 6 | 9 | 17.62 | 228.34 | 16.62 |
| 6 | 10 | 26.64 | 401.54 | 21.16 |
| 7 | 5 | 19.05 | 201.59 | 7.51 |
| 7 | 6 | 15 | 177 | 12 |
| 7 | 7 | 17.9 | 312.9 | 13.9 |
| 7 | 8 | 26.07 | 387.23 | 12.05 |
| 7 | 9 | 10 | 165 | 9 |
| 7 | 10 | 20.28 | 447.33 | 16.2 |
| 8 | 5 | 21.99 | 330.08 | 8.33 |
| 8 | 6 | 20.24 | 322.78 | 12 |
| 8 | 7 | 18 | 325 | 14 |
| 8 | 8 | 22.76 | 572.7 | 14.35 |
| 8 | 9 | 14.16 | 361.82 | 12.58 |
| 8 | 10 | 22 | 517.95 | 19.53 |
| 9 | 5 | 8.96 | 167.76 | 6.84 |
| 9 | 6 | 16.38 | 435.49 | 11 |
| 9 | 7 | 16 | 436.24 | 11.39 |
| 9 | 8 | 28.17 | 869.44 | 14.84 |
| 9 | 9 | 19.24 | 661.19 | 14.22 |
| 9 | 10 | 19.5 | 508.87 | 17.19 |
| 10 | 5 | 9.88 | 199.48 | 6.72 |
| 10 | 6 | 12.45 | 332.8 | 8.2 |
| 10 | 7 | 20.85 | 688.06 | 12.01 |
| 10 | 8 | 20.65 | 722.71 | 12.17 |
| 10 | 9 | 27.01 | 976.08 | 15.7 |
| 10 | 10 | 22.44 | 628.04 | 18.8 |

**Table 4:**

For a constant number of Blocks (i.e. 10) with a variable number of stacks (3 - 10):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Number of Stacks** | **Number of Blocks** | **Average Goal Tests** | **Average Maximum Queue Size** | **Average Actual Path Cost (Number of Steps)** |
| 3 | 10 | 129.84 | 340.74 | 28.41 |
| 4 | 10 | 34.75 | 216.9 | 21.15 |
| 5 | 10 | 23 | 249 | 21 |
| 6 | 10 | 20 | 289.83 | 19 |
| 7 | 10 | 22.52 | 411.34 | 19.38 |
| 8 | 10 | 29.64 | 826.35 | 18.45 |
| 9 | 10 | 22.01 | 621.25 | 18.01 |
| 10 | 10 | 23.23 | 802.27 | 17.94 |

Figure – 3: Plot of Number of Goal Tests vs Number of Stacks for 10 Blocks

Figure – 4: Plot of Number of Solution Cost vs Number of Stacks for 10 Blocks

**Table 5:**

For a constant number of Stacks (i.e. 10) with a variable number of Blocks (5 - 10):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Number of Stacks** | **Number of Blocks** | **Average Goal Tests** | **Average Maximum Queue Size** | **Average Actual Path Cost (Number of Steps)** |
| 10 | 5 | 6 | 104 | 5 |
| 10 | 6 | 11 | 189 | 10 |
| 10 | 7 | 19.08 | 572.04 | 13 |
| 10 | 8 | 21.01 | 1063.08 | 13.18 |
| 10 | 9 | 22.3 | 699.61 | 16.56 |
| 10 | 10 | 25.1 | 791.14 | 18.35 |
| 10 | 5 | 6 | 104 | 5 |
| 10 | 6 | 11 | 189 | 10 |

Figure – 5: Plot of Number of Goal Test vs Number of Blocks for 10 Stacks

Figure – 6: Plot of Number of Goal Test vs Number of Blocks for 10 Stacks

**Discussion:**