In this document, I will showcase the performance of two of the seven heuristics I came up with: heuristic 5 and heuristic 7. Heuristic 5 often takes a relatively small number of iterations to terminate, but at the cost of sub-optimal plan lengths. Heuristic 7 often provides near-optimal plan lengths, but often takes more iterations to terminate. Below are the implementation details for both of the heuristics. When my code is run without a specified heuristic, it will use heuristic 5 because it generally produces an output faster than the other heuristics. However, if I wanted to get a path whose length is close to optimal, I would specify the use of Heuristic 7.

**Heuristic 7** - This heuristic aims to reward block configurations that have their stack complete or mostly complete, and punishes configurations where blocks that are not in their correct position are under another block/set of blocks that are in the correct stack. There are three steps to the heuristic, described below.

1. The heuristic value starts equal to the number of blocks in the puzzle.
2. The algorithm iterates through all the blocks in each stack starting from the bottom of the stack. The algorithm will decrement the heuristic value by 1 for every block that is in the correct position and will move on to the next stack as soon as it finds a block that is not in the correct position. It will repeat this until all stacks have been checked.
3. The heuristic will iterate through the blocks in each stack again, but this time it will increment the heuristic value whenever it finds a block that does not belong in the current stack and that also has a block on top that does belong in the current stack. The heuristic value is incremented by a quantity equal to the depth of the block being “blocked” from moving by the block on top of it.

**Heuristic 5** - This heuristic rewards configurations that have a higher quantity of blocks in the correct position and punishes configurations that have blocks in the incorrect configuration stuck under other blocks. There are three steps to the heuristic, described below.

1. Set the heuristic value to 0.
2. Count the number of blocks that are in the incorrect place. Add this quantity to the heuristic value.
3. Count the number of blocks that are in the incorrect place and are under another block. Add this quantity to the heuristic value.

**Example Case for Heuristic 7**

| **Current State** | **Goal State** |
| --- | --- |
| 0 |A I  1 |G C E  2 |F  3 |D H B | 0 |A I J  1 |C E  2 |F G H  3 |D B |

Getting a heuristic value from Heuristic 7:

1. **Heuristic = 9**

* There are 9 blocks in this iteration of the problem, so we set the heuristic value to 9

1. **Heuristic = 9 - 2 - 0 - 1 - 1 = 6**

* Stack 0 has two consecutive letters in the correct order meaning we decrement the heuristic value twice. Stack 1 starts out with G, which is not in the correct spot and therefore the heuristic moves on to the next stack. Stack 2 only has F, which is in the correct position and therefore we subtract 1 from the heuristic. For stack 3, we subtract 1 for D since it is in the correct position and stop as soon as we reach H since it is not in the correct position.

1. **Heuristic = 6 + 0 + (1 x 3) + 0 + (1 x 2) = 6 + 3 + 2 = 11**

* For stack 0, there are no blocks in the incorrect stack, so we move on. In stack 1, we find that G is not in the correct stack and it is getting blocked by C, which is in the correct stack. Since G has a depth of 3 in this stack, we increment the heuristic by 3. In stack 2, there are no blocks in the incorrect stack, so we move on. In stack 3, we find that H is in the incorrect stack and it is being blocked by B, which is in the correct stack. Since H has a depth of 2 in this stack, we increment the heuristic by 2.

**The resulting heuristic value is 11 for this current state.**

**Example case for Heuristic 5**

| **Current State** | **Goal State** |
| --- | --- |
| 0 |A I  1 |G C E  2 |F  3 |D H B | 0 |A I J  1 |C E  2 |F G H  3 |D B |

Getting a heuristic value from Heuristic 5:

1. **Heuristic = 0**
2. **Heuristic = 0 + ( (0 + 0) + (1 + 1 + 1) + 0 + (0 + 1 + 1) ) = 5**

* All of the letters in Stack 0 are in the correct place, meaning we do not add anything to the heuristic value. In stack 1, we find that neither G, C, or E are in the correct place, so we increment the heuristic value three times. In stack 2, we find that the only letter F is in the correct place, so we continue without incrementing the heuristic value. In stack 3, we find that the letters H and B are not in the correct place, so we increment the heuristic value by 2.

1. **Heuristic = 5 + ( (0) + (1 + 1) + 0 + (0 + 1) ) = 5 + 2 + 1 = 8**

* In stack 0, A is the only letter with a block on top of it, but since A belongs in this position we do not increment the heuristic and move on to the next stack. In stack 1, G is stuck under C, and since G is not in the correct position we increment the heuristic. Similarly, since C is under E and C is not in the correct position, we increment the heuristic. In stack 2 there is only one block, so we skip it. In stack 3, we find that D is under H, but since D is in the correct position we do not increment the heuristic value. We also find that H is under B, and since H is not in the correct position we increment the heuristic.

**The resulting heuristic value is 8 for this current state.**

**Test Case Performance for Heuristic 7 and Heuristic 5**

**Note:** Blue coloring implies that the plan length metric was better than Dr. Ioerger’s heuristic. Red implies the plan length on the heuristic was worse than Dr. Ioerger’s heuristic.

**Note:** After comparing results obtained from the server, it appears that the results obtained in the compute.cse server WILL DIFFER from the ones below, which were obtained from an M1 Macbook Air.

|  | **Heuristic 7 Performance** | | | **Heuristic 5 Performance** | | |
| --- | --- | --- | --- | --- | --- | --- |
| **FileName** | **Plan length** | **Iterations** | **Max Queue Size** | **Plan Length** | **Iterations** | **Max Queue Size** |
| **probA03.bwp** | 3 | 3 | 10 | 3 | 4 | 13 |
| **probA04.bwp** | 4 | 7 | 16 | 4 | 7 | 18 |
| **probA05.bwp** | 5 | 22 | 39 | 5 | 7 | 20 |
| **probA06.bwp** | 6 | 26 | 47 | 6 | 23 | 44 |
| **probA07.bwp** | 7 | 16 | 28 | 7 | 28 | 64 |
| **probA08.bwp** | 8 | 58 | 107 | 8 | 56 | 113 |
| **probA09.bwp** | 9 | 70 | 106 | 9 | 62 | 99 |
| **probA10.bwp** | 10 | 206 | 284 | 12 | 263 | 353 |
| **probA11.bwp** | 11 | 88 | 154 | 12 | 183 | 310 |
| **probB03.bwp** | 3 | 5 | 58 | 3 | 5 | 58 |
| **probB04.bwp** | 4 | 5 | 75 | 4 | 5 | 75 |
| **probB05.bwp** | 5 | 6 | 82 | 6 | 13 | 150 |
| **probB06.bwp** | 6 | 8 | 83 | 6 | 6 | 70 |
| **probB07.bwp** | 7 | 43 | 375 | 7 | 16 | 178 |
| **probB08.bwp** | 8 | 81 | 660 | 9 | 57 | 576 |
| **probB09.bwp** | 9 | 300 | 2174 | 8 | 63 | 674 |
| **probB10.bwp** | 10 | 121 | 1135 | 9 | 106 | 1084 |
| **probB11.bwp** | 9 | 164 | 1399 | 9 | 31 | 306 |
| **probB12.bwp** | 9 | 36 | 422 | 10 | 84 | 1054 |
| **probB13.bwp** | 13 | 8787 | 51839 | 13 | 693 | 6583 |
| **probB14.bwp** | 14 | 6250 | 40492 | 17 | 17144 | 133577 |
| **probB15.bwp** | 14 | 4622 | 34427 | 16 | 1234 | 11588 |
| **probB16.bwp** | 15 | 12949 | 82971 | 18 | 5005 | 39950 |
| **probB17.bwp** | 16 | 98549 | 524843 | 16 | 773 | 7689 |
| **probB18.bwp** | 12 | 81 | 656 | 14 | 2644 | 22682 |
| **probB19.bwp** | 15 | 46349 | 205629 | 16 | 16031 | 101866 |
| **probB20.bwp** | 15 | 5464 | 37934 | 18 | 11288 | 85595 |