Artificial Intelligence 625 Project1

Student: Wei-Che Hsu

UIN: 826001725

**How to compile and use my submitted code:**

I compile and execute my code on the server linux2.cse.tamu.edu.

Put **BlockWorld\_Algo.cpp** and **BlockWorld\_Ds.h** to one folder

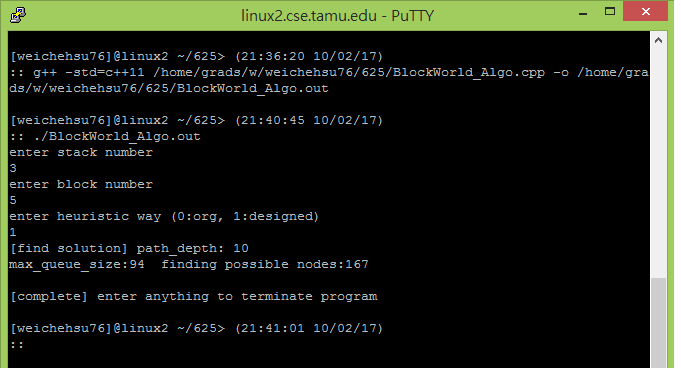
**Compile:** g++ -std=c++11 **Path/**BlockWorld\_Algo.cpp -o **Path/** BlockWorld\_Algo.out

For example, my folder path is: **/home/grads/w/weichehsu76/625/**

g++ -std=c++11 **/home/grads/w/weichehsu76/625/**BlockWorld\_Algo.cpp -o **/home/grads/w/weichehsu76/625/**BlockWorld\_Algo.out

Then, **./BlockWorld\_Algo.out** to execute my A\* program.

When executing this program, user should enter parameters as the below figure shows:



Ask your stack number? Just input one positive integer number and press enter

Ask your block number? Just input one positive integer number and press enter

Ask your heuristic way? input 0 and press enter🡺 Original heuristic (h3)

input 1 and press enter🡺 My heuristic (h1+h2+h3)

These 2 heuristic methods, I would explain in the next section.

After finished one test, the program will ask user to enter anything to terminate the program, just press enter is fine.

The **execution detail** will save in **proj1\_record.txt** including how to move blocks from the beginning states to the goal states. **proj1\_record.txt** will be generated in the folder user puts **BlockWorld\_Algo.cpp** and **BlockWorld\_Ds.h.**

If success, user can see the message **[find solution]** and the brief statistic result. Otherwise, you will see the message **[failure]** because the number of iteration is more than the number of MAX\_ITER. I define **MAX\_ITER** as 20000 now or there is no available node in priority queue.

I also put one **transcript,** which shows program traces, in the zip folder that I submit. In addition, I attached some partial **transcript** in the example section of the end in this report.

If user wants to change MAX\_ITER, it is in line 7 of **BlockWorld\_Algo.cpp.**

For waiting long time for some complex problems, I add **monitor message**

**[processing] iter\_cnt 500** for user to know the program is still running.

**Heuristic function:**

The function, h, I used can separate to 3 parts: h1, h2 and h3

h1: **How many blocks does player have to remove to put the target block in the first stack.** For example, h1 in the below state graph is 3. Because the target block is A, for making available position for A to put in the first stack, player needs to move C, E, and G to any other stack.

h2: **How many blocks does player have to remove to obtain the accessibility for the target block.** For example, h2 in the below state graph is 4. Because the target block is A, for obtain accessibility of A, player needs to move B, D, F, and H to any other stack.

h3: **this is the same as original heuristic**, just the number of out of order of blocks. For example, moving A to the first stack taking 1 step, and taking remain other blocks to the first stack in order at least take 7 steps. Thus, h3 in the below state is 7.

This is my log for one start state:

depth:0

1 | C E G

2 | A B D F H

3 |

f=g+h=15 = g + h1 + h2 + h3 = 0 + 3 + 4 + 8

f is total cost, g is the number of step, also equals to depth.

The depth of the root is 0, and the depth of first depth is 1.

**Priority queue always pop one node with the lowest f in each iteration.**

How do I decide the target block? just compare to the order blocks of the goal state, and choose the index of the first non-order blocks to find the corresponding block in the goal state.

1 | A B C D

2 |

3 | H G F E

For example, the target block is E in the above state graph.

The number of h1 and h2 is **essential steps** before putting the target block to the first stack. There is **no over-estimate** because there is some cost I don’t add to the proposed heuristic function. That is, After I take h1 and h2 to consideration for the current target block, moving remain out-of-order blocks would take more steps than h3-1.

Also, h3 is also the **essential** **steps** to put all blocks in order in the first stack.

Proposed heuristic function

Because Both of these heuristic functions do not over-estimate, and the value of Proposed heuristic function is always larger than that of for the same state graph. **I expect my method would have better performance.**

By definition of the textbook, admissible heuristic is one that **never overestimates** the cost to reach the goal. Therefore, my proposed heuristic function is **admissible**. Furthermore, it also satisfied **h(n) ≤ c(n, a, n’) + h(n’)** , n’ is successor node of node n. When player does a step which is followed my heuristic, **h(n) will transfer 1 cost to c(n, a, n’).** Like the below state graph, if **remain out-order blocks** are reverse in-order, and then the equal is satisfied that h(n) = c(n, a, n’) + h(n’), if remain out-order blocksneed more steps to take them in reverse order, then h(n) < c(n, a, n’) + h(n’)

State of n

1 |

2 | A

3 | E D C B

h(n)= h1+h2+h3 = 0 + 0 +5, f= g + h(n) = 0 + 5.

State of n’

1 | A

2 |

3 | E D C B🡸**remain out-order blocks**

h(n’)= h1+h2+h3 = 0 + 0 +4, f= g + h(n’) = 1 + 4. g is c(n, a, n’).

if blocks in the third stack is not in this reverse order to make h2 > 0

such that h(n) < c(n, a, n’) + h(n’)

Therefore, my heuristic also satisfied **consistency.**

Besides, I thought heuristic functions bring program **“knowledge or logic”** to move to the state which designers want. In the same depth, the program would take the decision step with high priority to remove non-order blocks in the first stack or remove the blocks on the target block.

**Program Flow:**

I use iterative way to represent A\* search method. I just briefly describe the program flow in pseudo code.

**My A\*()**{

Receive (stack\_no, blocks\_no)

Generate Start\_state and goal\_state.

Generate **root** node(Start\_state)

Add root node to **priority\_queue** and **visited\_vector**

Iter = 1

**While(Iter < MAX\_ITER)**{

If(priority\_queue without any nodes)

Goto **failure** // (break) message **[failure]** without possible path

Pop node **n** from priority\_queue

If(**n** has goal\_state)

Goto **[find solution]** // (break) **[find solution]** to print back-trace path

Successor(n)

Iter++

}

If (not go to goal)

Goto **failure** // message **[failure]** with iter = MAX\_ITER comes from here

}

**Successor (node n)**{

Expand possible **states** from n.state

For each **expand\_state** in **states{**

If **expand\_state** not exist in visited\_vector{

Add one node with **expand\_state** to priority\_queue and visited\_vector.

// the parent of this adding node is n

}

else{

if (cost of expand\_state < cost of existed node in visited\_vector)

update information in priority\_queue and visited\_vector

add one node into priority\_queue if this state is not in priority\_queue.

//for the existing state, especially the depth information.

// also update parent to n if necessary

}

}

}

}

**Table of Performance metrices:**

**heuristic: org = h3, heuristic: new = h1 + h2+ h3**

**[case] StackNum = 3**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **heuristic: org** |  | iter | queue\_size | path\_depth |
| stack = 3 | block = 5 | 14 | 22 | 5 |
| stack = 3 | block = 5 | 47 | 67 | 8 |
| stack = 3 | block = 5 | 391 | 353 | 10 |
| stack = 3 | block = 5 | 95 | 128 | 9 |
| stack = 3 | block = 5 | 220 | 218 | 10 |
| stack = 3 | block = 5 | 58 | 85 | 7 |
| stack = 3 | block = 5 | 59 | 75 | 8 |
| stack = 3 | block = 5 | 21 | 35 | 6 |
| stack = 3 | block = 5 | 34 | 59 | 8 |
| stack = 3 | block = 5 | 11 | 17 | 5 |
| avg |  | 95 | 105.9 | 7.6 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **heuristic: org** |  | iter | queue\_size | path\_depth |
| stack = 3 | block = 6 | 1755 | 1721 | 13 |
| stack = 3 | block = 6 | 1506 | 1416 | 13 |
| stack = 3 | block = 6 | 131 | 198 | 9 |
| stack = 3 | block = 6 | 130 | 216 | 10 |
| stack = 3 | block = 6 | 1533 | 1574 | 13 |
| stack = 3 | block = 6 | 112 | 189 | 10 |
| stack = 3 | block = 6 | 1416 | 1489 | 12 |
| stack = 3 | block = 6 | 719 | 817 | 12 |
| stack = 3 | block = 6 | 251 | 379 | 11 |
| stack = 3 | block = 6 | 475 | 587 | 11 |
| avg |  | 802.8 | 858.6 | 11.4 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **heuristic: org** |  | iter | queue\_size | path\_depth |
| stack = 3 | block = 7 | 11598 | 11332 | 16 |
| stack = 3 | block = 7 | 799 | 1084 | 12 |
| stack = 3 | block = 7 | 158 | 245 | 9 |
| stack = 3 | block = 7 | 5317 | 6082 | 15 |
| stack = 3 | block = 7 | 1093 | 1569 | 13 |
| stack = 3 | block = 7 | 34 | 56 | 7 |
| stack = 3 | block = 7 | 4778 | 5370 | 15 |
| stack = 3 | block = 7 | 2251 | 2838 | 14 |
| stack = 3 | block = 7 | 14520 | 13756 | 16 |
| stack = 3 | block = 7 | 642 | 1090 | 12 |
| avg |  | 4119 | 4342.2 | 12.9 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **heuristic: new** |  | iter | queue\_size | path\_depth |
| stack = 3 | stack = 5 | 16 | 30 | 9 |
| stack = 3 | stack = 5 | 6 | 14 | 5 |
| stack = 3 | stack = 5 | 41 | 50 | 10 |
| stack = 3 | stack = 5 | 7 | 15 | 6 |
| stack = 3 | stack = 5 | 33 | 46 | 10 |
| stack = 3 | stack = 5 | 11 | 19 | 8 |
| stack = 3 | stack = 5 | 5 | 11 | 4 |
| stack = 3 | stack = 5 | 22 | 41 | 8 |
| stack = 3 | stack = 5 | 22 | 40 | 9 |
| stack = 3 | stack = 5 | 29 | 37 | 10 |
| avg |  | 19.2 | 30.3 | 7.9 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **heuristic: new** |  | iter | queue\_size | path\_depth |
| stack = 3 | stack = 6 | 99 | 161 | 12 |
| stack = 3 | stack = 6 | 9 | 19 | 8 |
| stack = 3 | stack = 6 | 118 | 235 | 11 |
| stack = 3 | stack = 6 | 21 | 45 | 9 |
| stack = 3 | stack = 6 | 381 | 525 | 14 |
| stack = 3 | stack = 6 | 133 | 177 | 12 |
| stack = 3 | stack = 6 | 189 | 249 | 13 |
| stack = 3 | stack = 6 | 102 | 166 | 12 |
| stack = 3 | stack = 6 | 46 | 20 | 9 |
| stack = 3 | stack = 6 | 32 | 62 | 11 |
| avg |  | 113 | 165.9 | 11.1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **heuristic: new** |  | iter | queue\_size | path\_depth |
| stack = 3 | block = 7 | 239 | 301 | 14 |
| stack = 3 | block = 7 | 746 | 1156 | 14 |
| stack = 3 | block = 7 | 3348 | 4550 | 18 |
| stack = 3 | block = 7 | 535 | 765 | 15 |
| stack = 3 | block = 7 | 332 | 478 | 15 |
| stack = 3 | block = 7 | 41 | 67 | 12 |
| stack = 3 | block = 7 | 331 | 488 | 15 |
| stack = 3 | block = 7 | 780 | 1047 | 16 |
| stack = 3 | block = 7 | 264 | 580 | 13 |
| stack = 3 | block = 7 | 994 | 1336 | 15 |
| avg |  | 761 | 1076.8 | 14.7 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **heuristic: new** |  | iter | queue\_size | path\_depth |
| stack = 3 | block = 8 | 87 | 143 | 15 |
| stack = 3 | block = 8 | 6755 | 8551 | 18 |
| stack = 3 | block = 8 | 930 | 1560 | 17 |
| stack = 3 | block = 8 | 2666 | 3454 | 18 |
| stack = 3 | block = 8 | 30 | 74 | 11 |
| stack = 3 | block = 8 | 266 | 411 | 15 |
| stack = 3 | block = 8 | 1766 | 3263 | 16 |
| stack = 3 | block = 8 | 1854 | 2572 | 17 |
| stack = 3 | block = 8 | 48 | 72 | 14 |
| stack = 3 | block = 8 | 10866 | 14615 | 20 |
| avg |  | 2526.8 | 3471.5 | 16.1 |

**[case] StackNum = 5**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **heuristic: org** |  | iter | queue\_size | path\_depth |
| stack = 5 | block = 5 | 84 | 428 | 7 |
| stack = 5 | block = 5 | 182 | 718 | 8 |
| stack = 5 | block = 5 | 13 | 78 | 6 |
| stack = 5 | block = 5 | 152 | 558 | 8 |
| stack = 5 | block = 5 | 38 | 143 | 6 |
| stack = 5 | block = 5 | 16 | 105 | 6 |
| stack = 5 | block = 5 | 60 | 267 | 7 |
| stack = 5 | block = 5 | 68 | 318 | 6 |
| stack = 5 | block = 5 | 97 | 381 | 8 |
| stack = 5 | block = 5 | 9 | 65 | 6 |
| avg |  | 71.9 | 306.1 | 6.8 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **heuristic: org** |  | iter | queue\_size | path\_depth |
| stack = 5 | block = 6 | 975 | 3706 | 10 |
| stack = 5 | block = 6 | 650 | 2387 | 10 |
| stack = 5 | block = 6 | 144 | 800 | 8 |
| stack = 5 | block = 6 | 1100 | 3925 | 10 |
| stack = 5 | block = 6 | 946 | 3182 | 10 |
| stack = 5 | block = 6 | 968 | 3530 | 10 |
| stack = 5 | block = 6 | 865 | 3216 | 10 |
| stack = 5 | block = 6 | 362 | 1611 | 9 |
| stack = 5 | block = 6 | 369 | 1519 | 10 |
| stack = 5 | block = 6 | 1110 | 4352 | 10 |
| avg |  | 748.9 | 2822.8 | 9.7 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **heuristic: new** |  | iter | queue\_size | path\_depth |
| stack = 5 | block = 5 | 15 | 97 | 7 |
| stack = 5 | block = 5 | 10 | 78 | 8 |
| stack = 5 | block = 5 | 13 | 119 | 7 |
| stack = 5 | block = 5 | 9 | 65 | 8 |
| stack = 5 | block = 5 | 15 | 86 | 7 |
| stack = 5 | block = 5 | 8 | 54 | 7 |
| stack = 5 | block = 5 | 14 | 105 | 8 |
| stack = 5 | block = 5 | 33 | 200 | 8 |
| stack = 5 | block = 5 | 10 | 62 | 7 |
| stack = 5 | block = 5 | 8 | 66 | 7 |
| avg |  | 13.5 | 93.2 | 7.4 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **heuristic: new** |  | iter | queue\_size | path\_depth |
| stack = 5 | block = 6 | 7 | 84 | 10 |
| stack = 5 | block = 6 | 8 | 62 | 7 |
| stack = 5 | block = 6 | 71 | 371 | 9 |
| stack = 5 | block = 6 | 49 | 255 | 9 |
| stack = 5 | block = 6 | 23 | 161 | 9 |
| stack = 5 | block = 6 | 20 | 148 | 9 |
| stack = 5 | block = 6 | 14 | 129 | 8 |
| stack = 5 | block = 6 | 146 | 796 | 9 |
| stack = 5 | block = 6 | 80 | 406 | 9 |
| stack = 5 | block = 6 | 22 | 181 | 9 |
| avg |  | 44 | 259.3 | 8.8 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **heuristic: new** |  | iter | queue\_size | path\_depth |
| stack = 5 | block = 7 | 146 | 1032 | 11 |
| stack = 5 | block = 7 | 21 | 211 | 8 |
| stack = 5 | block = 7 | 88 | 691 | 12 |
| stack = 5 | block = 7 | 72 | 467 | 10 |
| stack = 5 | block = 7 | 320 | 2097 | 12 |
| stack = 5 | block = 7 | 60 | 521 | 11 |
| stack = 5 | block = 7 | 17 | 144 | 11 |
| stack = 5 | block = 7 | 125 | 793 | 11 |
| stack = 5 | block = 7 | 39 | 348 | 11 |
| stack = 5 | block = 7 | 330 | 2048 | 11 |
| avg |  | 121.8 | 835.2 | 10.8 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **heuristic: new** |  | iter | queue\_size | path\_depth |
| stack = 5 | block = 8 | 251 | 1575 | 13 |
| stack = 5 | block = 8 | 1454 | 9832 | 14 |
| stack = 5 | block = 8 | 315 | 2317 | 12 |
| stack = 5 | block = 8 | 158 | 1185 | 12 |
| stack = 5 | block = 8 | 1412 | 9463 | 13 |
| stack = 5 | block = 8 | 74 | 657 | 13 |
| stack = 5 | block = 8 | 55 | 573 | 11 |
| stack = 5 | block = 8 | 73 | 661 | 13 |
| stack = 5 | block = 8 | 403 | 2509 | 12 |
| stack = 5 | block = 8 | 406 | 2810 | 12 |
| avg |  | 460.1 | 3158.2 | 12.5 |

**[case] StackNum = 7**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **heuristic: org** |  | iter | queue\_size | path\_depth |
| stack = 7 | block = 5 | 9 | 93 | 6 |
| stack = 7 | block = 5 | 8 | 77 | 6 |
| stack = 7 | block = 5 | 54 | 370 | 7 |
| stack = 7 | block = 5 | 67 | 589 | 7 |
| stack = 7 | block = 5 | 402 | 2501 | 8 |
| stack = 7 | block = 5 | 237 | 1176 | 8 |
| stack = 7 | block = 5 | 12 | 164 | 7 |
| stack = 7 | block = 5 | 202 | 1207 | 7 |
| stack = 7 | block = 5 | 51 | 518 | 7 |
| stack = 7 | block = 5 | 158 | 1010 | 7 |
| avg |  | 120 | 770.5 | 7 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **heuristic: org** |  | iter | queue\_size | path\_depth |
| stack = 7 | block = 6 | 651 | 4817 | 9 |
| stack = 7 | block = 6 | 369 | 2949 | 9 |
| stack = 7 | block = 6 | 308 | 2537 | 9 |
| stack = 7 | block = 6 | 305 | 2409 | 9 |
| stack = 7 | block = 6 | 385 | 3060 | 8 |
| stack = 7 | block = 6 | 299 | 2355 | 8 |
| stack = 7 | block = 6 | 53 | 709 | 8 |
| stack = 7 | block = 6 | 430 | 3332 | 9 |
| stack = 7 | block = 6 | 142 | 1328 | 8 |
| stack = 7 | block = 6 | 46 | 450 | 7 |
| avg |  | 298.8 | 2394.6 | 8.4 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **heuristic: new** |  | iter | queue\_size | path\_depth |
| stack = 7 | block = 5 | 22 | 261 | 6 |
| stack = 7 | block = 5 | 20 | 230 | 7 |
| stack = 7 | block = 5 | 10 | 52 | 9 |
| stack = 7 | block = 5 | 13 | 175 | 6 |
| stack = 7 | block = 5 | 7 | 97 | 6 |
| stack = 7 | block = 5 | 8 | 102 | 7 |
| stack = 7 | block = 5 | 23 | 206 | 7 |
| stack = 7 | block = 5 | 58 | 501 | 8 |
| stack = 7 | block = 5 | 7 | 103 | 6 |
| stack = 7 | block = 5 | 33 | 263 | 7 |
| avg |  | 20.1 | 199 | 6.9 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **heuristic: new** |  | iter | queue\_size | path\_depth |
| stack = 7 | block = 6 | 16 | 209 | 7 |
| stack = 7 | block = 6 | 61 | 842 | 9 |
| stack = 7 | block = 6 | 10 | 136 | 9 |
| stack = 7 | block = 6 | 16 | 253 | 8 |
| stack = 7 | block = 6 | 35 | 529 | 9 |
| stack = 7 | block = 6 | 64 | 689 | 8 |
| stack = 7 | block = 6 | 12 | 225 | 7 |
| stack = 7 | block = 6 | 14 | 247 | 7 |
| stack = 7 | block = 6 | 90 | 965 | 10 |
| stack = 7 | block = 6 | 16 | 314 | 8 |
| avg |  | 33.4 | 440.9 | 8.2 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **heuristic: new** |  | iter | queue\_size | path\_depth |
| stack = 7 | block = 7 | 535 | 4800 | 11 |
| stack = 7 | block = 7 | 81 | 1490 | 11 |
| stack = 7 | block = 7 | 65 | 944 | 9 |
| stack = 7 | block = 7 | 118 | 1404 | 10 |
| stack = 7 | block = 7 | 35 | 615 | 10 |
| stack = 7 | block = 7 | 67 | 1096 | 11 |
| stack = 7 | block = 7 | 78 | 1302 | 8 |
| stack = 7 | block = 7 | 23 | 441 | 8 |
| stack = 7 | block = 7 | 484 | 5578 | 11 |
| stack = 7 | block = 7 | 28 | 475 | 8 |
| avg |  | 151.4 | 1814.5 | 9.7 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **heuristic: new** |  | iter | queue\_size | path\_depth |
| stack = 7 | block = 8 | 144 | 2238 | 11 |
| stack = 7 | block = 8 | 312 | 5855 | 13 |
| stack = 7 | block = 8 | 508 | 5964 | 12 |
| stack = 7 | block = 8 | 1538 | 17284 | 13 |
| stack = 7 | block = 8 | 76 | 1134 | 9 |
| stack = 7 | block = 8 | 1295 | 13173 | 12 |
| stack = 7 | block = 8 | 220 | 2955 | 11 |
| stack = 7 | block = 8 | 722 | 7649 | 13 |
| stack = 7 | block = 8 | 137 | 2390 | 11 |
| stack = 7 | block = 8 | 101 | 2252 | 12 |
| avg |  | 505.3 | 6089.4 | 11.7 |

**Discussion:**

**Did my heuristic work well as I expected?**

When under the same parameters (stack\_number, block\_number), the above performance metrics shows my heuristic is better because **the average queue size and iteration loops** under my heuristic is **obvious lower** than that under original heuristic. Of course, they have the **nearly same average path length**.

Average performance under original heuristic

|  |  |  |  |
| --- | --- | --- | --- |
| (#stacks, #blocks) | iter | queue\_size | path\_length |
| (3,5) | 95 | 105.9 | 7.6 |
| (3,6) | 802.8 | 858.6 | 11.4 |
| (3,7) | 4119 | 4342.2 | 12.9 |
| (5,5) | 71.9 | 306.1 | 6.8 |
| (5,6) | 748.9 | 2822.8 | 9.7 |
| (7,5) | 120 | 770.5 | 7 |
| (7,6) | 298.8 | 2394.6 | 8.4 |

Average performance under my heuristic

|  |  |  |  |
| --- | --- | --- | --- |
| (#stacks, #blocks) | iter | queue\_size | path\_length |
| (3,5) | 19.2 | 30.3 | 7.9 |
| (3,6) | 113 | 165.9 | 11.1 |
| (3,7) | 761 | 1076.8 | 14.7 |
| (5,5) | 13.5 | 93.2 | 7.4 |
| (5,6) | 44 | 259.3 | 8.8 |
| (7,5) | 20.1 | 199 | 6.9 |
| (7,6) | 33.4 | 440.9 | 8.2 |

**What were the largest size problem I tried?**

Under my heuristic, I have tried (#stack, #block) = (3, 8) with path length = 20 case, and I attached it in the example section. Also, I tried (#stack, #block) = (7, 9) and (5,9) and record the result in the attached transcript file.

**How does queue size grow?**

The queue size with grow like **initially** assuming all stack has at least one block, and then after some loops, there are generating some repeated states which does not add to the **priority queue** when the depth of generating node is larger than that of original node in **visited\_vector**. At that time, the growth of queue size would become slower sometimes even to 0 or less than ten.

**Did increasing the number of blocks make problem more difficult?**

Yes, it did. The average length of path from the beginning state graph to the goal state graph grows with the increasing number of blocks.

**Did increasing the number of stacks make problem easier?**

For my heuristic, when **the number of stacks from 3 to 5 and the number of blocks is larger than or equal to 5**, the problem becomes easier because the number of states increases bring free stack to adjust blocks character order easily. The length of path to goal decreases 4 implying that **the problem becomes easier.** Also, when the number of stacks from 5 to 7, there is some improvement, and it brings more space complexity to my heuristic. However, if the number of blocks is equal to or less than the number of stacks, it couldn’t gain benefit from increasing the number of stacks. For original heuristic, it has the similar phenomenon which I describe above.

Average performance under my heuristic

|  |  |  |  |
| --- | --- | --- | --- |
| (#stacks, #blocks) | iter | queue\_size | depth |
| (3,7) | 761 | 1076.8 | 14.7 |
| (5,7) | 121.8 | 835.2 | 10.8 |
| (7,7) | 151.4 | 1814.5 | 9.7 |
| (5,5) | 13.5 | 93.2 | 7.4 |
| (7,5) | 20.1 | 199 | 6.9 |

**Did your program generally find optimal solution?**

If the inference in heuristic section is correct, both of them are admissible. They can ensure player to find the **optimal** solution which only relates to **the length of path**. However, it can’t ensure the search speed. As I know, the speed of my heuristic is not good enough.

**Any Idea to improve my heuristic?**

When (#stacks, #blocks) becomes larger, like more than 7 stacks or more than 9 blocks, my heuristic tries many nodes with the same f value, which is g + h, that takes much iterations. Maybe I can try to **see the remain blocks order after taking target block** to reduce priority of some bad frontiers by adding more cost to them.

**Examples:**

1. **My heuristic applied on one random example. (length of path: 20)**

========[ New Test is coming ]========

stack number:3 block number:8 heuristic:1

1 | C H

2 | A E G

3 | B D F

=====[ expected goal ]=====

1 | A B C D E F G H

2 |

3 |

iter#:1 queue\_size:1

access node: f=g+h=12 depth:0

iter#:2 queue\_size:6

access node: f=g+h=12 depth:1

……

iter#:10866 queue\_size:14615

access node: f=g+h=20 depth:20

[find solution] path\_depth: 20 iter:10866

max\_queue\_size:14615 finding possible nodes:25480

depth:0

1 | C H

2 | A E G

3 | B D F

depth:1

1 | C H

2 | A E

3 | B D F G

depth:2

1 | C H E

2 | A

3 | B D F G

depth:3

1 | C H E A

2 |

3 | B D F G

depth:4

1 | C H E A

2 | G

3 | B D F

depth:5

1 | C H E A

2 | G F

3 | B D

depth:6

1 | C H E

2 | G F

3 | B D A

depth:7

1 | C H

2 | G F E

3 | B D A

depth:8

1 | C

2 | G F E H

3 | B D A

depth:9

1 |

2 | G F E H C

3 | B D A

depth:10

1 | A

2 | G F E H C

3 | B D

depth:11

1 | A

2 | G F E H C D

3 | B

depth:12

1 | A B

2 | G F E H C D

3 |

depth:13

1 | A B

2 | G F E H C

3 | D

depth:14

1 | A B C

2 | G F E H

3 | D

depth:15

1 | A B C D

2 | G F E H

3 |

depth:16

1 | A B C D

2 | G F E

3 | H

depth:17

1 | A B C D E

2 | G F

3 | H

depth:18

1 | A B C D E F

2 | G

3 | H

depth:19

1 | A B C D E F G

2 |

3 | H

depth:20

1 | A B C D E F G H

2 |

3 |