**CSCE 625**

**ARTIFICIAL INTELLIGENCE**

**PROGRAMMING ASSIGNMENT#1**

**Note:** It is recommended to go through the Instructions, if you want to test my program, which is present in the other file with .docx extension.

**Application of A\* Search for Blocksworld Puzzle:**

I have developed three heuristics which are admissible/consistent for solving the Blocksworld puzzle and all three of them are based on a single idea which is described in Heuristic-1. Heuristic-2 is an enhancement of the same idea but is a lot better than heuristic-1. Heuristic-3, which is very complex and computationally slightly slower, is developed for solving puzzles with very large parameters (block size and stack size). Heuristic-3 can solve a problem size of 22 blocks and 7 stacks without failure but requires 15-30 mins. So, for the detailed explanation, heuristic-1 is considered and other heuristics are not thoroughly explained because of the complex nature of the algorithms.

**Heuristic-1:**

The heuristic relies on the idea of estimating the number of moves required for each block present in each stack based on their relative position.

Consider the following scenario:

The blocks are represented in my model from 1 to n:

Let **5 6 7 4 3 2 1** be the blocks that are present on stack-n (n cannot be considered as 1 which will be explained later)

**Stack#1 - - - - -**

**…**

**Stack#n 5 6 7 4 3 2 1**

By looking at the scenario it is evident that the blocks **1, 2, 3 & 4** can be pushed directly into stack-1 which results in 1 step for each of those blocks to reach the goal state and the blocks **5, 6 & 7** require at least two steps for each to reach the goal state.

This idea can be summarized as the blocks which are in ascending order from the right may require 1 or more steps and the blocks which are in descending order may require at least two steps to reach the goal state.

To make the heuristic optimal, the lower bound of the number of steps each blocks takes, is considered.

Let us discuss about the blocks in Stack-1. In this case, all the blocks that are not already in the goal state may have to be moved at least twice to reach the goal state. So, in the heuristic, the number of blocks that are not in correct position multiplied by two is considered for the calculation.

A pseudocode for the algorithm is given below:

Since, in each step we are considering the lower bound on the number of steps required for each block to reach the goal position, we can claim that this solution is admissible and hence it is also optimal. Although this idea looks simple, it achieves solution for a 5-stack, 10-block puzzle within 5000 goal tests whereas BFS takes more than 1,00,000 goal tests (which is a failure).

This heuristic is further improved in upcoming algorithms.

**Heuristic-2:**

In this heuristic, there’s an improvement over the previous one in a way that the elements that are present on the right side of the minimum element in each stack are always required to be moved at least twice before reaching the goal state. Everything else remains the same. It makes sense that in each step the minimum block in each stack is the first block that must be pushed to the goal state. All the blocks to the left of the minimum block are again followed the same approach as the above. This heuristic outperforms the previous heuristic because the guessed number of steps are closer to the actual value in this case. Also, it is admissible since only the lower bound of the possible values of each block is considered in the calculation.

The pseudo code for this algorithm is not detailed since only the step to find the minimum element in each stack is additional in this heuristic.

**Heuristic-3:**

Among all, this heuristic is the best in the way that it takes less than half of the queue size or the goal tests as compared to the other heuristics. But it is computationally slower than heuristic-2 because this algorithm goes ahead of several steps and estimates the heuristic which is quite complicated.

This approach works in two steps. In the first step, all the blocks in the first stack except the ones that are already in the goal position are redistributed to the other stacks so that it foresees the step when the blocks are cleared out of stack-1 which eventually must happen. In the next step, heuristic-2 is run to compute the estimated number of steps. This algorithm is so efficient in the way that when there is a scenario where there are some blocks in the first stack, it clears out all the blocks that are not in the goal position in very few goal tests/iterations.

The crucial part of this heuristic is the approach of redistribution of the blocks. This happens in a way such that all stacks are evenly consumed and, in each stack it is ensured that minimum steps required to reach the goal position. The algorithm for step-1 is detailed below using a pseudo code.

Although the pseudo code doesn’t look efficient here, it is written efficiently in the project.

Heuristics-3 is more intelligent because it considers into account the number of stacks which was not considered into the previous ones. The admissibility remains from previous heuristics since the number of stacks has been considered which is different from others.

**Performance Measurement:**

The performances of the three heuristics are mentioned here to demonstrate the ability of the heuristics to solve different problem sizes below.

**Table-1:** Stack size – 3 and Blocks size - 5

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # Steps to reach goal state | Heristic-1  Max Queue Depth | Heuristic-1  # Goal tests | Heristic-2  Max Queue Depth | Heuristic-2  # Goal tests | Heristic-3  Max Queue Depth | Heuristic-3  # Goal tests |
| 9 | 87 | 37 | 76 | 31 | 50 | 20 |
| 11 | 173 | 70 | 121 | 48 | 80 | 34 |
| 11 | 189 | 86 | 123 | 49 | 39 | 16 |
| 7 | 22 | 9 | 22 | 9 | 22 | 9 |
| 9 | 65 | 24 | 64 | 23 | 55 | 20 |
| 7 | 24 | 9 | 24 | 9 | 24 | 9 |
| 7 | 31 | 12 | 30 | 11 | 26 | 10 |
| 9 | 68 | 26 | 48 | 17 | 28 | 11 |
| 5 | 18 | 7 | 18 | 7 | 18 | 7 |
| 9 | 76 | 26 | 76 | 26 | 32 | 11 |
| Mean: 8.4 | **Max:** 189 | **Max**: 86 | **Max**: 123 | **Max**: 49 | **Max**: 80 | **Max**: 34 |

From now, I’ll start ignoring heuristic-1’s performance since it takes few minutes for solving the problems

**Table-2:** Stack size – 3 and Blocks size - 10

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # Steps to reach goal state | Heristic-2  Max Queue Depth | Heuristic-2  # Goal tests | Heristic-3  Max Queue Depth | Heuristic-3  # Goal tests |
| 17 | 414 | 159 | 369 | 144 |
| 16 | 217 | 85 | 175 | 69 |
| 23 | 15235 | 5875 | 2397 | 902 |
| 23 | 26005 | 11277 | 22991 | 8893 |
| 22 | 3515 | 1329 | 1150 | 432 |
| 20 | 1896 | 679 | 555 | 205 |
| 22 | 3093 | 1351 | 424 | 178 |
| 19 | 653 | 251 | 422 | 186 |
| 18 | 398 | 143 | 187 | 73 |
| 21 | 2782 | 980 | 426 | 153 |
| Mean: 20.1 | **Max**: 26005 | **Max**: 11277 | **Max**: 22991 | **Max**: 8893 |

**Table-3:** Stack size – 5 and Blocks size - 8

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # Steps to reach goal state | Heristic-2  Max Queue Depth | Heuristic-2  # Goal tests | Heristic-3  Max Queue Depth | Heuristic-3  # Goal tests |
| 9 | 124 | 11 | 124 | 11 |
| 12 | 945 | 64 | 396 | 12 |
| 8 | 249 | 22 | 249 | 22 |
| 12 | 787 | 58 | 501 | 37 |
| 9 | 104 | 11 | 104 | 11 |
| 16 | 7121 | 594 | 7008 | 588 |
| 12 | 413 | 35 | 426 | 35 |
| 15 | 1620 | 127 | 1003 | 80 |
| 11 | 618 | 48 | 441 | 35 |
| 11 | 804 | 66 | 804 | 66 |
| Mean: 11.5 | **Max**: 7121 | **Max**: 594 | **Max**: 7008 | **Max**: 588 |

**Table-4:** Stack size – 5 and Blocks size - 13

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # Steps to reach goal state | Heristic-2  Max Queue Depth | Heuristic-2  # Goal tests | Heristic-3  Max Queue Depth | Heuristic-3  # Goal tests |
| 22 | 6475 | 416 | 1961 | 129 |
| 21 | 3918 | 254 | 1784 | 123 |
| 20 | 9924 | 739 | 9835 | 732 |
| 20 | 1789 | 125 | 1874 | 136 |
| 19 | 1308 | 88 | 1931 | 128 |
| 21 | 14556 | 936 | 1340 | 85 |
| 21 | 60009 | 4798 | 14674 | 1216 |
| 23 | 31890 | 3005 | 47880 | 4771 |
| 19 | 1915 | 125 | 1875 | 19 |
| 21 | 7366 | 571 | 6653 | 502 |
| Mean: 20.7 | **Max**: 60009 | **Max**: 4798 | **Max**: 47880 | **Max**: 4771 |

**Table-5:** Stack size – 7 and Blocks size – 20

This is the maximum problem size, the presented heuristic can handle within real time. Although, it can solve other problems such as 7 stacks and 22 blocks for which an instance is mentioned in the examples section, it takes quite an amount of time (~5-30 minutes, cannot guarantee that).

|  |  |  |
| --- | --- | --- |
| # Steps to reach goal state | Heristic-3  Max Queue Depth | Heuristic-3  # Goal tests |
| 25 | 7725 | 232 |
| 27 | 5950 | 181 |
| - | Max limit reached | - |
| 27 | 15148 | 442 |
| 32 | 13052 | 374 |
| 27 | 7535 | 250 |
| 26 | 7992 | 250 |
| - | Max limit reached | - |
| 26 | 16409 | 475 |
| 31 | 29273 | 826 |
| Mean: 27.625 | **Max**: 29273 | **Max**: 826 |

As you may notice that the heuristic is able to solve a problem of depth-32 which is quite high and at the same time requiring only 374 goal tests. This shows the capability of the heuristic. But, on the opposite side it is not able to solve few problems as noted as “max limit reached” in the table. Therefore, it is not quite guaranteed that it works well for all problems of the mentioned size when you try to test, but in most cases, it’ll solve.

Results apart, let’s come to the discussion part.

**Discussion of Results:**

**Note:** If I don’t mention anything about the heuristic in the upcoming report, it is assumed that I refer to heuristic-3

**Observations about the performance of heuristics:**

As I mentioned during the description of the heuristics, heuristic-3 achieves goal state by conducting less goal tests as compared to heuristics-1&2 because it is more intelligent in a way that it goes a lot ahead in the graph and computes its heuristic value, thus eliminating lot of subtrees, which may not lead to a solution/goal, in the initial phase. The same can be verified in all the tabular measurements presented above although there are few outliers.

**Did it work well as I expected?**

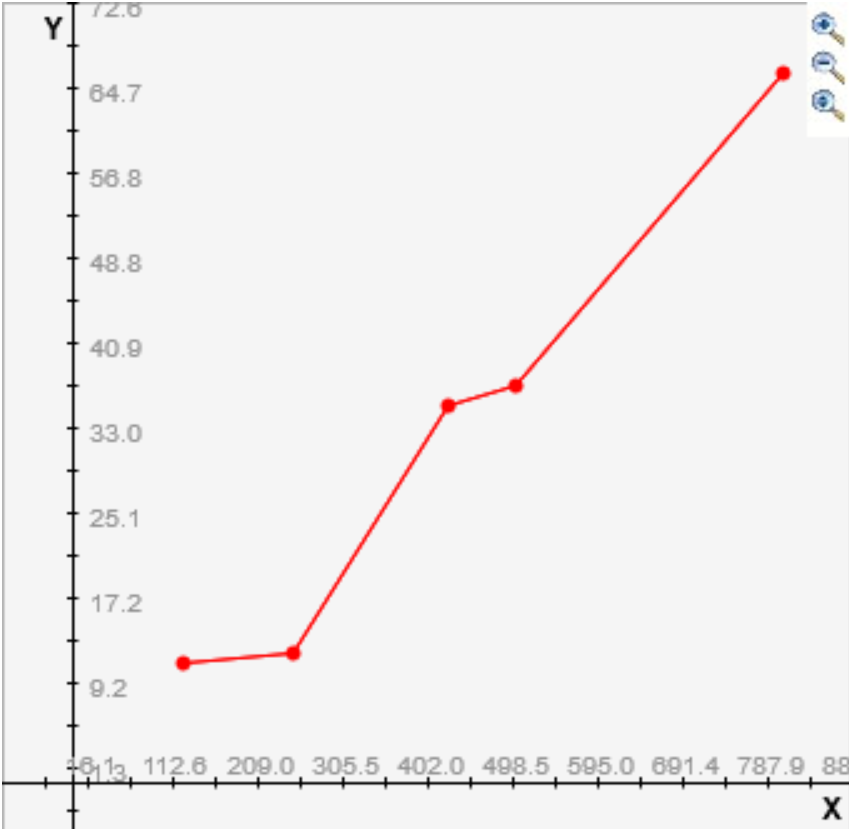
It worked better than what I have anticipated from all the heuristics. But heuristic-3 has surprised me more than the others.

**What are the largest problems the heuristic could solve?**

Heuristic-3 works for most of the problems up to the size of 7 stacks and 20 blocks problem and generates the path almost instantly. It also worked for 7 stacks and 22 blocks but took almost 10 minutes to generate the path and is shown in the example traces present in the other file.

**How large did the queue grow as a function of number of iterations?**

Although it is difficult to interpret how the queue size grew with the number of iterations but it can be estimated to be nearly linear for my heuristic which is plotted below for one of the problem size.



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

For my first and second heuristics which doesn’t consider into account the number of stacks in the problem, increasing the number of stacks definitely made the problem difficult to solve.

In my third heuristic, the process is divided into two steps in which one of the steps considers the stack size into account inherently thus making the problem easier to solve if the stack size grows but when it is coupled with the second step (heuristic-2) which makes the problem harder with increasing stack size, it finally makes the problem harder.

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

During the description of the heuristic, it has been proved that the functions are admissible thus indicating the solutions are optimal and I have verified the same matching with different heuristics.

**Scope of improvement of the heuristic:**

I believe there’s still a scope of improvement of the heuristic I have developed since I have not made the algorithm completely dependent on the number of stacks in the problem, which should have been considered for an ideal heuristic.