**RAW Data from 100 test cases across various depths**

Depth 2

Depth 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Array | Search cost H1 | Execution Time H1 (ms) | Search cost H2 | Execution Time H2 (ms) |
| 312645078 | 5 | 7 | 5 | 2 |
| 142305678 | 7 | 2 | 7 | 2 |
| 120345678 | 5 | 1 | 5 | 2 |
| 142305678 | 7 | 1 | 7 | 1 |
| 312405678 | 7 | 1 | 7 | 0 |
| 120345678 | 5 | 0 | 5 | 1 |
| 312645078 | 5 | 1 | 5 | 1 |
| 142305678 | 7 | 1 | 7 | 1 |
| 312405678 | 7 | 0 | 7 | 0 |
| 142305678 | 7 | 1 | 7 | 1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Array | Search cost H1 | Execution Time H1 (ms) | Search cost H2 | Execution Time (ms) |
| 312475680 | 12 | 8 | 12 | 3 |
| 142358670 | 12 | 2 | 12 | 2 |
| 032415678 | 15 | 2 | 12 | 2 |
| 125304678 | 12 | 1 | 12 | 1 |
| 312605748 | 12 | 1 | 12 | 1 |
| 125348670 | 10 | 2 | 10 | 1 |
| 142375680 | 12 | 1 | 12 | 1 |
| 310452678 | 12 | 1 | 12 | 1 |
| 125304678 | 12 | 1 | 12 | 1 |
| 312605748 | 12 | 2 | 12 | 1 |

Depth 6

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Array | Search cost H1 | Execution Time H1 (ms) | Search cost H2 | Execution Time (ms) |
| 142307685 | 27 | 2 | 24 | 2 |
| 142705368 | 35 | 2 | 24 | 1 |
| 312458067 | 17 | 2 | 17 | 1 |
| 125634078 | 17 | 2 | 17 | 1 |
| 310642785 | 15 | 1 | 15 | 2 |
| 014352678 | 32 | 2 | 24 | 1 |
| 312407685 | 27 | 1 | 24 | 1 |
| 125308647 | 17 | 1 | 17 | 1 |
| 432105678 | 69 | 2 | 34 | 2 |
| 312705468 | 48 | 1 | 21 | 1 |

Depth 8

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Array | Search cost H1 | Execution Time H1 (ms) | Search cost H2 | Execution Time (ms) |
| 235104678 | 112 | 3 | 32 | 1 |
| 312508467 | 64 | 2 | 26 | 1 |
| 140652738 | 40 | 2 | 29 | 1 |
| 162305748 | 117 | 3 | 71 | 2 |
| 350421678 | 80 | 1 | 53 | 1 |
| 035421678 | 70 | 2 | 29 | 1 |
| 432175068 | 96 | 2 | 39 | 1 |
| 032715468 | 89 | 2 | 26 | 1 |
| 351642078 | 76 | 13 | 49 | 5 |
| 312758460 | 56 | 2 | 26 | 1 |

Depth 10

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Array | Search cost H1 | Execution Time H1 (ms) | Search cost H2 | Execution Time (ms) |
| 425138067 | 121 | 7 | 27 | 3 |
| 041362785 | 107 | 4 | 50 | 3 |
| 125378460 | 135 | 4 | 48 | 2 |
| 184302657 | 173 | 4 | 43 | 1 |
| 025371468 | 116 | 16 | 41 | 5 |
| 254138670 | 149 | 11 | 27 | 2 |
| 250174368 | 89 | 5 | 27 | 2 |
| 310762485 | 127 | 6 | 47 | 2 |
| 025164738 | 96 | 4 | 39 | 2 |
| 325701468 | 139 | 4 | 64 | 2 |

Depth 12

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Array | Search cost H1 | Execution Time H1 (ms) | Search cost H2 | Execution Time (ms) |
| 042178356 | 305 | 9 | 80 | 2 |
| 371602845 | 446 | 9 | 151 | 4 |
| 158402367 | 420 | 9 | 151 | 3 |
| 450231678 | 503 | 7 | 159 | 3 |
| 128357460 | 276 | 5 | 99 | 2 |
| 120648537 | 258 | 4 | 76 | 2 |
| 428103657 | 424 | 7 | 51 | 1 |
| 042168537 | 268 | 3 | 78 | 1 |
| 310672845 | 387 | 10 | 134 | 4 |
| 320475168 | 441 | 10 | 119 | 3 |

Depth 14

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Array | Search cost H1 | Execution Time H1 (ms) | Search cost H2 | Execution Time (ms) |
| 045182367 | 1046 | 16 | 293 | 5 |
| 140378652 | 838 | 13 | 335 | 5 |
| 328501674 | 1205 | 17 | 227 | 5 |
| 362415780 | 1064 | 15 | 453 | 7 |
| 415308627 | 1318 | 17 | 417 | 6 |
| 182465370 | 1128 | 16 | 382 | 6 |
| 143672085 | 1043 | 14 | 231 | 4 |
| 142806735 | 1349 | 19 | 176 | 3 |
| 435628170 | 991 | 13 | 139 | 2 |
| 328507416 | 830 | 12 | 60 | 1 |

Depth 16

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Array | Search cost H1 | Execution Time H1 (ms) | Search cost H2 | Execution Time (ms) |
| 031782564 | 2506 | 33 | 303 | 5 |
| 415206378 | 3224 | 38 | 729 | 10 |
| 641205738 | 3270 | 42 | 768 | 10 |
| 625104378 | 2686 | 31 | 1193 | 15 |
| 412365078 | 3224 | 41 | 1294 | 16 |
| 351702684 | 2733 | 35 | 1189 | 15 |
| 258347061 | 1947 | 25 | 261 | 4 |
| 042713685 | 2285 | 30 | 716 | 9 |
| 237145680 | 2513 | 33 | 495 | 7 |
| 625184730 | 2365 | 29 | 332 | 5 |

Depth 18

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Array | Search cost H1 | Execution Time H1 (ms) | Search cost H2 | Execution Time (ms) |
| 640372815 | 5736 | 66 | 925 | 10 |
| 058241637 | 5114 | 61 | 672 | 10 |
| 287145360 | 4846 | 56 | 513 | 7 |
| 032845167 | 6504 | 74 | 876 | 11 |
| 315408726 | 6027 | 69 | 1689 | 21 |
| 150864327 | 5905 | 62 | 405 | 6 |
| 528167034 | 4409 | 50 | 635 | 9 |
| 614205378 | 7399 | 79 | 1777 | 23 |
| 610243785 | 4916 | 56 | 587 | 9 |
| 741638052 | 5605 | 64 | 856 | 11 |

Depth 20

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Array | Search cost H1 | Execution Time H1 (ms) | Search cost H2 | Execution Time (ms) |
| 530412867 | 12678 | 137 | 1570 | 21 |
| 348207651 | 17272 | 179 | 2198 | 27 |
| 564218037 | 10653 | 109 | 1200 | 16 |
| 710582364 | 13813 | 137 | 2689 | 32 |
| 164837052 | 9983 | 102 | 653 | 8 |
| 248175036 | 12432 | 126 | 1550 | 21 |
| 351682470 | 17302 | 166 | 3122 | 36 |
| 823401675 | 20291 | 205 | 2834 | 37 |
| 458607213 | 15462 | 147 | 651 | 8 |
| 568247130 | 11313 | 121 | 210 | 3 |

**Approach:**

My 8-puzzle solver program was created in Java, using Eclipse IDE. There are two main classes that I used: Board and Solver. The Board class handles the creation of board nodes. It has 3 total constructors: one of them is used for passing a user-generated input, one for passing an input from a file, and the last one is used to create the children nodes. There are also 5 private variables, as well as the methods to assign data to them. The first variable is an ArrayList of Integers, which stores the current configuration of the board. The distanceFromRoot variable stores the depth level the node is in. The other ArrayList of Board stores all the children nodes that are successors of the current node, and finally the integers h1 and h2 stores the Hamming function H1 and the Manhattan function H2. As for the solver class, the main algorithm is stored in the Solver constructor. First, an empty board is created. Then the isSolvable function will check whether that board can be solved or not. If it can, the root will be added to the frontier, which is a priority queue and operates with either H1 or H2 functions. The root state is then compared with the goal state. If they both match, the algorithm ends and the puzzle is done. If not, the root will get removed, and its children will be moved onto the frontier. The root node configuration data will get stored in a hash set called exploredSet. The process will now repeat, with the children node with the lowest g(n) + h(n) selected to be the current node. The process is completed when one of the generated child node equals to the goal state.

**Graphs and Analyses:**

Average search cost and execution time based on the raw inputs at every depth:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Depth | Average Search cost H1 | Average Execution Time H1 (ms) | Average Search cost H2 | Average Execution Time (ms) | Number of cases tested |
| 2 | 6.2 | 1.5 | 6.2 | 1.1 | 10 |
| 4 | 12.1 | 2.1 | 11.8 | 1.4 | 10 |
| 6 | 30.4 | 1.6 | 21.7 | 1.3 | 10 |
| 8 | 80 | 3.2 | 38 | 1.5 | 10 |
| 10 | 125.2 | 6.5 | 41.3 | 2.4 | 10 |
| 12 | 372.8 | 7.3 | 109.8 | 2.5 | 10 |
| 14 | 1081.2 | 15.2 | 271.3 | 4.4 | 10 |
| 16 | 2675.3 | 33.7 | 728 | 9.6 | 10 |
| 18 | 5646.1 | 63.7 | 893.5 | 11.7 | 10 |
| 20 | 14119.9 | 142.9 | 1667.7 | 20.9 | 10 |

A\* performance from the table in the book

|  |  |  |
| --- | --- | --- |
| Depth | A\*(h1) | A\*(h2) |
| 2 | 6 | 6 |
| 4 | 13 | 12 |
| 6 | 20 | 18 |
| 8 | 39 | 25 |
| 10 | 93 | 39 |
| 12 | 227 | 73 |
| 14 | 539 | 113 |
| 16 | 1301 | 211 |
| 18 | 3056 | 363 |
| 20 | 7276 | 676 |

The above graph shows the corresponding performance of 2 heuristic functions that I implemented compares to that of the book. This graph is not a great measure on the performance at depth less than 12, because the y-axis is divided in increments of 2000 search costs, which means we will not be able to see the variance of data at less than 2000 search costs. However, this graph is a good measure of higher depths, ranging from 12 to 20. The common pattern is that h2 from the book and the assignment could generate significantly less nodes than h1, therefore helping in reducing the execution time. It also confirms that H2 function at higher depth will always run faster than H1. The H1 and H2 functions from the code that I created seems to run slightly slower to the H1 and H2 functions from the book. One reason for that could be because the computer performance is slower on my PC, or it could be due to program implementations, as well as the chosen language and compiler.

Looking at the average search cost and execution time table, we can see that on most cases, the H1 function surpasses H2 on the number of nodes generated. As the depth increase, the search cost on H2, while still increasing, does so at a much lower factor than H1. At depth 10, the ratio between H1 and H2 search cost is already 3.03, while at depth 20, this ratio quickly jumped to 8.46. This means at depth 20, the number of nodes generated by function H1 is more than 8 times higher than the number of nodes generated by H2. Because of the lower nodes generated, function H2 consequently could finish executing in less time, as evident in all depth levels. At depth 20, the ratio of execution time between H1 and H2 is 6.83, which means the Manhattan H2 function can re-run almost 7 other depth-20 8-puzzle board before the H1 function could solve the first board. However, this is not to say that the H1 function is completely inferior to the H2 function. By looking at the smaller depth tables, we can see that at depth lower than 12, although the search cost is still significantly lower in H2, the execution time between the two functions is close to each other, and most of the time the puzzle is solved in less than 10 mili-seconds, which is very fast indeed. One other significant benefit to the H1 function is that it is easier to write and deploy compares to H2, because the programmer only need to specify whether the current state is different than the goal state, as oppose to in H2 function, where the programmer will have to specify the path cost from the current state to the goal state. As for the H2 function, because of the sheer speed advantage over H1, it is possible that the H2 function can also be used in a higher complexity puzzle, like the 15-puzzle problem, where the search costs and traversing choices between the hole and the numbers are much higher.