**NAME:Peter Chinetti**

**COLLABORATOR(S): *Krishen Blows. Krishen and I sat together to understand what this lab was asking for, and after understanding that it was to simply run simulations, decided to run the simulations entirely on my server. My server has a lot more 'horse power' than either of our two laptops. The analysis however, was done independently.***

**CS480 – HOMEWORK 4**

**Assigned on: Friday, 9/20/2013**

**Due: Sunday, 9/29/2013, 11:59pm**

There is only one question. Please submit your solution through black board assignment page.

1. **[100 points]** In this assignment, you will write code to compare Iterative Deepening Search and A\* search using three different heuristics. For randomly generated 8-puzzle boards, you will compare the total number of nodes expanded and the total time for these algorithms. Please download the attached java files.

Compare the following algorithms:

* Iterative Deepening Search
* A\* search with misplaced tile heuristic
* A\* search with Manhattan distance heuristic
* A\* search with iterative deepening search

To run the first three algorithms, the attached code is sufficient. For the last one, A\* search with iterative deepening search, you need to write code for the IterDeepHeuristicFunction.java. What you are supposed to do is to run iterative deepening search, and find the optimal path cost, and return that as your heuristic value.

Fill in the following two tables and answer the respected questions. If you are not able to run an algorithm due to time or memory issues, please note it on its row/column.

Do not submit code. Submit only the requested information.

**Question 1:** Please provide details on the computer you ran the simulations on.

**Processor: 2x Intel Xeon E5410**

**RAM size: 8GB**

**OS: Ubuntu Server 12.04.3 LTS**

**Table Average number of nodes expanded by each algorithm.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Depth | Iterative Deepening | A\* - Misplaced Tiles | A\* - Manhattan Distance | A\* - IterDeep |
| 1 | 1 | 1 | 1 | 1 |
| 2 | 3 | 2 | 2 | 2 |
| 3 | 11 | 3 | 3 | 3 |
| 4 | 33 | 4 | 4 | 4 |
| 5 | 109 | 6 | 5 | 5 |
| 6 | 300 | 10 | 7 | 6 |
| 7 | 824 | 20 | 10 | 7 |
| 8 | 1513 | 31 | 15 | 11 |
| 9 | 6800 | 125 | 34 | 9 |
| 10 | 16493 | 346 | 55 | 11 |
| 11 | 54251 | 442 | 77 | 14 |
| 12 | 104619 | 474 | 41 | 12 |
| 13 | 446391 | 2451 | 253 | 16 |
| 14 | 1267086 | 12659 | 1376 | 14 |
| 15 | 3346634 | 39080 | 3121 | Timeout |
| 16 | 6360708 | 73804 | 9970 |  |
| 17 | 24049131 | 120879 | 9768 |  |
| 18 | 64253216 | 266482 | 7794 |  |
| 19 | 201687132 | 834700 | 5543 |  |
| 20 | 816230824 | 2448255 | 185235 |  |
| 21 | Memory | 5097311 | 539616 |  |
| 22 |  | Timeout | 198678 |  |
| 23 |  |  | 303415 |  |
| 24 |  |  | 295509 |  |
| 25 |  |  |  |  |

**Question 2:** Which algorithm expanded the fewest number of nodes on average? Why?

A \* IterDeep expanded the fewest nodes, because its heuristic always choses the best path by repeatedly solving the problem.

**Table 2 Average time.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Depth | Iterative Deepening | A\* - Misplaced Tiles | A\* - Manhattan Distance | A\* - IterDeep |
| 1 | 0.006500 | 0.010500 | 0.009500 | 0.014500 |
| 2 | 0.000750 | 0.000750 | 0.000750 | 0.022250 |
| 3 | 0.001875 | 0.001375 | 0.001000 | 0.037375 |
| 4 | 0.005200 | 0.001700 | 0.001300 | 0.040200 |
| 5 | 0.007700 | 0.001900 | 0.001500 | 0.031800 |
| 6 | 0.017800 | 0.002800 | 0.001400 | 0.062100 |
| 7 | 0.025200 | 0.004800 | 0.001900 | 0.148400 |
| 8 | 0.016500 | 0.005000 | 0.002100 | 0.357500 |
| 9 | 0.018200 | 0.009500 | 0.003400 | 1.082000 |
| 10 | 0.038600 | 0.010600 | 0.004600 | 3.762700 |
| 11 | 0.128500 | 0.011100 | 0.004100 | 11.418700 |
| 12 | 0.227300 | 0.006100 | 0.001800 | 22.565500 |
| 13 | 0.893200 | 0.022400 | 0.001800 | 90.670400 |
| 14 | 2.608200 | 0.117900 | 0.019400 |  |
| 15 | 6.754000 | 0.414100 | 0.031800 |  |
| 16 | 13.231300 | 0.784200 | 0.112600 |  |
| 17 | 50.683000 | 1.392000 | 0.111800 |  |
| 18 | 136.265700 | 3.221000 | 0.081100 |  |
| 19 | 422.971000 | 10.843500 | 0.707400 |  |
| 20 | 1156.341800 | 38.848400 | 3.639500 |  |
| 21 | 3563.354600 |  | 15.637900 |  |
| 22 |  |  | 2.934200 |  |
| 23 |  |  | 4.733500 |  |
| 24 |  |  | 4.377000 |  |
| 25 |  |  |  |  |

**Question 3:** Which algorithm was the fastest on average, and why?

A \* Manhattan Distance because the Manhattan Distance heuristic is best suited to the problem.

**Question 4:** Which algorithm would you prefer and why?

A \* Manhattan Distance because it completes the problem most quickly without using obscene amounts of memory.