Nicolas Stoian CS780 AI

Easy Puzzle
1 3 4
8 6 2
7 0 5

	Nodes Expanded	Total Time
A* using tiles out of place	19	>1ms, 699951ns
A* using Manhattan	16	>1ms, 809618ns
IDA* with Manhattan	19	>1ms, 566501ns
Depth-first branch and bound	30	>1ms, 813581ns

Number of moves in solution = 5
Depth-first branch and bound best solution time = 549324ns

Medium Puzzle
2 8 1
0 4 3
7 6 5

	Nodes Expanded	Total Time
A* using tiles out of place	80	>1ms, 1751033ns
A* using Manhattan	42	>1ms, 1291226ns
IDA* with Manhattan	51	>1ms, 799708ns
Depth-first branch and bound	76	>1ms, 1158106ns

Number of moves in solution = 9
Depth-first branch and bound best solution time = 772622ns

Hard Puzzle
2 8 1
4 6 3
0 7 5

	Nodes Expanded	Total Time
A* using tiles out of place	257	>1ms, 4673387ns
A* using Manhattan	71	>1ms, 1940307ns
IDA* with Manhattan	117	>1ms, 1269425ns
Depth-first branch and bound	127	>1ms, 1470591ns

Number of moves in solution = 12
Depth-first branch and bound best solution time = 965199ns
Worst Puzzle
5 6 7

4 0 8
3 2 1

	Nodes Expanded	Total Time
A* using tiles out of place	286847	156055ms, 156049934349ns
A* using Manhattan	1969	31ms, 35750020ns
IDA* with Manhattan	3852	>1ms, 10071168ns
Depth-first branch and bound	4222	>1ms, 12591189ns

Number of moves in solution = 30
Depth-first branch and bound best solution time = 9482535ns

Analysis

The performance for all of the algorithm implementations was satisfactory on my computer. A* using tiles out of place heuristic takes a long time with the worst puzzle, almost 3 minutes, however I feel that this is acceptable considering the amount of nodes expanded. The A* algorithms typically take a bit longer than the DFBnB and IDA* implementations, I believe this is due to those algorithms in my implementation having to check the open list in each iteration. The DFBnB and IDA* implementations performed extremely well in all cases, including the worst scenario.

Questions and answers

1. What is the number of possible states of the board?

The maximum number of possible configurations of the board would be 9! = 362880.

2. What is the average number of possible moves from a given position of the board?

There are 24 total possible moves for the 9 positions of the board, so 2.66667 average

3. Estimate how many moves might be required for an optimal (minimum number of moves) solution to a “worst-case” problem (maximum distance between starting and goal states). Explain how you made your estimate (Note this is an open-ended question; any logical answer may suffice).

The farthest any given tile can be from its goal is 4 moves and there are 8 tiles on the board, so assuming that each tile is the farthest it can be from the goal state then an optimal solution to this would require at least 32 moves.

4. Assuming the answer to question #2 is the “average branching factor” and a depth as in the answer to question #3, estimate the number of nodes that would have to be examined to find an answer by the brute force breadth-first search.

This would require more than $\sum_{k=1}^{32} 2.6^k$, or approximately 3.0903× 10^13 nodes.

5. Assuming that your computer can examine one move per millisecond, would such a blind-search solution to the problem terminate before the end of the semester?

No, it is very doubtful that it would. 3.0903× 10^13 ms is approximately 979.93 years.

6. The “worst” example problem given above is actually one of the easiest for humans to solve. Why do you think that is the case? What lessons for AI programs are suggested by this difference between human performance and performance of your search program?

A human is able to look at the start board and the goal board and intuitively realize that all that is required is to rotate the tiles around the board. We are able to solve this linearly in 30 moves, which is pretty amazing. I believe that we are able to do this because we can look at all of the tiles at once and see the relationship between the start and goal states. Perhaps we can develop AI that considers multiple tiles simultaneously to help make decisions.

7. Compare A*, DFBnB, and IDA* and discuss their advantages and disadvantages.

A* expands the least amount of nodes out of the 3, however due to additional overhead required to implement it the time required is longer than DFBnB, and IDA*.

DFBnB expands the most amount of nodes out of the 3 algorithms, however it is not too many more than IDA*. Performance is slightly slower than the IDA* algorithm, though the overall run time is longer DFBnB finds the optimal solution in the fastest time.

IDA* has the fastest overall times of the algorithms and expands slightly less nodes than DFBnB. Overall it offers the best performance while expanding a reasonable amount of nodes.

Easy Puzzle Solution path

1 3 4
8 6 2
7 0 5

1 3 4
8 0 2
7 6 5

1 3 4
8 2 0
7 6 5

1 3 0
8 2 4
7 6 5

1 0 3
8 2 4
7 6 5

1 2 3

8 0 4
7 6 5

Medium Puzzle Solution path

2 8 1
0 4 3
7 6 5

0 8 1
2 4 3
7 6 5

8 0 1
2 4 3
7 6 5

8 1 0
2 4 3
7 6 5

8 1 3
2 4 0
7 6 5

8 1 3
2 0 4
7 6 5

8 1 3
0 2 4
7 6 5

0 1 3
8 2 4
7 6 5

1 0 3
8 2 4
7 6 5

1 2 3
8 0 4
7 6 5

Hard Puzzle Solution path

2 8 1

4 6 3
0 7 5

2 8 1
4 6 3
7 0 5

2 8 1
4 0 3
7 6 5

2 8 1
0 4 3
7 6 5

0 8 1
2 4 3
7 6 5

8 0 1
2 4 3
7 6 5

8 1 0
2 4 3
7 6 5

8 1 3
2 4 0
7 6 5

8 1 3
2 0 4
7 6 5

8 1 3
0 2 4
7 6 5

0 1 3
8 2 4
7 6 5

1 0 3
8 2 4
7 6 5
1 2 3
8 0 4

7 6 5

Worst Puzzle Solution path

5 6 7
4 0 8
3 2 1

5 6 7
4 8 0
3 2 1

5 6 7
4 8 1
3 2 0

5 6 7
4 8 1
3 0 2

5 6 7
4 8 1
0 3 2

5 6 7
0 8 1
4 3 2

0 6 7
5 8 1
4 3 2

6 0 7
5 8 1
4 3 2

6 7 0
5 8 1
4 3 2

6 7 1
5 8 0
4 3 2

6 7 1
5 8 2
4 3 0

6 7 1

5 8 2
4 0 3

6 7 1
5 8 2
0 4 3

6 7 1
0 8 2
5 4 3

0 7 1
6 8 2
5 4 3

7 0 1
6 8 2
5 4 3

7 1 0
6 8 2
5 4 3

7 1 2
6 8 0
5 4 3

7 1 2
6 8 3
5 4 0

7 1 2
6 8 3
5 0 4

7 1 2
6 8 3
0 5 4

7 1 2
0 8 3
6 5 4

0 1 2
7 8 3
6 5 4

1 0 2

7 8 3
6 5 4

1 2 0
7 8 3
6 5 4

1 2 3
7 8 0
6 5 4

1 2 3
7 8 4
6 5 0

1 2 3
7 8 4
6 0 5

1 2 3
7 8 4
0 6 5

1 2 3
0 8 4
7 6 5

1 2 3
8 0 4
7 6 5