AI Project 1: Sliding Tiles

Roy Howie

February 8, 2017

The Puzzle

History

The 15-puzzle harkens back to the late 19th century, when it was invented by Noyes Chapman, the Postmaster of Canastota, New York, who applied for a patent in March of 1880.

It is often misattributed, however, to Sam Loyd (1841-1911), who is widely known as being one of America's greatest puzzle-writers. This is due to Loyd's (debunked) misinformation campaign—beginning in 1891 and continuing until his death 20 years later—that he was the puzzle's sole inventor.

The puzzle, along with the smaller 8-puzzle variant, has been enjoyed throughout the years for its simplicity. The 15-puzzle is sometimes called the 16-puzzle; similarly, the 8-puzzle is occassionally referred to as the 9-puzzle. It is but a matter of semantics: to count the blank tile or not?

How it works

The concept is simple. Given a starting state—such as the one below—move tiles horizontally or vertically into the blank space until the goal state is reached. For example, in Figure 1, the player would first move the **14** tile to the left. Next, the player would move the **15** tile to the left. This last action would complete the puzzle.

But how does a player know whether a puzzle is solvable? Indeed, Figure 3 presents a puzzle state which is unsolvable, i.e., one which can never be permuted into the "goal state" represented by Figure 2.

1	2	3	4
5	6	7	8
9	10	11	12
13		14	15

Figure 1: Starting State

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Figure 2: Goal State

1	2	3	4
5	6	7	8
9	10	11	12
13	15	14	

Figure 3: Unsolvable State

Determining a puzzle's solvability

As there are $16! = 20,922,789,888,000$ different puzzle board arrangements, listing which starting arrangements are good and which are bad is clearly unfeasible. Indeed, half of all possible arrangements—about 10 trillion—cannot be permuted into the goal state represented in Figure 2. For convenience, we shall henceforth take "solvable" to mean "can be permuted into the (standard) goal state represented in Figure 2."

A puzzle state is solvable if it is of *even* permutation, as proven by W.E. Story in 1879. Conversely, a puzzle state is unsolvable if it of *odd* permutation, as proven by W.W. Johnson in 1879.

Determining a puzzle's permutation

Let P be a puzzle and let $T_i \in P$ be the tile containing the number i. Reading from left to the right and from top to bottom, let n_i be the amount of tiles with numbers less than i which occur after T_i. Ignore the blank tile. Let N be the sum of tile inversions:

$$N = \sum_{i=2}^{15} n_i$$

Note that the sum begins at $i = 2$, as $n_1 \equiv 0$, for no tiles contain numbers of value less than 1.

Let $e \in \{1 \dots n\}$ be the row number of the blank tile. Thus, Permutation$(P) = N + e \pmod 2$ and the puzzle is solvable if and only if $N + e$ is even.

An example

Consider the puzzle state represented in Figure 4. We then have that $n_6 = 3$, as three tiles of value less than six occur after T_6: T_2, T_3, and T_4. See Table 1 for an enumeration of all T_i.

1	8	7	13
5	6	10	15
2	14	12	
3	9	4	11

Figure 4: Determining n_6

Table 1: n_i for Figure 4

n_2	0	n_6	3	n_{10}	4	n_{14}	5
n_3	0	n_7	5	n_{11}	0	n_{15}	7
n_4	0	n_8	6	n_{12}	4	e	3
n_5	3	n_9	1	n_{13}	9		

Hence, Permutation$(P) = N + e = 50$, which is even, so the puzzle represented in Figure 4 is solvable.

Generalizing puzzle solvability to other dimensions

As mentioned previously, other similar forms of the 15-puzzle exist, such as the smaller 8-puzzle. In general, if a puzzle is of dimension $k \times k$, determining its solvability is not much more complicated than the process detailed above for the 15-puzzle of size 4×4.

Let N be the sum of tile inversions n_i, as before, and let e be the row number of the blank tile. Consider a puzzle P of size $k \times k$. When k is odd, P is solvable if N is even; when k is even, P is solvable when $N + e$ is even.

Indeed, one can even consider non-square puzzles of size $k \times l$, but that is outside the scope of this paper.

Solving the puzzle

Goal

Given a *valid* puzzle starting state, find a path of individual tile movements which leads to the goal state. A *valid* puzzle is said to be one which has an even permutation and can therefore be permuted into the standard goal state.

Constraints

Due to the prohibitive amount of states—over 20 trillion—the 15-puzzle can inhabit and the corresponding amount of memory and time a personal computer would require to process said states, we will consider the smaller 8-puzzle, as it has only $9! = 362,880$ different states.

Approach

We will create a search tree out of the puzzle. The search operator will be all previously unseen moves available to a given state via sliding a single tile. The root node will be a random valid starting state.

The puzzle states will be (conveniently) stored in memory as hashes; this will allow for easy lookup in a set to determine whether a state has already been visited and should therefore be passed over by the search operator. For example, the puzzle state from Figure 4 would be represented by the hash `187d56af2ec0394b`. This is obtained by reading from left to right and from top to bottom, considering the numbers in base-16; the blank tile is read as zero.

Algorithms

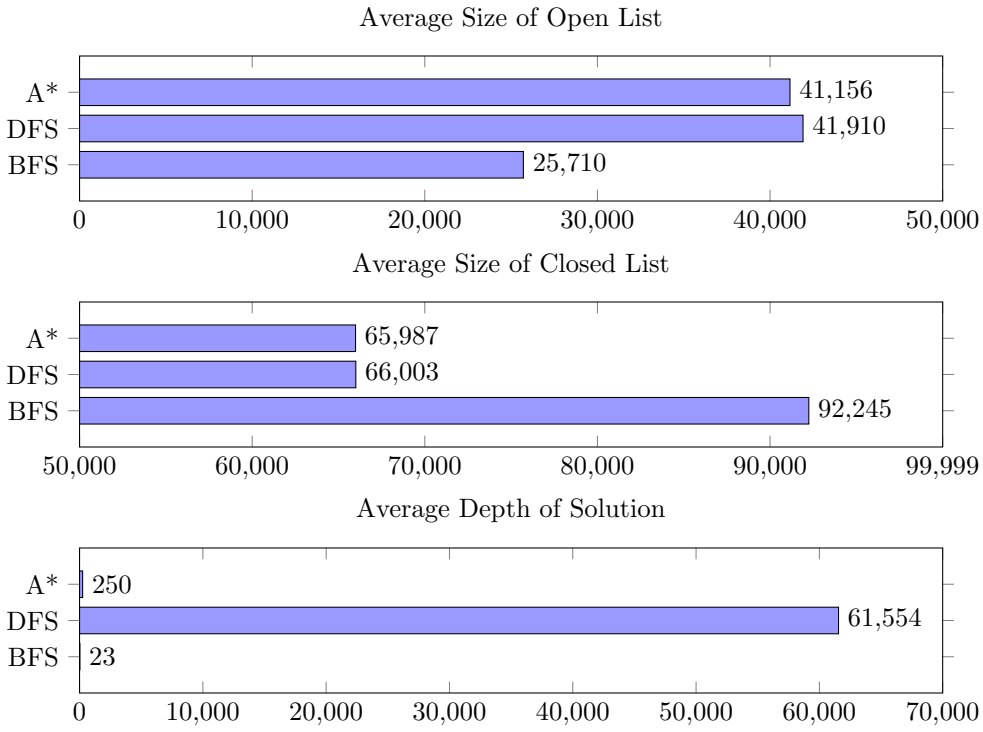
The search tree will be searched using DFS (depth-first search), BFS (breadth-first search), and A*. Iterative deepening search was determined not to be suitable to the puzzle, as it resulted in a search tree no different from BFS; this is due to the nature of the search tree, which will be explained in detail later.

Rubric

The three algorithms will be run for 1300 trials each. Each will be evaluated based on the the length of the closed list (total number of nodes visited), the maximum length of the open list (maximum number of nodes present at any point in the search tree), and the depth of the tree to the desired goal state (the number of tiles moved by the algorithm to solve the puzzle).

Results

Figure 5: Average Size of Open & Closed Lists, Solution Path



Discussion

DFS was by far the least adequate search algorithm tasked with finding a solution to the 8-puzzle. A* and BFS were much closer in their performance, with BFS providing a much more practical solution. A* used the Taxicab Metric as its heuristic; this heuristic is known to be admissable, but perhaps there are better, more sophisticated heuristics available. If the algorithms were ranked based on the abstract concept of "usability," they would be placed in the following order: BFS, A*, DFS.

The search tree for the sliding tiles puzzles has a low branching factor of zero to three—the fourth branch always corresponds to the previous state—but an extremely high depth. This was most likely the reason for the poor showing by DFS. As DFS traverses the tree, plunging downwards, it skips very few states. In fact, it is very likely to hit the children of previously unexplored children, as the graph of puzzle states is highly connected, i.e., a puzzle state can be easily reached from most other states with very few intermediary steps.

This resulted in DFS's ambling about, exploring long, winding "caverns" or "tunnels" until it either reached a cul-de-sac or the solution. Upon reaching a dead-end, DFS would simply backtrack, beginning its pursuit anew. This ultimately caused DFS to have an average solution depth of over 60 thousand; compare this to the average solution depths of A* and BFS at 250 and 23, respectively. Hence, DFS is completely impractical, as no human would suffer through so many moves to solve the simple 8-puzzle. Indeed, even A* would be unbearable with its 250 moves, as most players would easily find a solution requiring a number of steps on an order of magnitude similar to that of BFS.

Ignoring the practicality of the solutions presented by the three search algorithms, the results were much more even-handed, but still lopsided. A* and DFS both had an average open list size of around 41k, whereas

BFS had an average open list size of 25k. This was a surprising result, as BFS typically requires more memory to store the search tree, i.e., has a larger open tree size.

As previously seen states were not revisited by the algorithms, BFS most likely had a smaller search tree because it more quickly eliminated dead-ends and bad choices, preventing those paths from being searched. This was also the reason the Iterativing Deepening Search algorithm was not included, as it was practically the same as BFS due to the low branching factor and the small depth of potential solution tree (an 8-puzzle can always be solved in 31 moves or fewer).

On the other hand, A* and DFS visited about 70% as many states as BFS did, with 65k and 92k states visited on average, respectively. Indeed, BFS took, on average, about 38% longer to complete its search than A* and DFS did.

Despite being somewhat slower, BFS was the clear winner, as it was able to present a user-friendly solution using less memory.

Bibliography

- http://mathworld.wolfram.com/15Puzzle.html
- http://www.geeksforgeeks.org/check-instance-15-puzzle-solvable/
- http://www.geeksforgeeks.org/check-instance-8-puzzle-solvable/