Mini-Project 1: Heuristic Search.

Marceau TONELLI (40102633)
Clément PÉAU (40102553)

In this paper we will discuss the way we solved and implemented the solution to the 11 tiles problem. The problem consists of a 4 * 3 board containing 11 tiles numbered from 1 to 11 and one empty tile (numbered 0). The objective is to reach certain configuration of the board where the tiles match a specific order. To do so, the “player” can switch the empty tile with one of the adjacent tiles (diagonals included).

The objective of the project was to solve this problem via different algorithms, including Depth First Search, Best First Search, and A* ; using different heuristics to inform the search. This report will be divided in three sections, one dedicated to the algorithms used and the way we implemented them, as long as the heuristics we chose to use and why we chose to use them, another one in which we will discuss the results we obtained with those algorithms and heuristics, and finally we will analyse and discuss these results.

1. Implementation:

As said in the introduction, we used different State Search algorithms in order to solve this problem. For the code, we chose to work with Python, as it is a very easy language to adopt and is very handy to setup a project from scratch (plus we were both already familiar with the language).

We first started by implementing non informed Depth First and Breadth First algorithms, in order to layout the basic functionalities we needed (such as how to manage the nodes, how to keep trace of the search path, how to keep the state of the board in memory and how to make permutations of tiles, etc…). Once implemented, these two algorithms didn’t have good performances at all, but were a good base to start building from.

The main problem we had with depth first search was that with such a high branching factor we wouldn't be assured of finding a proper solution without running out of ressources first. We added a maximum depth on the Depth First, and when the algorithm run out of states to search, it will increase the maximum depth by 5 until it finds a solution

Once those two base algorithms were implemented, we needed to define heuristic functions in order to implement more informed algorithms, such as Best First Search and A*. We chose to make a quickly computed but not very precise heuristic function, and another one, heavier to compute but which will give a more informed result. We settled with the following two:

Misplaced tiles heuristic:

This heuristic simply counts the number of tiles that aren't at the same position as they are in the goal state. It is very easy and cheap to compute but it doesn't make the difference between a tile being just next to the goal position, and a tile being at the other end of the board. From now on, we will call this heuristic **h1**.

Tiles distance heuristic:

Unlike the previously described heuristic, this one doesn't only counts the misplaced tiles but also computes the distance between their position in the current and their position in the goal state. If we assign to each tile two **x** and **y** coordinates corresponding to their position in the board, we can compute the said distance the following way (given that this heuristic is called **h2**):

h2(n) = sqrt((x_goal - x_state) ^ 2 + (y_goal - y_state) ^ 2)

As for each node, if it already is in the same position the distance will be zero (same as with h1), and if it is not, the distance will always be greater or equal to 1 (greater than with h1). Therefore we can say that for each node **n**, we can affirm the following: **h2(n) > h1(n)**, thus making the second heuristic more informed than the first one.

Once these heuristics defined and implemented, we implemented both Best First and A* algorithm, running them with each one of these heuristics.

2. Results:

After all algorithms were implemented and tested, we observed that not some were more suited to solve this problem than others. We are thus going to detail each algorithm's pros and cons.

Depth First Search:

This algorithm was the slowest to run, and the one that led to the largest search path, as a branch can be very deep before reaching a leaf or a state already visited, thus in the closed list. We observed that it ran through more than 200.000 states before finding a solution. However, as soon as we added an iterative depth to it, it became significantly more performant, as it wasn't going too deep in branches leading nowhere. We can see the results of iterative Depth First Search below (**0** being the initial state):

```
0 [1, 0, 3, 7, 5, 2, 6, 4, 9, 10, 11, 8]

b [1, 0, 3, 7, 5, 2, 6, 4, 9, 10, 11, 8]
f [1, 2, 3, 7, 5, 0, 6, 4, 9, 10, 11, 8]
g [1, 2, 3, 7, 5, 6, 0, 4, 9, 10, 11, 8]
d [1, 2, 3, 0, 5, 6, 7, 4, 9, 10, 11, 8]
h [1, 2, 3, 4, 5, 6, 7, 0, 9, 10, 11, 8]
l [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 0]
Number of room visited: 2033
Length of the solution: 6
```

However, due to its uninformed nature, the performances deeply vary from one initial state to another. For instance, with the following initial state: **[10, 8, 7, 3, 5, 2, 6, 4, 11, 1, 9, 0]**, the algorithm ran through more than 300.000 nodes without finding any solution before we shut it down.

Breadth First Search:

Even though it wasn't required in the project's subject, we chose to implement a Breadth First Search algorithm, in order to test and compare its performances with the other algorithms. We observed better results (in terms of performance) than with the Depth First Search Algorithm, but only if the solution was reachable in a reasonable number of moves. Here we see that with the same initial state given to

the DFS algorithm, the results are better if the goal state's depth is small enough. However with another initial state it runs through many nodes and never seems to find the answer. For example with the same second initial state given to the DFS algorithm (**[10, 8, 7, 3, 5, 2, 6, 4, 11, 1, 9, 0]**) 300.000 nodes were also visited unsuccessfully before we shut down the algorithm. We observed later with the informed algorithms that the goal state was reachable in around 35 moves, which is a too high depth, thus leading to a very high branching factor.

```
0 [1, 0, 3, 7, 5, 2, 6, 4, 9, 10, 11, 8]

b [1, 0, 3, 7, 5, 2, 6, 4, 9, 10, 11, 8]
f [1, 2, 3, 7, 5, 0, 6, 4, 9, 10, 11, 8]
g [1, 2, 3, 7, 5, 6, 0, 4, 9, 10, 11, 8]
d [1, 2, 3, 0, 5, 6, 7, 4, 9, 10, 11, 8]
h [1, 2, 3, 4, 5, 6, 7, 0, 9, 10, 11, 8]
l [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 0]
Number of room visited: 1205
Length of the solution: 6
```

Best First Search:

This is the first informed algorithm we have implemented, with the two heuristic functions described previously. We observe significant performance enhancements compared to the previous uninformed search algorithms, as the heuristic is supposed to make one node preferable to another one by evaluating how close it is to the goal state.

Here are the results obtained with the two heuristics for two different initial states. The first initial board state is optimal for the misplaced tiles heuristic function, as for each state there is a possible move that places a tile to its right position. However we see with the second example that it is not always the case, as the visited nodes and result path are significantly smaller. We can note nevertheless that the execution time is still higher even when visiting a smaller number of nodes as the distance heuristic is heavier to compute than the misplaced tiles one.

```
0 [1, 0, 3, 7, 5, 2, 6, 4, 9, 10, 11, 8]

Misplaced Heuristic
b [1, 0, 3, 7, 5, 2, 6, 4, 9, 10, 11, 8]
f [1, 2, 3, 7, 5, 0, 6, 4, 9, 10, 11, 8]
[...]
h [1, 2, 3, 4, 5, 6, 7, 0, 9, 10, 11, 8]
l [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 0]
Number of room visited: 6
Length of the solution: 6

Distance heuristic
b [1, 0, 3, 7, 5, 2, 6, 4, 9, 10, 11, 8]
f [1, 2, 3, 7, 5, 0, 6, 4, 9, 10, 11, 8]
[...]
h [1, 2, 3, 4, 5, 6, 7, 0, 9, 10, 11, 8]
l [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 0]
Number of room visited: 9
Length of the solution: 6

0 [10, 8, 7, 3, 5, 2, 6, 4, 11, 1, 9, 0]

Misplaced heuristic
l [10, 8, 7, 3, 5, 2, 6, 4, 11, 1, 9, 0]
g [10, 8, 7, 3, 5, 2, 0, 4, 11, 1, 9, 6]
[...]
k [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 0, 11]
l [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 0]
Number of room visited: 478
Length of the solution: 36

Distance heuristic
l [10, 8, 7, 3, 5, 2, 6, 4, 11, 1, 9, 0]
i [10, 8, 7, 3, 5, 2, 6, 4, 0, 1, 9, 11]
[...]
g [1, 2, 3, 4, 5, 6, 0, 8, 9, 10, 11, 7]
l [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 0]
Number of room visited: 327
Length of the solution: 34
```

A* algorithm:

A* is the second informed search algorithm we implemented.
The main change over the BFS we add the node depth to the heuristic to make sure we find the shortest solution possible. Depending on the heuristic the A* can have a bigger search space than BFS. However A* is completely dependent of its heuristic.

We observe identical results than with the BFS algorithm with the first example, however we observe a performance bottleneck with bigger search paths, making the algorithm run very slowly. Therefore we don't have the results output to present for this algorithm, as we didn't have the time to wait until the algorithm completion.

Bibliography and acknowledgements

As for references, we used this post which discusses the ideas behind A*.
http://theory.stanford.edu/~amitp/GameProgramming/ImplementationNotes.html