SOLVING PUZZLE PROBLEM USING HEURISTIC SEARCH

Daniel Adinugroho¹ & Esti Haryono²

¹ Fakultas Ilmu Komputer Unika Soegijapranata Semarang
Jl. Pawiyatan Luhur IV/1, Bendan Duwur, Semarang 50234

² Alumnus Fakultas Ilmu Komputer Unika Soegijapranata Semarang
adinugroho@unika.ac.id, esti1706@yahoo.com

ABSTRAK

Puzzle is common example problems in basic Artificial Intelligence study. There are many types of puzzle problems. The one that will be discussed in this paper is one similar to the Tetravex game. The puzzle has nine tiles. Each tile has four faces, top, bottom, left and right. Each tile also contains a number from 1 - 9. In the beginning, the puzzle is empty and the user should put each tile in the puzzle with the only rule is that two tiles can only be placed next to each other if the numbers on adjacent face match. Basically, this problem is just a searching problem or more specifically searching a path put all tiles in the puzzle as quick as possible where adjacent face tile’s will have the same number. There are many ways to get the solution to this problem. The easiest way is using uninformed search or sometimes called blind search, such as Depth First Search or Breadth First Search. However, both searching techniques are not guarantee to get the solution. Even, when a solution is found, it may not have the lowest cost or the shortest path to the solution. To get the better solution, we could use informed search, such as: Hill Climbing Search, Best First Search, Beam Search, Branch and Bound and A* algorithm. Informed search or sometimes called heuristic search is differed to the uninformed search because it uses heuristic function. Informed search use heuristic function to decide which path will be explored next. The best algorithm to get the shortest path or the lowest cost is A* algorithm. In this paper, it will be discussed how to solve the puzzle problem using A* algorithm. To get the best of this algorithm, it should be decided what heuristic function will be used. This paper will show using the calculating correct tiles’ position as the heuristic function. However, the tile will not be placed in random, but first the tiles will be stored in four arrays according to the four face its has, up, bottom, left and right. The placing process is depends on the requirement by looking a tile in the corresponding array starting from center position that has the most adjacent face tiles.. This should reduce the numbers of trials, which hopefully will get the goal quickest than comparing to all possible tiles.

Keywords: Heuristic Search, A* Algorithm, Puzzle Problem

1. INTRODUCTION

1.1. Puzzle Problem

Puzzle problem is one of the examples which often used to introduce Artificial Intelligence. The puzzle problem discussed in this paper is one similar to Tetravex game with some modification. Tetravex [1] is a 3 x 3 grid board game which has nine square tiles. Each tile has a number on each edge. The objective of this game is to place the tiles into the grid in the proper position as fast as possible. Two tiles can only be placed next to each other if the numbers on adjacent faces match. In most cases, there is only one solution to the problem. Still, few cases can have more than one solution.
Above all, there is always at least a solution to the puzzle problem.

The modification done in this paper are, the numbers can only be 1 to 9 as 0 is used to represent empty tile. Scores are calculated as number of tries until meeting the goal. In this paper, we concentrate more on getting the solution in the shortest time indicated by lower number of tries.

1.2. Puzzle Solving Techniques

Examining the puzzle problem, we could generalize the problem as searching problem. The goal is to get all tiles are placed in the puzzle where the numbers of adjacent face tiles are matched.

The easiest way is by using uninformed or blind search such as Depth First Search (DFS), or Breadth First Search (BFS). However, this can be very daunting for some cases as DFS and BFS do not utilize any information.

The other techniques are by using Informed Search. The best search path algorithm known is A* Algorithm. This algorithm will guarantee to get the lowest cost to the goal.

2. SOLVING PUZZLE PROBLEM

2.1. Puzzle Class

In order to solve the puzzle problem, we will design the Puzzle class first. Puzzle class has nine tiles. Each tile will hold information about the numbers located on its edge: left, top, right and bottom.

2.2. Uninformed Search

BFS[1] is one example of uninformed search. It is simple. BFS tries every possible combination of the tiles to be positioned in the puzzle. However, the program is run out of memory as the numbers of node to be explored is huge. It is 9! or 362,880 puzzle configuration. Even, during the running of the program, BFS cannot be started as it could not get one fully configurations.

The other kind of uninformed search is DFS[2]. DFS will reduce the number of node to be explored as it tries to get one fully-filled puzzle before exploring others.

DFS is better than BFS is in this case as it can save the memory by exploring the depth first. Therefore, the numbers of nodes being expanded are less than one on BFS.

Still, DFS will not solve this problem efficiently. As it will examine every possibility in unintelligent way, all possible configuration of puzzle will be tried. In case of the same puzzle problem with different positioning tiles, DFS will solve the problem differently.

---

Figure 2. Puzzle 1, example of Puzzle Problem

---

Figure 3. Puzzle 2, the same puzzle of Figure 2 but having different tiles positioning

---
problem but having different tiles position. Using DFS, Puzzle 1 was solved in 11 tries, while Puzzle 2 was solved in 21 tries.

This is an example of why uninformed search is not intelligent. It is fast in starting the search, no processing required before searching. However, it could not identify the same puzzle problem should the placing configuration are different.

```
+-------+-------+-------+
|   2   |   1   |   9   |
| 7   3 | 3   5 | 5   8 |
|   6   |   2   |   7   |
+-------+-------+-------+
|   6   |   2   |   7   |
| 8   3 | 1   5 | 5   3 |
|   7   |   4   |   2   |
+-------+-------+-------+
|   7   |   4   |   2   |
| 3   4 | 2   3 | 3   1 |
|   2   |   9   |   5   |
+-------+-------+-------+
```

*Figure 4. Solution for Figure 1 and 2*

2.3. Informed Search

Information is required to reduce the number of nodes being expanded. First, we are going to revisit BFS. However, this time we try to reduce the number of nodes being expanded by eliminating unmatched tiles to the adjacent cells. Consequently, we will have fewer nodes than original BFS.

This idea was researched by Esti in her thesis[5] to reduce the number of nodes being expanded by BFS.

The original idea was to have four arrays, each consist of number in four way direction: left, up, right and bottom. During the implementation of this algorithm, it can be simplified in programming term by just matching the numbers on the particular direction without creating separated arrays. Unmatched adjacent tiles will not be considered in searching process.

As the result, we could run this modified BFS and got the solution. The number of trial will be very high as all possible configurations will be explored. However, this modified BFS can be run as the number of nodes being expanded are not as many as original BFS.

Further, to make it more efficient, we could change this algorithm into A* algorithm[4] by adding heuristic function to measure the condition of current configuration of puzzle compare to the solution. The higher the value is produced by this function, the higher its chance for being explored first.

Basically, this algorithm will place a tile in a certain position by examining whether it has a match number with adjacent face cells. While it is true that we may have more than one possible candidate for a certain position, we could cut the number of tries by limiting ourselves to only exploring the match numbers. In this case, we will have a 'thinner' tree.

This time, we will start trying from the top-left position as the root. It will produce a tree with two children (Figure 5). Starting from tile 0, we build our solution to get tile 3 and 1, then tile 6, 4 and two, then tile 7 and 5, finally we get the tile 8. This algorithm should produce more efficient number of tries than previous algorithms.

Unfortunately, there is a constraint for Child 7, 8, and 9. They should match with the other two parent node not just one.

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

*Figure 5. Tree-like model from a Puzzle with the Top-Left tile as the root*

Having constraint during generating the solution is a disadvantage as more processing required to do. During the development of this algorithm, it was found out that the most problematic tile is the center tile. It is very disturbing as it must match in all four different directions.
However, the constraint could become an advantage. We know that the tile in center position should match all four directions. Should we start our searching from the center tiles, we will further reduce the number of nodes being expanded as not many tiles could be on the center position.

The algorithm is modified to start looking the possible candidate for center position. The tile should match in all four directions. Then the adjacent tiles will be examined. The testing function will calculate the correct match tiles. The higher number of correct match will be explored first until a goal is met.

More constrains can be found here as tile 6, 7, 8 and 9 must match with the other two tiles. Although tile 6 only attached to tile 7, it should also match with tile 3. However, as the nature of the game is that the puzzle will have at least one or more solution, automatically tile 6 will match tile 7 and tile 3.

The result of this algorithm for Puzzle 1 and Puzzle 2 are 9 tries or exactly one tile place in one position.

3. CONCLUSION

It can be seen that Uninformed or blind search is easy to be implemented but it will not guarantee a solution efficiently. In comparison to Informed search such as A* Algorithm, Uninformed Search will be not efficient. It is true some puzzle problem like can be solved quickly by DFS. However, it is not always the case.

On the other hand, A* Algorithm can solve all puzzle problems, even it solve them in the shortest path to the goal. The efficiency of this algorithm depends on what heuristic function is used.

The heuristic function used in this paper show us that it is important in reducing the amount of processing time. It also helps in calculating the current condition compare to the goal, so that we could reach the goal in the shortest path.

In conclusion, this paper would like to show that Informed Search can guarantee a solution in the shortest path, if any. In addition, this paper could be used to introduce Artificial Intelligence in solving problem like but not limiting to puzzle.

4. REFERENCES