Finding Solutions for N-Puzzle Problem using Breadth First Search and Branch and Bound

Deo Fetalvero

April 26, 2019

1 Introduction

The N-Puzzle Problem is a sliding puzzle that consists of N moving square tiles in arranged to make make up a larger puzzle that has equal rows and columns of those tiles. In the puzzle, a tile that along with all other tiles complete all the spaces in the rows and columns is deliberately removed so that all other tiles can be rearranged or disarranged by moving adjacent tiles to the space of the supposed tile. Each tile is labeled by its original position in the grid (top to bottom, left to right). To solve the puzzle, the tiles should be moved one by one such that the original position of each tile correspond with it current position in the grid (top to bottom, left to right).

The puzzle was invented by a postmaster in New York as early as 1874. Since then, the puzzle has become popular by the spread of copies thus, making its way to the manufacturers and sellers. Since its invention, the puzzle started with labels using numbers from 1 to N where 1 is the position at the top left corner and N is at the bottom right (top to bottom, left to right). With the position labels, it was relatively easy to find where a out of place tile should go. Further improvement of the puzzle was done by utilizing parts of images such that when arranged correctly would result to a complete cohesive image. This way increases the difficulty of the puzzle depending on how recognizable each image part of every tile that could connect to other image parts to form the bigger image.

2 Methodology

The current discovered and proposed solutions to this puzzle problem are graph searching algorithm. Each node in the graph is an arrangement of the tiles labeled by its original position. The node would then have children nodes of another arrangement with the connecting edge as the move made to achieve another arrangement. Each move is done by sliding adjacent tiles to the empty tile space or by swapping the empty tile's position to one of the adjacent tiles in the arrangement. Of course, the tiles can't go outside the borders of the puzzle reducing the adjacent tiles that could move (or slide) when the empty tile space is beside the border of the puzzle.

Every move is done by sliding a square tile to the empty square tile space. Since the puzzle is composed of square tiles, the maximum moves that can be done by sliding to the empty tile space on a specific tile arrangement is 4. These moves that are possible with the puzzle can be labeled as the left move, right move, top move and down move. Each movement is based on which adjacent tile will move to the empty tile space in the arrangement. For example, the left move is done by moving the left adjacent-to-the-blank-space tile to the blank space.

To find the solution from a shuffled arrangement, where not all tiles are in their original or supposed position, to the original arrangement, where all tiles are in their supposed position, arrangement nodes are traversed from the shuffled arrangement node to search for the original arrangement node with each move possible from the previous arrangement as the connecting edge of the previous arrangement (parent) node to the next arrangement (child) node. The traversing or search only stops when the original arrangement node is found connected using a specific order of edges or moves done from the said shuffled arrangement.

When using a graph searching algorithm, an increased search space composed of all nodes also increases the computational difficulty of finding nodes in the space. This is also happens to the graph searching algorithm when the number of tiles increases by adding rows and/or columns of square tiles to the puzzle. Since adding more tiles add more possible disarrangements that can be done by moving the tiles anywhere in the grid. It can be put simply that since there are more tiles, there would be more permutations from an increased tile count and more permutations would result to a larger search space.

2.1 Graph Searching Algorithms: Breadth First Search

One of the very basic and easily implementable solution of the N-Puzzle Problem is the Breadth First Search (BFS). A BFS is done by checking every

node by each level of node depth. Since the search starts from a specific node, node depth describes the number of edges that are required to reach a another node from that specific node. This only means that BFS checks the nodes with the least amount of edges traversed from the specific starting node (or the "nearest" nodes) first before checking other nodes.

This can computationally done by queueing nodes by their level of node depth. A basic implementation of BFS is:

1. Label the tree node or starting node as the current node. 2. Create a queue that would contain nodes to be checked. 3. Check if the current node is the node being searched for. a. If it is, skip to step 7. b. If it isn't, continue to step 4. 4. Get all the possible children nodes or nodes that can be traversed by an edge from the current node. 5. Add all the nodes from the previous step to the end of the queue. 6. Remove the node from the start of the queue, set this node as the current node and go back to step 3. 7. Collect the edges that in-order traverses the starting node to the node being searched for.

Step 7 in the implementation can be easily done by alternatively storing the edges that have been traversed to reach the node (in step 4) inside that same node and then accessing that information in step 7 also.

It is also important to note that some graphs have node cycles such that blindly traversing nodes can make the traversal to loop back to a previously checked node (which can also cause infinite loops). One example of such graph is from the N-Puzzle Problem since each arrangement can be redone using a few number of moves and one of these few number of moves is by simply doing a set moves on an arrangement and then reversing those moves such that the left move reverses right move v.v. and up move reverses down move v.v.

Preventing looping checks described previously is also important to not waste compute power on traversing from a previously searched node for its children and "grand"-children which are also assumed to be already checked. The best way to mitigate this is by allowing the nodes to be serialized uniquely by how each node can be distinguished from each other (note: this can't be traversed edge information) and each checked serial should be included in a hash set (fast set similar to a hash map which stores a unique item only once). For the N-Puzzle Problem this should be the original positions of each tile in order of how they are currently arranged spaced out by a separator. This serialization method also can be used to identify the node being searched for in step 3.

For solving the N-Puzzle Problem, the implementation of BFS should allow: nodes to be serialized, node's children be identified and instantiated, edge traversal information be recorded in nodes and a hash set to be utilized for containing information of nodes already checked. These requirements are already described in the previous paragraphs.

2.2 Graph Searching Algorithms: Branch and Bound