Deep Parekh

Dominick Atanasio

CS 4200

23 February 2020

**Approach:**

I have created a PuzzleState class that is in charge of storing the state of the puzzle as a byte array with a fixed length of 9. There are 3 separate constructors for this class depending on the use cases. All the state related methods can be found in this class. Some of the methods include, getNextStates which gets the immediate successor states of the current PuzzleState, isGoal which checks whether the PuzzleState is a goal state, and a getPath function that returns the path from the initial state to the current state.

Additionally, the PuzzleState class also has a solve method that includes a call to the solveTree and solveGraph functions that take a HeuristicInterface as a parameter. The HeuristicInterface interface is an abstract class with a single getHeuristic method which is called in the solve Graph and solveTree methods to find the total cost of a PuzzleState. This total cost is also used by the PuzzleComparator that compares two PuzzleStates.

I also have two classes Manhattan and Hamming that implement the HeuristicInterface and the getHeuristic function that is used to get the total cost of a state. The PuzzleSolver class is the entry point of the program. There are multiple ways to solve the puzzle, if needed we can add a text file containing initial states of the puzzle on a separate line as a command line argument. We can also enter an integer as a command line argument which will test that number of random puzzle states. If there are no command line arguments the program will prompt the user if they want to enter their own initial state or if they want to solve a randomly generated state. If the user enters anything other than a Y, or N then the program will prompt the user to enter a valid answer.

**Hamming Distance vs Manhattan Distance:**

Hamming Distance: Hamming Distance was the heuristic that counted the number of tiles that were out of place. Using this function as a heuristic results in more nodes searched than the Manhattan distance and it also results in more time taken to reach the goal state. However, using the Hamming distance as a heuristic with the Tree Search strategy leads to even more nodes searched and even more time taken to find the path to the goal state for search depth greater than 10.

Manhattan Distance: Manhattan Distance is the heuristic that is based on the distance of a misplaced tile from its expected position. The Manhattan distance as a heuristic is better than the Hamming distance heuristic because the cost of the search is significantly cheaper while also keeping the time taken to get to the solution being about 10 times lesser. As observed with the Hamming Distance heuristic, the Tree Search strategy leads to more nodes searched and overall more time taken than the Graph Search Strategy when the search depth goes above 10.

**Analysis:**







