Programming Assignment 1

Student: Oliver Cooksey
Course: CSCI 48700: Artificial Intelligence
Instructor: Dr. Snehasis Mukhopadhyay
Due date: September 23, 2025

The purpose of this assignment is to implement and compare two common search algorithms: breadth-first search and A* search. Both of these algorithms are used to find optimal search paths given a graph. In this case, the 8-puzzle problem will be used to test each algorithm.

The efficiencies of each algorithm will be compared. Because both of these search algorithms attempt to find the optimal solution, the cost of their solutions should be the same given the same initial state. However, the number of iterations it took to find the optimal solution is expected to be different. The implementation created for this assignment tracks the number of "expansions" a search requires. An expansion is defined as the number of times a node is evaluated for possible moves. The more nodes expanded, the more costly the algorithm.

Both algorithms will also implement a closed list. The closed list improves search time by refusing to expand nodes that have been searched before. Every time an expansion is about to begin, the node to be expanded checks against the closed list. If its current state is found, the algorithm will neglect to expand it because a more optimal path to that node must already exist.

3 configurations were randomly generated and evaluated by both algorithms. The algorithms are both set to halt search if a maximum number of expansions is reached. 1 of the 3 configurations is known to be unsolvable, so both algorithms will give up while solving it.

As for the two solvable configurations, breadth-first search was able to find solutions in 28,567 expansions and 71,912 expansions. However, A* search was able to find solutions in just 1,301 expansions and 3,394 expansions. Breadth-first averaged 50,240 expansions while A* search averaged just 2,348 expansions for solvable configurations. That is a 95.327% decrease in the average number of expansions from breadth-first search to A* search. It is important to note that this is not the case for the 1 unsolvable configuration, as both algorithms will need to expand an entire tree that is the same size to determine that a configuration is unsolvable.

These results are as expected. Though both algorithms find an optimal solution, breadth-first search effectively scans all paths blindly without taking into account if the moves are making any kind of progress. A* search avoids paths that get further from a solution until it has searched paths that the heuristic deems efficient. Even with a heuristic as simple as the one used, the results are substantial.

However, specific data on how much faster cannot be concluded from this experiment. 3 test cases is hardly enough to create a solid projection for how both algorithms will perform on all possible initial start states.

It is clear that the simple addition of a heuristic can drastically increase the performance of a search algorithm over more blind or procedural search options. A* search can decrease execution time substantially over breadth-first because its average decision is much more informed due to the use of the heuristic.