John Pignato

Artificial Intelligence

Spring 2022

Rush Hour Problem

The purpose of this project was to write a program to solve any rush hour board in an optimal way using the A\* algorithm. For A\* to work efficiently a suitable heuristic must be designed and applied to the algorithm. After several iterations of the game, a simple heuristic became evident. The numbers 1, 2, 3, 3.5 are used in the heuristic to distinguish better moves from worse ones in the order of lowest(best) to highest(worst). The number 1 is assigned to the goal car if it moves towards the direction of the goal state(positive). If the goal car moves the inverse direction of the goal state(negative) then the number 3.5 is assigned to the move. The number 3 is assigned to horizontal cars if they move in the negative direction in hopes it will free up space for vertical cars to move in front of the goal car. The number 2 is assigned to vertical cars if they are trucks(size 3) and can move down or are cars(size 2) and can move up. This is done in hopes that the cars in front of the goal car will be able to free up space. The number 3.5 is assigned to all other moves as they are deemed less important.

This heuristic is admissible because it never “over-estimates” the value of a move. For the number 1 heuristic if the goal car is moving towards the goal state so it is at least one move away. For the number 2 heuristic, the car is at least two moves away from the goal state as we are moving a non-goal car. For the number three heuristics since we are moving a horizontal car that means we must be freeing up space in order to move a vertical car than the goal car in a best-case scenario. Lastly, the 3.5 heuristics is assigned to all other moves because it is assumed that these are only sought when we are far from the goal state.

Hash maps, queues, arrays, heap, and graphs were used to solve this problem. Arrays were used in order to create matrixes that represented board states. Queues were used inside the heap. The heap was used as a priority queue inside the A\* function. Hashmaps were used for a fast look-up of car information. Lastly, the graph was used to represent the entire state space.

The heuristic functions to reduce the number of states in my state-space that are visited. In a breadth-first search, more nodes are visited. The heuristic directs the algorithm to visit the nodes most likely to be the shortest path first. This will lead to a faster result and a breadth-first search only if worst-case scenarios. A more detailed heuristic could definitely be applied to the algorithm. Looking at the position of the goal car and scaling the heuristic according to which side the current car is on may yield a better solution. There is a high memory overhead for my implementation. A lot of data is stored that is only accessed once or even never accessed at all in the cars hashmap and the graph itself. Instead of using an explicit graph in the future, the use of an implicit one may have large speed and memory improvements.

I solved my algorithm on paper and implemented it a week before the due date. It took close to four days to debug. This was due to the way I defined my nodes using my grid “fingerprint” idea. It was a clever idea in design but became difficult to understand when stepping through the output. In the future, I may use a unique id for the node that is significantly shorter than the encoded matrix string.

I would like to acknowledge and thank Dr. Huelsman for his help on and design of a challenging yet fun assignment.

## 