IT3105: AI Programming - Module 1
Rush Hour Puzzle

Lasse Henriksen
Ingunn Vallestad

September 2017

1 The heuristic function

The heuristic function is a function of the current state of the game (node n) and is the distance from car-0 (its rightmost occupying square) to the goal position, plus the total number of vehicles currently blocking the straight path between car-0 and the goal. Mathematically this can be written as

$$h(n) = (Car(0).x - goal.x) + \sum_{i=1}^{k} B(0, Car(i)) \tag{1}$$

where

$$B(0,i) = \begin{cases} 1 & \text{if Car}(i) \text{ blocks Car}(0) \\ 0 & \text{if Car}(i) \text{ doesn't block Car}(0) \end{cases} \tag{2}$$

and k is the total number of vehicles in the puzzle.

This heuristic satisfies the condition that the heuristic must be admissible, as to never overestimate the cost of reaching the goal. All vehicles in the puzzles are either of length 2 or 3 such that it takes 1 or 2 moves to get them out of car-0's path.

If all the vehicles blocking car-0 are cars (of length 2) and if there is nothing to stop these vehicles from being moved out of the way (and moving any of them won't stop the others from being moved away) this is in fact the true heuristic. However, some of these vehicles may be trucks (of length 3). Also, in most cases there will be other vehicles blocking the blocking vehicles and hence in reality it is normally necessary to make more moves than this before reaching the goal. This shows that the true heuristic will always be larger than or equal to the heuristic in equation (1).

Additionally one could add extra cost of each vehicle blocking the vehicles that is blocking car-0, thus computing the cost of moving the second-order blocking cars, or even third-order blocking cars. But this quickly increases the complexity of the system and eventually becomes just as complex as the initial problem. We chose to not go down this path.

2 Expanding a node and generating successors

Expanding a node consists of finding feasible movement of any vehicle on the board, which must satisfy three conditions

a Any movement of the vehicle must be within the game board,

b it can not occupy a square already occupied by another vehicle, and

c a horizontal vehicle may only move left or right and a vertical vehicle may only move up and down.

Hence, finding successors consists of iterating through every vehicle on the board, finding feasible moves, and for every one feasible move create a new game node with the new move made on the game board and finally appending this to the successors list. Thus all successors differ from their parent by exactly one move.

3 Comparison of BFS, DFS and A*

	Puzzle Variants			
Search Method	Easy-3	Medium-1	Hard-3	Expert-2
Breadth-1st	(100, 16)	(805, 24)	(2038, 33)	(10934, 73)
Depth-1st	(107, 42)	(1128, 278)	(3035, 795)	(10153, 2338)
Best-1st (A*)	(89, 16)	(656, 24)	(1063, 33)	(7089, 73)

The table above shows the main result for each puzzle with all three methods of searching with (searchNodesGenerated, movesFromStartToGoal). Both best-first and breadth-first searches find the optimal solution, but best-first generates roughly 20-50% less nodes varying between the different puzzles. Even though A* and BFS both finds the optimal solution, A* generates a smaller search tree than BFS because it uses its heuristic to evaluate each node and can therefore choose to proceed with the node that looks the most promising.

DFS does not find the optimal solution, and if it ever does it's by pure luck. This is because DFS just dives in the first node it finds and keeps searching until it finds a solution, and thus may (and most probably will) miss the optimal solution. It's worth mentioning that the number of moves found from DFS is also dependent on the order of which successors is appended to the queue.