This assignment has been submitted late, due to Ashwin Rajendran being sick and this disrupted the workflow of the group. Dr.Ron Petrick kindly granted an extension via email for 1-2 days after the original deadline (which was end of day 13/11/17). Please exempt any lateness penalties.

Part 1: Programming with A* Search

Implementation

The A* search was implemented in Java. Please also see the source code submitted as well.

A **node class** was created which stores the position of all the blocks in a 3*3 array, the robots current position, the nodes parent node, the action the robot did to come from parent node to current node, the Heuristic value of the node, the cumulative cost of the node, and the F value (Heuristic value + Cumulative Cost). We can visualise each node as shown in the diagram below (each node also contains the parent node, which is not in the diagram):

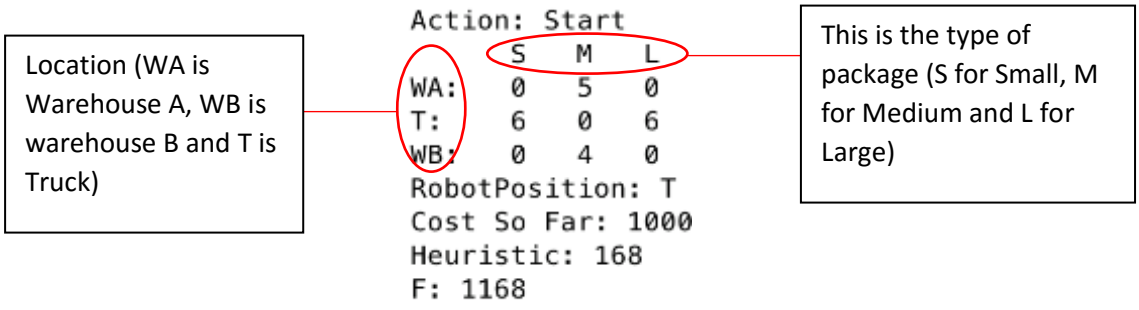

Figure 1: Representation of a node

There is a **createChildren** function, which calculates the possible the child nodes (transitions) from a given node:

From position T, the robot can:

1. Move to B
2. Move to A
3. Move a small package to A
4. Move a large package to B

From position B, the robot can:

1. Move to T
2. Move to A
3. Move medium package to T

From position A, the robot can:

1. Move to B
2. Move to T
3. Move medium package to T

We did not encode moves that are obviously detrimental to achieving the goal (such as moving small package from T to B, large package from T to A, small package from A to B, etc.). As the A* search will never choose such paths, and it will only take up valuable memory space and slow down performance of the A* search.

There are two list of Nodes called **openlist** and **closedlist.** The openlist contains a fringe of all unexplored nodes and the closedlist contains a list of all the nodes explored so far.

Whenever the createChildren function is called on a node, that node is added to the closedlist and removed from the openlist. All the child nodes, which are created, are checked against the closedlist and if they don't exist in the closed list then they are added to the openlist (the fringe).

There is are two global variables - Boolean found which is used to indicate if we have reached the goal state and Node final_node which stores the final node once the goal state is found.

The createChildren node also has a **goal test** (All medium in truck, small in Warehouse and Large in Warehouse B), this goal test is done every time the createChildren function is called. If the goal test is passed, then the global variable found is set to true and the final node is set to the equal the node that has reached the goal state.

Inside the main function, there is a while loop which loops while found = false. The openlist is sorted to put the node with the lowest F value (Heuristic value + cumulative cost) at the top of the list. (The tie breaking for nodes with equal values is dependent on the java built-in sorting algorithm). Then createChildren is called on the node at the top of the openlist. This process repeatedly happens until the goal state is reached.

When the goal state is reached, the while loop stops. Then we call the **constructPath** function on the final_node. This function adds the parent node of the final node to a **path** list, then the parent node of the parent node of final node and so on… (recursively finds the parent node attached to each node). This repeats until the start node is reached where the parent node is null.

On the path list that was created, another function called **printPath** is called on it. This reverses the path list so that it starts at the start node, and prints out the states one by one. This gives us the output on the terminal, which shows the path the A* has chosen as the optimal path.

Calculating Heuristics & Costs

In our design, The cost of moving a package is set to 1 and the cost of just moving between locations is set to 2. This was chosen because moving between two places doesn't directly get us closer to the goal but moving a package does, hence just moving between two places has a higher cost. Each node keeps a cumulative cost of the path so far, when a node is created it adds the cost of the action to move from parent node to current node to the cost stored in the parent node. This way a cumulative cost of path to reach state is stored.

We tried two heuristics:

Heuristic 1: Misplaced Packages (an Admissible Heuristic)

> For each node, the heuristic is calculated to be the number of packages misplaced. Since the cost of moving a package from the wrong place to the right place is 1, we find that the heuristic of number of packages misplaced is always equal to or less than the actual cost. This way we know that the heuristic is admissible.

Heuristic 2: Misplaced Packages * 8 (an Inadmissible Heuristic)

> This heuristic is the same as the above, except it is multiplied by 8. This definitely overestimated the actual cost, so it is an inadmissible heuristic. We found that this heuristic made the program very fast, and able to work with larger numbers (packages in each locations). This is because the huge heuristic value overshadows the cost value, so the A* search works as a greedy/best first search. This might not return the path with the lowest cost.

Example Solution from the A* Search

Using Heuristic 1, where the initial start state has 1 medium package in Warehouse A, 2 small and 1 large package in Truck and 1 medium package in Warehouse B (and the Robot Starting position is Truck):

Start -> Moved medium package from A to T -> Moved large package from T to B -> Moved medium package from B to T -> Moved small package from T to A -> Moved from A to T -> Moved large package from T to B. (FINAL STATE reached, with a cumulative cost of 7 and in 7 steps from start node)