Exercise 3
Implementing a deliberative Agent

Group №76: Simon Honigmann, Arthur Gassner

October 15, 2018

1 Model Description

1.1 Intermediate States

For this exercise, the state representation could be simplified compared to the previous because task information is available to the agents. State was represented by the current city of a given agent, a list of all available tasks, and a list of all tasks currently being carried by the agent. A `State` class was created to conveniently encode this information.

In addition to the information included in the vehicle's state, we added some information to the nodes to reduce the complexity of planning algorithms. This information included: the total distance travelled from the root node, the node's parent, the total weight of tasks being carried (which can be computed directly from the list of carried tasks), a list of actions, *actions required*, that the agent needs to take to transition from the parent node, and the tree level on which the node exists. A `Node` class was created to encapsulate the state and supplemental information.

1.2 Goal State

The goal state for the agent is to have no remaining tasks to pick-up or deliver. This is represented in the tree as reaching a node which has no children nodes. Nodes satisfying this goal state are not unique. The optimality of a node is determined by the magnitude of the distance travelled to reach the node.

1.3 Actions

At any given state, an agent is given N+M possible transitions, where N represents that number tasks available for pick-up and M represents the number of tasks currently being carried which can be dropped-off. Each of these possible actions will cause the agent to transition to the city of the respective pick-up or drop-off task.

2 Implementation

Prior to implementing either search algorithm, a `Tree` class was created, which established the functions and data structures required to create a tree of `Node` states. The `Tree` can be initialized differently, to either generate and store the whole tree immediately or to generate the tree incrementally, depending on the search algorithm being implemented. The `Tree` class also has functions to remove nodes, return all nodes at a given level, check the goal condition for a node, and to generate children for a node.

2.1 BFS

The Breadth First Search (BFS) algorithm determines the absolute optimal pick-up and delivery plan for an agent by searching through all possible paths. This is done by starting at the tree's root node and working down level by level, evaluating each node. On every level, the algorithm will first check the distance to root parameter for a node and compare it to the current best distance (initialized as `double.MAX_VALUE` such that any path reaching the goal state can trump the default value). Any node with a distance greater than the current best will be removed, along with it's children. If the node's distance to root is lower than the current best, the algorithm checks if the node satisfies the goal condition of having no children. If the node has no children, then it is the new best node and the best distance parameter is updated accordingly. If the node has children, a flag is set to continue the search of remaining nodes on the next tree level. On each level the flag is reset. The algorithm continues checking subsequent levels of the tree until the flag is not reset. Once this condition is met and every node on the level has been explored, the best node is returned. A list of actions required to get to the optimal node is generated by travelling up the tree until the root is found, and storing each node's *actions required* in an ArrayList. Finally, a plan is generated using the agent's starting city and the generated list of actions. This plan is returned and is implemented by the agent.

2.2 A*

2.3 Heuristic Function

3 Results

3.1 Experiment 1: BFS and A* Comparison

3.1.1 Setting

3.1.2 Observations

3.2 Experiment 2: Multi-agent Experiments

3.2.1 Setting

3.2.2 Observations