# Exercise 3: Implementing a Deliberative Agent

Group №: 80, Vernet Arthur, Wüst Matthias

October 22, 2019

## 1 Model Description

A deliberative agent possesses the complete relevant knowledge about a simulation world and can therefore preemptively find the optimal strategy (plan) to fulfill a given set of tasks. To do so, a tree data structure is set up, consisting of all possible actions (edges) and states (nodes). They are linked in a parent-child fashion and the initial state defines the root node. Additionally, the nodes contain the path from the root to the current node. With this structure the breadth-first search and A* algorithm are implemented.

### 1.1 Intermediate States

The state class has three variables: city (City) - the agent's current location, currentTasks (TaskSet) - the tasks that are currently loaded and remainingTasks (TaskSet) - the tasks that still need to be picked up. For example, the initial state generally contains the starting location, an empty currentTasks TaskSet and a TaskSet provided by the deliberative.xml file as the remainingTasks TaskSet.

### 1.2 Goal State

Goal states are defined as having empty currentTasks **and** remainingTasks sets. The city variable has no influence, because it will always match a city where the delivery by another agent took place. Then the currentTasks set will be empty at the moment the vehicle reaches this last city.

### 1.3 Actions

In every state there are three possible transitions: delivering a task, picking up a task or moving to another city. The way our tree is structured is the following. Initially, it is checked if a current task can be delivered. If this is the case, it will **always** be delivered. After that, a check is made if a task can be picked up. Here we considered two options: Either to always take a task (same as with delivery) or to consider all cases (not taking the task, taking one or all). The first option would lead to better computation efficiency but does not always guarantee the algorithm to find the best possible path. This is increasingly noticeable when either the tasks are very heavy or the agents capacity is low. So the second option was the way to go, increasing the computation time but guaranteeing the optimal path. Refer to figure 1 (appendix) for a visual description of the tree structure.

## 2 Implementation

Implementation is fairly straight-forward, except maybe for the custom classes we developed in order to store State objects in a proper data structure (i.e. a Tree with Nodes) to retrieve easily the optimal series of actions our agents have to follow to complete the delivery of their tasks.
Note that, for both BFS and A* we used the same *successors* method to generate child nodes, i.e. the possible next states agents can be given an initial state. Finally, once the optimal series of nodes that

lead to a final state is found, the method *planGivenFinalNode* takes care of translating it into a series of Actions, which results in a Plan.

### 2.1 BFS

The Breadth First Search algorithm was implemented following the given pseudo-code, with two noticeable differences since it was asked to return the optimal solution and not the shallowest solution. First off, the algorithm doesn't stop when it finds a solution but instead stores it in a HashMap alongside its cost (i.e. the covered distance by the vehicle). Then, once all solutions have been found out, the algorithm returns the least costly. The algorithm returns every existing paths to every possible final states. It is possible two distinct paths end in the same final state, hence the need to compare their total cost.

### 2.2 A*

Again, the implementation of A* follows the given pseudo-code. Note that we had to implement a custom comparator dedicated for this algorithm for our Node objects so that they can be ordered in a fashion that respects A* conditions (i.e. they have to be ordered according to $f(n) = g(n) + h(n)$.

### 2.3 Heuristic Function

The heuristic function we developed works as follows. First, we list every cities our agent has to deliver tasks. We then compute the distance from where the agent is to the closest city it has (or will have, if the task has yet to be picked up) to deliver a task. We repeat this process, i.e. by computing the distance from this new city to the closest one, until there are no more cities to consider. We sum up the distances and this gives us our estimation of the remaining cost until our agent reaches a final state.
While being sub-optimal, this heuristic ensures us we do not overestimate the cost until a final state since it is impossible the agent finds a faster path to deliver every tasks since we just used the shortest one as a heuristic by assuming the agent did not have to pick up any tasks.

## 3 Results

### 3.1 Experiment 1: BFS and A* Comparison

Both algorithms find the optimal path way. We noticed that for certain setups multiple optimal paths exist (meaning the same optimal final reward/km). Thus, in some cases, the BFS found another way of fulfilling the tasks than the A* algorithm, but both being equal in the total delivery efficiency. However, BFS performs better than the A* in terms of computational efficiency (at least for the tested setups).

#### 3.1.1 Observations

In the same setup (Switzerland, initial city: Basel) with six tasks, BFS takes 244 ms to finish, whereas A* finds a plan in 256 ms. Also the maximum number of tasks that the algorithms can handle before a timeout in the same setup as above are: BFS - 12, A* - 9. In retrospect it is clear that either the heuristic is badly designed or the code logic is incorrect, because a well chosen heuristic in A* should lead to less computation than with BFS.

### 3.2 Experiment 2: Multi-agent Experiments

Setup: Topology - Switzerland, Agent 1 - Basel, Agent 2 - Zürich, Agent 3 - Bern, eight tasks need to be delivered, BFS algorithm.

### 3.2.1 Observations

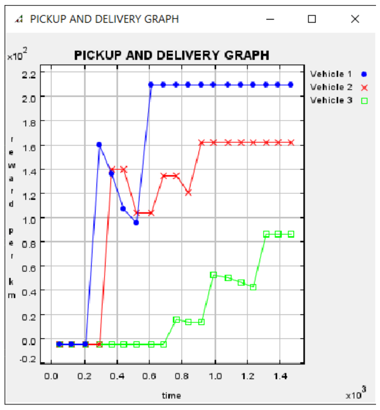

Figure 1: The performance of three agents together.

The agents travel a lot unnecessarily since only upon arrival in a city where a task should be, the remainingTask list is updated and a new plan computed. Therefore, the averaged reward/km of the three agents is largely lower than in a setup with less agents. This however, clearly depends on the task distribution. Overall an increasing number of agents will deliver the tasks in shorter time.

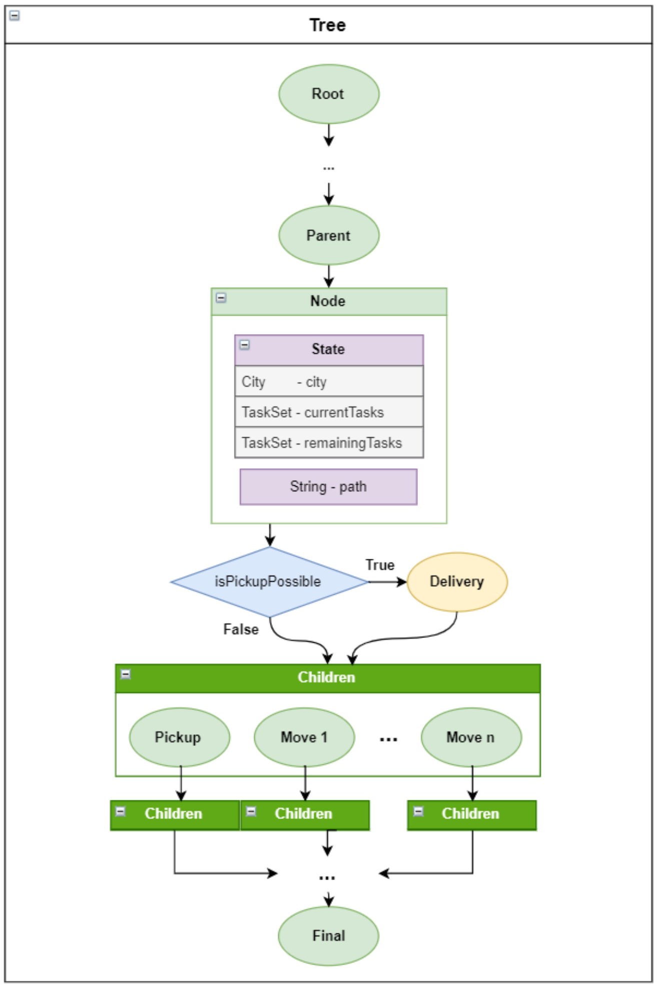

Figure 2: Tree structure - All light green objects are nodes, the yellow delivery node is a special case because when a delivery is possible, it is the only child of the parent node.