Excercise 3
Implementing a Deliberative Agent

Group 21 : Martin Vold & Kristoffer Landsnes

October 23, 2018

1 Model Description

1.1 Intermediate States

In this task we ended up with six attributes to represent a state.

- The tasks the agent currently are carrying and that are ready to be delivered; **TaskSet currentTaskSet**
- Tasks that still are in the world and not picked up by our agent; **TaskSet tasksInTheTopology**
- The capacity of the agent; **int capacity**
- The current city the agent are located in; **City currentCity**
- The total cost of travelling to this state(distance · cost/km): **float costOfGettingToState**
- The sequence of actions that took us from the initial state to this state; **ArrayList<Action> actionsToGetToState**

When comparing states, the currentTaskset, tasksInTheTopology and currentCity are used since they yield sufficient information about the state of the world. If this comparison is insufficient, hence the states are similar, the costOfGettingToState determines the best state.

1.2 Goal State

A goal state is defined as when there is no more tasks for the agent to pick up or to deliver, i.e. both the tasks that the agent are carrying and the tasks left in the world are empty. Then the objective is fulfilled and we have reached a goal state.

1.3 Actions

The different actions our agent can take are: 1. Move to a City and PickUp a Task. 2. Move to a City and Deliver a Task. This makes one state have as many successors as there is Tasks in the TaskSets for the world and agent. To reduce states, only legal actions are possible to execute, e.g. it's only possible to pick up a Task if the weight of that Task is less than the current capacity of the agent. Furthermore, its only possible to deliver and pickup in the corresponding deliver and pickup city of the task.
They way successors are created is as follows:

- First a new DeliberativeState is initialized with attributes as 1.1, with a parent state.

- For a task in either the world or agents TaskSet the moves to get to the corresponding pick up or delivery states is added to the new states list of actions to get to the state. At the end the pick up or delivery actions is added based on if the Task is in the world or agents TaskSet.
- The cost of getting to this new City from the previous one is added to the states cost and the current City is set to the new City. The agents capacity is also updated.
- If the task was delivered, it is removed from the agents TaskSet. If it was picked up, it is added to the agents TaskSet and removed from the worlds TaskSet.

2 Implementation

2.1 BFS

The BFS algorithm is implemented as presented in class with minor changes. To avoid a non-optimal end state we are keeping the end states in a ArrayList. Before adding a new state we compare costs to find out of it's better than the first state in the list (if there is any). It's added to the front if it's better, unless it's added to the back.
Also if we reach a state that we already have visited, we want to check if we managed to arrive there cheaper. Again, we compare costs. If we arrived cheaper, we add this new state and its successors to the ArrayList. Also, we remove the old state. If not, we don't add the new state to the ArrayList.

2.2 A*

The A* algorithm is implemented as presented with pseudo code presented in class. This algorithm will return the first end state it reaches if our heuristic, which is described in 2.3, is admissible, i.e. it never over-estimates the cost of reaching a end state from the current state.

2.3 Heuristic Function

The heuristic function returns 0 if the TaskSet for both the world and the agent are empty, i.e. we are in a end state. If the TaskSet of the world still contains tasks, then it returns the highest sum of the cost of going from the agents current state to the pickup City of a Task and going from the pickup City to the delivery City of the Task. If there are no Tasks left in the world, but the agent has Tasks for delivery it returns the larges distance between the agent's current City and the Tasks delivery City.
This heuristic is admissible, never overestimates the cost of getting to a end state, because choosing the Task with the longest distance (pick up and delivery or just delivery) from the current City of the agent will either return the exact distance, if there is only one task left in both TaskSets, or a distance which only takes into account the distance for one Task. This last distance is smaller than the distance we get if we take all remaining task into account, as these can be seen as a detour from the longest distance path and will add more distance to our path if included.

3 Results

3.1 Experiment 1: BFS and A* Comparison

Both algorithms are tested for different amount of tasks.

Tasks	BFS	A-star	BFS cost	A-star cost
4	0.024s	0.011s	6100.0	6100.0
5	0.167s	0.034s	6100.0	6100.0
6	2.713s	0.122s	6900	6900.0
7	17.286s	3.930s	∞	8050.0

3.1.1 Setting

Topology: $< topologyimport = "config/topology/switzerland.xml" / >$. Seed: $rngSeed = "23456"$. The only change was the amount of tasks, as shown in the result table, and the algorithm.

3.1.2 Observations

It is observed that the BFS is performing well for a low number of tasks, quite similar performance with A*. Also the same optimal plan is returned for both algorithms, as can be seen from the plots, which implies both algorithms reaches optimality. For the BFS algorithm, we have found a bound of 8 tasks. This is mainly because we found out, a bit too late that, that the use of arraylist is not especially fast when modifying objects in the list. Since this is a big implementation change, we haven't been able try with anything else, but realise that hashmaps would speed up the searches. For A-star the bound was found to be 9, which reflects its enhanced performance over BFS.

3.2 Experiment 2: Multi-agent Experiments

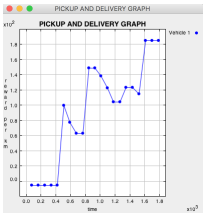
We decided to run a multi-agent experiment with the BFS algorithm.

3.2.1 Setting

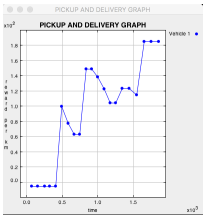
All multi-agent experiments were executed with 3 tasks, BFS algorithm and same setting as 3.1.1. The agents are *deliberative-main* and one/two exact duplicates of this.

3.2.2 Observations

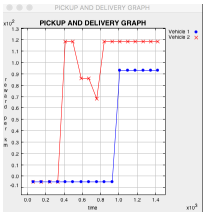
Testing with only three tasks the main problem quickly surfaces: The agents aren't aware of each others presence. Hence, in figure 1 (c) we can observe how vehicle 1 is not getting any tasks but recomputing its plan. In 1 (d) the same incident reoccurred, but with three tasks and three agents, one agent doesn't perform any pickups and becomes redundant.



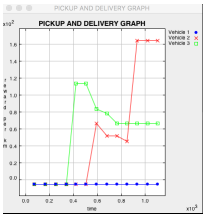
(a) BFS single agent. Cost:6900



(b) A-star single agent. Cost:6900



(c) Multi-agent BFS: 2 agents



(d) Multi-agent BFS: 3 agents

Figure 1: Single Agent BFS & A-Star