Exercise 3
Implementing a deliberative Agent

Group №70 : Pierre-Antoine Desplaces, Julien Perrenoud

October 23, 2018

1 Model Description

1.1 Intermediate States

As seen in class, possible state representations for a deliberative agent in the delivery problem are based on the vehicle and package positions. In our case, we chose to define our state using the position of the vehicle, the set of available tasks it has yet to deliver, the set of tasks it is already carrying, and the remaining capacity of the vehicle.

1.2 Goal State

A state is a goal state if all the packages are located at their delivery point. With our implementation, this is equivalent to having both set of tasks empty, meaning they have all been picked up and delivered. There are several goal states, as it does not depend on the final location of the vehicle.

1.3 Actions

From an arbitrary state, the possible actions and their resulting state are :

- **Pickup a task** : If the vehicle is in the same city as an available task and has enough room, the task goes from the available set to the transported set and the vehicle capacity reduces accordingly.
- **Deliver a task** : If the vehicle is in the same city as the destination of a transported task, the task is removed from the transported set and the capacity increases according to its weight.
- **Move** : The location of the agent changes to a given city. (This means there is one such action per neighboring city.)

2 Implementation

2.1 BFS

In order to implement BFS, we define hashmaps `parents`, `causes`, and `costs` to associate, with each state, its predecessor, the action that led to it, and its cumulative cost. Then, we create a FIFO queue (`ArrayDequeue`) that will store the states that we should visit next (and enqueue `initialState`).

Then, we dequeue a state one at a time and test if it is a goal state, in which case we are done. Otherwise, we enqueue all of its successor states (based on available actions), after testing if we haven't visited them already.

From the goal state found, we finally then iterate backwards, using `parents` and `causes` to reconstruct the plan.

2.2 A*

A* is very similar to BFS, except that we add another hashmap `f` that will keep track of $f(n) = \text{cost}(n) + \text{heuristic}(n)$ for each node. Moreover, our queue is replaced by a `PriorityQueue` that will sort states according to their value in `f`.

Additionally, instead of simply ignoring states that were already visited, we also check if the current state has a lower cost than during the previous visit, in which case we update its cost, parent and cause and re-queue it.

2.3 Heuristic Function

Our heuristic calculates a lower bound on the total cost to deliver remaining packages. In order to avoid counting edges twice, it calculates a (sort of) minimum spanning tree on the set of cities the agent still has to visit. However, as computing MST on a subset of nodes is NP-complete (Steiner Tree), we instead compute the MST on a complete graph of the given cities where the edge weight is the distance between them.

We believe that this heuristic is admissible - the path of the vehicle must go through all the cities at least once and will therefore always be larger or equal to the smallest possible path connecting such cities. If this is the case, then A* should always find an optimal plan and this is indeed what we have observed.

3 Results

3.1 Experiment 1: BFS and A* Comparison

3.1.1 Setting

We compare the optimality and speed of BFS and A* in different settings with varying number of tasks and capacity. All tasks have weight 1.

3.1.2 Observations

	N. of Tasks	12			14			16		
	Capacity	1	3	12	1	4	14	1	4	16
A*	# steps	322,412	185,398	11,321	1,602,456	N/A	162,517	N/A	N/A	531,347
	time [ms]	6,116	5,644	495	38,864	>60,000	7,757	>60,000	>60,000	30,401
	cost	14,900	7,700	5,950	20,050	N/A	7,650	N/A	N/A	7,650
BFS	# steps]	343,919	2,503,323	6,377,114	1,572,545	N/A	N/A	7,077,207	N/A	N/A
	time [ms]	696	10,129	44,893	4,227	>60,000	>60,000	38,930	>60,000	>60,000
	cost	17,050	8,000	6,450	23,100	N/A	N/A	24,300	N/A	N/A

We observe that BFS is faster with capacity 1. The reason is that our A* heuristic does not take capacity into account, and hits a lot of dead-ends while BFS benefits from a lower branching factor. In all other settings, A* is more performant both in computation speed and final cost, which is optimal for all test cases. A* was also able to plan for up to 17 tasks in less than a minute (with capacity 17).

3.2 Experiment 2: Multi-agent Experiments

3.2.1 Setting

We measure the total cost for 1, 2 and 3 agents using BFS or A*. There are 12 tasks of weight 1 and vehicles have capacity 3.

3.2.2 Observations

Nb of agents	1	2	3
BFS (first)	9 300	16 750	20 250
A* (first)	9 100	15 250	20 050

We observe that the total cost more than doubles after adding two additional agents. The reason is that all agents compute a plan to complete all tasks, which is redundant and pointless as they get in the way of each other and end up each delivering only a fraction of the tasks with a lot of overhead cost.