Exercise 3
Implementing a deliberative Agent

Group 47 : Elia Anzuoni (312827), Francesco Ballestrazzi (312494)

October 20, 2020

1 Model Description

1.1 Intermediate States

We modelled a state as a triplet containing the current location of the agent (a `City`), the set of tasks picked up but not yet delivered (a `TaskSet`), and the set of tasks yet to be picked up (a `TaskSet`).
This representation comprises all the information relevant for the planning activity.
The total number of states is $N \cdot 3^T$, where N is the number of cities and T is the number of tasks: this is because, for each possible current location, each task can be in one of three stages: yet to be picked up (1), picked up but not delivered (2), delivered (3).

1.2 Goal State

With this representation, a goal state is simply a state in which both sets of tasks are empty.

1.3 Actions

We chose our action set so that, after each action, exactly one task does exactly one jump: a task is said to "jump" if it goes from stage 1 to stage 2, or from stage 2 to stage 3 (no other jump is possible). To achieve this, we set the following rules:

1. If there are some tasks in stage 2 which can be delivered in the current location, then the only possible action is to deliver the one with the smallest ID among them. This is because, if we can deliver some tasks, it makes no sense not to (we choose the one with the smallest ID to avoid useless different paths to the state where all such tasks have been delivered).

2. Otherwise, if a task can be picked up in the current location without overflowing the capacity, picking it up is one possible action.

3. Additionally, we line up all the cities where we can deliver a task in stage 2. For each of these cities, one possible action is to move directly to this city (on the optimal path) and to deliver the deliverable task with the smallest ID (again, to avoid path multiplication).

4. Finally, we line up all the tasks in stage 1 (that do not overflow the capacity) that have to be picked up in a city where we can deliver no task (to avoid path multiplication). For each of these tasks, one possible action is to move directly to its pickup city (on the optimal path) and pick it up.

2 Implementation

2.1 BFS

We implemented a plain BFS with one modification: we keep a set "visited" containing both the visited and the enqueued states. This way, we can never dequeue, at the beginning of the iteration, a duplicate of an already-visited state; instead, we may find out a better path for a state in "visited" when enqueueing the children of a state: in this case, we just update the old copy's attributes "father" and "cost", and drop the new copy.
This is justified by the following observation: from the "one-jump-per-action" property discussed above, we have that all the paths to a state have the same number of arcs (because there is a well-defined number of jumps to be done). Thus, a node may be discovered more than once as a child, while it is in the queue, but always before any of its children are even enqueued.

2.2 A*

We implemented a plain A*, with no modifications. Like before, when we discover a better path to an already-visited node, we keep the old copy and update its attributes, rather than substituting it with the new one (only this time we do re-enqueue the children, because A* does not visit nodes in order of depth).

2.3 Heuristic Function

For each of the tasks in stage 2, we consider the cost of reaching its delivery city (on the optimal path). For each of the tasks in stage 1, we consider the cost of reaching its pickup city plus the cost of reaching (from there) its delivery city. We take the heuristic of a state to be the maximum of all these quantities. This clearly never overestimates the cost to reach a goal state, since the most expensive of these (sequences of) actions need to be done in any case. Thus, the heuristic is admissible. This, by a known result, guarantees optimality of the path returned by A*.

3 Results

3.1 Experiment 1: BFS and A* Comparison

As discussed in the previous section, both BFS an A* return the optimal plan. The two algorithms, in fact, return two plans that have the same cost. Below we present results obtained for two different topologies: Switzerland and France.

3.1.1 Setting

$Topology = \{Switzerland, France\}; \#Tasks = \{9, 12\}; \#Agents = 1; Taskconfiguration = default$

	Switzerland		France	
	BFS	A*	BFS	A*
Iters	73387	30536	72658	14334
States visited	73387	15039	72658	7439
Cost	8600	8600	23955	23955

	Switzerland		France	
	BFS	A*	BFS	A*
Iters	2030509	913799	2039721	48803
States visited	2030509	384853	2039721	26992
Cost	9100	9100	23955	23955

Table 1: Results for both BFS and A* in the two topologies and with different numbers of tasks (left: 9, right: 12).

Observations The only differences between BFS and A* are the number of iterations and the states visited. The latter, by far, requires less iterations and nodes exploration in all different configurations.

Cost vary a lot among the two topologies. Being distances in Switzerland very small compared to France, we obtained cost that are tripled for the latter.

In less than one minute, we could compute the plan in France for 16 tasks maximum and for 12 in Switzerland.

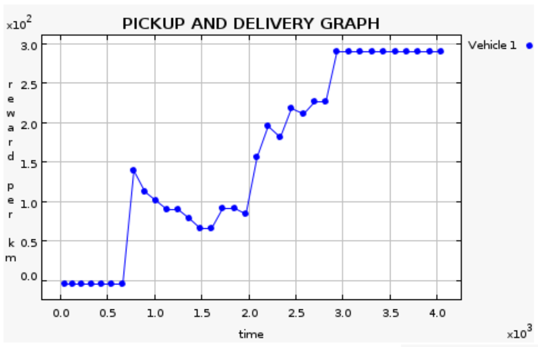
3.2 Experiment 2: Multi-agent Experiments

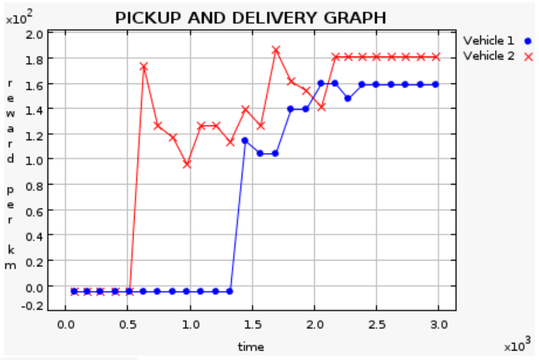
We show here how adding multiple agents to the platform affects the results. The comparison measure is the time it took to the agents to delivery all the tasks. Since the cost per km of all vehicles is the same, the time is directly proportional to the number of traveled kilometers, and thus it's a good indicator of the overall performance of the system.

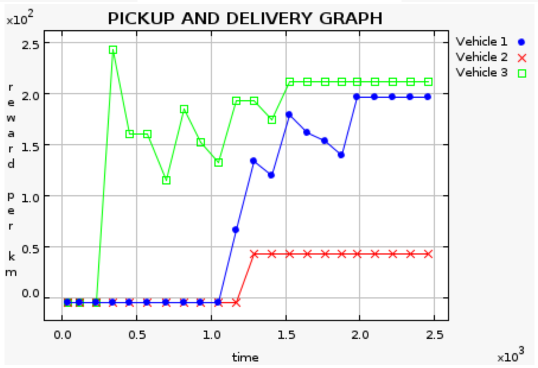
3.2.1 Setting

$Algorithm = A*; Topology = Switzerland; \#Agents = \{1, 2, 3\}; \#Tasks = 10; Taskconfiguration = default$

3.2.2 Observations







We observe that the time taken to deliver all the tasks for the single agent (3000) is less than the sum of times of agents in a multi-agent setting (4600 for 2 agents, 4800 for 3 agents). If we have more than one agent, the total number of traveled kilometers and the cost will be higher than the single-agent case.

Thus, there is no benefit in adding many agents, if those don't collaborate to optimize their resources.