Excercise 3
Implementing a deliberative Agent

Group №66: Amaury Combes, Théo Nikles

October 23, 2018

1 Model Description

1.1 Intermediate States

Every city we go through in the plan we chose is an intermediate state. Every state knows which tasks are taken by our agent (attribute holdingTasks in State class), which tasks are still available (worldTasks), in which city we are and how much weight is carried by our agent (sum of the tasks being carried).

1.2 Goal State

The goal of our simulation is to have all the tasks delivered. So to determine if we have reached our goal state, we simply check whether there are still some tasks available in the world and if there are some tasks hold by the agent. If both of these are false, we know that we've reached a goal state and our agent can stop.

1.3 Actions

We have 3 different types of action, namely **Move**, **Pickup**, **Deliver**. We use them to minimize the overall cost of the path our agent will choose. We will briefly describe them now.
The **Move** action allows us to move from one city to another. The cost of this action corresponds to the distance that separates the two cities.
The **Pickup** action can occur when we are in a city that proposes a task for another one. Since we just pick the task and don't move anywhere, but do not deliver either, the cost of this action is 0.
For **Deliver**, this time we have a task and are in the city where it needs to be delivered. We simply deliver the task without moving. However this task does not grant any reward as we are interested in minimizing the overall cost (although in this problem minimizing the cost and maximizing the reward are equivalent).

This world representation let us model all situations and optimize the cost of our plans.

2 Implementation

2.1 BFS

Our BFS implementation is similar to the description in the slide. The differences are that we stop searching only when there are no new nodes to visit and we always take the cheaper plan to an intermediate node.

2.2 A*

Our A* implementation follows the pseudo-code presented in the slides. As in BFS, we added a new object called ExplorationNode that let us keep track of the paths from the root to the current leaf nodes.

2.3 Heuristic Function

Our heuristic function is defined as follow (in a functional paradigm):

```
def h(S) = {
    val longestWorldTask = max(S.worldTasks.map(task =>
    task.pathLength + S.currentLocation.distanceTo(task.pickupCity)))

    val longestHoldingTask = max(holdingTasks.map(task =>
    currentLocation.distanceTo(task.deliverCity)))

    return max(longestHoldingTask, longestWorldTask)
}
```

We find the max between the longest task that was not picked up plus the shortest path to this task and the longest task that was picked up. This heuristic is guaranteed to underestimate the true cost left as it corresponds to a lower bound of the real path that must be taken. Note that this heuristic is not optimal w.r.t. the true cost as one could do better using for example a minimum spanning tree on the cities that must be visited to reach our goal. However such heuristic is longer to compute making A* slower depending on the implementation (a version with a MST heuristic can be found in our code in the class Transitioner under the name hPrime).

3 Results

3.1 Experiment 1: BFS and A* Comparison

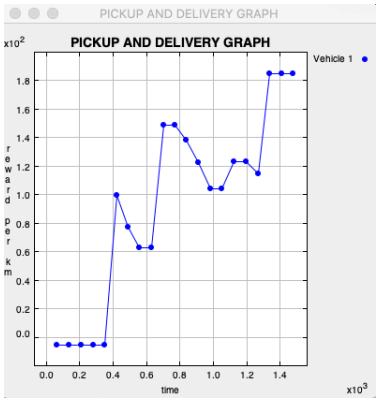
3.1.1 Setting

For this experiment, we ran BFS and A* with 6, 9 and 12 tasks and default settings.

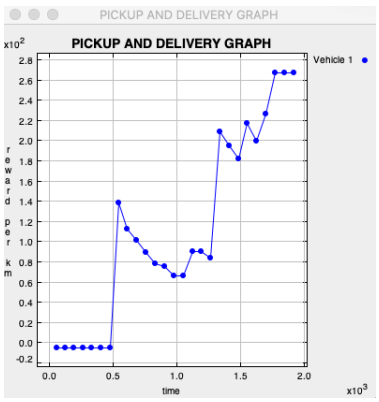
3.1.2 Observations

We observe that for small tasksets (6 and 9 tasks), BFS and A* run in similar time. BFS might be a little bit faster as it doesn't have to merge lists and the memory required is manageable. An optimal plan is found in both cases. However for a larger taskset (12 tasks), BFS is not capable of computing an optimal plan while A* achieves the result in about 2 minutes 38 secondes.

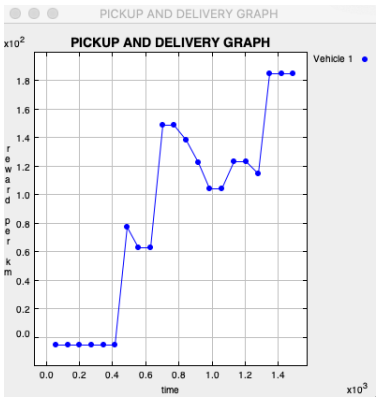
The maximum number of tasks for which we could build a plan in less than one minute under the default settings is 11 for A* (50 sec) and BFS (53 sec).



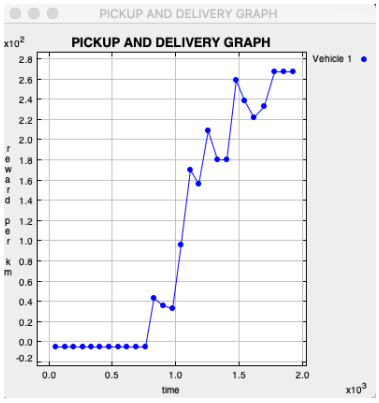
BFS with 6 tasks



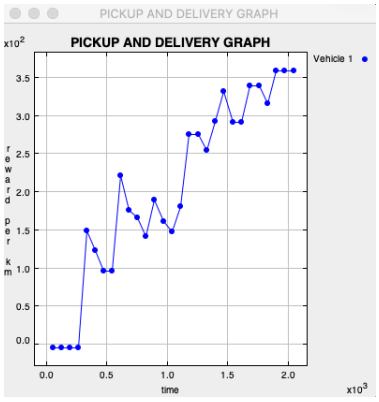
BFS with 9 tasks



A* with 6 tasks



A* with 9 tasks



A* with 12 tasks

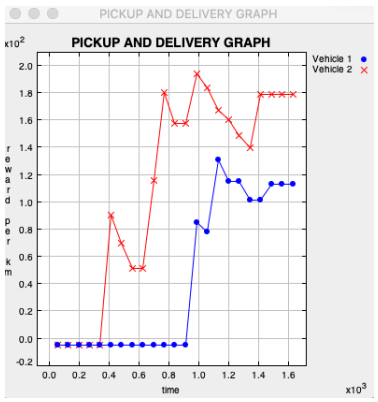
3.2 Experiment 2: Multi-agent Experiments

3.2.1 Setting

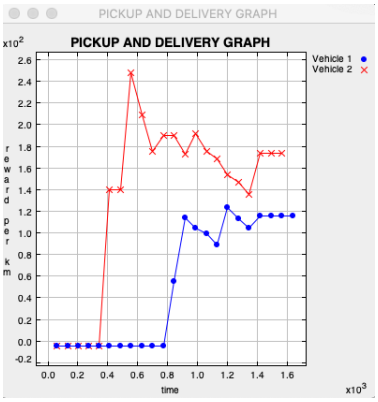
For this experiment, we ran two A* agents, then two BFS agents and then one BFS and one A* agent with 9 tasks and default settings.

3.2.2 Observations

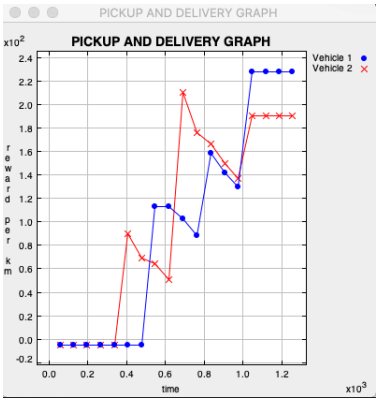
In all three cases, the agents' plan were conflicting and thus had to be recomputed during the simulation. We notice that BFS and A* seems to have less overlap between in their schedule but no conclusion can be taken out of this experiment.



A* and A*



BFS and BFS



A* (red) and BFS (blue)