Excercise 3
Implementing a deliberative Agent

Group №87 : El Abridi Ali, Kalim Tariq

October 22, 2019

1 Model Description

1.1 Intermediate States

In our implementation of the state-based search algorithm, a state is mainly defined by 4 attributes:

- ***The current city***: the location of the vehicle/agent in the topology. **(City)**
- ***Remaining tasks***: the set tasks still available to be picked-up and delivered. **(TaskSet)**
- ***Carried tasks***: the tasks already loaded in the vehicle by the agent at that state. **(TaskSet)**
- ***Accumulated Cost***: the distance in km already travelled from home city to the current city multiplied by cost per km of the vehicle already. **(Integer)**

In addition to that, secondary attributes were also added: The **action** that resulted to that state, i.e, ***PickUp*** or ***Delivery***, and the **heuristic cost** of that state that represent an under estimate of the cost to reach the goal state, picking-up and delivering all remaining and carrying tasks (discussed further in the A* algorithm section).

1.2 Goal State

A goal state is defined by a state in which both remaining task set and carried task set are empty, i.e., all remaining and carrying tasks have been delivered.

1.3 Actions

In this model, we three possible actions:

- ***PickUp(task)***: The agent picks up a task from the remainingTaskSet if it has enough capacity.
- ***Deliver(task)***.: The agent delivers a task to its deliveryCity.
- ***MoveTo(city)*** The agent moves from its current city to the desired city (used to construct a plan).

2 Implementation

In the following section, we describe the overall structure of the different state-based search algorithm, omitting the Naive algorithm as it was already in *Logist Platform*.

2.1 Breadth-First Search

The Breadth-First search (BFS) algorithm implementation is similar to the one presented in the course. The nodes of the BFS algorithm are represented by the different states of the agent as described earlier. In addition to that, the BFS algorithm is enhanced with a *VisitedSet* that store the different states that have been already visited, which prevent the algorithm to explore states that have already been visited. Furthermore, in order to obtain the most optimal goal state i.e., lowest distance travelled to pick-up and delivery all the tasks by the agent, the implementation keeps track of the different nodes *(States)* that have reached a final state *(Goal state)*, and return only the one that has the minimum travelled distance.

2.2 A* Search

The A* search algorithm implementation is similar to the one presented in the course. The nodes are defined as the different states of the agent. A heuristic specific to each search is used to guide the search, and decide which path to extend at each iteration. The A* algorithm bases its selection of the next node to explore on the cost of the path, and an estimate of the cost from the current state to the goal state, minimizing the following $f(n) = g(n) + h(n)$, where n is the next node to explore, h(n) is the heuristic function that optimistically estimate the cost to reach the goal state, and g(n) is the accumulated cost. $cost = distance(homeCity, CurrentCity) * costPerkm$. In the experimentation, we consider two kinds of A* star search, AStar with the heuristic function described in the next section, and AStarConstant, which assign a heuristic value $h(n)$ of 0 to all states.

2.3 Heuristic Function

The heuristic function idea is based on the fact that the cost to reach the goal state (picking up and delivering all the tasks) has to be at least the cost of travelling the max distance that exists between a pick-up and delivery city in case of a remaining task, or the the maximum distance that exist between any carrying task to be delivered and the home city of the agent; therefore, the heuristic is both admissible as it does not overestimate the minimum possible cost, and also consistent as for every state n, and every of its successor n′ by any type of action the following holds: $h(n) \leq c(n, a, n') + h(n')$; therefore, the heuristic is optimal.

Algorithm 1 Heuristic calculation for one state

```
heuristicCarryingT = 0, heuristicAvailableT = 0
for task ∈ carryingTasks do
    heuristicCarryingT = max(heuristicCarryingT, costPerKm*currentCity.distanceTo(task.deliveryCity))
end for
for task ∈ availableTasks do
    heuristicAvailableT = max(heuristicAvailableT, costPerKm*task.pickupCity.distanceTo(task.deliveryCity))
end for
return max(heuristicCarryingT, heuristicAvailableT)
```

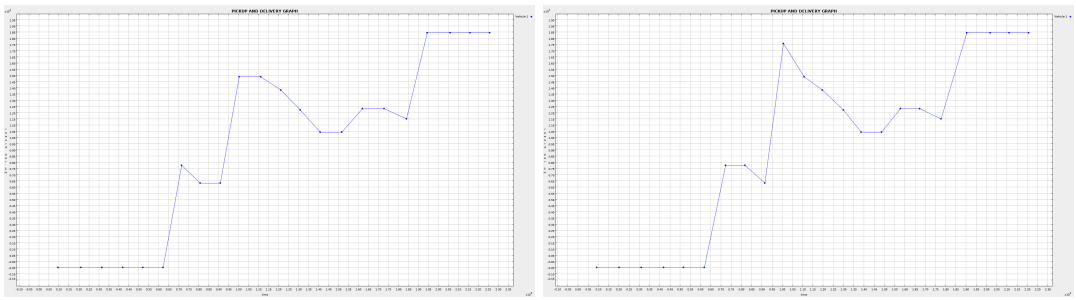
3 Results

3.1 Experiment 1: BFS and A* Comparison

3.1.1 Setting

The topology used for all experiments was of Switzerland to have consistency. For the task configuration, the experiments have been conducted for 4, 6, 8, 10, 11 number of tasks which gave a good idea of the performance of each algorithm in finding the optimal distance, elapsed time under 1 minute. the cells marked with **(x)** mean that the algorithm has failed to produce results under a minute.

3.1.2 Observations



(a) BFS - 6 tasks (b) A* - 6 tasks

Figure 1: Evolution graph for one agent

Algorithm/# of tasks		4 Tasks	6 Tasks	8 Tasks	10 Tasks	11 Tasks
Naive	***optimal distance (km)***	2210	3840	4420	5180	5550
	elapsed time (ms)	1	2	1	2	2
BFS	***optimal distance (km)***	1220	1380	1710	x	x
	elapsed time (ms)	114	223	10049	x	x
AStar	***optimal distance (km)***	1220	1380	1710	1820	x
	elapsed time (ms)	10	66	1076	53754	x
AStar Constant	***optimal distance (km)***	1220	1380	1710	x	x
	elapsed time (ms)	22	81	3383	x	x

Table 1: Comparison of the different state-search algorithms for different number of tasks

The results show that A* and AStar Constant have been consistent in producing the optimal distance equal to the exhaustive BFS for the different number of tasks, which support the idea that the heuristic function optimistically underestimate the cost of the optimal goal state (admissibility) while decreasing the elapsed time by at best a factor of 10 compared to BFS, and also finding more optimal solutions under a minute compared to BFS for different number of task. Moreover, we also that the Naive solution has been produced optimal distances far greater than exhaustive BFS, A*, and A* Constant, which confirms that it is not the best strategy to find the most optimal distance. (Experiments run on an "Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz")

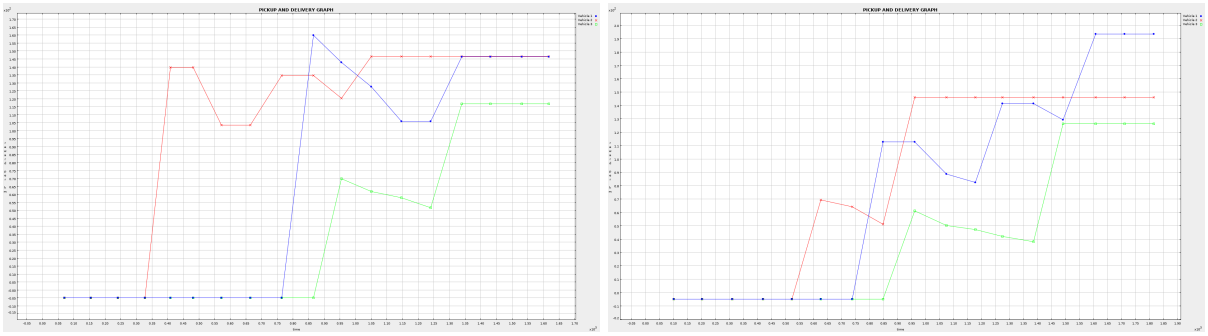
As far as mono-agent simulations, A* shows a net advantage in computation speed compared to BFS and A* Constant. Unfortunately the time-outs prevent us from assessing the speed difference between BFS and A* for more than 10 tasks, and the optimal distance difference between the Naive and A* algorithm. The speed and efficiency of A* could be improved by changing the heuristic and tweaking it to obtain a tighter bound, but that may come at the expense of losing optimality, and producing a sub-optimal result.

3.2 Experiment 2: Multi-agent Experiments

3.2.1 Setting

We are using 3 agents with 10 tasks for both BFS and A*.

3.2.2 Observations



(a) BFS - 10 tasks - 3 vehicles

(b) A* - 10 tasks - 3 vehicles

Figure 2: Evolution graph of 3 agents

Once again, BFS and A* produce different graphs even though they both achieve optimal results. And unsurprisingly vehicles behave differently from each others since they don't necessarily start in the same city. Which counter-intuitively translates in an imbalanced reward/km (efficiency) among the fleet but which results in a higher overall reward/km