Excercise 3
Implementing a deliberative Agent

Group 39 : Hugo Bonnome, Pedro Amorim

October 24, 2016

1 Model Description

1.1 Intermediate States

First of all let us specify the elements that we need to define the state. C is the set of all cities in the topology. D is the set of all the delivery tasks currently in the topology including the ones that are being delivered. A task t is defined by $t \coloneqq (o, d, w)$ where $o, d \in C$, with o being the city where the task originated and d the destination city of the task, w is defined as the weight associated with this task. The state s of an agent is defined by

$$s \coloneqq (c, A, T)$$

where $c \in C$ is the city where the agent currently is, $A \subseteq D$ is the set of avalaible tasks in the topology, $T \subseteq D$ is the set of tasks that the agent is currently delivering.

1.2 Goal State

With the intermediates states defined we can define the goal state by

$$g \coloneqq (c, \emptyset, \emptyset)$$

where $c \in C$ can be any city of the topology and the empty sets representing the fact that all tasks of the topology have been delivered.

1.3 Actions

An agent can do the following actions:

- Move to a city d where a task is available to pickup or deliver.

$$(c, A, T) \rightarrow (d, A, T)$$

- Pick up a task t in the current city c.

$$(c, A, T) \rightarrow (c, A', T')$$

where $A' = A \setminus (t)$ and $T' = T \cup (t)$ with the constraint that the sum of the weights of the tasks in T should not exceed the maximum carrying capacity of the vehicle.

- Deliver a task t in the current city c.

$$(c, A, T) \rightarrow (c, A, T')$$

where $T' = T \setminus (t)$

2 Implementation

2.1 BFS

BFS is implemented in the Solver class in the execute method. It generates successors beginning in the initial node and visits every one of them before generating new succesors. It stops as soon as it find a goal node.

2.2 A*

A* is also implemented in the Solver in the execute method. It generates the successors of the visited node with the lowest estimated cost.

2.3 Heuristic Function

The heuristic considers the minimal distance required to transport every task from its origin to the destination while only counting each route once i.e. no backtracks. This models a vehicle with infinite capacity. Since agents do not have infinite carrying capacity the possibility of backtracking exists and thus the heuristic underestimates the true cost, meaning that it will always find an optimal solution.

3 Results

3.1 Experiment 1: BFS and A* Comparison

3.1.1 Setting

- Topology: Switzerland
- Task distribution: constant
- Weight distribution: constant

3.1.2 Observations

BFS runs out of memory as soons as the number of tasks exceeds 6, the agent does not even manage to compute a plan. This is understandable since in BFS there is a combinatorial explosion of states to explore. With more than 7 tasks A* plan computation takes more than one minute, but its memory consumption is far lower than BFS since it only generates succesors for states that seem promising.

BFS only computes the plan with the least number of actions needed to get to the goal state; there is no guarantee about optimality in relation to the distance travelled.

A* is guaranteed to find the optimal solution as long as the heuristic function underestimates the true cost. We can see this easily by imagining an heuristic that always returns 0: A* will always follow the nodes with the current lowest cost and so the first goal state node it finds will be the optimal one. The whole challenge lies in finding an heuristic that is close to the true cost as this improves the overall efficiency of the algorithm.

3.2 Experiment 2: Multi-agent Experiments

3.2.1 Setting

- Topology: Switzerland
- Task distribution: constant
- Weight distribution: constant

3.2.2 Observations

With only one agent the reward stabilizes at 180 units while with multiple agents it averages at a value of 100 units. This is understandable since there is no coordination between the agents, they interfere with each other's plans. Each time a plan must be recomputed it means that the agent made an unprofitable move that could have been prevented if the agents communicated.