Excercise 3
Implementing a deliberative Agent

Group №: 272257, 262609

October 20, 2020

1 Model Description

1.1 Intermediate States

Here we integrated the intermediate states with the following attributes:

- logist.topology.Topology.City *city*, the city the agent is currently in,
- LinkedList<Task> *ctask* (default value = *empty*), that represent the list of tasks the agent has picked up. Note that the elements of the list also have 3 attributes : *pickupCity* that represents the city the agent can pick up the contract at, *deliveryCity* that represents the city the agent must deliver the contract at, and *weight* that represents the weight of the contract,
- LinkedList<Task> *free_tasks* (default value = *tasks present in the environement*), that represents the list of tasks that have not been picked up yet. The elements of the list have the same attributes as in the list mentionned before,
- State *parent* (default value = *null*) is the predecessor of the current state. This will be used after the application of the *BFS* and *A** algorithms to build back the plan from the goal state we find using those algorithms,
- double *cost* (default value = 0), that is equal to the sum of all steps needed to get to the state,
- Act *act* (default value = $START$), represents the action taken at the given state. This value can vary between *START*, *PICKUP*, *MOVE* and *DELIVER*. The meaning of these actions will be given in a section below.
- int *depth* (default value = 0), that represents the depth of the nodes in the tree,
- Boolean *heuristic*, that represents the method used by the search algorithm. If *heuristic* = *true*, then the *A** algorithm is used. Otherwise the *BFS* algorithm is used,
- a method long *cweight*(), that returns the sum of the weight of the current tasks.
- a method Stack<State> *succ*(int *capacity*), that returns all the possbile children of the current node, where the parameter is the maximal capacity of an agent.

Note that we also have a constant called *capacity* stored in the *Deliberative* class (see below) that represents the maximal weight that can be loaded into an agent. This constant will be passed in the succesor function each time it is called.

1.2 Goal State

A goal state is defines as a state at wich all tasks have been picked up and delivered. Translated , this gives the condition $ctask.size() == 0$ && $free\_tasks.size() == 0$.

1.3 Actions

The agent has 4 different possiblities of action. An action is implemented by the *act* variable and it defines what the chilren of a node can be, i.e. the result of the succesor function. We provide a clearer explanation. Suppose we have a state *state*. We will only give the attributes that change, the rest is assumed to remain constant except for the *parent* attribute that is set to *state* and *depth* that is set to $state.depth + 1$ for the child. Note that all combination of mentioned cases can happen and thus we generate a child node for each individual possibility if not mentionned otherwise.

- <u>If $state.act = START$:</u> Then we can have

$$s'.act = \begin{cases} PICKUP & \text{, if } state.city \in \{task.pickupCity \mid task \in state.free\_tasks\}, \\ MOVE & \text{, in any case.} \end{cases}$$

- <u>If $state.act = PICKUP$:</u> Note that it means that we have that there exists $task \in state.free\_tasks$ such that $task.pickupCity = state.city$ and $capacity - state.cweight() \geq task.weight$. Then we can have

$$\begin{aligned} s'.ctask =& s.ctask.add(task), \\ s'.free\_tasks =& s.free\_tasks.remove(task), \\ s'.act =& \begin{cases} PICKUP & \text{, if } state.city \in \{task.pickupCity \mid task' \in state.free\_tasks.remove(task) \\ & \&\&\ capacity - state.cweight() - task.weight \geq task'.weight\}, \\ MOVE & \text{, in any case,} \\ DELIVER & \text{, if } state.city \in \{task.deliveryCity \mid task \in state.ctask\}. \end{cases} \end{aligned}$$

- <u>If $state.act = DELIVER$:</u> Note that this means that we have a task $task \in state.ctask$ such that $task.deliveryCity = state.city$. Then we can have

$$\begin{aligned} s'.ctask =& state.ctask.remove(task), \\ s'.act =& \begin{cases} PICKUP & \text{, if } state.city \in \{task'.pickupCity \mid task' \in state.free\_tasks \\ & \&\&\ capacity - state.cweight() + task.weight \geq task'.weight\}, \\ MOVE & \text{, in any case,} \\ DELIVER & \text{, if } state.city \in \{task.'pickupCity \mid task' \in state.ctask.remove(task)\}. \end{cases} \end{aligned}$$

- <u>If $state.act = MOVE$</u>: Then for each $city'$ such that $state.city.isNeighbour(city')$, we can have

$$\begin{aligned} s'.city =& city', \\ s'.cost =& state.cost + state.city.distanceTo(city'), \\ s'.act =& \begin{cases} PICKUP & \text{, if } city' \in \{task.pickupCity \mid task \in state.free\_tasks \\ & \&\&\ capacity - state.cweight() \geq task.weight\}, \\ MOVE & \text{, in any case,} \\ DELIVER & \text{,if } city' \in \{task.deliveryCity \mid task \in state.ctask\}. \end{cases} \end{aligned}$$

Note that this generate a lot of nodes at each iteration.

2 Implementation

We implemented those two algorithms in the *Deliberative* class. We suppose that we begin at the city *city*.

2.1 BFS and A*

Algorithm 1: BFS and A^*

initialization: Set Q a queue of $State$ objects with one element $state$ abd default value with $state.city = city$ and $state.heuristique = true$ if A^* and $false$ if BFS

$s \leftarrow nil$

while $Q.size() \geq 1$ *and s is not a goal state* **do**

  $s \leftarrow Q.pop()$

  $Q \leftarrow Q + succ(s)$

  if $s.heuristique$ **then**

    Q.sortBy(f)

  end

end

return s

2.2 A*

2.3 Heuristic Function

3 Results

3.1 Experiment 1: BFS and A* Comparison

3.1.1 Setting

3.1.2 Observations

3.2 Experiment 2: Multi-agent Experiments

3.2.1 Setting

3.2.2 Observations