Assignment 3

Programming: Implementing a Deliberative Agent

Intelligent Agents Course 2009/2010

Radoslaw Szymanek
Brammert Ottens

G10

Bello Ruiz, Javier
Pocard Du Cosquer De Kerviler, Tugdual

A Deliberative Agent for the Pickup and Delivery Problem

In this exercise we have learned how to use a deliberative agent to solve the Pickup and Delivery Problem. A deliberative agent has an explicit model of the world in which it lives so he is fully aware of all available tasks; he can build a plan that specifies the sequence of actions to be taken in order to have the optimal behavior to reach a goal; in our case the goal is to deliver all the tasks in an optimal way.

This plan is a list or sequence of actions that form a path for moving through the network, delivering or picking up tasks in an optimal way to minimize the total cost. But the agent should be also aware of the changes coming in the world, because if a change affects his plan, he must recompute it. A serious limitation of deliberative agents is their lack of speed when faced with uncertainty. Every time the situation is different from the one anticipated by the agent's plan, the plan must be recomputed. The deliberative agent can compute only a plan for a particular scenario only, the states are known with certainty, and the states transitions are deterministic. While this plan is optimal for one vehicle, the presence of another vehicle can greatly influence the overall performance of the company.

In order to compute this plan we have implemented two state-based search algorithms to minimize the agent's total cost for delivering the set of tasks: a breadth-first search with cycle detection and the A* heuristic search.

Algorithm A* (best-first)

```
1.Q ← initial node
2.C ← empty
3.repeat
4.   if Q is empty, return failure
5.   n ← first element of Q, Q ← rest(Q)
6.   if n is a final node, return n
7.   if n ∉ C, or has lower cost than its copy in C then
8.      add n to C
9.      S ← succ(n)
10.     S ← sort(S,f)
11.     Q ← merge(Q,S,f)
      (Q is ordered in increasing order of f(n) = g(n) + h(n))
12.  endif
13.end
```

The breadth-first algorithm explores the tree layer by layer. The advantage of this algorithm is that it always finds the shortest path.
However it requires a large amount of memory to store all the nodes of one level. And furthermore, it is very slow as it tests every node so it is impossible to compute a plan for more than 4 tasks.

At first we defined on paper the following representation of the world:

Agent,

A vehicle can transport multiple tasks at a time if they do not exceed the vehicle capacity; a vehicle has to deliver all available tasks. When more than one deliberative agent is present in the environment, the lack of coordination can determine a very inefficient outcome.

States,

State = {(listOfTasks, listOfPickedUpTasks)}

First State = {(allAvailableTasks, NULL)}

Goal (Final State) = {(NULL, NULL)}

Where 'listOfTasks' is a list of the tasks that are still available in the world to be picked up, and 'listOfPickedUpTasks' is the list of the tasks that the agent/truck already has picked up, always considering the capacity in weight of the truck. Initially in the first state, the listOfPickedUpTasks is empty, and the 'listOfTasks' contains all the tasks in the world that should be delivered. The final state or goal of the plan should contain neither a task to be picked-up nor a task to be delivered. Due to needs of implementation we finally have the following code for the class state:

```
public class State extends Node
{
	private ArrayList<TaskDescriptor> tasks;
	private ArrayList<TaskDescriptor> assignedTasks;

	private double cost;
	private AgentProperties agentProp;
	private City currentCity;
	private City destination;
	private Topology topology;
	private ArrayList<IGenericAction> actions;

}
```

Transitions,
The transitions between the states should be the following:

Transition= {pick-up(idOfTask), deliver(idOfTask)}

In order to build the child states of the tree, the possible transitions that make it possible to change between two states are to pick-up one of the available tasks of the 'listOfTasks' or to deliver one of the 'listOfPickedUpTasks'. This decision is finally determined in the plan established after applying the state-based search algorithms.

Plan,
The plan developed after calculating the search algorithms is composed of these possible actions:

Action= {pick-up(idOfTask), move(idCity), deliver(idOfTask)}

We recompute this plan when the agent reaches a city if it realizes that the environment has changed.

Heuristic function,

h(n) = estimate of the minimal cost from node n to a goal node
= distanceAlreadyTraveled in our implementation, that is the distance already traveled to reach node n multiplied per the costPerKm, or as a matter of fact the cost of the parent node.

g(n) = cost of the best path to node n
= shortestDistanceBetween(previousCity, currentCity)*costPerKm

f(n) = g(n) + h(n)
is an estimate of the cost of the best path from initial to goal node passing through node n.

The performance/optimality of A* depends largely on the quality of the heuristic function. If we always have h(n) = 0, nodes are explored in the order of their cost: any path found will always be the shortest possible, thus optimal. We can prove that A* always finds the optimal solution as long as h(n) under-estimates the true cost.

The following table compares the performances of the breadth-first search and the A* algorithms for one agent and several tasks:

	number of tasks	2	3	4	5	6	7
A*	time (ms)	0	7	75	331	3368	too long
	states explored	8	26	50	538	6141	?
BFS	time (ms)	2	42	290	1977	out of mem	out of mem
	states explored	14	182	3862	91287	?	?

In one minute, we cannot find the optimal plan for more than 6 tasks.

After simulating the PDP with 1 and 2 deliberative agents for each of the algorithms we can see that the total performance of the company is worse than for one vehicle because they have to recompute the plan every time that the other agent interferes in the plan already computed.