Deliberative Agents

Liang Zixuan, Wang Junxiong 10/26/2015

Model

States

Follow is the basic properties of a state:

1. $Location$. It is the current city of the agent. Picking up tasks needs to know the current location of the agent.
2. $Cost$. It is the total travel cost of the agent till current city. Our goal is to minimizing total cost, thus we need to compare cost of different states.
3. $AccumulateWeight$. It is the total weight of tasks the agent is carrying. If it exceeds the capacity of the agent, the agent will refuse picking up the task.
4. $TaskSign$. It represents the status of all the tasks. For each task, "0" stands for not being picked up, "1" stands for having been picked up but not delivered, "2" stands for having been delivered.

After constructing state tree, we have to produce a best plan from this state tree. Thus it is necessary to add the following properties to the states:

1. $ParentState$. It is the closest previous state of the current state.
2. $Task$. It represents which task the agent picks up/delivers to transfer parent state to current state.
3. $IsPickup$. It represents, in the state transfer phase, whether the agent is going to pick up a task in the next city or it is going to deliver a task to the next city.

Transitions

A new state can be produced from an old state via:

1. Going to a new city and picking up new task.
2. Delivering a carried task to a new city.

The agent may pick up multiple tasks along a path among several cities before delivering one of them, which leads to two state transfer possibilities described above.
Which one happens depends on the status of the certain task. If the task is not yet picked up, it is the first case. Current state becomes the parent state of next state. The current location

property of next state is the pick up city of the task. The cost property of next state is updated by adding cost from current location of current state to the pick up city of the task.
On the other hand, if the task has been picked up but not yet delivered, it is the second case. In this situation, the current location property of next state is the deliver city of the task. The cost property of next state is updated by adding cost from current location of current state to the deliver city of the task.

Final States

Our goal is to deliver all tasks. Clearly, in a state, as long as all tasks have been delivered, it is a final state. In our code, if each character of the string $TaskSign$ is '2', the state is a final state.

Heuristic for A*

In our A* algorithm, we implement the cost function of state n: $f(n) = g(n) + h(n)$ as follow.
$g(n)$ is the cost of state n of the agent so far.
$h(n)$ is defined as follow. For each unpicked up task, we compute the sum of distance from current city to the pick up city and distance from pick up city to deliver city as candidate heuristic. For each carried task, we compute the distance from current city to the deliver city as candidate heuristic. Then we choose the biggest one from all candidates as $h(n)$.
To prove that our algorithm preserves the optimality, we can assume the most extreme case in which all unpicked tasks and carried tasks share an overlapped path, thus the remaining future cost will be exactly the same as our $h(n)$. Any other heuristic functions greater than our $h(n)$ will overestimate in such extreme case. Therefore, our heuristic can guarantee optimality as well as satisfactory compute time.

Performance of BFS and A* for One Agent

For tasks number ranging from 3 to 9, the performance of BFS and A* is shown in Table 1.

Task Number		6	7	8	9	10
BFS	Iteration Number	199957	1837128	11205132	Time out	Time out
	Execution Time (ms)	459	3208	24157	Time out	Time out
	Minimum Cost	6900	8050	8550	NAN	NAN
A*	Iteration Number	3910	29384	161571	522935	3221301
	Execution Time (ms)	124	379	1671	6588	49562
	Minimum Cost	6900	8050	8550	8600	9100

Table 1. Iteration number and execution time for different tasks

We can see that the iteration number and execution time of BFS grow exponentially as the number of tasks increases. While the iteration number and execution time of A* grow much slower than BFS.

Maximum Number of Tasks of A* for One Agent

The maximum number of tasks for which I can build a plan in less than one minute is 10. When the number of tasks is 10, running our A* algorithm to construct the best plan costs 49 s.

Multiple Deliberative Agents

How Do We Solve Conflicts

Before demonstrating agents' joint performance, it is necessary to describe how they update their plan when they realize that the environment has changed.
Once a gent realizes the task that the environment has changed, it will remember which tasks it has picked up but not yet delivered, by the variable $carriedTasks$. Then the agent executes the plan method again in the current position, with the remaining unpicked tasks plus the carried tasks. We use task signs '0', '1', and '2' to distinguish among unpicked, carried and delivered tasks. Figure 1 shows how agents replan when conflicts happen.

```
LP: Stuck action : Pickup (Task 5)
LP: Tasks        : [(Task 4, 3 kg, 18465 CHF, Lausanne -> St-Gallen)]
A* algorithm start
Iteration Number: 23 Minumum Cost: 4340.0
Execution Time: 0 ms
[(Task 2, 3 kg, 59433 CHF, Sion -> Genève), (Task 3, 3 kg, 59433 CHF, Basel -> St-Gallen), (Task 4, 3 kg, 18465 CHF, Lausanne -> St-Gallen)]
LP: Stuck action : Pickup (Task 2)
LP: Tasks        : [(Task 5, 3 kg, 47123 CHF, Sion -> Basel)]
A* algorithm start
Iteration Number: 6 Minumum Cost: 2725.0
Execution Time: 0 ms
```

Figure 1. Replan when conflicts happen

Joint Performance

Without loss of generality, we produce 6 tasks to observe multiple agents' performance. Figure 2 shows the baseline, which is the performance of one agent.

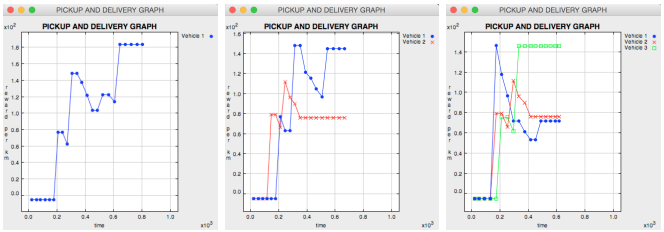

Figure 2. Performance of One, Two and Three Agents for 6 Tasks

We can see that agent 2 replaces agent 1 in some tasks, leading to decrease of reward of agent 1 in the second case. Also, agent 3 replaces agent 1 in some tasks while agent 2 is not affected by agent 3 because the locations of agent 3 is relatively closer to agent 1. Therefore, tasks are tend to be picked up and delivered by the nearest agent, and other agents just stop considering tasks that have been delivered, thus increasing efficiency for multiple agents.