Exercise 3
Implementing a deliberative Agent

Group №25: Boris Goullet, Julien Grondier

October 21, 2017

1 Model Description

1.1 Intermediate States

A state is defined by the TaskSet of the tasks not picked up, the TaskSet of the picked up but not delivered tasks, the current city, the current capacity of the vehicle, the total cost collected till now, and a list of Actions taken to get to the current task. However, the current capacity, the state's total cost, and its sequence of actions are not taken into account when determining whether two states are equal; although the cost is used to order states between themselves.

1.2 Goal State

The goal state is when both TaskSets is empty, which means there is nothing left to do.

1.3 Actions

For each state, we find its successors like so:

- For each task currently picked up, moving to that task's delivery point and delivering it is a successor state.
  The successor state computed this way will be created from the current state and modified in the following ways:
  - The picked-up task chosen to be delivered is removed from the set of picked tasks
  - The location of the vehicle is set to the delivery city of the chosen task
  - The cost of the state is increased by the distance to the delivery city
  - All the movement actions necessary to get to the delivery city + the delivery action are added to the list of actions of the state
  - The capacity is increased by the task's weight
- For each task currently available in the world that we have enough capacity to left to pick up, moving to that task's pickup point and picking it up is a successor state.
  The successor state computed this way will be created from the current state and modified in the following ways:
  - The task chosen to be picked up is removed from the set of available tasks and added to the set of picked up tasks
  - The location of the vehicle is set to the pickup city of the chosen task
  - The cost of the state is increased by the distance to the pickup city

- All the movement actions necessary to get to the pickup city + the pickup action are added to the list of actions of the state
- The capacity is decreased by the task's weight

2 Implementation

Both of our algorithms aim to find a state without any successor states, while taking the current state of the agent as a root node.

2.1 BFS

To implement BFS, we followed the slides' pseudo-code BFS implementation. It takes the first goal state it finds (which won't necessarily be the best one) and returns its list of actions.

2.2 A*

Similarly, to implement the A* algorithm, we simply followed the general A* algorithm described in the slides. It take the first goal state it finds, which is the best one as long as the heuristic function does not overestimate the true cost, and returns its list of actions.

2.3 Heuristic Function

Our heuristic function returns the following:

- If there is no item to pick up, we simply return the cost of delivering the furthest away picked-up task.
- If there are items to pick up, we return the maximum cost of going to an available item's pickup point, then delivering it.
- If there are no items to pick up and none to deliver either, then we return 0 (which doesn't matter, as that means our agent has finished)

It is admissible as it can never overestimate the true final cost: indeed, we will always have to at least pickup/deliver that "furthest" task.
If it is the last task to be completed, the heuristic estimates exactly the true final cost.
If it is not, then the agent will still have to, at some point, go to the city, pickup, and move to the delivery city. The heuristic will underestimate the final cost in proportion to how many other tasks there are left to complete (as those are like "detours" compared to the heuristic's cost).

3 Results

3.1 Experiment 1: BFS and A* Comparison

3.1.1 Setting

This experiment was done using the Switzerland topology, and rngSeed 23456 for the task distribution. The only modified variables were which algorithm our agent uses, and the number of tasks to be completed.
These experiments were run on a desktop PC with an i7-7700K@4.8GHz CPU.

3.1.2 Observations

Tasks	Loops w/ BFS	Time w/ BFS	Cost w/ BFS	Loops w/ A*	Time w/ A*	Cost w/ A*
6	9568	119s	19200	826	26ms	6900
7	38642	1167ms	21100	6330	120ms	8050
8	147010	13596ms	22100	26623	1282ms	8550
9	N/A	>60s	N/A	70072	8284ms	8600
10	N/A	>60s	N/A	N/A	>60s	N/A

We observe that our A* algorithm is both much faster and much optimal (in fact, completely optimal) in reducing travel cost than BFS.
We tried using h=0 as heuristic (aka Dijkstra's algorithm) and the cost result was, unsurprisingly, the same, but the time to compute the plan was much longer (while still shorter than BFS).

3.2 Experiment 2: Multi-agent Experiments

3.2.1 Setting

This experiment uses the exact same settings as the first experiment, but also varies the number of agents.

3.2.2 Observations

BFS Agents	A* Agents	Tasks	Total Loops	Total Time	Total Cost
2		7	87218	2405ms	24500
2		8	333693	23504ms	27000
3		7	128060	3396ms	29950
3		8	489852	33346ms	33400
	2	7	7287	130ms	10600
	2	8	33647	1083ms	13150
	2	9	121651	10239ms	15150
	3	7	7877	129ms	16050
	3	8	35451	1159ms	14250
	3	9	185609	18872ms	20350

BFS Agents	A* Agents	Tasks	BFS Profit	BFS Cost	A* Profit	A* Cost
1	1	8	108696	9750	169193	7550

We observe that multiple agents do not mean a better performance: for the same number of tasks, multiple agents will globally perform worse than a fewer number of agents. The average total cost also increases with the number of tasks. One exception we saw in our experiment is A* with 3 agents: 7 packets cost 16050 to collect, and 8 packets cost 14250.
We can also see that with multiple agents, A* is way faster to compute: with 3 agents, we can pickup 9 packets with A* with a computation time of 18872ms compared to 8 packets with BFS with a computation time of 33346ms.
When comparing directly A* to BFS, we also notice that A* gets a much bigger profit than BFS for a lesser cost.