

Excercise 3

Implementing a deliberative Agent

Group №23: Jean-Thomas FURRER, Emily HENTGEN

October 22, 2017

1 Model Description

1.1 Intermediate States

In our model, a state is characterized by the following six attributes:

- int[] **tasksStatus**: keeps track of whether each task has not been picked up yet, has been picked up but not delivered, or has been delivered
- int **taskIndex**: the index of the task being currently handled in this state
- City **departure**: the current city where the agent's vehicle is
- double **cost**: the accumulated cost in this state
- double **charge**: the accumulated charge in this state
- State **previous**: the previous state the agent's vehicle was in

1.2 Goal State

A goal (or final) state is a state where all tasks have been delivered. For a given state, this can be checked using the *tasksStatus* attribute.

1.3 Actions

There are three possible actions in our model: moving from one city to a neighbouring city, picking up a task and delivering a task.

2 Implementation

2.1 BFS

The main issue with the BFS algorithm is that the number of states grows exponentially with the number of tasks. The implementation must hence create states occupying few memory and each state visit must be as fast as possible. To this end, we use a *LinkedList* for storing the the set of all states that are yet to be visited and *HashSets* for storing the successors of the current state as well as the set of all the already visited states. In the previous *State* definition, we further use an array of *int* for keeping track of each task status: it occupies few memory, and updating it induces less overhead than *Lists* or *Sets* for instance storing the yet to be picked up/picked up but not yet delivered tasks. Following the same idea, instead of storing the complete list of actions that lead to the state, we store a reference to the previous state, which is less memory consuming.

2.2 A*

The A* algorithm is similar to the BFS algorithm. The main difference is that the queue of states is reordered as soon as new (successors) states are added to it. Moreover, once a final state is reached, the algorithm does not further visit other states, but terminates and uses this final state for building a plan.

Algorithm 1 BFS

```
INPUT
  Vehicle vehicle                                ▷ the agent's vehicle
  TaskSet tasks                                  ▷ the set of tasks to deliver
OUTPUT
  Plan plan                                       ▷ the plan found by the BFS algorithm

1:  $Q \leftarrow \{initialState\}$ ,  $S \leftarrow \{\}$ ,  $loopCheck \leftarrow \{\}$ 
2:  $minimumCost \leftarrow \infty$ ,  $finalState \leftarrow null$ 
3: while  $Q$  is not empty do
4:    $currentState \leftarrow Q.pop$ 
5:   if  $currentState$  is a final state then
6:     if  $currentState.cost < minimumCost$  then
7:        $minimumCost \leftarrow currentState.cost$ 
8:        $finalState \leftarrow currentState$ 
9:     end if
10:  end if
11:  if  $loopCheck$  does not contain  $currentState$  then
12:     $loopCheck \leftarrow loopCheck \cup \{currentState\}$ 
13:     $S \leftarrow currentState.successors$ 
14:     $Q \leftarrow Q \cup S$ 
15:  end if
16: end while
    $plan \leftarrow buildPlan(finalState, plan, tasks)$ 
17: return  $plan$ 
```

2.3 Heuristic Function

3 Results

3.1 Experiment 1: BFS and A* Comparison

3.1.1 Setting

Topology: Switzerland map

Tasks configuration: tasks uniformly distributed

Vehicle: default configuration

3.1.2 Observations

The BFS algorithm runs out of memory when the number of tasks to deliver exceeds 8; the A* algorithm can go up to 9 tasks before it throws an *OutOMemoryError* as well, or times out. More detailed performance results for both algorithms with 6 up to 10 tasks and a seed of 23456 are shown in the Table 1.

Number of tasks		6	7	8	9		6	7	8	9	10
Minimum cost		6900	8050	8550	-		6900	8050	8550	8600	-
Execution time (ms)	BFS	113	1155	9723	-	A*	102	230	2146	18147	-
Number of states		49 993	435 391	2 517 570	-		22 097	222 660	1 351 656	6 420 867	-

Table 1: Performance of BFS and A* for a seed equal to 23456

By changing the seed to 65432 for instance, the number of tasks for which the agent can compute a plan increases: the BFS agent can now handle up to 9 tasks, and the A* agent up to 10. We notice the cost for 9 and 10 tasks is the same with the A* algorithm: as the number of tasks increases, the probability that the agent will have to pick up or deliver a task to a city it previously only passed through increases as well. If its capacity allows it, it can thus design a plan which includes the additional task with a cost no higher than before (Table 2).

Number of tasks		7	8	9	10		7	8	9	10	11
Minimum cost		5700	6600	7150	-		5700	6600	7150	7150	-
Execution time (ms)	BFS	260	3122	31 627	-	A*	68	472	4501	62 165	-
Number of states		115 157	991 288	5 307 498	-		46 464	437 334	2 488 459	10 144 872	-

Table 2: Performance of BFS and A* for a seed equal to 65432

Compared to the BFS algorithm, the A* algorithm can only design a plan with one additional task before it eventually times out or throws an *OutOfMemoryError*. However, in overall, since it visits less states, the A* algorithm also takes less time to compute an optimal plan the BFS algorithm finds by visiting all the possible states.

3.2 Experiment 2: Multi-agent Experiments

3.2.1 Setting

3.2.2 Observations