Excercise 3
Implementing a deliberative Agent

Group №: Liangwei CHEN, Siyuan LI

October 22, 2019

1 Model Description

1.1 Intermediate States

We define our state as following,

- currentCity: Current city of vehicle
- notDeliveredTask: Set of tasks that have not been delivered
- deliveringTask: Set of tasks being carried by the vehicle
- parent: Parent state of current state, a state s is the parent of another state t iff s is the most recent state updating/creating t.
- actionFromParent: Action has been done by its parent state to reach current state, we define this for the convenience to back-trace.
- cost: Current accumulative cost from initial state to current state,
  $:= distance(initState.City, currentState.city) * costPerKm$
- capacity: Capacity of the vehicle
- key: String formed by currentCity.name + notDeliveredTask.toString() + deliveringTask.toString(), used to identify a state.
- costPerKm Cost per km of vehicle.

1.2 Goal State

Goal states are the states with empty $notDeliveredTask$. The target state is the goal state with lowest cost.

1.3 Actions

The possible actions are pickup a unpicked task at certain city and deliver a carrying task at certain city. In short, only the events changing the notDeliveredTask or the deliveringTask are considered as action.

2 Implementation

2.1 BFS

- Data Structures: Map$< String, State > S$ holding current visited states, Queue$< State >$ $remainingStates$ holding queue of states to be checked by the searching algorithm

- Optimization: In order to obtain an optimal path to some goal states, the algorithm does not returns once it reach some terminals. Instead, it continues searching and updating till the remainingStates become empty. After that, all goal states has been visited and the cost to them are well stored inside the states. An optimal action path is then computed by finding the goal state with minimum cost and tracking back towards the root.

- Cycle Prevention: To prevent indefinitely adding a cycle into the remainingStates, two requirements have been imposed for enqueuing into that holder. When a neighbour state n is evaluated by current state s, it is enqueued iff $n \notin remainingStates$ or $n.cost > s.cost + distance(n.city, s.city)$. The two conditions ensure that only new states or states having their cost updated will be enqueued.

- Optimal Path Construction: The parent field of a state is set to its detecting or updating states. Moreover the Pickup/Delivery action happening in current state is stored in field actionFromParent. Thus in the end a sequence of Pickup/Deliver actions can be generated from initial state to terminal state. In order to obtain the Move actions between Pickup/Deliver action pairs, the pathTo method has been exploited to generated the shortest path between cities of two states. Eventually a sequence of Move/Pickup/Deliver actions from initial state to optimal terminal will be generated in such way.

2.2 A*

- Data Structures: Map$< State, Double > C$ stores current cost from initial state to a state, PriorityQueue$< State > pq$ stores states to be evaluated with heuristic function h.

- Optimization: The algorithms construct paths from initial states to the first terminal state retrieved from the pq. The optimality is ensured by the well-defined heuristic function.

- Cycle Prevention: The neighbours n of one state s is enqueued into pq iff $s \notin C$ or $s.cost < C.get(s)$. Moreover, same techniques as in BFS are used to further prevent cyclic enqueuing. If one of the above two conditions holds, s with most updated cost will be put into C. This mechanism ensures that only new states or updated states can lead to expansion of pq.

- Optimal Path Construction: Same as BFS.

2.3 Heuristic Function

Let D, U represent delivering tasks and un-pickedup tasks separately.
$\forall state\ s$,

$$d(s) := max_{t \in D}(distance(s.currentCity, t.deliveryCity)$$

$$u(s) = max_{t \in U}(distance(s.currentCity, t.pickUpCity) + distance(t.pickUpCity, t.deliveryCity))$$

$$h(s) = max(u(s), d(s))$$

In short, it represents the greatest cost to deliver a task from current state.

- Admissibility: Follows from the observation that $h(s)$ is indeed the cost to deliver certain undelivered task from current state, which is not greater than the cost to deliver all the undelivered tasks from current state. So the heuristic function never over estimates the remaining cost at certain state.

- Optimality:$h(s)$ is actual cost to terminal state if there is only one task to be delivered. This observation follows from the fact that $h(s)$ is the maximum cost to deliver one of the tasks at current state. With vehicle's knowledge of the system, a heuristic that performs better than h with more than one tasks to be delivered is complicated and hard to evaluated since it has to consider the topology of the underlying map. While in h, one only need to use $distanceTo$ function to obtain all necessary information.

3 Results

3.1 Experiment 1: BFS and A* Comparison

3.1.1 Setting

All the experiments are done on the Switzerland map with default task configuration. The only variable is the agent type and number of tasks available.

3.1.2 Observations

Task number	Type	Round	Time(ms)	Cost
7	AStar	2468	161	8050
7	BFS	15133	293	8050
8	AStar	9036	539	8550
8	BFS	49628	1136	8550
9	AStar	22844	1617	8600
9	BFS	167059	3782	8600
10	AStar	94443	6656	9100
10	BFS	578040	19910	9100
11	AStar	277332	28492	9100
11	BFS	NA	TIMEOUT	NA

- Optimality: From the results one may observe that our BFS construct a path with same cost as AStar, thus satisfies optimality requirement.
- Efficiency: By comparing running time and rounds of two algortihms with same number of task, one may discover that AStar has much lower computing round than BFS. However, the running time difference between this two algorithms is not so big as that of computing rounds. That is because the Enqueue operation of AStar takes $O(logN)$ time while the AddtoQueue operation of BFS only takes $O(1)$ time.

3.2 Experiment 2: Multi-agent Experiments

3.2.1 Setting

In the following experiments, the Switzerland map is used. The task configuration follows the default setting. For efficiency, only AStar algorithm is used to compute the optimal path. For simplicity, all vehicles are set to have same cost per km.

3.2.2 Observations

Task Number	Vehicle Number	Average reward per Km
10	1	$2.9 * 10^2$
10	3	$1.58 * 10^2$
8	1	$2.3 * 10^2$
8	3	$1.43 * 10^2$

One may observer that with increase of number of vehicles, the avg reward per km decreases. This is due to useless moving in the sense that a task to be picked up has already been picked up by others when one vehicle is moving towards it. In the setting of deliberative agent, the vehicles have no efficient way of communication and thus will inevitably conduct useless moving.