Exercise 5: An Auctioning Agent for the Pickup and Delivery Problem

Group №9: Fengyu Cai, Wanhao Zhou

November 24, 2019

1 Bidding strategy

1.1 General architecture

The main idea of our bidding strategy is to maintain two `Bidder` objects, one for each agent. Since we do not possess information regarding the opponent agent, we need to constantly refine our guess of the opponent only via the bidding information after the auction result has been posted.

Key members of of a `Bidder` object includes {vehicles, home cities, tasks, total bids, plan solver}. Vehicles and home cites are estimated for our opponent, and are updated if errors exceed a certain range. We take the average vehicle model of our agent to estimate the opponent's vehicles. Tasks and total bids information are updated once the auction result is given.

For plan solver, we reuse the SLS algorithm implemented in the previous exercise. We do not start from scratch to search for the optimal solution of a given set of tasks; instead, we add the new task to the previous optimal solution, and resume from that point.

In the following sections, we explain our strategy to compute bids for tasks and how we refine our guess of the opponent every time the result is given.

1.2 Bidding price computation

For the first round, since we have no information about the opponent but we also would like to acquire more tasks at the beginning stage, we simply return a price regarding the task cost, defined as the total length of this task times `costPerKM`. We select a coefficient of 0.7 and multiplies with the cost to be our first bid. In the following rounds, however, we constantly speculate our marginal cost for adding this task and the opponent's bid.

We try to strike a balance between our guess of the opponent's bid, namely `bidOp`, and ourself's guess `bidSelf`. We first give a rough estimate of the bid, generated via `estimateSelfMC()` or `estimateOpMC()`, which runs in three steps:

(i) Estimate the marginal cost via SLS algorithm.

(ii) Estimate the average cost of the task, i.e. total cost of tasks divided by the number of tasks.

(iii) Keep a running average of these two components, and output a rough guess.

In step (iii) in particular, in the beginning stage when the total number of tasks is small, we generate the random dummy tasks from the task distribution and the topology information, i.e. sampling from the distribution information to get the pickup city, delivery city and the task weight. Also, there are two ratios attached to the two estimate functions, which are dynamically modified after the auction result.

We have mainly 5 functions to refine our guess. The following explains how we deal with the `bidSelf`; the same applies for `bidOp` as well.

(i) `bidSelf = Math.min(bidSelf, bidOp`$*1.2$`);`

(ii) `bidSelf = Growth(bidSelf, round);`

(iii) `bidSelf = SinFlutuate(bidSelf, round);`

(iv) `bidSelf = Math.max(bidSelf, Math.max(10, taskCost));`

(v) `bidSelf = Math.max(bidSelf, 0.2 * (nextCost - totalBid))`.

`Growth` function allows our bid to grow as round goes up and is relatively small at the beginning. `SinFlutuate` is basically a shifted sine function to add some randomness to our bid, fluctuating between `[bidSelf, 1.15 * bidSelf]` with a period of 15 rounds. Function (v) might increase a little bit of our bid if the gap between the cost of accepting the next task and our total reward so far is big.

After these operations, we output the final bid.

1.3 Updating parameters for `Bidder`

We keep modifying the `Bidder` object corresponding to our opponent when there is a gap between the guess and reality. Two ratios described above, `selfRatio` and `opRatio` are also modified in this process.

In the first round, after we get the result of the auction, we initialise the opponent bidder object `bidderOp`. We estimate first the most possible city (i.e. having the lowest cost of carrying out the previous task) and also other candidates. We place dummy vehicles (generated from the average model of our vehicles) at such cities to form our opponent.

In the following rounds, we keep refining our model. We keep track of our guess of the opponent's bid and the actual bid given in the result. If our guess is smaller, we increase the `opRatio` which will in turn gives a higher estimate in the next round; if our guess is much bigger than the actual value, we modify the `bidderOp` object. This is achieved by placing a new vehicle for our opponent at the previous task's pickup city and recompute the cost via SLS.

If we successfully win a task, we increase our `selfRatio` to potentially generate a larger bid next time and vice versa. We also keep track of the percentage of tasks won by us: if our winning rate is relatively high, we can potentially increase our bid.

1.4 Plan generation

In the auction process, our plan as well as the cost estimation is generated by adding task one by one in our modified SLS algorithm. We run a maximal of 1000 rounds each time and choose the best neighbour with a probability of 0.7, which is the best parameter from our previous exercise.

When outputting our final plan for all the accepted tasks, we run another 1000 rounds to search for other possibilities, and finally choose the best plan for the tasks.

2 Results

2.1 Experiment 1: Comparisons with dummy agents

In this part, we will try to compare our auction model with the dummy agents.

2.1.1 Setting

We select two topologies, England and the Netherlands. And the total number of task in each auction will be set 10, 20, and 30. In each competition between our trained auction and the dummy agent, we will run 5 times and record the results from the perspective of our trained model.

2.1.2 Observations

From the Table 1, we could found out that in every setting, our trained model can beat the dummy agents. In the most cases, our trained agent will never lose to the dummy agents. When the number of task is low, it gets more difficult to win the competition. The first reason is randomness, and secondly, it is hard for our trained agent to precisely estimate the opponent parameters.

Topologies\Number of Tasks	10	20	30
England	3-2-0	5-0-0	5-0-0
The Netherlands	5-0-0	5-0-0	5-0-0

Table 1: Competition Results with Dummy Agent (X-Y-Z indicates the number of X winnings, Y draws, and Z lost from the perspective of our trained agent)

2.2 Experiment 2: Compare with model without opponent simulation

In this part, we would like to compare our trained model with the simpler model which does not consider the margin cost of the opponent. In our trained model, we simulate an opponent bidder, and every time we calculate the bid, we will also consider the estimated opponent's bid to adjust our bid. We do this experiment to find out the effect of this mechanism.

2.2.1 Setting

We choose England as the background topology, and run the agent with and without the estimation of the opponent agent for 5 times with different random seed. Each time, we set the number of task 20 which is the median in the range of the task number.

2.2.2 Observations

In Table 2, we run the experiment with different seeds, and the model with opponent simulation is in the front in each cell. We find in the six experiments, our trained model with opponent simulation wins five times. From the comparison above, the main effect of opponent simulation will help generate more reasonable bid and seize more tasks. Meanwhile, we find that the simulation of opponent can also prevent the model from seizing some very expensive tasks, which will make the opponent's cost very high. Therefore, more number of tasks and relative low cost will bring our smart agent a higher profit.

	Total Tasks	Total Distance	Total Cost	Total Reward	Total Profit
Seed 1234	15-5	1602.2-1635.8	8011-8179	8326-6727	315-(-1452)
	12-8	1167.6-1506.0	5838-7530	7272-7121	1434-(-409)
Seed 12345	14-6	1370.4-1659.0	6852-8295	6756-6429	(-96)-(-1866)
	11-9	1275.8-1335.8	6379-6679	6351-6672	(-28)-(-7)
Seed 123456	12-8	1328.8-1609.9	6644-8050	6068-7328	(-576)-(-722)
	13-7	1163.3-1668.8	5817-8344	5873-8344	56-(-1600)

Table 2: The comparison between our trained model and the model without opponent simulation