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

Group №70: Julien Perrenoud, Pierre-Antoine Desplaces

November 25, 2018

1 Bidding strategy

1.1 Setup

First let's look at the setup we have. The bidding process goes over an unknown number of rounds (between 10 and 50) where a new task is selected from a known probability distribution. Both companies have to submit a bid, the lowest of each wins the task and has to deliver it for the price given. At the end of the game, the company with the highest revenue wins. This setup is somewhat equivalent to a multi-round first-price sealed bid auction where the history is known and the rounds are not independent from each other.

1.2 Task Cost and Utility

Our first step towards a reasonable bidding strategy was to define a fair definition of the "true cost" and utility of a task upon which to base our bid. This is not trivial as the real delivery cost depends on the future of the game and is therefore uncertain at the time of the bid. Nonetheless, we tried to approximate it by making use of the following elements,

- The immediate **marginal cost** $\text{MarginalCost}_A(t) = \text{Cost}_A(T_A^{(i)} \cup \{\ t\ \}) - \text{Cost}_A(T_A^{(i)})$ for delivering task t given our other tasks T_A.
- The **current round** of the game i. Having more tasks is a clear competitive advantage as it allows us to deliver tasks at a lower cost later on. Therefore, we want to bid more aggressively in the initial rounds in order to "capture the market". We implement this behavior by defining an initial ratio r_0 that will linearly increase over a given horizon h so that $r^{(i)} = \min(1, r_0 + (1 - r_0) \cdot (h - i))$
- The **probability distribution** of the tasks and **topology** of the map. The intuition here is that the cost of a task greatly depends on our ability to bundle it with other similar tasks. For instance, tasks connecting cities with higher transit will be easier to deliver. One way to calculate this future cost would be to look at the expected cost of any set T of an arbitrary size n containing our task $E_{T,t\in T}[\text{Cost}(T)] = \sum_{T,t\in T} p(T)\text{Cost}(T)$. Unfortunately, in practice this would require us to run our CSP solver an exponential number of times. Here, we decided to focus on an easier computation to approximate this value and only look at the joint cost of each pair of tasks by computing $c(t) = \frac{\sum_{t'\in T} p(t')\text{Cost}(t\cup t')}{\sum_{t_1,t_2\in T} p(t_1)p(t_2)\text{Cost}(t_1\cup t_2)}$.

We then combine these elements to compute an approximated long-term cost of the task $\gamma_A^{(i)}(t) = r^{(i)} \cdot \text{MarginalCost}_A^{(i)}(t) \cdot c(t)^\alpha$ where α is a coefficient of the joint cost.

1.3 Considering the Opponent

1.3.1 Guessing their Cost

The goal of the game is to have the highest profit $p = \text{Reward} - \text{Cost}$. Therefore, we should also consider minimizing the opponent's profit (in order to maximize $p_A - p_B$). Our intuition here is to place a bid that will minimize the opponent's ability to make a profit.

In order to do this, we first need a way to estimate their expected cost for each task. This cost is different than ours as it depends on their current tasks $T_B^{(i)}$ and vehicles V_B. The former is easy to calculate $T_B^{(i)}$ given the history of bids, however estimating V_B can be more challenging. We could analyze their first (few) bid(s) and attempt to derive their home cities from this information, but it does not work well in practice as the first bids might only loosely be correlated with the marginal cost.

Instead, we decided to simply assume that the opponent has a vehicle in each of the city. This gives us a nice lower bound which tends to be fairly accurate as the game goes on, and since it has a known downward bias that we can try and compensate for.

1.3.2 Minimizing their Profit

Now that we have a reasonable estimate of our true cost $\gamma_A(t)$ and the opponent's true cost $\gamma_B(t)$. The strategy that minimizes the maximum loss is a bid $b_A = \frac{\gamma_A(t)+\gamma_B(t)}{2} = \bar{\gamma}$. A way to see that is that if $\gamma_A(t) \leq \gamma_B(t)$, then we agent A can make a maximum profit up to $\bar{\gamma} - \gamma_A(t)$. Either we win the auction and make this profit, or we lose but B had to take a loss of $\bar{\gamma} - \gamma_B(t)$ which is an equivalent profit for us. If we deviate from $\bar{\gamma}$, then it allows the opponent to bid between $\bar{\gamma}$ and b_A and make a better profit or smaller loss. This is symmetric when $\gamma_A(t) > \gamma_B(t)$ except that this time we have a disadvantage and try to minimize our loss. This highlights the need to grab more tasks early which we discussed above, as it allows us to lower our γ_A for future tasks and give us an advantage in the negotiation.

Finally, there are cases where $\gamma_A(t) = \gamma_B(t) = 0$. In such cases, it does not make sense to bid $\bar{\gamma} = 0$ as there is always a slight cost to any task taken. We handle such cases by taking the lowest known bid of the opponent discounted by 1.

1.4 Competitive Bidding Strategy

Putting it all together, this gives us the following formula.

$$b_A^{(i)} = r^{(i)} \cdot \frac{\gamma_A^{(i)}(t) + \gamma_B^{(i)}(t)}{2} \cdot c(t)^{\alpha}$$

Now before we submit that bid, we do have a couple of sanity checks in places. Mainly,

- If the bid is negative, we apply the same strategy as if we had 0. Such cases tend to happen more when the task set grows larger because the CSP solver might wrongly calculate a negative cost if it does not have enough time to converge to a good solution.
- In order to counter the fact that we underestimate the actual cost from the opponent, we apply a small bias $\beta = (\beta_0, \beta_1)$ to the bid $b_A^{(i)}$. This bias modifies the bid as $(b_A^{(i)})' = \beta_0 \cdot b_A^{(i)} + \beta_1$

2 Results

2.1 Experiment 1: Comparisons with dummy agents

2.1.1 Setting

In this experiment, we create a tournament over 4 configurations between our agent `Competitive` and several dummy agents with different behaviors. `AFK` is an agent that does not make any bid, `Naive` attempts to deliver each task sequentially, `Zero` always bids at his marginal cost and `Coeff` uses a coefficient to multiply his marginal cost and adjust it according to the bids of the opponent. We run the tournament on 3 different fair configurations with various number of tasks, vehicle and topologies.

The 3 configurations are : 20 tasks in England with 2 vehicles for both (2 of capacity 40 vs 1 of capacity 30 and 1 of capacity 50), 30 tasks in the Netherlands with 3 vs 4 vehicles and finally 40 tasks in France with 4 vehicles for both.

2.1.2 Observations

Fortunately, we can see from the table of results that our agent won all of the matches played (except a close one against the best of dummy agents `Zero`). It is also interesting to note that `Coeff` lost all of its matches. The reason is that it loses all the early tasks because of an initial coefficient of 1. This causes it to aggressively lowers its coefficient, from which it cannot recover quickly enough and always end up with a negative profit.

Agent \Opponent	**Competitive**	**Naive**	**Marginal**	**AFK**	**Coeff**	**# Victories**
Competitive	-	6	5	6	6	**23**
Naive	0	-	3	6	6	**15**
Marginal	1	3	-	3	6	**13**
AFK	0	0	3	-	6	**9**
Coeff	0	0	0	0	-	**0**

2.2 Experiment 2

2.2.1 Setting

In this experiment, we now take our `Competitive` agent and try different values for its parameters.

2.2.2 Observations

Here, we can see that using the joint cost as coefficient seems to provide a small improvement on the basic algorithm. We also see that the negative bias fairs well, which indicates that we could potentially slash prices even more aggressive in early game.

Agent $(r_0, h, \alpha, \beta_0, \beta_1)$ **\Opponent**	**Future**	**Negative**	**Positive**	**Neutral**	**# Victories**
Future $(0.5, 15, 0.5, 50, 1.05)$	-	5	4	3	**12**
Negative $(0.5, 15, 0, -50, 0.95)$	1	-	4	4	**9**
Positive $(0.5, 15, 0, 50, 1.05)$	2	2	-	5	**9**
Neutral $(0.5, 15, 0, 0, 1)$	3	1	2	-	**6**