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

Group №2: Cosmin-Ionut Rusu, Sorin-Sebastian Mircea

November 24, 2019

1 Bidding strategy

The bidding strategy is split into two major parts. The first one focuses on computing the cost of the possible plan, and then, the following receives this value and decides the profit margin that is to be added depending on multiple factors.

1.1 Computing the cost

For every new task that the bidding system exposes, we temporarily simulate adding it to the current plan and compute the new cost (also called our marginal cost).

Choosing the algorithm

Since we know the set of tasks and we need to pick all of them, we must minimize the cost to deliver all of them, which is defined as the sum of all the distance traveled by each vehicle weighted by their corresponding cost per km.

For this task we have chosen to implement *Stochastic Local Search* on the grounds that it has a flexible run-time, being able to adapt to different time limits (more time allowed increases the possibility of reaching an optimal cost plan).

1.2 Computing the margin

After we have computed the cost of the plan that can be formed by possibly adding the current bidding task to it, an initial value of our bid is formed in the following way:

$$bid_i = costPlanWithTask_{(i)} * expectedMargin - totatAmountBiddenSoFar$$

After more experiments we found an *expectedMargin* value of *1.2* to provide the best results.

This bid value is not final yet, during the next subsections we are going to present the transformations that will occur on top of this initial value.

Dealing with tasks that do not bring any profits to our plan

A negative bid shows that the task does not bring any profits to the last plan (i.e benefits of taking it do not overcome the costs), in this case, we decided to not take the task, so we bid *null* for it.

In this context, by knowing the probability distribution of the tasks, we have also considered looking into all the cities that our plan takes us into, this allows us to compute an expected value of possible tasks that we can pick up on our road. By analyzing the magnitude of this value we thought about making exceptions to the above rule, as we couldn't find such a threshold that would render positive results we abandoned the initiative.

Adaptive margin based on bidding performance

As the goal is to maximize our profits, a greedy strategy must be implemented. If we win the bid multiple times, we slowly increase our confidence (base on the winning streak) and so we increase the bid by a small percentage (5%) every time we win, when we lose a bid we reset this confidence level and for us to build it again we must regain the winning streak.

This algorithm works the same for the opposite case of losing multiple times in a row, a case where we would decrease our confidence and bid a little bit more as multiple loss streak accumulate.

Of course, we have capped the streak to a parametrized value allowing the margin to fluctuate between $(-40\% : +40\%)$ of the margin values.

Taking into account the adversary margin

On the grounds that the performance of the adversary and our cost computing algorithm can be considered constant throughout the biddings, even more, by knowing which tasks our agent and the adversary won in the previous bets, we can compute the adversary approx margin from the following formula:

$$adversaryMargin = adversaryBid - planCost$$

This allows us to compute the mean and standard deviation of its margin.

We choose to use this value to further limit the range of our computed margin in the following way

```
// if my_marginal_cost is lower than other_next_bid_estimate-const*variance
// bid with something between marginal cost and lower than other_next_bid_estimate-const*variance
if (bid < mean - 2 * std) {
  bid = mean - 2 * std;
}

if (bid > mean + 2 * std) {
  bid = mean;
}
```

Figure 1: Adjusting the bid based on adversary's bid mean and std

2 Results

2.1 Experiment 1: Comparisons with dummy random agent

In this experiment we are juggling with the *expectedMargin* parameter

2.1.1 Setting

The experiments are run with the default values found in *action2.xml* file. The xml will load Netherland's topology with a timeout of setup/plan/bid of *500ms*

2.1.2 Observations

Agents	Win - Draw - Lose	Adversary	OurAgent
Adversary	2 - 0 - 0	-	WIN (936 : 614)
OurAgent	0 - 0 - 2	LOSE (286 : 705)	-

Figure 2: expectedMargin of *0.8*

Agents	Win - Draw - Lose	OurAgent	Adversary
OurAgent	2 - 0 - 0	-	WIN (2399 : 708)
Adversary	0 - 0 - 2	LOSE (1046 : 1591)	-

Figure 3: expectedMargin of *1.2*

Agents	Win - Draw - Lose	Adversary	OurAgent
Adversary	2 - 0 - 0	-	WIN (1729 : 577)
OurAgent	0 - 0 - 2	LOSE (1516 : 1767)	-

Figure 4: expectedMargin of *2.0*

The results were predictable, a too-small value of expectedMargin will lead to negative profits, and too high value will lead to a high chance for losing bids (especially against a capable adversary).

2.2 Experiment 2: Comparisons with dummy constant margin agent

This experiment is going to test our agent's performance against an adversary that has a different constant margin (through a round).

2.2.1 Settings

The same settings as the above experiment were kept.

2.2.2 Observations

All the adversary margin values lead to wins for our agent. One interesting aspect we can observe is given by the increase in bids for our agent as adversary margin increases, thus proving that our adaptive algorithm works.

Agents	Win - Draw - Lose	OurAgent	Adversary
OurAgent	2 - 0 - 0	-	WIN (-152 : -2693)
Adversary	0 - 0 - 2	LOSE (-2949 : 122)	-

Figure 5: adversary margin of *0.8*

Agents	Win - Draw - Lose	OurAgent	Adversary
OurAgent	2 - 0 - 0	-	WIN (3850 : 1103)
Adversary	0 - 0 - 2	LOSE (869 : 2481)	-

Figure 6: adversary margin of *1.2*

Agents	Win - Draw - Lose	OurAgent	Adversary
OurAgent	2 - 0 - 0	-	WIN (2866 : 2147)
Adversary	0 - 0 - 2	LOSE (5441 : 5896)	-

Figure 7: adversary margin of *2.0*