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

Group №70: Jelena Banjac, Stratos Triantafyllou

December 8, 2019

1 Bidding strategy

Here we explain the details of our bidding strategy. For more information regarding the implementation itself, please refer to the code.

Q1: Do you consider the probability distribution of the tasks in defining your strategy?
Yes, we are considering the **probability distribution** defined in the configuration files of the project. The strategy is implemented as a part of `askPrice` method. We compute the future solutions using this simulated distributed tasks in order to **predict** the best cost and therefore bid with the best value. For more details on the implementation see next questions.

Q2: How do you speculate about the future tasks that might be auctions?
To speculate about the future tasks, we use the `TaskDistribution` given from the Logist platform. Since our method for computing the marginal cost initially was extending the current solution with the one task only, we tried to improve it by adding additional random tasks generated using the Logist distribution option and afterwards recomputing their costs to calculate new marginal costs. We also use the information that the lower bound of minimal auction tasks is which can also be manually increased as a parameter of this program. Additional parameter that is specified is the parameter saying how many simulated solutions we will calculate for each simulated future task.

Q3: How do you use the feedback from the previous auctions to derive information about the other competitors?
We collect the feedback from the previous auctions in `auctionResult` method. After saving the solution that will be used for the plan, we collect the bids of agents and their ID in `bidHistory` structure that will be later used to make a bid in a next tournament cycle (bidding for the next task during the one tournament).

Q4: How do you combine all the information from the probability distribution of the tasks, the history and the planner to compute bids?
To compute bids, we first start by running the method to calculate our marginal cost from the SLS solution we implemented in the previous exercise with centralized agent. The implementation of this method is located in the `computeMarginalCost_advanced()` method. In this method we start from the initial solution given we know the vehicles and current number of tasks. Then we extend that solution by adding the new task to the biggest vehicle. Again, we recompute the solution of our SLS algorithm. After that we have a marginal cost described as a difference between extended solution and old solution. To improve this implementation, we do several iterations that will try to predict the marginal cost of some tasks that would be accepted in the future. We add new *simulated tasks* imitating the future tasks using the `TaskDistribution` we have in Logist platform that represents the **probability distribution**

of the tasks. All these simulated solutions are calculated in order to predict the future marginal cost and give the best bid. In this case we don't save the **solution of the SLS** on the simulated tasks since for the real plan it will need to recompute the whole plan. Therefore, after we get the marginal cost, we can calculate the bid using the **history of bids** we collected from the previous cycles. Our marginal cost will be initially returned as a bid in the first cycle. However, after the first cycle we already have some bids and winners collected so we use this opportunity to adjust our bid. We do it by finding the agent that has the best bid history, we get his minimal bid, and only if his minimal bid is larger than our marginal cost we continue by increasing our bid with some tolerance specified as an input to the program.

2 Results

In this section we explain the results from experiments with our auctioning agent.

2.1 Experiment 1: Comparisons with dummy agents

We test our agent against two different agents, the one provided in the template, and an agent that utilizes SLS to compute an efficient plan but only bids its marginal cost for each auctioned task.

2.1.1 Setting

We compare the three agents by running the tournament. Aside from our agent *SLS*, we use two other agents:

- *Template* is the agent provided in the skeleton code. It uses a single vehicle and works in a linear way by appending each new task at the end of its current plan. When a task is auctioned, *Template* computes the marginal cost of delivering the task after all previous tasks have been delivered. It computes the bid based on this cost, by multiplying it by a random number between 1 and 2. After tasks have been assigned, *Template* computes its final plan the same way, by adding each of the tasks linearly.

- *Simple*, on the other hand, uses the SLS algorithm to compute the marginal cost of each auctioned task. It also generates a bid randomly, exactly like *Template*, but in the end it builds an efficient plan based on SLS.

2.1.2 Observations

As seen in Table 1, our agent *SLS* beats both *Template* and *Simple* every time. *Template* does not benefit from SLS optimization, which results in lower computed marginal costs during the auction. Both *SLS* and *Simple*, when competing against *Template*, can offer lower bids, and thus win more tasks. In addition, they are able to build an optimal plan in the end, that optimizes profit. As such, *Simple* beats *Template*.

SLS beats *Simple*, because the former one takes into account the probability distribution of tasks, and is also able to offer lower bids based on its risk strategy.

Agents	SLS	Simple	Template
SLS	-	WIN (5643 : 1610)	WIN (9902 : 3019)
Simple	LOSE (491 : 1423)	-	WIN (6437 : 3353)
Template	LOSE (2539 : 16224)	LOSE (3160 : 7184)	-

Table 1: Performance (profit) of the three agents against each other

2.2 Experiment 2: Varying the internal parameter values

Here we examine the sensitivity of our agents in regards to the following parameters:

- `minTasks` indicates the number of tasks included in a temporary solution, below which we extend the plan with simulated tasks (based on the probability distribution) in our computation of the marginal cost. Increasing this value essentially means that we use this strategy for a longer part of the auction.
- `numPred` indicates how many different "futures" the agent considers in computing the bid when using the simulated task scheme.
- `riskEpsilon` essentially refers to how "aggressively" the agent attempts to further cut down the value it bids for a task. A higher `riskEpsilon` means that the agent might win more tasks, but make less profit out of them.
- `margEpsilon`, on the other hand, is associated with the agent's attempt to raise the bid close to the minimum winning bid from previous rounds. A larger value here represents an attempt to gain more from each auctioned task, provided the marginal cost is low enough.
- `depth` indicates how many previous rounds the agents considers in finding the minimum winning bid. A higher value means that the agent refrains from raising its bid even when immediately previous winning bids were higher.

2.2.1 Setting

For the experiment, we run a tournament in which agents with the same parameter set to different values compete against each other. The initial values of the parameters were: `minTasks` = 5, `numPred` = 10, `riskEpsilon` = 0.7, `margEpsilon` = 0.5, and `depth` = 5.

2.2.2 Observations

minTasks *(by default* 5*, also tested for* 10*)* When increasing the number of tasks, the performance decreases.

numPred *(by default* 10*, also tested for* 5 *and* 20*)* The agent for which `numPred` was set to 5 performed best out of the three.

riskEpsilon *(by default* 0.7*, also tested for* 0.3 *and* 0.9*)* The best performing agent had its `riskEpsilon` set to 0.9.

margEpsilon *(by default* 0.5*, also tested for* 0.2 *and* 0.8*)* We got maximum performance for `margEpsilon` set to 0.8.

depth *(by default* 5*, also tested for* 3 *and* 10*)* Performance increased when we opted for `depth` = 10.

After the tournament finished, we determined the optimal parameters for our agent are: `minTasks`**=5,** `numPred` **= 20,** `riskEpsilon` **= 0.9,** `marginEpsilon` **= 0.8,** `depth` **= 10.**