An auction agent for the PDP

Prisca Aeby, Alexis Semple

Our task for this lab was to design an agent capable of making online bids for tasks within a world, knowing at the time of the bid only the details of the task, and after each bid is made, the bids made (including the opponent) and the winner of the bid. The model of bidding we follow here is a *closed-bid-first-price reverse auction*.

The problem, then, was to design the agent such that it would be able to determine a bid for which it would make a net profit, that is larger than its marginal cost for that task. Furthermore, it would need to estimate in some way the behaviour of the opponent. This is necessary for our agent to obtain tasks when facing an agent which would systematically bid lower than us and still obtain a good profit. If our implementation can decide to bid below our marginal cost (meaning a net deficit for one turn) while a profit could be made as more tasks are won, it would be able to circumvent such situations.

Bidding

Our main strategy for `askPrice` the method that proposes a bid for a given task, relies on our agents estimation of the opponent's bid, learning from past bids that were made, and our own marginal cost calculated for this task.

Bid decision for our agent

The computation for our agent's bid was made by creating a plan for a potential agent, i.e. one which we assume has the proposed task in its `TaskSet` along with all previous tasks won in the auction so far. The plan is created using the SLS planning algorithm we created for the previous lab, which we tweaked a bit to make it perform faster and yield better results. Since this is not the object of this lab, we will not dwell on its implementation here.

Given a `TaskSet` $\mathcal{T}$ and a proposed task $t \notin \mathcal{T}$, we already know the cost computed for the plan using $\mathcal{T}$ and keep track of it using a class called `IncrementalAgent`. We can then compute the cost of $\mathcal{T} \cup \{t\}$ (our so-called potential agent) and use both of these to compute their difference, i.e. the *marginal cost*.

Bid decision for the opponent

Since we do not know the exact configuration of the agent we are facing in an auction, our strategy relies on n different agent configurations (with $n = 3$ for our tests). The idea behind this is that if we compute the marginal cost for several agents, all with randomized parameters within certain constraints, we would get a more general estimation of the opponent's bid than if we'd used only one random configuration.

By the problem definition, we knew that the vehicle configurations for the agents we would be using weren't fixed, so we decided, starting from our own vehicle set, to randomize their details. The number of vehicles is chosen between 2 and 5. The capacity of each vehicle is defined randomly between 75% and 125% of each 'original' vehicle, while keeping the sum of the capacities the same. The home city of each vehicle is also picked at random. For more details, see the `randomizeVehicles` function in the `IncrementalAgent` class.

By computing the plans for these different agent configurations, and their mean cost, we can obtain an estimated marginal cost for the opponent in the same way as for our agent (i.e. using the previous costs and the potential costs), multiplied by a factor `ratio` which is explained below.

We split the time available for the computation of the bid according to the current task set size of the opponent and our own task set size.

We set our bid to be $0.85 * opponentMarginalCost$, so that we bid just below our estimated bid for the opponent. However, if this bid is lower than our computed marginal cost multiplied by a factor `moderate` (explained below), it is set to that value instead, so that our bid doesn't result in a net loss for us (at least not in the long run).

Moderation and learning

An exception to the general behaviour of our agent described above is allowed by the use of the `moderate` factor, which is designed to let the agent bid below his computed marginal cost. This is applied mostly in the beginning of the auction, in order to have a greater chance of getting the early tasks. We thus set `moderate` such that $0.5 \leq$ `moderate` < 1 (good performance in tests for $0.6 \leq$ `moderate` ≤ 0.8), starting at a lower value to begin with, and increasing it by a fixed increment (say 0.15) every time a task is won, and decreasing it by a smaller value (say 0.05) every time the opponent wins one. The moderation later on in the auction can still happen, but the factor is often weaker. The factor is updated in `auctionResult`.

The marginal cost of the opponent in the `askPrice` phase is multiplied by a factor `ratio`, which is defined as the ratio between the actual bids of the opponent and our estimated bids. We use this value of past bid ratios as a correction for the current ratio, where more recent ratios have a greater weight. The value is bounded such that $0.75 \leq$ `ratio` ≤ 2.5 in order to not let its effect be unconstrained. This value is updated in the `auctionResult` method.

Testing the agent

In order to assess the efficiency of our auction agent, we tested it against some agents with simple behaviours. These agents were : the `AuctionTemplate` given for the lab (called **AT** hereafter), which computes a bid based on the distance of delivery of the task; a random bidder (called **RB**) which bids a random value between 5 and 5005 for each task; a marginal bidder which bids 5% above its computed marginal value (called **MB+**; another that bids 5% below its marginal value (called **MB-**). MB+ and MB- both use our SLS algorithm to compute their marginal cost, so one must expect more unpredictable values in a real situation. We call our agent **MA** (for MyAgent) and show the results of the tests in the following table.

Opponent	Winner	Profit	Tasks
AT	**MA**	6938, 787	17, 3
	MA	8957, 1512	17, 3
	MA	3733, 473	9, 1
	MA	5584, 473	9, 1
RB	**MA**	11330, 1225	17, 3
	MA	11242, 943	18, 2
	MA	3047, 473	9, 1
	MA	4730, 473	9, 1

Opponent	Winner	Profit	Tasks
MB+	**MA**	7589, 291	15, 5
	MA	6158, -1590	14, 6
	MA	4038, 77	8, 2
	MA	4475, 91	9, 1
MB-	**MA**	8902, -150	16, 4
	MA	16004, -188	19, 1
	MA	5161, -143	7, 3
	MA	2512, -143	7, 3

The results of the profit and tasks are noted first for **MA** and then for the opponent. Our agent won the match in every case, so he performs well against basic behaviours. We ran 4 different matches for each dummy agent, with either 10 or 20 tasks, and changing the value of `moderate` between 0.7 and 0.8 (the difference is inconclusive).

Conclusion

This auction agent needs to essential strong components. First it needs the strong backbone of the planning algorithm (the SLS in our case). The better this algorithm, the lower he'll be able to bid for a task, thus giving him a greater possibility to win tasks. Second, he needs to be able to adapt to the behaviour of the opponent, such that, even if the opponent has a better planning algorithm, he can still win tasks and make a profit. Working with the past of the opponents bids is an efficient way of doing this, alongside with estimating the bids he's making, although the latter can be very time-consuming.

Other strategies could have included working with the task distribution provided, but we decided to not implement this in our case, assuming the net result would not be much better than working with just the estimations and history of bids.