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

Group № 23: Alessandro Bianchi, Davide Nanni

November 22, 2020

1 Bidding strategy

Our bidding strategy combines the following three three information:

- Topology
- Bids history
- Task Probability Distribution

1.1 Topology

The algorithm starts with a one-time computation of a table, which maps a topology-dependent score to a city in a one-to-one fashion. In particular, for any given City A, the following value is computed:

$$avgScore(a) = avg(dist(A, B)) \ , \quad B \in Topology \ \wedge \ B \neq A \, .$$

Then, for each City A,

$$\frac{avgScore(A) - mean(avgScore)}{max(|\ avgScore\ |)}$$

is stored. The normalization over the maximum is actually applied ignoring values that are further from the mean than 2 standard deviations.

When a new task is auctioned, the algorithm tries to compute a new optimal plan for all its vehicles in a centralized manner. In particular, the result with the lowest marginal cost is picked among the results of two strategies: a deterministic approach, which tries to insert the task at various positions in the previous existing plan, and, if there's enough time, the SLS algorithm, adapted from the previous exercise. We use both these approaches together as the SLS may still find a better plan, since it also shuffles tasks between vehicles.

1.2 Bids history

In addition to calculating its own marginal cost, the algorithm attempts to find the opponents' ones. As no information is known concerning the opponents' starting cities, those are determined in the following manner:

1. As a task is won by an unknown competitor, its bid b is considered.
2. For up to 50 times, a random mapping of home cities and vehicles starting city is chosen and the cost of delivering the task is computed.
3. The configuration that gives the closest result to the computed bid (thus, supposing it does not differ too much from the marginal cost) is saved.

Once this is done, the supposed marginal cost of all the opponents are computed at each step, and only the lowest one is considered. We will be referring to this as $advMarginalCost$.

The last decision-parameter computed at each turn is $min(bids)$, which is then stored in a size-limited queue $minBidHistoryWindow$ which serves as a sliding window. When a price has to be computed, $min(minBidHistoryWindow)$ is retrieved: we will be referring to this parameter as $historyMin$.

After all the marginal costs have been computed, the algorithm behaves differently based on the value of the marginal cost (whether it is 0 or strictly positive), as well as the amount of new cities encountered and their appeal, which is based on connectivity and tasks' probability distribution:

- The first case involves a new computed plan with a non-positive marginal cost. This may happen if then new task is on the way of the already computed path, or if a better new plan is found through SLS; in this last hypothesis, the marginal cost may even be negative. For such a condition, the bid is formulated based on the opponent's marginal cost and the minimum calculated over the sliding window. In particular, we have:

  $$targetBid = max(1,\ min(advMarginalCost,\ avg(advMarginalCost,\ historyMin)))\,.$$

  Both $advMarginalCost$ and $historyMin$ are attenuated with safety margins before the computation of the min.

- The second case regards a computed plan with positive marginal cost. Here, two scenarios unravel:
  - If no new cities are present, the same algorithm as above is used, although with $marginalCost + 1$ as the first parameter of the max function instead of 1, representing the lower bound of the offer.
  - If one or more new cities are new to the plan, two case-specific attenuating factors are used. The first one is computed as the average of the cities' pre-computed scores. Considering p_i as the probability of there being a task to a nearby city ($distance \leq median(avgDistances)$) from the city i, we compute the second score as $1 - (1 - p_1) \cdot\ \ldots\ \cdot (1 - p_n)$. These are also attenuated by some multiplicative factor. The goal of the first factor is to favor well connected city and ask a higher price for isolated ones; the goal of the second one is to slightly lower our bid, proportionally to the probability that a new city will give tasks to nearby cities (and that will therefore have a low marginal cost). Finally, the $targetBid$, defined as $max(marginalCost,\ avg(marginalCost,\ advMarginalCost))$, is multiplied by the two factors, thus giving the actual bid.

The final bid is ultimately edited so that it is always non-negative and so that it never causes a deficit (not even temporarily).

It is worth noting that, except for ADV_RETAIN, all of the attenuating factors are adaptive, and increase or decrease based on their effectiveness. If a factor has been used and has led to a task being taken, it is modified so that the next time it will allow for a higher bid. If, on the contrary, its use has led to losing a task, it will cause a more conservative behavior the subsequent time.

2 Results

2.1 Experiment 1: Comparisons with dummy agent

2.1.1 Setting

In this experiment, we have run multiple tournaments, in order to see how our Auction Agent here named *htd_adv_2* fares against the *dummy* agent provided in the handout. In order to provide clear results, we have paired the two agents in the England topology, with a number of tasks going from 10 to 30 with step 10. Other topologies have been tested with similar results and have therefore been deemed uninteresting.

Task #	Agent	Win	Draw	Lose	htd_adv_2	naive	Total gain
10	htd_adv_2	2	0	0	-	WIN (7906 : 1077)	11,627
	naive	0	0	2	LOSE (29 : 3721)	-	1,106
20	htd_adv_2	2	0	0	-	WIN (14199 : 1077)	20,993
	naive	0	0	2	LOSE (29 : 6794)	-	1,106
30	htd_adv_2	2	0	0	-	WIN (20828 : 1077)	34,305
	naive	0	0	2	LOSE (29 : 13477)	-	1,106

Table 1: htd_adv_2 vs naive

Task #	Agent	Win	Draw	Lose	htd_adv_2	safe	Total gain
10	htd_adv_2	2	0	0	-	WIN (266 : 2)	3,397
	safe	0	0	2	LOSE (165 : 3131)	-	167
20	htd_adv_2	2	0	0	-	WIN (2939 : 4)	8,461
	safe	0	0	2	LOSE (168 : 5522)	-	172
30	htd_adv_2	2	0	0	-	WIN (4697 : 8)	19,847
	safe	0	0	2	LOSE (254 : 15150)	-	262

Table 2: htd_adv_2 vs safe

2.1.2 Observations

As we can see from Table 1, *htd_adv_2* is the clear winner, dominating both as the first and second company. The naive agent appears to only be a able to pick tasks between the first 10, as we can observe that its gain does not increase with the number of tasks. This might be due to the effect of the adversarial algorithm, which correctly guesses the marginal cost of the dummy agent and correctly bids under it. Securing advantageous cities at the beginning may also help to keep a smaller marginal cost, thus allowing to subsequently being able to afford lower bids.

2.2 Experiment 2: Comparisons with safe agent

2.2.1 Setting

The setting of this experiment has been kept the same as Experiment 1. In this case, our competitor is a $Safe$ agent, an early implementation of our agent which only aims at bidding $max(marginalCost + 1, historyMin)$, with $historyMin$ being the same described for *htd_adv_2*.

2.2.2 Observations

As we can see from Table 2, the *htd_adv_2* agent proves to be the best one once again, easily winning all of the tournaments. In this case, the average gain for both agents is much lower, as it is driven down by the $safe$'s strategy: using its adversarial policy, combined with heuristics, *htd_adv_2* will try to make sure to pick tasks, and therefore often bid low amounts by averaging the computed $advMarginalCost$ with the $historyMin$. This will then trigger the safe's algorithm, which will adapt and lower its bid again when possible. In this case, getting a higher gain when the marginal cost allows it, as well as accepting a loss to visit favorable cities, is what leads the *htd_adv_2* to succeed.