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

Group №16: Léopold Bouraux, Vincent Rinaldi

November 24, 2019

1 Bidding strategy

Our agent for this assignment reuses the Stochastic Local Search (*SLS*) algorithm we implemented in the *Centralized* Project part. In addition, we though about and added a bidding strategy on top of our last assignment implementation. Our *SLS* algorithm for which all methods used are mainly located in the *CSP* class remains almost exactly the same, with the only difference that we don't remember anymore all non optimal neighbors solution to remove them. This is due to the very few tasks we have at the beginning of an auction plan.

We added other methods related with the auction rounds and bids, in order to update the tasks set of our *Constraint Satisfaction Problem* model, or add a task to the most convenient vehicle for cost optimization, or also computing the total estimated gain according the expected task weight on different specific paths the vehicle takes and the estimated remaining bidding round. The bidding strategy is based on the fact that we want to win the most tasks as possible during the very first rounds, even if we don't make profit at this time, to lower as much as possible the marginal cost subsequently. When the vehicle is travelling around the map to pickup or deliver some packets, usually its path overlaps with other packets journey, meaning that the cost of the part of this other packet path delivery is already paid, giving us the possibility to bid on this task delivery it at a lower price than the cost.

At each bidding round we compute two values : the marginal cost and the marginal estimated gain, and then we calculate the final bid, such that this calculation depends on three different phases the agent can be depending on the number of rounds that already passed, and the number of tasks he already won, or the current gain the agent generated at this moment.

When we receive a task for which we have to bid during an auction round (method *askPrice*), we compute the new optimized plan for every vehicles of the current agent, using the previous list of optimized plan per vehicle, but with the addition of the new task that is proposed for bidding during this round. We then get the marginal cost by computing the difference between the total cost of the new optimized list of plans and the previous list of optimized plans (without the new added task). Actually, this marginal cost does not take into account future tasks than can be located on the same path as the one to reach the task we are actually bidding for, since that, if it is the case, we would be able to deliver those future task without increasing the vehicle cost, except if we reach the capacity limit of the vehicle.

We then introduced the expected maximum gain E[*MaximumGain*] for future tasks handling. The expectation measures how much more gains could be made if all our vehicles were full at each move (such that each move m is the path from a city to a pickup or delivery city depending if the given task has to be picked up or to be delivered), through the formula :

$$\sum_{v \in vehicles} \sum_{t \in tasks} \sum_{m \in moves} distance(m) * costPerKm(v) * bundleFactor$$

where

$bundleFactor = min(1, max(0, (E[totalWeightCarriedOnMove(v,m)] - totalWeightCarried(v))/capacity(v)$

and

$E[totalWeightCarriedOnMove(v,m)] = min(capacity(v), E[taskWeightOnMove[m]] * E(remainingRounds)$

such that *E(remainingRounds)* is represented by the function *estimatedRemainingRound(roundNumber)* which computes the expected remaining rounds given the current round we actually are into (calculating the expected number of remaining rounds is like calculating the expected number of remaining tasks, since that we don't know at the beginning how much tasks there will be in total). We define

$$estimatedRemainingRound(roundNumber)) = 15e^{-\frac{\sqrt{x}}{2}}$$

. We note that *totalWeightCarried(v)* is the current total task weight the vehicle is carrying at this moment, and is increased (respectively decreased) by the given picked up (respectively delivered) task weight after each *task* iteration.

Since E[totalWeightCarriedOnMove(v,m)] takes into account the probability distribution of tasks and their expected weights from *TaskDistribution* given by the platform, but also the guessed E[remainingTasks], it helps us to record the potential future apparitions of tasks that can be placed at interesting locations.

We then obtain the marginal expected gain at a given round by simply subtracting the expected maximum gain obtained from the new list of optimized plans for each vehicle of the agent, by the one obtain with the previous list of optimized plans. If this quantity is large, then taking the task will increase the expected generated profit with the future tasks. If it is small or negative then the current task we have to bid for would most probably not gives any benefits following the tasks proposed in the next rounds, and we would not focus ourselves too much on it since it is less valuable to us.

Now that we have computed our marginal expected gain and marginal cost, we can calculate our bid for the given task following three different cases. We don't want the agent to make a bid equal to 0, so we always, in each case, return at least a lower bound that is difficult to beat by other opponents to prevent the case when the marginal cost is equal to 0. This lower bound is a fraction of the path cost computed with the function *miniPathCostNormalised* where with simply take the vehicle with the smallest costPerKm attribute, multiply it by the path length of the task, and divide the result by the capacity attribute of this vehicle.

As a first case, for the very first rounds, since we want to win the most tasks as possible, we bid at prices that would lead us to some deficit. This decit will depends on how much the task we won will be useful and lead use to high profit with future tasks. If the task has high utility, our bid will be equal to *marginalCost - marginalExpectedGain * α* (the bid is lowered by a fixed ratio to increase our chance to win the task).

As a second case, when the sum between the number of tasks we won and the number of rounds that has passed is larger than a fixed value *deficitRounds*, we now aim to recover from our decits that were generated in the first case. We want to always be profitable after a fixed number of rounds, defined by the value *deadlineToProfit*. The formula for the bid in this case then reuses our current decit (that is represented by the value *totalGains* in our code), and is *marginalCost + totalGains/max(1, deadlineToProfit roundNumber))*

The third and last case happens when the agent recovered from its losses and became protable again. This case is similar as the first one since the bids are adjusted following how much the auctioned task will bebeneficial for future tasks. The only difference is that now, those bid will not cause losses. The formula is *marginalCost + max(1, 1 marginalExpectedGain$\cdot\beta$)* (here, the bid is also lowered by a fixed ratio to increase our chance to win the task).

2 Results

2.1 Experiment 1: Comparisons with dummy agents

2.1.1 Setting

We first used 20 tasks along with the seed 123456, and the location of default vehicles in the England.xml topology. For this experiment we use our agent with the "aggressive" technique such that we take as much as possible of the task at the beginning and then have a lower marginal cost for the future tasks. We have kept here an acceptance probability of 0.5 for the choose of new solutions for the agent's vehicles in the *localChoice* function. Concerning the parameters of our bidding strategy, α (for the first case) is set to 0.1, β (for the third case) is set to 0.2, *deadlineToProfit* and *deficitRounds* (for the second case) are respectively set to 14 and 8. As a reminder, as seen in the *CentralizedAgent* project Observations, with a higher probability of choosing the new optimized plans for our vehicles, the algorithm tends to converge more quickly, and can be easily trapped into a local minima.

2.1.2 Observations

There are 4 agents here but we only want to focus ourselves on the 2 most important : our main agent named BourauxRinaldiAuctionMain16 and the dummy agent from the template named BourauxRinaldiAuctionRandom.

```
### auction.xml ###
          Agents            |      Win - Draw - Lose      | BourauxRinaldiAuctionMain16 |          YourName
BourauxRinaldiAuctionMain16 |          5 - 0 - 1          |              -              |       WIN (3254 : 0)
         YourName           |          5 - 0 - 1          |       WIN (3085 : 0)        |               -
BourauxRinaldiAuctionRandom |          1 - 0 - 5          |      LOSE (15 : 1069)       |       LOSE (15 : 952)
         YourName2          |          1 - 0 - 5          |       LOSE (15 : 850)       |       LOSE (15 : 982)
          Agents            |      Win - Draw - Lose      | BourauxRinaldiAuctionRandom |          YourName2
BourauxRinaldiAuctionMain16 |          5 - 0 - 1          |      WIN (4505 : 132)       |       WIN (2974 : 0)
         YourName           |          5 - 0 - 1          |       WIN (5167 : 0)        |       WIN (3237 : 0)
BourauxRinaldiAuctionRandom |          1 - 0 - 5          |              -              |      WIN (5413 : 2883)
         YourName2          |          1 - 0 - 5          |      WIN (5413 : 2883)      |               -
```

Tournament: intelligent agent vs naive agent

We notice, according to those results, that our main agent won the majority of his confrontations (by winning 5 out of 6 battles according to the Figure), but what is important is that it mostly won against the naive agent, which proves that we actually got improvements. If we look closely on the value obtained, we can see that our main agent won against the dummy one after generating a profit of 4505 against 132 which is a huge victory. The dummy agent recorded a loss against our main agent by having only generated a profit of 15 against 1069 for our agent which is again a huge victory by our main agent.