PeakPact: An agent submitted to the ANAC 2023 SCM league

Barış Arat, Alara Baysal, Ahmet Çağatay Savaşlı

April 14, 2024

1 Core Strategy

We detected four major problem points in the strategy of our base agent, EqualDist. Our development process was to introduce new strategies and features addressing these shortfalls to get a higher score in the competition.

In this scope, the main problem points are addressed:

1. Rejecting or accepting all offers at once without the flexibility to select good offers in between
2. Having an acceptance threshold that causes large shortfall penalties
3. Sending an equally distributed quantity offer without considering the partner's history
4. Having only the option to send a random price or the best price for us without considering the partner's history

The high-level solutions we implemented for these problems are as follows:

1. If the agent can't fulfill its needs at the given step, selects the best offers and makes counter-offers to the rest. Therefore, it keeps the good offers and has progress at each step of the negotiation.
2. Keep a tight acceptance threshold and aggressively target no-penalty goals.
3. Sending a proportional response to the offers in quantity by considering the agent's last offer from its partners.
4. Sending a higher price counter-offer while keeping the last price it got from its partners. The result is a balance between cooperation and price maximization.

The rest of the section explains the implementation of these solutions in detail.

1.1 Threshold of Acceptance and Progressive Selection

Our agent's goal is to minimize the sum of the shortfall penalty and disposal cost. To meet that goal, we started by setting an ambitious sub-goal for our agent, aiming for almost zero penalty until it gets the 60th percent of the steps during the negotiation. Then, it voluntarily accepts a certain penalty level by accepting a mismatch threshold between the needed and the agreed quantity. This threshold value increases from 0.6 relative time until the deadline in an exponential form.

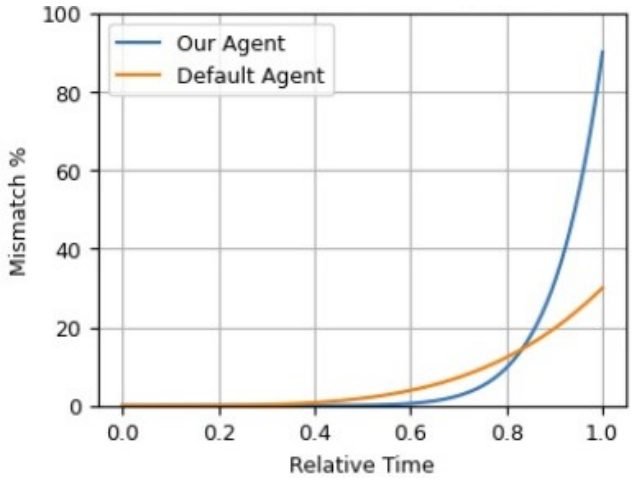

Figure 1: Allowed Mismatch % over time

Our initial threshold function concedes later but faster than the EqualDist agent's threshold function. The initially tighter model provided us with an improvement over the default model. However, we tested further increasing the starting point of the increase in the threshold and found that a value closer to zero from the beginning to the end works best in our case. Therefore, our final model uses a fixed threshold value of 1 unit difference from the total need.

We should look at our agent's decision mechanism more closely to understand why this is the case. This threshold affects the decision to accept all offers at once only. The agent counts all the quantities offered to it and compares that to the total quantity it needs to fulfill. The threshold value sets the maximum difference between the need and the offered quantity our agent can accept. This difference will result in a shortfall or a disposal penalty.

Naturally, we would like to keep this difference as low as possible. However, not accepting a slight difference, therefore a penalty, can cause getting a larger penalty at the later steps if a better opportunity doesn't arise.

Keeping the one-unit fixed threshold worked better for us because of our progressive offer selection mechanism. When we don't accept all offers simultaneously, we select the best offers (we'll explain in the next section what we

mean by best) and send counter-offers to the rest of the partners. Therefore, at each step, we partially fulfill our needs and lower our risk of penalty. On the contrary, the EqualDist agent takes a larger risk by not accepting the given offers since it rejects all of them at once. This mechanism requires an allowed mismatch to be accepted for EqualDist to lower this risk.

In summary, this threshold was critical for the base agent because it accepted or rejected all offers simultaneously. In our case, we introduced a step-by-step progressive offer selection mechanism and an accept-all option. As a result, we decreased the occurrence of high penalties and increased our overall score in the competition.

1.2 Stepwise Best Offer Acceptance Mechanism

The default agent EqualDist's behavior is to accept all offers at once if they meet the target quantities or reject all of them until it gets the desired quantities in one step. In the case of rejection, this agent divides its total quantity needs into equal pieces and sends them as counteroffers to its partners (we'll discuss the pricing strategy later). This behavior has two significant drawbacks.

Firstly, it increases the risk of a higher loss by giving other agents more chances to make agreements with a broader choice in the market since this agent doesn't block any quantity in the market by rejecting all offers.

Secondly, it makes non-optimized offers due to its equal distribution mechanism. The mechanism gives all partners the same amount of counteroffer regardless of their prior offers. As a result, some partners get counteroffers that are lower than their initial offer from us in terms of quantity. This offer can motivate them to give back lower quantity counteroffers, or they can accept these lower quantity offers. As a result, this agent signs a lower quantity agreement with a partner that offered it a higher quantity in the first place.

To solve these problems, we developed a simple acceptance mechanism:

1. We assume that the agent will act like our base agent, rejecting all offers and sending them equal-sized offers to meet our quantity goal. As a result, the agent gets a hypothetical offer to send to the partners. We can call this amount the expected quantity.

2. The agent compares the offers from our partners and accepts them if they are above the expected quantity.

3. The agent sends counteroffers to the rest of the partners.

After accepting the best offers, the counteroffers are based on the remaining quantity agent needs. We explain the details of the distribution of these needs in the next section.

1.3 Proportional Distribution In Counter Offers

EqualDist agent distributes its total quantity needs equally over its partners. This approach is blind to the preferences of the partners. As an improved version

of this method, our agent considers the last offer it got from its partners.

The counter-offer quantity is based on need; the last partner offers, and the accepted amount at this step.

First is the remaining need. The remaining need is simply calculated by the total needs deducted from the amount our agent will accept at this step. The accepted amount is calculated based on the expected value above. Simply put, we count all offers to be accepted if they exceed the expected quantity.

Then, we find the shortage we have after accepting the top offers and considering the remaining offers. The shortage is the amount our agent will ask from its partners in addition to what's been offered to it in the last step. As a result, we increased our offer by an equally distributed shortage. Finally, we increase the offer even more by multiplying this amount with a constant value based on the market condition. This over-ordering multiplier will be explained in the next section.

1.4 Equal Distribution In First Offers

In the first offer, when we're the initiator of the negotiation, we don't make any assumptions about the partners and distribute all needs equally among the agents. In this case, we are experimenting with distributing randomly, too. However, this approach is riskier due to the rare high and low occurrences in the random distribution, and the experimental results didn't show any improvements.

1.5 Over Ordering In Counter Offers

The total extra amount we'll ask from the partners depends on our total shortage and the over_order multiplier. The multiplier is a conditional set from 0% to 20% based on the number of available partners left after our agent accepts the top offers. The over-ordering amount increases as the number of available partners increases because more partners mean a lower probability of acceptance by all partners at the same time.

$$f(n\_partners\_remaining) = \begin{cases} 0 & \text{if } n\_partners\_remaining < 5 \\ 0.05 & \text{if } 5 < n\_partners\_remaining < 10 \\ 0.15 & \text{if } 10 < n\_partners\_remaining < 15 \\ 0.2 & \text{if } 15 < n\_partners\_remaining \end{cases}$$

1.6 Over Ordering In Initial Offers

In addition to the strategy to set the quantities for counter offers, we also have a distinct method to set the initial quantities when we are the negotiation initiator. In this case, we divide the quantity offered equally over the partners; however, instead of sending as much as we need, we offer 50% more than we need.

We hypothesized that we needed to offer more than we needed, especially at the beginning of the negotiation, because many partners would be available, and we could get many rejections. Also, considering the other strategies that utilize selecting higher quantity offers early on, we are more likely to be accepted by these agents by offering more.

Our early experimentation in the local environment showed that an increase in the initial quantity offered by up to 50% had a visible positive effect on the score. However, beyond this point, the results started to have higher penalties, and we kept the initial increase at 50%.

1.7 Setting Price In Counter Offers

Our primary goal in this tournament is to match the quantity we buy and sell. However, optimizing price still provides an improvement in the overall score. Our pricing strategy is simple and based on experimentation in our local environment.

We have two distinct strategies for the price. The first is for the cases in which we are the negotiation initiators. In this case, we will offer the best price possible. The motivation is to exploit the agents, not considering price as a significant decision criterion.

However, assuming that no agent uses price as a decision factor is dangerous. Therefore, after the first step, we switched rules and made a more balanced price offer. For cases, the price offer is calculated using a simple average of the last offer from the partner and the best possible price we can get.

As a result, in every counter-offer, we ask for a better price for us. Experimentally, we found that this approach works better than always asking for the best price, which can be explained by the partners that prefer something other than our high-price strategy. The other alternative we considered was to give the same price we got back from the partner. Although this seemed like a better idea to us initially, considering its positive effect on getting more acceptance, the experimental result of the balanced average price was slightly better than giving the same price-back strategy.

The lack of interest in price optimization in the current Oneshot setting can also explain these results. Since agents are more interested in the quantity than the price, small price increases can lead to slightly higher scores because the partners mostly ignore the price change.

1.8 Summary of Algorithm

Algorithm 1: Counter_All()

Input :
Offers: all offers we got at this step containing Price and Quantity information
Output: Responses

$Partners \leftarrow$ Get id of all partners giving the offer;
$BestPartners \leftarrow$ Get subset of Partners that gives the highest utility;
$Need \leftarrow$ Get the total number of products we need to sell or buy based on our position;
$Threshold \leftarrow 1$;
$BestDifference \leftarrow |Need - \text{GetTotalQuantity}(BestPartners)|$;
if $BestDifference < Threshold$ **then**
    foreach $Partner$ ***in*** $Partners$ **do**
        if $Partner$ ***in*** $BestPartners$ **then**
            ACCEPT_OFFER();
        else
            END_NEGOTIATION();
        end
    end
end
$N\_partners \leftarrow \text{len}(BestPartners)$;
$NeedPerPartners \leftarrow Need/N\_partners$;
$OffersToAccept \leftarrow []$;
foreach $Partner$ ***in*** $BestPartners$ **do**
    if $GetQuantitiy(Partner) > NeedPerPartners$ **then**
        ADD($OffersToAccept$);
    end
end
$Shortage \leftarrow$ GetRemainingNeedsAfterAccepting() − GetTotalQuantityWillBeRecejted();
$DistributedShortage \leftarrow Shortage/$GetNPartnersAfterAccepting();
$Over \leftarrow$ GetOverOrderMultipler();
foreach $Partner$ ***in*** $BestPartners$ **do**
    if $Partner$ ***in*** $OffersToAccept$ **then**
        ACCEPT_OFFER();
    end
    else
        $Quantity \leftarrow (\text{GetLastOfferQuantitiy}(Partner) + DistributedShortage) \times (1 + Over)$;
        $Price \leftarrow (\text{GetLastOfferPrice}(Partner) + 2 \times \text{BestPriceForUs})/3$;
        COUNTER_OFFER($Quantity, Price$);
    end
end