DistRedistAgent for ANAC 2024 SCML OneShot Track

Hajime Endo
Tokyo University of Agriculture and Technology
endo@katfuji.lab.tuat.ac.jp
Japan

April 16, 2024

1 Introduction

In the ANAC 2024 SCML OneShot track, Official documentation[1] says that the range of unit prices for trading products has become very narrow, placing greater emphasis on adjusting the quantities traded and this is a different point from the usual rules. Therefore, the DistRedistAgent (Distribution and Redistribution Agent) developed this time aims to distribute its own required contract quantities among the offers submitted to each trading partner in an efficient and easily acceptable method. Additionally, it implements functionality to redistribute any remaining quantities that cannot be accepted to other agents once.

This report will explain the specific strategies. Furthermore, the DistRedistAgent created this time is based on the OneShotSyncAgent in the SCML 2024 OneShot tutorial, but it has been significantly expanded upon in its development.

2 Strategy

The DistRedistAgent performs two main actions. The first one is the First Proposal. This is a proposal action conducted after each step begins, where the agent proposes the initial offer to its partner agents. The second action is Counter All. This is a response action where the agent decides whether to accept the proposal (ACCEPT OFFER), reject it and make another proposal (REJECT OFFER), or terminate negotiations without agreement (END NEGOTIATION) based on proposals made by the opponent.

Both of these actions involve the proposal of offers. In proposing offers, it is necessary to distribute contract quantities to ensure that the agent's required contract quantities are met, and there are several distribution strategies for this purpose. Furthermore, some distribution strategies may utilize information about other agents accumulated during past negotiations.

In this chapter, we first explain how past information about other agents is collected, followed by an explanation of distribution strategies and strategies for First Proposal and Counter All.

2.1 How to accumulate Information

We will explain the method of collecting information about other agents involved in past negotiations. This agent collects the following four types of information from all agents involved in negotiations in the past.

- Info[0] : Number of all the successful contracts with the agent
- Info[1] : total quantity contracted
- Info[2] : Number of all of unsuccessful contracts with the agent

Info[0] and Info[1] are accumulated each time the on-negotiation-success() method is executed, while Info[2] is accumulated each time the on-negotiation-failure() method is executed.

2.2 Distribute Strategy

The DistRedistAgent has three different distribution strategies, and selects among them depending on the situation.

2.2.1 Random Distribution

The first strategy is a random distribution strategy. This distribution method is implemented concerning the distribute function in the tutorial[2]. In this distribution method, 1.35 times the required contract quantity is allocated to the negotiating partner agents randomly. Although the allocation is random, the specification ensures that no agent is allocated a quantity of 0. Also, this method offers quantities exceeding the required amount, but this is based on my empirical observation that the likelihood of all of my proposals being accepted is low, so I judged there is no problem. Additionally, there is an advantage in distributing slightly more quantities in advance, it allows for securing a certain amount of contracts even when some offers are rejected. This method is implemented in distribute_needs_randomly().

2.2.2 Distribution by Information

The second strategy utilizes information accumulated from past negotiations with partner agents. As explained in Chapter 2.1, the accumulated information includes Info[0] to Info[2]. Using this information, the agent calculates which partner agents are more likely to accept its proposals.

The specific method involves calculating a "priority" for each negotiating partner agent using the following formula and allocating a larger quantity to agents with higher priority values.

"Priority" is expressed by the following expression. As mentioned earlier, Info[0] [2] accumulates for each agent of the negotiating partners throughout a single simulation.

$$\left(\frac{Info[1]}{Info[0]}\right)\left(\frac{Info[0]}{Info[0]+Info[2]}\right) \tag{1}$$

The above expression allows us to calculate values such as the expected value of the quantity of contracts formed with each agent. This enables us to estimate how likely it is for that agent to form contracts.

For each agent of the negotiating partners, we calculate the "Priority" at that moment, and then calculate the ratio of those values. We then divide the range [0, 1) according to this ratio. For example, if we calculate the Priority for three agents(agentA, agentB, agentC), and the ratio is 5:3:2, the range [0, 1) is divided into three segments: [0, 0.5), [0.5, 0.8), and [0.8, 1.0). Next, we use Python's random.random() function to generate a uniform random number in the range [0, 1) for each desired quantity of contracts to be distributed. We then determine which of the divided ranges the generated random number falls into. Finally, we allocate the corresponding quantity to the agent corresponding to the range in which the random number falls. For example, if the required contract quantity is 3 and the generated random numbers are 0.2, 0.4, and 0.65, then 2 units would be allocated to agentA, 1 unit to agentB, and 0 units to agentC.

Finally, we check if there are any agents for whom no quantity has been allocated. If there are such agents, we forcefully allocate quantity 1 to them. This is done to ensure that as many negotiating partner agents as possible remain in the negotiation process. While this may result in an excess allocation of quantities compared to the required amount, I think there is no problem since it is extremely rare for all of the offers to be accepted. This method is implemented in distribute_by_info().

2.2.3 Distribution by Previous Offer

In addition to using a custom-calculated "Priority" for distribution, another method distributes quantities based on the ratio of the quantities offered by that agent in their previous offer. This distribution method is figured out from my assumption that agents who have offered more quantities are more likely to accept offers with more quantities. The distribution method is the same as introduced in Chapter 2.2.2 and it uses the random function. Furthermore, here as well, distribution is carried out to ensure that each agent receives a quantity of 1 or more. This method is implemented in distribute_by_offers().

2.3 First Proposal

In the First Proposal, we utilize two distribution strategies to create offers as follows.

$$\begin{cases} \text{Call distribute\_needs\_randomly()} & \text{if current step} < 10 \text{ or random} < 0.07, \\ \text{Call distribute\_by\_info()} & \text{otherwise} \end{cases} \tag{2}$$

Also, In this proposal, the best price will be used for prices of all the offers.

2.4 Response Strategy(Counter All)

Here, we will explain the strategy for accepting offers from the opponent. First, we generate the power set consisting of offers received from the opponent agent. For example, if offers A, B, and C are received, the power set would be as follows: $\{\emptyset, \{A\}, \{B\}, \{C\}, \{A, B\}, \{B, C\}, \{C, A\}, \{A, B, C\}\}$. Next, agents calculate the utility function values for each of these power set elements when all offers they contain are accepted. Then, agents sort the power set in descending order based on these utility values and get the list made from elements of power set called as "offers_list".

Next, following the code provided below, we will proceed with the acceptance process and redistribute the quantity, and make a new offer according to the following code.

```
need: int
round: int

for i, offer in enumerate(offers_list):
    if i == len(offered_list) - 1 and need != 0:
        if random.random() < 0.07:
            distribute_needs_randomly()
        else:
            distribute_by_offers()

    if needs > offer[QUANTITY]:
        if i == len(offered_list) - 1:
            if random.random() < 0.07:
                distribution = distribute_needs_randomly()
            else:
                distribution = distribute_by_offers()

            accept(offer)
            reject_and_offer(distribution)

    elif needs < offer[QUANTITY]:
        if round <= 16:
            continue
        else:
            if offer[QUANTITY] - needs <= 2:
                accept(offer)
                end_negotiation()
            else:
                continue

    elif needs == offer[QUANTITY]:
        accept(offer)
        end_negotiation()
```

"need" refers to the required contract quantity, and "round" indicates the current turn number of the negotiation. Additionally, "accept(offer)" means the acceptance of the offer, "reject_and_offer(distribution)" means the redistribution of the remaining quantity to the remaining opponent agents according to the distribution result, and making a new offer, while "end_negotiation()" means rejecting the remaining offers and finish the negotiation with that agent.

	agent_type	score
0	DistRedistAgent	1.09072
1	SyncRandomOneShotAgent	1.06112
2	BetterSyncAgent	1.05671
3	RandomOneShotAgent	0.81646

Figure 1: experimental result

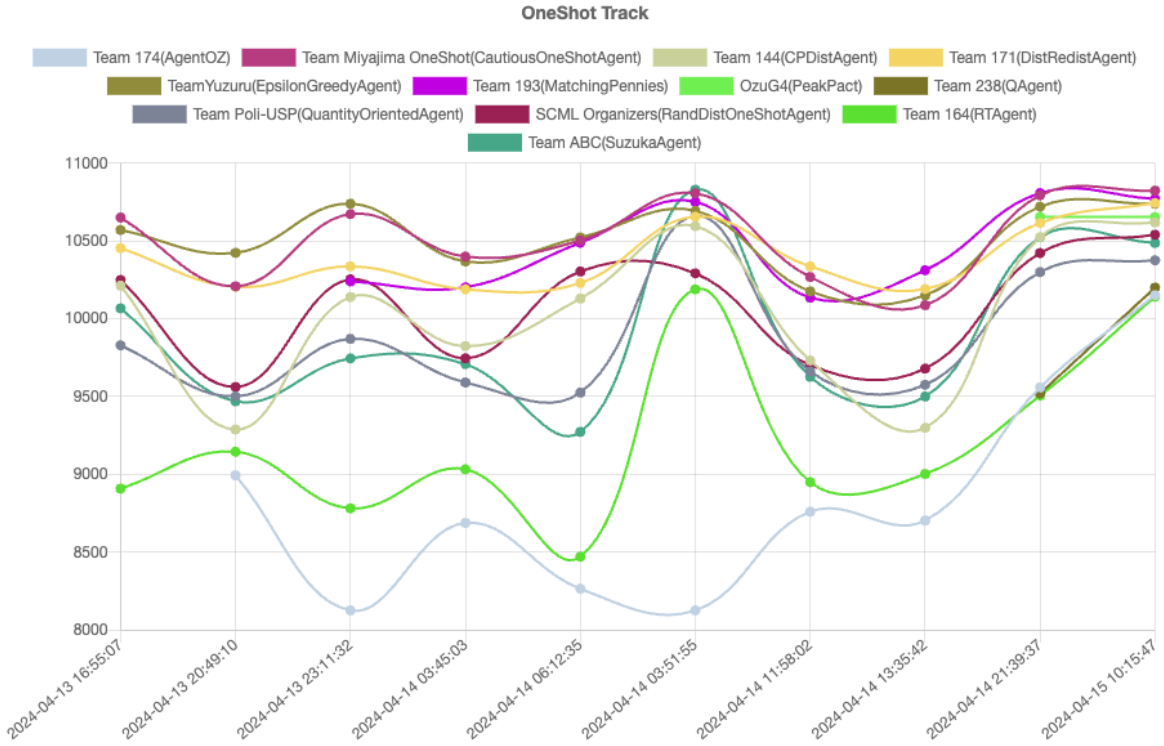

Figure 2: result of live competition

These processes enable the agent to select the best offer while ensuring that the contracted quantity does not exceed the required amount from the opening of the negotiation. As the negotiation progresses, concessions are made even if the quantity is slightly excessive and it is accepted if it leads to an increase in utility. If not, the agent redistributes the necessary contract quantities and proposes a new offer.

3 Evaluation

Here are the experimental results of competing the DistRedistAgent created this time with basic sample agents, shown in Figure 1 below. Experiments were conducted with a step count of 100 and a configuration of 10. From the experimental results, it is evident that DistRedistAgent got better scores than other basic implementation sample agents (RandomOneShotAgent, SyncRandomOneShotAgent, BetterSyncAgent).

Figure 2 below illustrates the results obtained upon submission of the DistRedistAgent to Live Competition. From the figure, it is evident that the DistRedistAgent has consistently performed well in many tournament runs, often ranking among the top agents. However, due to the influence of factors such as the proposal strategies of other agents and random elements like randomness, the ranking of the DistRedistAgent remains unstable, showing frequent fluctuations. Addressing this instability will likely be a key focus for future improvements.

4 Conclusion

In the ANAC2024, SCML (OneShot) competition, I created and submitted an agent called DistRedistAgent. This agent employs a strategy of distributing the necessary contract quantities to multiple opponent agents based on past information and random elements, and then proposes offers. Additionally, during acceptance, it selects the best offer combination from all offers made by the opponent and, to minimize unnecessary costs, may redistribute quantities and propose offers again. Experimental results have confirmed that this agent can achieve higher profits compared to basic sample agents. Moreover, it has demonstrated the ability to perform well in Live Competitions, consistently ranking among the top competitors. Finally, I extend my heartfelt gratitude to all those involved in organizing the competition and to all the participants who competed.

References

[1] SCML Organizing Committee. Supply chain management league (oneshot). 2023.

[2] Yasser Mohammad. scml tutorials, 2024. Accessed: April 15, 2024.