ANAC 2024 SCML Automated Negotiating Coyote Agent

Mehmet Altuğ Karataş
Computer Science
Ozyegin University
Istanbul, Turkiye
altug.karatas@ozu.edu.tr

Metin Arda Koker
Computer Science
Ozyegin University
Istanbul, Turkiye
metin.koker@ozu.edu.tr

Ahmet Utku Erşahin
Computer Science
Ozyegin University
Istanbul, Turkiye
utku.ersahin@ozu.edu.tr

***Abstract*—In this paper, a basic explanation is given about agents which were used for the ANAC 2024 SCML competition. We have designed an agent for automating the negotiation. It is called Coyote Agent. It is making offers depending on the opponent agent. The mathematical explanation and pseudo-code can be found. Our outputs and comparison with several agents that were created in SCML libraries are shown. Coyote agent can give great results under correct circumstances and of course there are some circumstances which is not able to give great results.**

***Index Terms*—Agent, SCML, ANAC, Negotiation**

I. Introduction

Our main goal in relation to the Supply Chain Management League was to improve our negotiating tactics. Our attention was focused on creating an algorithm that made use of the financial resources that agents had access to in order to accomplish this goal. Having developed a sophisticated mathematical framework, our methodology recognized the strategic benefit that comes with having excess cash.

Our strategy's main component was the introduction of a dynamic pricing mechanism. Our algorithm was designed to take a proactive approach to negotiations by setting greater initial demands. This proactive strategy was carefully calibrated to take advantage of the psychological dynamics of negotiation, especially in the early going when agents tend to set pricing benchmarks.

But we also understood how crucial flexibility is to the dynamics of negotiation, particularly in times of high stress or tension. We used a parameterized approach that allowed for pricing strategy flexibility in order to address this. Two crucial parameters had to be introduced in order to accomplish this: the "stubbornness point" and the "fear point." These parameters were carefully adjusted to react to changes in the dynamics of the negotiation over time, guaranteeing a flexible and subtle approach to pricing.

Moreover, we added more parameters to our algorithm to maximize offer generation and price acceptance, taking inspiration from the SyncRandomStdAgent model. Our objective in incorporating these variables was to achieve a nuanced equilibrium between assertiveness and adaptability, thereby augmenting the effectiveness of our algorithm in maneuvering through the intricacies of negotiation situations.

In conclusion, we developed our negotiation strategies within the Supply Chain Management League using a nuanced combination of strategic foresight, adaptability, and aggressiveness. Our algorithm aims to maximize negotiation outcomes and promote value generation for participating agents by carefully calibrating parameters and integrating important aspects.

II. Literature Review

When examining reward-based negotiating agent strategies as outlined in the article "Reward-Based Negotiating Agent Strategies," [1] we opted to establish a relationship between our balance and that of our opponent as an approach. Our approach dictated how early we would reduce prices. Rather than employing reinforcement learning, we endeavored to reach a similar outcome using constants and equations.

III. Strategy design

A. Description

Our agent's name is coyote agent, it is taking actions depending on the opponents' attributes. Trying to create an opportunity based on opponents' weak points. Basically, if the opponent has a shortage of products, then the coyote agent will try to sell the product more expensively. And also in the buying phase, the agent checks the seller's attributes. If the seller has plenty of products then the coyote agent will try to buy the product more cheaply.

B. Mathematical Equations

$$U_{Coyote} = \begin{cases} U_{RandAgent} * 0.3, & \text{if } t \leq 0.2 - \frac{My}{My+Opponent} \\ U_{RandAgent} * 1.5, & \text{if } t \geq 0.2 + \frac{Opponent}{My+Opponent} \\ U_{RandAgent}, & \text{else} \end{cases}$$

```
1: function GOOD2BUY($p, t, mn, mx$, today)
2:     return $p - 0.0001 \leq$ buy_price($t, mn, mx$, today)/ multiplier $\times 2$
3: end function
4: function GOOD2SELL($p, t, mn, mx$, today)
```

```
5:      return $p + 0.0001 \geq$ sell_price($t, mn, mx$, today) $\times$ multiplier $\times 0.5$
6:  end function
7:  function GOOD_PRICE(nmi, today)
8:      $initial\_cash \leftarrow 15000$
9:      $state \leftarrow nmi.state$
10:     $my\_cash \leftarrow$ awi.current_balance
11:     $other\_cash \leftarrow 0$
12:     if other_report is not None then
13:         $other\_cash \leftarrow other\_report.cash$
14:     end if
15:     if $state.relative\_time > (0.2 * my\_cash/initial\_cash)$ then
16:         multiplier $\leftarrow 0.3$
17:     else if $state.relative\_time < 0.2 * (other\_cash/initial\_cash)$ then
18:         multiplier $\leftarrow 1.5$
19:     end if
20:     $mn \leftarrow nmi.issues[UNIT\_PRICE].min\_value$
21:     $mx \leftarrow nmi.issues[UNIT\_PRICE].max\_value$
22:     if is_supplier(partner_id) then
23:         return buy_price($relative\_time$, $mn, mx$, today)/multiplier
24:     else
25:         return sell_price(get_nmi($relative\_time$, $mn, mx$, today $\times$ multiplier)
26:     end if
27: end function
```

IV. Evaluation

A. Comparison Metrics

There are several agents given to us. These are RandomOneShotAgent, SimpleAgent, BetterAgent, AdaptiveAgent. Simpleagent is extending from OneShotAgent, and other agents are extending SimpleAgent. Each agent adds a couple of things to the previous one. BetterAgents creates a max and min value as an addition to SimpleAgent and creates a threshold. AdaptiveAgent changes its min and max values in every offering stage. There are so many different ways of implementing the max and min values, but these are the most frequently used in graphs and tables.

In the SCML library, there are predefined agents. The agents that we compared with the coyote agent are EqualDistOneShotAgent, SyncRandomOneShotAgent,and GreedySyncAgent. In every test, the output score can be changed regarding the seed of the run.

In general coyote agents can achieve better scores against some agents and cannot against some agents. In the next figure 1 there are 6 different graphs can be seen. The first one is for the. In the next one, which is the storage cost, our agent has a really high storage cost in general which has a negative effect on the score. In the third one, because of storing too many items, the penalty of storage is high. In the score graph at a point the coyote agent becomes better than the greedy agent. In general, coyote agents store too many items and sell them in cases of getting profit.

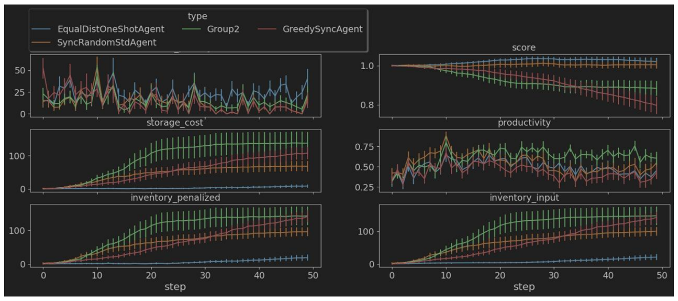

Fig. 1. SCML world run with multiple agent outputs

In the process of developing the coyote agent, the storage part is not improved well. That's why it is losing too much point in that area. However, the buying phase has improved. So, coyote agents can beat some agents in some cases.

References

[1] R. Higa, K. Fujita, T. Takahashi, T. Shimizu, and S. Nakadai, "Reward-Based Negotiating agent Strategies," Proceedings of the ... AAAI Conference on Artificial Intelligence, vol. 37, no. 10, pp. 11569–11577, Jun. 2023, doi: 10.1609/aaai.v37i10.26367.

[2] Y. Mohammad, S. Nakadai, and A. Greenwald, "NegMAS: a platform for situated negotiations," in Studies in computational intelligence, 2021, pp. 57–75. doi: 10.1007/978-981-16-0471-3_4.