CostAverseAgent for ANAC2025 SCML Oneshot Agent Strategy

Yuzuru Kitamura
Jun 12, 2025
Tokyo University of Agriculture and Technology
kitamura@katfuji.lab.tuat.ac.jp

Introduction

In SCML 2025, as in the previous year, the impact of transaction prices on scores is much smaller than that of transaction volume. Therefore, a strategy was created to negotiate primarily on transaction volume. The absolute values of disposal cost and shortfall penalty do not necessarily match. Agents adopted a strategy of comparing the sizes of these two values and adjusting the transaction quantity in favor of the smaller one. This aversion to costs is the origin of the agent's name. CostAverseAgent inherits class BetterSyncAgent.

Strategy

Proposal Strategy

Like the BetterSyncAgent, it distributes its current needs to all negotiators. However, unlike BetterSyncAgent, the distribution is not random. The agent equally distributes the quantity demanded q among n counterparties. The quantity allocated to one counterparty will be $\lfloor \frac{q}{n} \rfloor$ or $\lfloor \frac{q}{n} \rfloor + 1$.

Response Strategy

Like BetterSyncAgent, the power set of offers from the counterparties is obtained and the combination that is closest to the quantity demanded is selected from among them. In this agent, the criteria for selecting combinations were changed.

In BetterSyncAgent, among all possible offer combinations, the one with the smallest absolute gap between total offers and the agent' s needs is selected. In contrast, CostAverseAgent uses a threshold-based strategy: it selects the combination whose total offer is as large as possible without exceeding a threshold, and responds "Reject" to all other offerers to encourage the overall offered amount to approach its needs.

The threshold varies with the current round number t. Denoting it by $\mathrm{th}(t)$, it is defined as:

$$\mathrm{th}(t) = \begin{cases} \left\lfloor \frac{1}{a}(t-1) \right\rfloor, & \text{if } \texttt{is\_first\_level} = \texttt{true}, \\ \lfloor a(t-1) \rfloor, & \text{otherwise.} \end{cases}$$

Here, the pacing factor a is defined as:

$$a = 0.5 \times \frac{\text{shortfall\_loss}}{\text{disposal\_loss}}.$$

The term `shortfall_loss` refers to the utility decrease incurred when a shortfall penalty is triggered, while `disposal_loss` refers to the utility decrease due to disposal. By using this value a, when the disposal loss is large, the agent avoids leaving inventory surplus, which makes shortfall loss more likely to occur. Conversely, when the disposal loss is small, the agent holds more inventory to prevent shortfall loss from occurring.

Evaluation

We tested `EpsilonGreedyAgent` in a simulation against *RandomOneShotAgent*, *BetterSyncAgent*, and *EpsilonGreedyAgent* (which I made last year). The condition of the simulation are shown in Table 1.

The results of this simulation are shown in Figure 2.

The results are shown in Table2. This result shows that CostAverseAgent is superior to all other agents, especially the original EpsilonGreedyAgent.

Table 1: The condition of simulations

Condition	Value
n_configs	10
n_steps	100

Table 2: Simulation Results

Index	Agent Type	Score
0	CostAverseAgent	1.10835
1	EpsilonGreedyAgent	1.09947
2	SyncRandomOneShotAgent	1.0735
3	RandomOneShotAgent	0.792878

References

For implementation details, we referred to the official SCML 2024 OneShot agent tutorial:

`https://scml.readthedocs.io/en/latest/tutorials/02.develop_agent_scml2024_oneshot.html`
Accessed: June 12, 2025.