Report of UcOneshotAgent for SCML 2021 competition

By Yuchen Liu

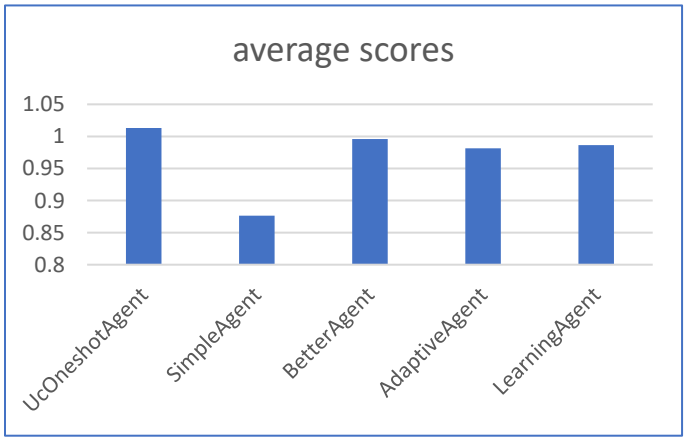
July-28-2021

Introduction

The UcOneshotAgent is developed for the SCML 2021 competition Oneshot track, it was developed for scml.oneshot.SCML2020OneShotWorld, therefore it should not run in other simulators (worlds). This agent is developed by myself under the instruction of professor Rafik Hadfi and professor Ito Takayuki. In this report I will mainly introduce performance, developing process and the strategies used in the agent.

Performance

Since this year is the first year of oneshot competition, there is no agents from past competition available at scml package. However, I could borrow the example agents from scml documentation website, there are four agents in the tutorial section: SimpleAgent, BetterAgent, AdaptiveAgent and LearningAgent, running a 100-day(step) SCML2020OneShotWorld simulation for 300 times with UcOneshotAgent and these four agents, and the average scores show in right diagram.

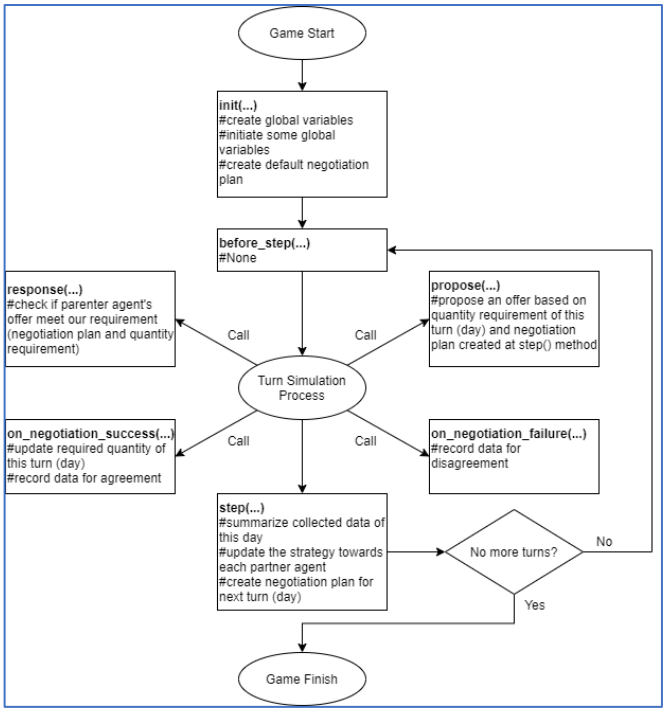


UcOneshotAgent looks just a bit better than other four agents. But as UcOneshotAgent is able to adjust its concession policy for different partners, it performs better in complex situations. According to 10 online tournaments from 27th June to 2nd July, UcOneshotAgent's score is not bad, with the ranking hovered between first and third place.

Structure and working flow

SCML2020OneShotWorld simulates oneshot games by calling agents' different methods at different simulation stage. The figure below (from official oneshot game documentation) shows the execution order of methods in an agent.

Thus, it is essential that UcOneshotAgent's structure and working flow fit the execution order shows above. The figure on the right shows the working flow and each function's duty of UcOneshotAgent. The overall structure of UcOneshotAgent is creating a negotiation plan (negotiation plan means offer price and acceptable price) at the end of each day, and follow the plan when negotiating with partner agents at the next day. At the end of each day, the negotiation plan will be updated based on the negotiation results of this turn. The before_step() function do nothing in UcOneshotAgent, this is because before_step() function was added to OneshotAgent at later version of scml, and it is not necessary to separate step() function into two functions.



Among these functions, step() function is the most important one, it adjust

negotiation strategy at the end of each turn. How to adjust negotiation strategy will significantly influence the performance of our agent.

Developing and optimizing process

The developing process was inspired by the process of reinforcement learning. The initial agent is nothing but a basic concession strategy agent, then different strategies were added to the agent. When a new strategy was added to the agent, I will run competitions between new agents with different coefficient, old agents (agents without new strategy) and agents from scml tutorial to see if the new strategy casts positive effects to the performance, if the new strategy casts positive effects, it will be remained and best score agent's coefficient will be adopted, but if the new strategy casts negative effects, it will be dropped. Adding agents from scml tutorial can provide uncertainty to the simulation process, which makes it close to real competition environment.

Strategies

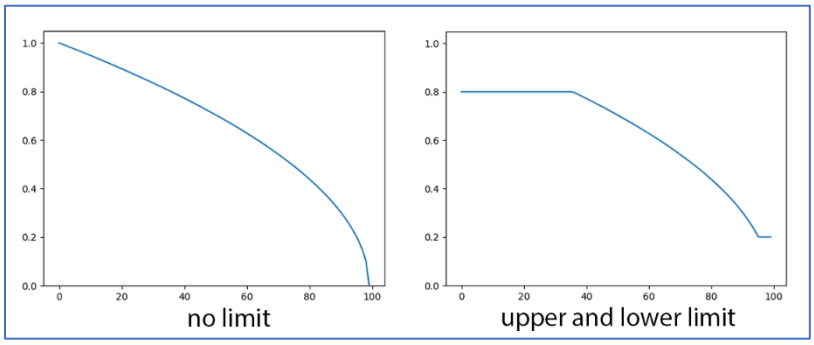
- *Strategy #1: rapid reaction strategy*

Based on the results of world simulation, I found out that an important factor that influences agent's performance is the timeliness. If you can reach an agreement before your competitors (agents in the same production level), your agent is likely to earn more money. Thus, to reduce the reaction time of propose() and response() function, UcOneshotAgent build a negotiation plan at step() function and follow that plan at next simulation day to save reaction time.

- *Strategy #2: concession rate adjusts strategy*

Different partners have different strategy, some partners are easier to compromise, for these partners we should be more aggressive to gain more profit; Other partners are stubborn, and it is our turn to make some compromise. The concession rate adjust strategy adjusts the rate of concession at the end of every step, based on the negotiation results. The basic idea is to be more aggressive if the negotiation success, and more defensive if negotiation failed. Adjust will continue through the whole game, since the strategy of partners towards our agent are also keep changing throughout the game.

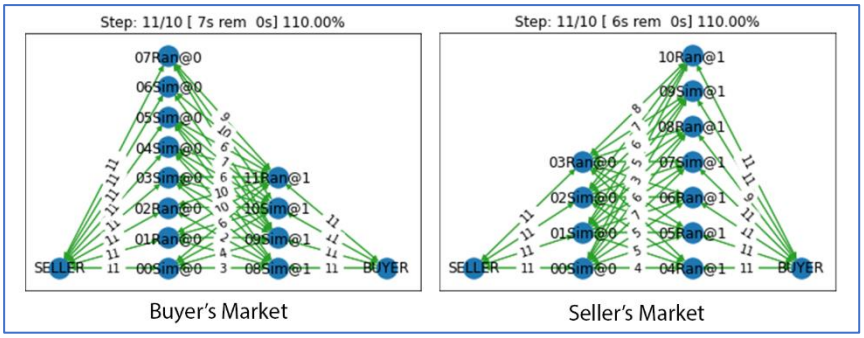
The agent also adjusts the offer's price towards category price, since it is nearly impossible for partner agents to accept an offer that maximize our utility value. And it is also not wise to accept such offer given by partners. Figure on the right shows a normal concession curve and the concession curve after applying upper (0.8) and lower (0.2) limit.



The toughness of each partner is not the only concern, if the number of partner agents is

large, we do not need to concede to tough partners but try to reach agreements with other partners. In short, a connection can be built between number of partners and concession rate.

Another factor we can put into concern is the negotiation results of previous turns. We can adjust our concession rate based on the ratio of agreement reached quantity and exotic quantity of previous turns, the perfect condition is that the ration equals to 1, which means we do not to make further modifications of our concession rate. If the ratio is lower than 1, our agent should reduce concession rate and vice versa.



Finally, I want to mention a global coefficient adjust strategy, at early version of scml package, there are fixed three agents at each level in SCML2020OneShotWorld. But the number of L0 and L1 agents are not fixed anymore at current version, in this case, "buyer's market and seller's market" can be put into concern, which means our agent should be more aggressive if the number of competitors is less than the number of partners, and if the number of partners is less than the number of competitors, our agent should be more defensive.

- *Strategy #3: over quantity strategy*

This strategy is relatively simple, I think most of the agents should already include it. This mechanism's main idea is agent should accept offers that exceed required quantity but still offer a positive utility value (profit is larger than deposit cost or shortfall penalty).

Shortages and future improvement plans

- *Noise of concession rate adjustment strategy*

There are many sub-strategies of concession rate adjusts strategy in previous section. All these sub-strategies make modifications to concession rate. Thus, they will influence each other and produce noise which will cause the overall performance to drop. Thus, it will be helpful if those strategies could combine in a reasonable way, and there are several ways to do so:

- Linear combination:

$$E = k_1 M_1 + k_2 M_2 + k_3 M_3 + \cdots$$

  E: overall effect towards concession rate

  k: coefficients

  M: concession rate adjustment by a strategy's mechanism

- Polynomial (assume there are 2 mechanisms):

$$E = k_1 M_1 + k_2 M_2 + k_3 M_1 M_2 + k_4 M_1^2 + k_5 M_2^2$$

- ANN:
  Construct an ANN network for regression task, with mechanism adjustments (M) as input neurons and overall adjustment (E) as output.

To implement these three methods, we need a huge amount of test runs to adjust coefficients in linear functions and polynomial functions, ANN training process also requires test runs as training materials. Unfortunately, the simulating process is very time-consuming, a 100-day simulation run takes around 150 to 400 seconds to finish with scml v_0.4.5. But scml version 0.4.6 has a huge performance boost, same simulation only takes about 20 to 40 seconds. So, I think I could implement these methods on next year's competition.

- *No sync algorithms*

Another shortage of UcOneshotAgent is it does not contain any sync algorithm other than "required quantity", dynamic price adjustments always play negative effects to the overall performance according to my test results. Take the test run in Performance section as an example, the score of AdaptiveAgent is lower than the score of BetterAgent, where AdaptiveAgent just added sync price adjustment algorithm based on BetterAgent.

But refer to real world, it is important to compare the price of different offers offered by different clients/suppliers. So, the key point is to figure out the reasons that cause score to drop and see if those can be recovered.

Reference

1. Developing an agent for SCML2021 (OneShot)

http://www.yasserm.com/scml/scml2020docs/tutorials/02.develop_agent_scml2020_oneshot.html

2. SCML 2021 League

https://scml.cs.brown.edu/

3. Y. Mohammed, E. Areyan Viqueira, A. Greenwald, K. Fujita, M. Klein, S. Morinaga, S. Nakadai (July 5, 2021) Supply Chain Management League (OneShot)

http://www.yasserm.com/scml/scml2021oneshot.pdf