SteadyMgr: An agent submitted to the ANAC 2020 SCM league

Masahito Okuno[1], Takanobu Otsuka[2]
Nagoya Institute of Technology, Aichi, Japan
[1]okuno.masahito@otsukalab.nitech.ac.jp, [2]otsuka.takanobu@nitech.ac.jp

July 13, 2020

Abstract

Our Agent purchases the maximum number of materials available for production and produces as much as possible. And by actively trading, it is possible to make a lot of profit. It is also possible to make profits in any environment, as the trade price is changed based on the success or failure of the transaction. Experiments showed that our agent got high scores at all times, and outperformed the default agents.

1 Introduction

According to the game description, in SCML2020, the profit (score) is calculated as follows:

$$\text{Profit} = \frac{B_N + \frac{1}{2} I_N - B_0}{B_0}, \tag{1}$$

where B_0 and B_N represent the first and last balance, and I_N represents the value of the product as calculated from the trading price. The important thing here is that the I_N has been halved. Therefore, to make a larger profit, the product needs to be sold as well as manufactured. We focused on this and created an agent that doesn't finish the game with products in their possession.

And, of course, it's also important to buy materials cheaply and sell products at a higher price to make a big profit. However, the optimal price will vary from simulation to simulation, as trading prices are constantly fluctuating. This made it difficult to use the results of past transactions and we found it less desirable to fix the price. Therefore, we have created an agent that can perform well in any simulation by implementing the function to flexibly change the trade price.

2 Agent Strategy

2.1 Agent Configuration

The SteadyMgr class consists of the following four classes:

- MyTradingStrategy
- MyNegotiationManager
- SupplyDrivenProductionStrategy
- SCML2020Agent

This agent was created based on the DecentralizingAgent, but the "Trading Strategy" and "Negotiation Manager" have been replaced with ours. We will discuss MyTradingStratedy in chapters 2.2 and 2.3. Most of MyNegotiation-Manager is the same as StepNegotiationManager, but we have changed the _urange, acceptable_unit_price, and target_quantity.

The _urange used the catalog price to specify a range of prices for negotiations but has been changed to use the trade price, which is updated by the function described below. Also, the acceptable_unit_price was subtracting the production cost from the trade price in the function, but we removed it because it is considered in the function to update the price. Then, target_quantity is capped at twice the number of production lines. As we'll discuss below, our agent tries to gather many materials in a hurry to produce as much product as possible. However, transactions in which quantity too much could be rejected, thus we set a cap.

2.2 Trade and Production Management

One way to make more profit is to sell more products. This requires us to actively purchase materials and manufacture products. However, buying more material than you need will rather reduce your profits. In SCML2020, the number of production lines is predetermined and each of them can produce one product in one step. Therefore, the maximum number of products that can be produced is (number of production lines) $\times$ (number of steps). Also, for each step of the simulation, the maximum number decreases by the number of lines. My agents count the number of materials purchased and do not sign contracts that exceed this maximum.

The agent uses an already-prepared strategy, the SuppleyDrivenProduction-Strategy, as its production strategy. As we mentioned earlier, this agent moves to buy the maximum number of parts to manufacture and sell more products. In this game, it should also be noted that the materials do not count as profit. This means that the production line needs to be running at maximum capacity at all times. This strategy meets the requirements because it is a strategy that seeks to turn all the materials in one's possession into a product.

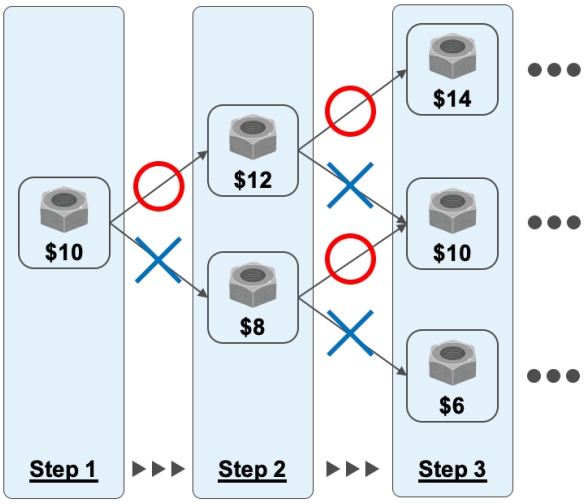

Figure 1: Example of changes in trade price (seller)

In terms of product sales, my agent determines whether it will be able to carry out or not based on the number of products it can manufacture before the delivery date, the number of products in stock, and the number of sales contracts it has already signed, and it only signs reliably viable contracts. There are penalties for breach of contract, and this agent tries to avoid them.

2.3 Trade Price Management

Trade pricing is very important in this game. However, trade prices vary greatly depending on the products you are responsible for and the people you are dealing with, and they change differently in each simulation. Therefore, pre-setting the trade price is not considered to be stable. My agent changes the trade price during the simulation depending on the success or failure of the trade, allowing it to make high profits in any environment.

An example is shown in Figure 1. Here the initial price of the product is set at \$10. If the product sold, the price of the next step is raised to \$12, and if it doesn't sell, the price will be reduced to \$8 (if the agent is a buyer, the ups and downs are reversing). This will be repeated in the next step and beyond. Varying the trading price in this way and my agent can always make a larger profit. In practice, the initial price is calculated from catalog prices and production costs. The increase or decrease is set at 10 percent of the current price and limited to varying between the initial price and twice the catalog price when selling, and between half the catalog price and the initial price when buying.

3 Experiments

To evaluate the agent's performance, we experimented with the run() function present in the template. The parameters are as follows:

- competition=std,
- reveal_names=True,
- n_steps=50,
- n_configs=2,
- max_n_worlds_per_config=None,
- n_runs_per_world=1,

and DecentralizingAgent and BuyCheapSellExpensiveAgent were specified as competitors. The scores of each agent for the five experiments and their means are shown in Table 1. This table shows that my agent has the best score all five times. The flexibility to change the trading price allows for high performance in different simulations. Also, DecentralizingAgent has both positive and negative values for scores, but my agents are all positive and the results are stable. This has been contributed to by limiting the number of purchases and moving to prevent violations.

Experiments	SteadyMgr(MyAgent)	DecentralizingAgent	BuyCheapSellExpensiveAgent
1	**0.15262**	-0.01194	-1.70316
2	**0.38456**	0.12701	-1.75007
3	**0.14949**	-0.22764	-1.57793
4	**0.14062**	-0.26910	-1.51617
5	**0.25419**	0.17639	-1.85530
Average	**0.21629**	-0.04106	-1.68053

Table 1: Scores of experimental results

4 Conclusions

In this report, we described our agent, SteadyMgr. SteadyMgr trades aggressively but its performance is stable because it doesn't cause any breaches. Also, it flexibly changes the trade price and gets a big profit in any environment. Its current live-competition score is high and it is considered a strong agent. Improvements include optimizing the increase or decrease in trade prices. Varying the width and upper limits of the changes depending on the simulation could yield greater profits.