Agent Merchant for SCML, ANAC 2020

Ayan Sengupta
a-sengupta@nec.com

NEC corporation

July 2020

1. Introduction

The SCM world simulates a supply chain consisting of multiple factories that buy and sell products from one another in a simulated market [3]. The factories are represented by autonomous agents from participants in SCML that act as factory managers. Each agent decides which other agents to buy and sell from, and then negotiates with them. The goal is to have the highest median score across all simulations. The score calculated in SCML is based on a percentage of profit and remnant unsold products. The simulation proceeds in discrete time steps during which multiple simultaneous negotiations take place. There are two separate tracks in SCML, 2020: standard track and collusion track. In the standard track, at most one instantiation of each team's agent will run in each simulation whereas, in the collusion track, multiple instantiations of the same team's agent will run during a single simulation.

This report describes the strategy of Agent Merchant in the following sections. Section 2 describes the basic strategy and algorithm. Section 2.1 and Section 2.2 describes the algorithm for standard track and collusion track respectively. Section 3 shows the result of Agent Merchant against builtin agents.

2. Strategy and Algorithm

For an agent to be profitable in the SCML world, it needs to buy input materials through negotiation, manufacture them, then sell output products through negotiation. The tutorial by the organisers in [2] broke the overall strategies into following components

1. Trading Strategy: This component decides the quantity and price to buy and sell at every time-step. A trading strategy can be based on future market prediction or partner behaviour prediction.
2. Negotiation Control Strategy: This component is responsible for proactively request negotiations, responding to negotiation requests and actually conducting concurrent negotiations.
3. Production Strategy: This component decides what to produce at every time-step.

Agent Merchant comprises of similar components. Along with these components, more stress will be given to signing strategy. It is a post-negotiation component that decides what agreements to sign as contracts.

Since SCML involves a lot of negotiation, one can assume that an appropriate negotiator and a complicated utility function is required to be successful in the market. But on the contrary, we use very basic negotiator and utility function for this agent. The negotiator used here is a modified Aspiration type negotiator (Boulware) [1]. The negotiator behaves as an aspiration Negotiator for certain conditions and as a tough negotiator for other conditions and it switches dynamically within a negotiation session depending upon a threshold value as shown in Algorithm 1. The algorithm takes the type of opponent and a threshold value as input. Moreover, if the opponent is an instance of the merchant agent it will always behave as an Aspiration type agent. Utility function used is a linear utility function. Negotiator and utility function are kept the same for both tracks. The rest of the algorithm for agent Merchant can be divided into two separate categories, one for standard track and another for collusion track.

2.1. Algorithm for Standard track

The basic algorithm for the standard track is divided into four components as mentioned in Section 2.

1. Trading Strategy: No special trading strategy is used. Most of the functionalities are in signing strategy.
2. Negotiation Control Strategy: The agent requests multiple negotiations with all its customers and suppliers at every time step with different issues as shown in Algorithm 2. If the agent fails to get any agreement from the

Algorithm 1: Merchant Negotiator

```
input  : threshold, opponent
output: Negotiator
Negotiator = Aspiration Negotiator;
if opponent != Merchant type then
    if aspiration value < threshold then
        Negotiator = ToughNegotiator(utility
          value = threshold);
    else
        Negotiator = Aspiration Negotiator;
    end
end
```

customers or the suppliers in a certain time step, the issue space is dynamically increased in the next time step for possible agreements. Certain abnormal offers are sent to customers and suppliers to check if those agents are exploitable. Abnormal offers include a buying offers of large quantity at a very low unit price or a selling offer of small quantity with large unit prices. Note here that if the offers are too abnormal that causes the exploitable agents to go bankrupt on a single contract, then it is not profitable due to bankruptcy rules. As mentioned in Section 2 the negotiator itself is a dynamic entity. If the agent fails to get any agreement in each time step the negotiation strategy becomes more conceding in the next time step. In the same way, if the agent receives successful agreements, it will behave more like a tough negotiator in the next time step. Moreover, the agent declines all negotiation requests from other agents to avoid exploitation. Negotiation control strategy also decides the threshold value for the negotiator in Algorithm 1.

3. Production Strategy: The agent produces whenever input products are available.

4. Signing Strategy: Most of the decisions are made by this component. Once a number of successful negotiations have been conducted, we have a pool of agreements. This component signs strategically to make profits in the long term and is described in Algorithm 3. Signing strategy has four sub-components:

    - A greedy signing policy that signs pairs of contract if available at each time step, one from buying side and one from selling side that ensures sure profit.
    - An inventory price based signing component signs any agreement that has the total cost of the input product and the production cost less than half of the catalog

Algorithm 2: Negotiation Control Strategy for Standard Track

```
input  : list of opponents o, acceleration
         factor a, deceleration factor d, list of
         small issue spaces i_s, list of large
         issue spaces i_l, boolean variable
         stating if any buy contract signed in
         last step buy, boolean variable
         stating if any sell contract signed in
         last step sell
if track == standard track then
    Cancel all negotiation requests;
    for ( items in o ) {
        for ( issue space i in i_s ) {
            Request negotiation with i issue
              space;
        }
        Send abnormal offers;
        if not buy then
            for ( issue space i in i_l ) {
                Request buy negotiation with i
                  issue space;
            }
            threshold = max(0.4, threshold -
              0.05 * d);
        else
            threshold = min(0.9, threshold +
              0.05 * a);
        end
        if not sell then
            for ( issue space i in i_l ) {
                Request sell negotiation with i
                  issue space;
            }
            threshold = max(0.4, threshold -
              0.05 * d);
        else
            threshold = min(0.9, threshold +
              0.05 * a);
        end
    }
end
```

price of the output product. Since remnant products are valued at half the value of trading price for that product, this component ensures profit for any such agreement.

- A over selling balancer component sign agreements to buy early in the case that the agent signs more contracts for selling. This minimises the risk of breaches.
- A over buying balancer component sign agreements to sell in the case that the agent signs more product for buying. This minimises the risk of a large quantity of unsold products.
- Finally a profit estimate component that estimates the agent profit at the end from the current signed contracts. If the profit estimated is greater than 0.8, no more contracts are signed throughout the simulation. So in the early time steps if an agent makes some profit it will backout from further negotiations to minimise risk.
- A dead lock breaker component that signs a few buy and sell contract to get the signing strategy work in case no buy agreements and sell agreements have been signed.

Algorithm 3: Signing strategy for Standard Track

```
input : List of agreements A, cost of
        production c_p, catalog price of
        output product p_o, variable stating
        total buy contract signed bought,
        variable stating total sell contract
        signed sold
if track == Standard track then
    if bought==0 and sold == 0 then
        sign few cheap buy agreement and
         few expensive sell agreement
    end
    for agreement in A and profit_estimate()
     < 0.8 do
        greedy_signing_policy();
        if agreement["price"] + c_p < p_o/2 then
            sign agreement;
        end
        if over sold then
            over_selling_balancer();
        end
        if over bought then
            over_buying_balancer();
        end
    end
end
```

2.2. Algorithm for Collusion track

Agent Merchant is able to collude in a market where more than one instance of the same agent is present. We have introduced collusive behaviour algorithm for any configuration where multiple instances of agents are present in two or more consecutive levels as shown in Figure 1 and Figure 2. For all other cases agent uses algorithm for standard track. Figure 3 shows two such collusion cases where each agent will behave independently.

An agent's final score is the median of the profits accrued by all the factories it is assigned to manage across all simulations. So the idea is to improve the median as much as possible. So in a collusion track simulation if one instance of agent sustains losses while providing profits to more than one instance of same agent, it ensures that the median score will increase. One thing to notice here is factories at a higher level has more starting balance than the factories at a lower level. This will be leveraged in the collusion algorithm. The configurations(positions of agents in a single world) in a collusion track can be divided into four case as follows:

1. If there are instances of agents in two or more consecutive levels as shown in Figure 1 (Top), then the agent in the highest level $(k + 1)$ buys a single quantity of product at a very high price like 80% of initial balance. These agents do not participate in any further negotiations. If there are more than two consecutive instances of agent this ensures that median score goes up as only one agent incurs losses while the other agent earn some profit. The profit margins depends on the initial distribution of balance. Since the distribution of prices increases proportionally with level, the profit percentages are high for each agent.

2. If there are two or more instance of an agent in the same level and a single instance of the agent at an adjacent higher level as shown in Figure 1 (Bottom), then the agent at the highest level buys a single product from each of the instance of the same agent in the lower level at a high price. Similar to the last case, only one agent incurs loss where as other agents gains a profit thereby increasing the median profit. These agents do not participate in any further negotiations.

3. The third case occurs when there are more than one instance of agent in each of the two consecutive levels as shown in Figure 2. In

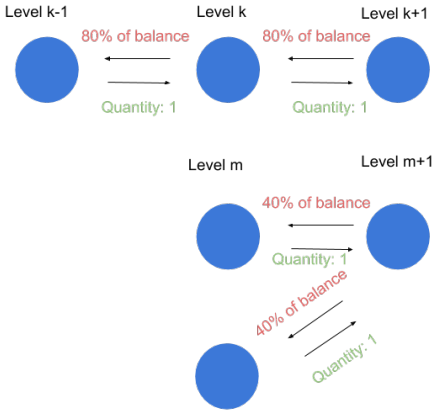

Figure 1: Top: Collusion strategy for two or more instances of same agent in consecutive levels namely $k-1, k$ and $k+1$. Bottom: Collusion strategy for two or more agents in a lower level m and one agent in the next higher level $m+1$

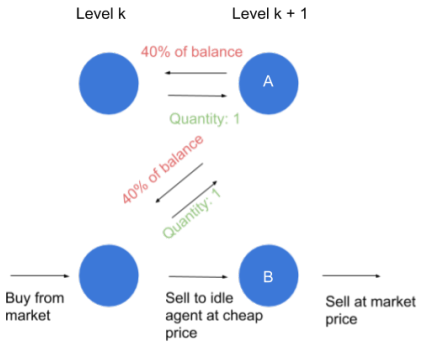

Figure 2: Top: Collusion strategy when there are more than one instance of agent in each of two consecutive levels k and $k+1$. In the initial step B is the ideal agent.

this case the algorithm works in two parts. In first part one or more agent in the highest level stays idle (agent B) and the problem becomes similar to case 2. After the initial transaction, agents in the lower level (level k) buy from market and sell it at a cheap price to the idle agent(s) (agent B) at the higher level. From next step these agents negotiate in the market like standard track and try to sell the output product.

4. All configurations where there are no two instances of agents in consecutive levels are treated as standard track problem.

3. Results against in built agents

Table 1 shows the result against the two inbuilt agents Decentralized agent and BuyCheapSellExpensive (BCSE) Agent for the standard track and collusion track. The configuration for competition are n_steps: 60, n_configs: 50, n_runs_per_world:

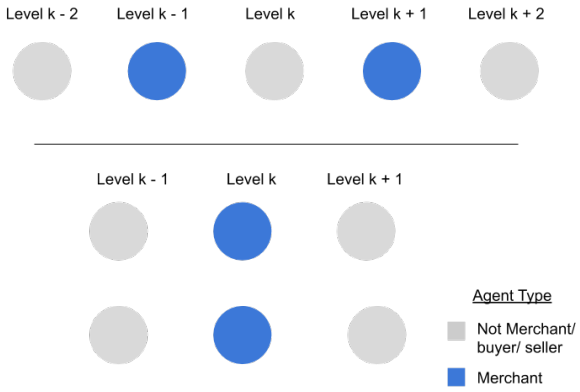

Figure 3: Top and Bottom: Not considered in collusion strategy. Each agent independently will behave according to standard track.

Table 1: Median score against inbuilt agents

Track	Merchant	Decentralized	BCSE
Standard	0.227	-0.049	-1.535
Collusion	0.005	-0.068	-0.527

2. As per the last live competition, agent Merchant stands at 12^{th} position in standard track and at 2^{nd} position in collusion track.

References

[1] P. Faratin, C. Sierra, and N. R. Jennings. Negotiation decision functions for autonomous agents. *Robotics and Autonomous Systems*, 24(3-4):159–182, 1998.

[2] Y. Mohammed. Supply Chain Management League Tutorial. `http://www.yasserm.com/scml/scml2020docs/` (July 2020).

[3] Y. Mohammed, A. Greenwald, K. Fujita, M. Klein, S. Morinaga, S. Nakadai. Supply Chain Management League. `http://www.yasserm.com/scml/scml2020.pdf` (July 2020).