report scml 2021

Emanuel Nafcha, Eyal Cohen, Eliyahu Zilberbrand

July 2021

1 Introduction

For building our agent, we used pre-built strategies and tried to improve some of them.

2 Strategy

2.1 Trading strategy

The pre-built strategy used for trading was PredictionBasedTradingStrategy. We changed it by applying the following limitations:

2.1.1 Blocking breaching agents

For every negotiation request, we checked if the agent of that contract has breached in the last n steps, if so, the contract will be restrained.
In our experiments, we tried tuning the number of steps of looking for violation, although we didn't find significant improvement between n=25 to unlimited, but on the live tournament, surprisingly we saw some improvement in the standard tournament.

We also checked dynamic breach blocking - being more forgiving to breaches in proportion to the game steps. The reason is that as the game progresses, it makes more sense that an agents would breach a contract from time to time after some steps.

We ran this agent alongside the same agent with static breaching agents blocking (of all agents that breach in the last 25 steps) and with some additional built-in agents (DecentralizingAgent, BuyCheapSellExpensiveAgent, RandomAgent, DoNothingAgent, IndDecentralizingAgent and IndependentNegotiationsAgent), and we found better results for the agent with static breaching agents blocking, so we didn't implement this method.

The limitation for breaching contracts was applied on incoming contracts only, for the following reasons:

- We could not limit both sides, otherwise the agent wouldn't have enough contracts. One solution was to limit only up to n steps back, but it didn't seem to perform too well

- Blocking incoming contracts from breaching agents led to better results, although we assume the results may have occurred (also) due to the limited amount of incoming contracts. We tried limiting the number of all incoming contracts but it seemed like the combination of limiting the amount of contracts and blocking breaching agents led to the best result.

- Blocking breaching agents on outbound contracts only but it performed worse than blocking incoming contracts only. We did not block breaching agents incoming contracts, but we did clock outbound contracts, this experiment resulted in a worse results than doing the opposite.

2.1.2 Preferring contracts from other instances of our agent

In order to save the slot for our agent, we didn't sign on incoming contracts if another agent was found, that offers the same product.
However, taking this approach we might cancel incoming contracts in order to save slots for our instances of our agent.
Applying this to outgoing contracts however, might solve part of the issue, but we still need to manage the amount of contracts, because it is possible that only one of our agents have a specific product to offer, and as an outcome we will block many of our other agent's contracts.
As for blocking breaching agents, giving priority to contracts from other instances of our agent gave the best results on incoming contracts only.
This fact strengthened our belief that limiting the incoming contracts might help.

2.2 Production Strategy

For our production strategy we used the pre-built SupplyDrivenProductionStrategy, which we kept unchanged.

2.3 Negotiation Manager

after trying out some different negotiation managers (for example: IndependentNegotiationsManage, StepNegotiationManager) we chose StepNegotiationManager after it gave us the best results.
This kind of negotiation manager creates controllers named StepController. The regular controller uses linear utility function called LinearUtilityFunction, we chose to change this to NonlinearHyperRectangleUtilityFunction with exponential function, and got better results. we used it for both seller and buyer positions.