Final Project: Agent Negotiation Simulation

1 Introduction

This final project option is based on the SCML (Supply Chain Management League) Agent Negotiation Competition, which builds on the work we did in Lab 5.

2 Setup

Please refer to the handout of Lab 5 if you do not have python or pip installed.

Then, run the command: pip install scml.

3 Game Description

The SCM OneShot world simulates a supply chain consisting of multiple factories that buy raw materials from, and sell final products to, one another. The factories are managed by autonomous agents. These agents are assigned a target quantity (drawn at random) to either buy or sell. They then negotiate with other agents to reach agreements, which become binding contracts that specify the terms of trade. A simulation comprises multiple days, during each of which the OneShot game is played. All agents have the same goal each day, namely to turn a profit. The agent with the highest total profit summed over all days, and then averaged across multiple simulations, wins. Learning is permitted from one day to the next during a single simulation; however, learning is not permitted across simulations.

Products There are three product types: a raw material, an intermediate product, and a final product.

Production There are two manufacturing processes, one for converting the raw material into the intermediate product, and a second for converting the intermediate product to the final product.

Factories Factories convert input products into output products by running their manufacturing processes on their production lines. All processes run convert exactly one unit, instantaneously, at a predefined cost.

Production Graph Factories are organized in two layers L0 and L1 (see Figure 1). L0 factories receive exogenous contracts to buy the input (raw material), and then negotiate with L1 factories to sell them the intermediate product. L1 factories receive exogenous contracts to sell their output (final product), and then negotiate with L0 factories to buy the intermediate product.

Agents The agents in the SCM world function as factory managers. They negotiate to reach agreements to buy and sell the intermediate product, which automatically become binding as contracts.
Negotiation Protocol  Agreements are negotiated using a variant of the bilateral alternating offers protocol, typical of ANAC competitions. Each offer specifies a buyer, a seller, a quantity, and a unit price. The sequences of offers and counteroffers in a negotiation are private to the negotiating parties.

Negotiation Issues  All negotiations concern two issues: quantity and price. Delivery dates are not a negotiation issue in the OneShot game, as all products are assumed to be delivered on the same day.

Quantity  An integer between 1 and the factory’s number of production lines.

Unit Price  An integer between 1 and $\lceil \eta(tp) \rceil$, where $\eta > 0$ is a configuration parameter (e.g., 2) and tp is the trading price of the intermediate product.

Utility Functions  An agent’s utility function represents its profits. As such, it is simply the total revenue it receives from any sales less its total expenses, the latter of which includes the contracted cost of the input product as well as the agent’s private production costs, disposal costs, and shortfall penalties.

N.B. While each agent’s production costs, disposal costs, and shortfall penalties are private information, the distributions from which these values are sampled are common knowledge.

Trading Price  The trading price (tp) of a product is a weighted average of its past prices, which weighs newer contract prices more heavily than older ones. The trading price is used by the simulator to set the price range of all negotiations, and for calculating penalties.

Balances  Factories have an associated balance—seeded at the start of the game with some finite amount—from which they withdraw to pay for supplies, etc., and into which their sales revenue is deposited.

Bulletin Board  The SCM world contains a world-readable bulletin board that conveys both static and dynamic information about the game environment and all factories over the course of the simulation.
The static information includes the simulator settings (e.g., number of simulated days), and product information, namely a list of the consumers and producers of all products (i.e., all factory’s positions in the production graph), and the initial trading prices (called catalog prices).

The dynamic information includes a trading price list (per product), which reports a weighted average of each product’s past prices; and a financial reports section (also per agent), which is updated only periodically, that summarizes the financial standing of all factories (e.g., their balances).

Finally, the bulletin board also contains an exogenous contract summary, which reports the total quantity and average unit price of exogenous contracts each day.

**The Simulation**  Each simulation of the SCM world runs for multiple (say, 100) days. Before the first day, each agent is initialized and assigned a production cost. Each day the following takes place (Figure 2):

1. All products’ trading prices are updated.
2. The world generates exogenous contracts, and samples disposal costs and shortfall penalties for all agents from their corresponding distributions.
3. All agents are initialized for the day.
4. Agents engage in multiple (say, 20) rounds of negotiations with their negotiating partners. They can also read the bulletin board.
5. All contracts are executed: i.e., products are moved from the seller’s inventory to the buyer’s, and money is moved from the buyer’s account to the seller’s.
6. All agents are finalized for the day.
7. The bulletin-board is updated, most notably to reflect new trading prices, updated financial reports, and the day’s exogenous contract summaries.

**4 Game Specifics**

- An agent’s performance will be measured by its score, which will be computed as the mean of the utilities (i.e., profits) accrued by all the factories it is assigned to manage across all simulations.
- At most one instantiation of each team’s agent will run in each simulation, together with an unknown mix of additional agents.
- The number of simulation days in a single run will be 50.

**5 Strategy and Considerations**

This is a complex game. Perhaps the biggest challenges your agent will face is learning about its negotiating partners. These partners will have different consistent tendencies—“personality traits,” you might call them. Some will be simple, some sophisticated, some cooperative, some competitive. They will also be subject to different transitory circumstances: the same agent might have lots of goods to sell one day, and very few the next. A successful agent will be able to learn something about its partners’ personality traits over the course of the game, and will also be able to quickly recognize its partners’ circumstances (e.g., rich or poor) within the 20 rounds of daily negotiation.

**6 Game API**

Your agent should extend the base class `OneShotAgent` (code here). You will need to implement methods that are called by the system throughout the negotiation. The full list of methods can be found in the source
code of the **OneShotAgent** class. The most important ones are:

- **init()** called once at the beginning of the simulation (i.e. before the first day starts).
- **before_step()** called at the beginning of each day. At this point, your **ufun** is set and market information is made available.
- **propose()** called when you need to offer something to your negotiating partners during negotiation. The method receives the negotiation state (an object of type **SAOState**, including among other things the current negotiation step, relative time, last offer, etc.) and returns an **Outcome**.
- **respond()** called when you need to respond to an offer proposed by a negotiating partner. The method receives a negotiation state and an offer (an object of type **Outcome**) from the opponent, and responds with a decision from the **ResponseType** enumeration (**REJECT_OFFER**, **ACCEPT_OFFER**, and **END_NEGOTIATION**).
- **on_negotiation_success()**/**on_negotiation_failure()** called after each negotiation is concluded to let your agent know what happened.
- **step()** called at the end of each day. Your agent can use this information to analyze what happened during the day and modify its strategy for the future.

Refer to Figure 2 to see where each of these methods is called in relation to the simulation steps.

![Figure 2: Order of execution of events during a simulation.](image)

7 Tier 1 Agent

You can/should test your agent against our Tier 1 TA Bot, **LearningAgent**, whose code can be found [here](#). At a high-level, **LearningAgent** operates as follows:

- Its offer policy concedes over time, meaning it offers its partner something less and less appealing to itself over time, but hopefully more and more appealing to its partner, with the intent that the negotiation does not end in disagreement.
- Each day, it updates the best price received so far from each negotiating partner, and then it limits the minimum price (for selling) and maximum price (for buying) based on this price.
- It also uses information from one negotiation to inform other concurrent negotiations—e.g., if it sees a good price in one, it will demand better prices in others.
8 Testing

To test your agent, refer to Running a One-Shot tournament. At the top of your testing file, be sure to import all the relevant classes: `from scml.oneshot import *`.

As with any programming project, you should test your agent extensively. In addition to the usual unit tests, etc., you should play test games against other agents. You can test against multiple copies of your own agent, multiple copies of our Tier 1 TA Agent, or any combination thereof. You can also create your own dummy agent or use any of the simple agents implemented here.

9 Useful References

SCML2022 League Website
SCML Github Repository
SCML Code Documentation/Tutorials
Supply Chain Management League (OneShot) Document