Supply Chain Management League (OneShot)

Automated Negotiating Agents Competition

SCML Organizing Committee:

Y. Mohammed, A. Greenwald, K. Fujita, M. Klein, S. Morinaga, S. Nakadai

September 25, 2023

Abstract

This document describes the Automated Negotiation Agent Competition (ANAC) Supply Chain Management League OneShot track (SCML-OneShot). The game is intended to further research on agent negotiation. As such, the game design emphasizes negotiation and de-emphasizes operations (e.g., production, scheduling, etc.).[1]

N.B. There are two tracks in SCML 2024. This document pertains *only* to the OneShot track.

1 Overview

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 L_0 and L_1 (see Figure 1). L_0 factories receive exogenous contracts to *buy* the input (raw material), and then negotiate with L_1 factories to sell them the intermediate product. L_1 factories receive exogenous contracts to *sell* their output (final product), and then negotiate with L_0 factories to buy the intermediate product.

[1] A shorter game overview is available here.

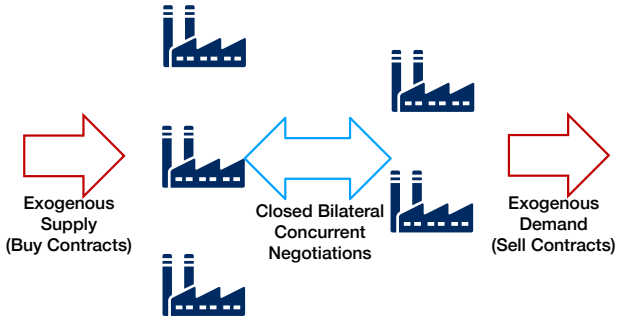

Figure 1: SCML-OneShot world. Each factory is represented by an agent, whose goal to negotiate buy and sell contracts that maximize profits.

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 [1, 2]. 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.

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** (see Figure 2) 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 assigned a production cost. During each day:

1. The world generates exogenous contracts, and samples disposal costs and shortfall penalties for all agents from their corresponding distributions.

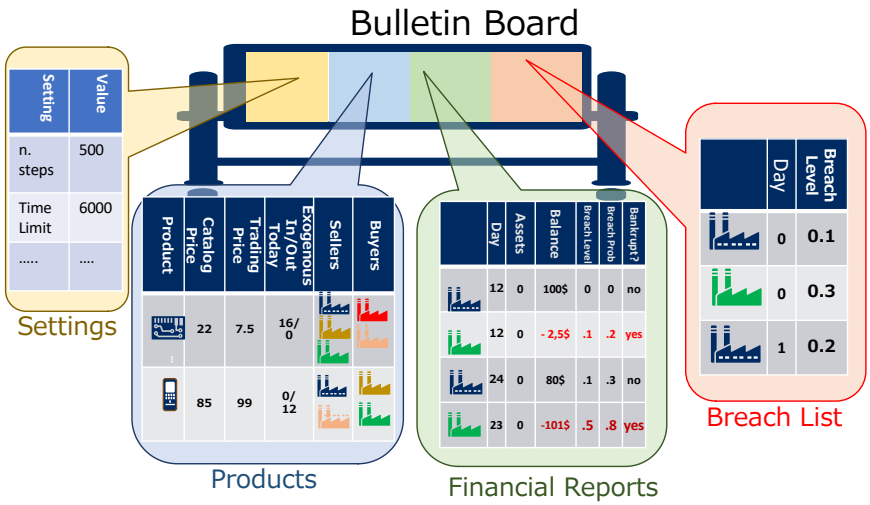

Figure 2: The bulletin board includes Simulator Settings, Product Information (includes trading prices, catalog prices, and exogenous contract summaries), Financial Reports, and the Breach List.

2. Agents engage in multiple (say, 20) rounds of negotiations with their negotiating partners. They can also read the bulletin board.

3. 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.[2]

4. The bulletin-board is updated, most notably to reflect new trading prices, updated financial reports, and the day's exogenous contract summaries.

Differences from SCML 2023 The SCML OneShot 2024 game has the same rules as SCML OneShot 2023. This year we provide a framework for building RL agents for the competition. Using this framework is not required but will probably make life easier for people intending to use RL for their agents.

2 Game Entities

We start by describing the components of the SCM world, namely the environment and the agents. By the **environment**, we mean the manufacturing structure—what can be manufactured and how. By the **agents**, we mean autonomous entities that make decisions about what to buy and what to sell.

2.1 The Environment

The manufacturing structure of an SCM world—what can be manufactured and how—is represented by a **production graph**. This graph comprises a set of **products** P and a set of **manufacturing processes** M, and specifies which inputs and processes are used to produce which outputs.

Although the SCM world supports directed acyclic graphs,[3] the production graph used in OneShot is a **chain**, with but one raw material, one intermediate product, and one final product. Correspondingly, there are two manufacturing processes, each one advancing the product one level in the chain. These manufacturing

[2]Contracts are executed in order, starting from the raw material and ending with the final product. Production is run in the same order, which guarantees the accuracy of the utility functions. (See Equation 5.)

[3]See `http://www.yasserm.com/scml/scml.pdf` (Chapter 1) for a detailed description of the space of manufacturing structures.

processes are instantaneous in OneShot. They transpire as soon as all the day's negotiations end, at which point all resulting products are delivered (instantaneously) as well.

Each factory in the SCM world operates at exactly one of the two levels in the chain, running one of the two manufacturing processes on its various production lines. In OneShot, all factories are endowed with the same number of lines (λ),[4] but product costs vary across levels, and across factories/manufacturing processes at the same level in the chain. In all SCML tracks, production becomes more and more costly, on average, the closer a product comes to being finished.

2.2 Agents: The Decision-Makers

Manufacture and trade in the SCM world are directed by autonomous, decision-making agents, who function as **factory managers**. As such, at the start of a simulation, each agent is assigned a single factory to manage. The factory that agent a manages is announced publicly, but each factory's manufacturing profile, most importantly the cost m_a of running its manufacturing process, is private information, known only to its factory manager (i.e., the agent).

As agents/factories in OneShot do not accrue inventory, agents that overprovision incur disposal costs at the end of a day, while agents that underproduce incur shortfall penalties. In addition to its manufacturing profile (i.e., the number of lines, the manufacturing process, and the production cost), an agent's private profile also contains four parameters indicative of its disposal costs and shortfall penalties, namely the mean and standard deviation of a normal distribution corresponding to each. Each day, a particular disposal cost and shortfall penalty is sampled from these distributions.

Each agent/factory is also assigned an initial balance at the start of the game (i.e., an initial endowment) that it can spend buying inputs and paying for production. This initial balance is the same for all the factories at a given level in OneShot, so it is known with certainty to all factory managers at this level, but not to their trading partners at adjacent levels. Throughout a simulation, the only state that agents maintain from one day to the next is their balance.

3 Negotiation

At a high-level, negotiating to reach an agreement proceed as follows:

1. Every day one negotiation is started between every L_0 agent and every L_1 agent. Negotiation proceeds via the alternating offers protocol (see Section 3.1).
2. If the negotiation successfully reaches an agreement, that agreement is binding.

3.1 Mechanism

The negotiation mechanism adopted by SCML 2023 [3] is a variant of Rubinstein's alternating offers protocol [4]. It involves two agents, who take turns making offers for a finite number of rounds and/or seconds. One agent opens the negotiation with an offer, after which the other agent takes one of the following actions:

1. Accepts the offer
2. Responds with a counteroffer, thus rejecting and overriding the previous offer
3. Walks away, thus declaring an end to the negotiation, without having reached an agreement

[4] In future iterations of the game, a single factory may be operate at multiple levels in the chain, and/or be capable of running different manufacturing processes on its various production lines (even at the same level).

This process repeats until either an agreement or a deadline is reached. To reach an agreement, one party must accept the offer proposed by its partner. If no agreement is reached by the deadline, the negotiation fails[5].

All negotiations in the SCM world must be completed within a fixed number of rounds (e.g., 100) and within a fixed amount of time (e.g., 2 minutes). Additionally, each agent has a fixed amount of time (e.g., 10 seconds) in which to respond to an offer or propose a new one.

Note that anything pertaining to negotiations between agents is private information. No other agents except those negotiating see any intermediate offers or responses, nor any final agreed-upon contracts.

Important features to keep in mind about negotiations in SCML:

- All negotiations at a given day proceed in one direction. This means that either all negotiations are opened by the offer from L_1 agents or all of them are opened by offers from L_0 agents.
- It is not guaranteed that the opening agent will be the same everyday. This means that some days L_1 agents will start all negotiations and in some other days L_0 agents will start all negotiations.
- You should not assume that all negotiations proceed at the same speed. This means that a negotiation between agent A and B can be on round 10 while another between agent A and agent C can be on round 5 [6].

3.2 Data Structures

A **negotiation agenda** ν is a tuple (p_ν, q_ν), where:

Negotiation Issues All negotiations (between L_0 and L_1 agents) have the following issues:

Unit Price an integer between $\lceil \kappa \operatorname{tp}(s) \rceil - 1$ and $\lceil \kappa \operatorname{tp}(s) \rceil$, where $\kappa > 0$ is a configuration parameter and $\operatorname{tp}(s)$ is the trading price of the intermediate product (see Section 4)[7].

Quantity an integer between 1 and λ_a (where λ_a is the number of lines in factory a).

An **offer** o is a tuple (p_o, q_o) consisting of a unit price p_o and a quantity q_o. Note that the precise value of each issue must be precisely specified in an offer; offers cannot include any ranges.

A **contract** c is a tuple $(s_c, b_c, o_c, \ldots)$, where

Seller $s_c \in \mathcal{A}$ The seller agent (an L_0 agent).

Buyer $b_c \in \mathcal{A}$ The buyer agent (an L_1 agent).

Offer o_c The agreed upon offer.

An **exogenous contract** is a contract in which one of the buyer or the seller is the simulator. Each exogenous input contract issued to an L_0 agent a includes a quantity of the raw material constrained not to exceed λ_a, and its unit price. Exogenous output contracts issued to L_1 agents are similar in structure, but pertain to the final product rather than the raw material.

[5] In SCML 2021, negotiators did not know which of them would initiate the negotiation because a random offer from one of them was dropped at the first round. Since SCML 2022, the standard alternating offers protocol is used with no such measures.

[6] In SCML 2023, it is the case that all negotiations proceed at the same speed though.

[7] This is the only difference between 2023 and 2022 versions of SCML OneShot. In 2023, the price range is guaranteed to be small (only 2 values) which means that the main determining factor in achieving high score is being able to match the total quantity bought and sold. Moreover, the value of κ will be around one which means that trading prices are not expected to change much from catalog prices.

3.3 Bankruptcy and Contract Breaches

All contracts in OneShot are binding. Moreover, agents honor all buy contracts, whether or not they have sufficient funds. Likewise, they honor all sell contracts, whether or not they have the requisite input products, or the funds to cover production costs.

If, at the end of a day, an agent's balance is negative (i.e., it did not have sufficient funds), it is declared **bankrupt**. Bankrupt agents cannot engage in any further negotiations; their balance is frozen (at a negative value) for the remainder of the simulation.

A **breach** occurs if ever an agent commits to selling more of a product during a day than it can produce, either because of an insufficient quantity of input product or manufacturing lines. An agent that commits a breach incurs shortfall penalties, which are intended to mimic the idea of an agent scrambling somehow to make good on a promise.

N.B. Since all contracts are honored in OneShot, each agent's utility function calculation is independent of all the others'.

4 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.

In this section, we present the decision problem embedded in computing an agent's profits, which is used to determine which buy and sell contracts the agent can satisfy, given its finite balance and limited production capabilities. We then explain how how these decisions feed into the agent's utility.

At the start of each day, each agent is issued an exogenous contract. Moreover, at the end of each day, each agent will have a set of *negotiated* input and output contracts. We refer to the set of negotiated input (output) contracts together with the exogenous input (output) contracts for agent a as C_a^{in} (C_a^{out}).

Decision Problem Because agent a can only buy what it can afford, and its balance b_a is finite, we define the set of satisfiable input contracts $C_a^{*\text{in}} = \{(p_c, q_c^{*\text{in}})\} \subseteq \{(p_c, q_c^{\text{in}})\} = C_a^{\text{in}}$ that minimize costs as follows:

$$\begin{aligned} C_a^{*\text{in}} &\in \arg\min_{\tilde{C}^{\text{in}} \subseteq C_a^{\text{in}}} \sum_{c \in \tilde{C}^{\text{in}}} p_c q_c^{*\text{in}} \\ \text{s.t.} \quad & q_c^{*\text{in}} \leq q_c^{\text{in}}, \forall c \in \tilde{C}^{\text{in}} \\ & \sum_{c \in \tilde{C}^{\text{in}}} (p_c + m_a) q_c^{*\text{in}} \leq b_a \end{aligned} \tag{1}$$

The variable $q_c^{*\text{in}}$ denotes the optimal quantity of the input product that it is feasible for a to buy to satisfy contract c, subject to the constraint that this quantity is *producible*, meaning the agent does not spend more than its balance buying its total producible quantity and converting its inputs to outputs.

Calculating $C_a^{*\text{in}}$ is actually easy. It can be accomplished greedily by sorting all input contracts in ascending order by price, and then inserting them into $C_a^{*\text{in}}$ as long as the balance constraint is not violated. Note that only the quantity of the last *satisfied* contract c in the sorted list is potentially less than q_c^{in}.

We write $Q^{*\text{in}} = \sum_{c \in C_a^{*\text{in}}} q_c^{*\text{in}}$ to denote the total quantity of the input product the agent considers buying and converting into outputs, as a result of executing this greedy algorithm. Similarly, we write $Q^{\text{in}} = \sum_{c \in C_a^{\text{in}}} q_c^{\text{in}}$ to denote the total quantity of the input product the agent contracted to buy.

Because agent a can only sell what it can produce, we define the set of satisfiable output contracts

$C_a^{*\text{out}} \equiv \{(p_c, q_c^{*\text{out}})\} \subseteq \{(p_c, q_c^{\text{out}})\} = C_a^{\text{out}}$ that maximize revenue as follows:

$$\begin{aligned} C_a^{*\text{out}} \in & \arg\max_{\tilde{C}^{\text{out}} \subseteq C_a^{\text{out}}} \sum_{c \in \tilde{C}^{\text{out}}} p_c q_c^{*\text{out}} \\ \text{s.t.} \quad & q_c^{*\text{out}} \leq q_c^{\text{out}}, \forall c \in \tilde{C}^{\text{out}} \\ & \sum_{c \in \tilde{C}^{\text{out}}} q_c^{*\text{out}} \leq \lambda_a \\ & \sum_{c \in \tilde{C}^{\text{out}}} q_c^{*\text{out}} \leq Q^{*\text{in}} \end{aligned} \tag{2}$$

The variable $q_c^{*\text{out}}$ denotes the optimal quantity of the output product that it is feasible for a's factory to produce to satisfy contract c, subject to two constraints, which together ensure that a does not produce beyond its means: 1. it cannot produce a quantity greater than its number of lines; and 2. it cannot produce a greater quantity of output product than its total producible quantity of input product, which we take to be the optimal quantity $Q^{*\text{in}}$ a can afford to produce, as per Equation 1.

Calculating $C_a^{*\text{out}}$ is also easy. It can be accomplished greedily by sorting all output contracts in descending order by price, and then inserting them into $C_a^{*\text{out}}$ as long as the constraints are not violated, with (as above) only the quantity of the last *satisfied* contract c in the sorted list potentially less than q_c^{out}.

We write $Q^{*\text{out}} = \sum_{c \in C_a^{*\text{out}}} q_c^{*\text{out}}$ to denote the total quantity of the output product the agent produces and sells, as a result of executing this greedy algorithm. Similarly, we write $Q^{\text{out}} = \sum_{c \in C_a^{\text{out}}} q_c^{\text{out}}$ to denote the total quantity of the output product the agent contracted to sell.

Putting it all together, the decisions made via this two-step greedy approach to deciding what to buy (among all buy contracts) and what to sell (among all sell contracts) comprise $C_a^{*\text{out}}$ and $Q^{*\text{out}}$.

Penalties As all agents in OneShot buy all input products Q^{in}, they may be left with excess inventory when $Q^{\text{in}} > Q^{*\text{out}}$. Similarly, as agents sell all output products they manage to produce ($Q^{*\text{out}}$), they experience a shortfall when $Q^{\text{out}} > Q^{*\text{out}}$. Agents incur penalties in their utilities based on the differences between the amount they contracted to buy (sell) and the amount they actually buy (produce/sell):

$$Q_a^{\text{excess}} = \max\{0, Q^{\text{in}} - Q^{*\text{out}}\} \tag{3}$$

$$Q_a^{\text{shortfall}} = \max\{0, Q^{\text{out}} - Q^{*\text{out}}\} \tag{4}$$

Utility Function Agent a's utility u_a can now be defined as a's profits, i.e., its revenue less its costs and its penalties:[8]

$$u_a(C_a^{\text{in}}, C_a^{\text{out}}) = \underbrace{\sum_{c \in C_a^{*\text{out}}} p_c q_c^{*\text{out}}}_{\text{revenue}} - \underbrace{\sum_{c \in C_a^{\text{in}}} p_c q_c^{\text{in}} - m_a\, Q^{*\text{out}}}_{\text{costs}} - \underbrace{\left(\alpha_a\, \text{tp}(\rho_a^{\text{in}}, d)\, Q_a^{\text{excess}} + \beta_a\, \text{tp}(\rho_a^{\text{out}}, d)\, Q_a^{\text{shortfall}}\right)}_{\text{total penalties}}, \tag{5}$$

where ρ_a^{in} and ρ_a^{out} are factory a's input and output products, respectively, and $\text{tp}(\rho, d)$ is the **trading price** of product ρ on day d, defined below.

$\sum_{c \in C_a^{*\text{out}}} p_c q_c^{*\text{out}}$ The total **revenue** it earns by selling its outputs.

$\sum_{c \in C_a^{\text{in}}} p_c q_c^{\text{in}}$ The total **cost** it incurs to buy its inputs.

$m_a\, Q_a^{*\text{out}}$ The production **cost**. Note that factories produce exactly what they can sell on the current day, as inventory does not carry over from one day to the next.[9]

[8] The term in red, namely C_a^{in}, indicates that agents buy *all* inputs they contracted to buy.

[9] Equation 1 ensures that the agent *can* produce output products corresponding to all its input products, but Equation 5 assumes production of only those output products the agent sells.

α_a $\mathbf{tp}(\rho_a^{\mathbf{in}}, d)$ $Q_a^{\mathbf{excess}}$ The total buy-side **penalty**, which is incurred on any output products that are not sold. Note that these penalties depend on the trading price of the input product.

β_a $\mathbf{tp}(\rho_a^{\mathbf{out}}, d)$ $Q_a^{\mathbf{shortfall}}$ The total sell-side **penalty** incurred by the factory for failing to deliver its output product. Note that these penalties depend on the trading price of the output product.

Trading Prices The **trading price** (tp) for product ρ at the beginning of day d is calculated as follows:

$$\text{trading price}(\rho, d) \equiv \text{tp}(\rho, d) = \frac{\gamma^d\, Q_{-1}(\rho)\, \text{cat}(\rho) + \sum_{i=0}^{d-1} \gamma^{d-i}\, Q_i(\rho)\, \mu_i(\rho)}{\gamma^d\, Q_{-1}(\rho) + \sum_{i=0}^{d-1} \gamma^{d-i}\, Q_i(\rho)} \quad , \tag{6}$$

where $\text{cat}(\rho)$ is the catalog price of product ρ, $\gamma \in [0, 1]$ is a discount factor (*trading price discount factor*), $Q_{-1}(\rho)$ is a weight representing the *effective* quantity that is represented by the catalog price (*prior catalog price quantity*), $Q_i(\rho)$ is the total quantity of product p traded on day i (in contracts executed even partially on that day), and $\mu_i(\rho)$ is the average price per item at which product p traded on day i. More specifically,

$$Q_i(\rho') = \sum_{\{c \in C^i | c.\rho = \rho'\}} c.\bar{q} \qquad \text{and} \qquad \mu_i(\rho') = \frac{\sum_{\{c \in C^i | c.\rho = \rho'\}} c.\bar{q} \times c.p}{Q_i(\rho')} \quad ,$$

where C^i is the set of all agreements reached on day i, $c.\rho$ is the product traded via contract c, $c.p$ is the unit price of contract c, and $c.\bar{q}$ is the actual quantity exchanged (which is less than the agreed upon quantity whenever a breach occurs).

5 Information

Some of the information in an SCML simulation is private to the agents to which it is germane (e.g., balances), while other information is public (e.g., trading prices). Moreover, some of the private information is summarized for public consumption periodically.

5.1 Private Information

All negotiations and ensuing contracts are private to the parties involved. All agents' profiles are also private. In OneShot, these profiles are characterized by five numbers: production cost, and the mean and standard deviation of disposal costs and the shortfall penalty.

Additionally, an agent's state information is private. This state includes the agent's current account balance, its current disposal costs and shortfall penalty, and its exogenous contracts.

5.2 Public Information

The SCM world maintains a bulletin board that broadcasts all public information. Some of this information is static, such as the simulator settings (see Table 1). Other information is dynamic.

The simulator publishes three types of dynamic information on the bulletin board regularly: the market's status, the agents' status (in the form of financial reports), and the breach list.

Market Status The simulator publishes the following market statistics on the bulletin board at the end of *every* simulation day for each product:

Exogenous Contract Summaries The total quantity and average price across all exogenous contracts.

Trading Prices The current trading price (See Equation 6), which is a weighted sum of past trading prices.

Financial Reports The simulator also publishes information about each agent's financial status every *reporting_period* days. These reports include the agent's current balance, the value of its inventory (which is always zero in OneShot), whether or not it is bankrupt,[10] and two indicators of its past breaches:

Breach probability The fraction of the agent's contracts that it has breached thus far in the simulation.

Breach Level The agent's breach level averaged across all days. The breach level on any given day is calculated as $(Q_s - Q_p)/Q_s$, where Q_s is the total quantity the agent committed to selling and Q_p is the total quantity it can produce.

Breach List Finally, the simulator publishes a list of all breaches committed by all agents each day. Each entry in this list is an agent name and its breach level that day.

6 Simulation Steps

Figure 3: Order of execution of events during a simulation.

All SCM agents implement an initialization function and a step function. The former is called by the simulator to initialize agents' behavior, before day zero.[11] After initialization, the simulator repeats the following loop every day, which calls the agents' step functions, among other things (see Figure 3):

1. Update all products' trading prices as per Equation 6.
2. Assign each agent an exogenous contract for this day. (See Appendix B.)

[10] Unlike in the standard SCML world, there is no need in OneShot to notify agents immediately about another agent's bankruptcy, or liquidate (e.g., via a spot market) and compensate others, because there are never any outstanding contracts.

[11] Following NegMAS, and to be consistent with most programming languages, the first simulation day in SCML is day zero.

3. Assign each agent a a disposal cost α_a and a shortfall penalty β_a. Each is sampled from a per-agent normal distribution (see Table 2).
4. Call all agents' *before-step* functions in an unspecified order.
5. Run all negotiations until completion. Successful negotiations are binding. Exogenous contracts are also binding.
6. Calculate each agent's profits (i.e., utility) using Equation 5 and add the ensuing amounts to the corresponding agents' balances.
7. Call the agents' *step* functions in an unspecified order.
8. Publish financial reports, a summary of recent exogenous contracts, and trading prices (every *reporting_period* days).

7 The SCML Platform

Like SCML 2019 and SCML 2020, SCML 2021 will run on top of NegMAS [5], which is a Python framework for developing autonomous negotiation agents embedded in simulation environments.

7.1 Negotiators

A **negotiator** is an entity that conducts negotiations on behalf of an agent. All negotiators must implement the following interface:

Propose Proposes an offer, which is an assignment of values to all negotiation issues.

Respond Either accepts, rejects, or ends the negotiation, in response to an offer.

Negotiators are dynamic entities created by an agent for the purpose of negotiating a contract on its behalf. Two types of negotiators are supported:

1. Empowered negotiators are full-on decision makers. They are assigned a utility function when they are created, and they use their utility function to make offers and respond to others' offers.
2. Pass-through negotiators merely pass offers and counteroffers through to the agent that created them. Decision making is therefore wholly the responsibility of the creating agent, which is called a **controller**. A controller typically manages multiple negotiators, deciding how to propose and respond for all of them. Together, controllers and pass-through negotiators can be used to implement a centralized negotiation strategy.

7.2 Agents (Factory Managers)

An agent (also called a factory manager) controls a factory in the SCM world.

Callbacks Agents can implement a variety of callbacks. The simulator calls them at appropriate times during the simulation. The callbacks starting with **On** need not return anything; they are merely informative. Other callbacks require the agent to take some action (e.g., respond to a negotiation request, etc.).

The first two callbacks are called by the simulator's main loop:

Init Called after the world is initialized, but before the simulation begins.

Befor Step Called in the simulation loop at the beginning of the day after exogenous contracts are created and ufuns are constructed and before any other calls to the agent.

Step Called in the simulation loop at the end of the day after all other calls to the agent are completed.

The next callback is event driven; it are triggered by the event its names suggest:

On Negotiation Success/Failure Called when a negotiation the agent is involved in terminates.

Actions Agents can gather information about their factory and other agents using the following methods:

Get State Reads the factory state.

Bulletin Board Access the bulletin board to read simulation settings (e.g. number of days, current day), product information, breach list, financial reports, exogenous contract summaries, and trading prices.

8 Tournament Mechanics

To participate in the Supply Chain Management League (SCML), you should write and submit code for an autonomous agent that acts as a factory manager.

In the OneShot track, at most one instantiation of each agent will run in each simulation, together with an unknown mix of additional agents prepared by other participants and by the organizing committee. An agent's performance will be measured by its score, which will be computed as the *truncated mean*[12] of the utilities (i.e., profits) accrued by all the factories it is assigned to manage across all simulations. The profit accrued by an agent during one day in one simulation is calculated according to Equation 5.

All tournaments will be conducted in two rounds, a qualifying round and a final round. All entrants that are not judged to break any of the SCML and ANAC submission rules will be entered into the qualifying rounds. Top-scoring agents in the qualifying round will then be entered into the final round.

The final results will be announced at IJCAI 2021. It is expected that finalists will send a representative to the ANAC workshop (at IJCAI 2021), where they will have the opportunity to present their agent.

A Simulation Parameters

The behavior of the simulator is controlled by the following hyperparameters. In this list, we first describe the hyperparameter, and then indicate its variable name in the SCML code base (in parentheses).

Number of simulation days (*n_steps*) $\in \mathbf{Z}^+_\infty$ The maximum number of simulation days in a single run.

Total simulation time (*time_limit*) $\in \Re^+_\infty$ The maximum number of seconds in a single run.

Reporting period (*reporting_period*) The number of days between periodic financial reports.

Negotiation rounds limit (*neg_n_steps*) $\in \mathbf{Z}^+_\infty$ The maximum number of rounds in a negotiation.

Negotiation time limit (*neg_time_limit*) $\in \Re^+_\infty$ The maximum number of seconds in a negotiation.

Offer Time Limit (*neg_step_time_limit*) $\in \Re^+_\infty$ The number of seconds between acceptable offers. If an offer is not received within this time limit, the negotiation ends.

Negotiation speed multiplier (*negotiation_speed*) $\in \mathbf{Z}^+_\infty$ The number of rounds in a negotiation per simulation day.

Table 1 lists the simulator parameters and their settings for SCML 2021 OneShot.

[12] An agent's truncated mean will be calculated by first sorting that agent's scores in all the simulations, and then removing the top and bottom x_t and x_b scores from that agent's sorted list, where x_t and x_b are values selected by the organizing committee to balance test efficiency (taking into account scores from as many simulations as possible) and robustness (insensitivity to outliers, or to a few simulations in which the agent realizes extremely high or low profits).

Table 1: Simulator parameter settings for SCML 2021 OneShot.

Setting	**Value**	**Notes**
Number of simulation days (S)	$50 < S < 200$	Based on available computational resources.
Total simulation time (in seconds)	7200	Two hours
Reporting period	5	
Negotiation Settings		
Negotiation rounds limit	20	
Negotiation time limit (in second)	120	Two minutes
Offer time limit (in seconds)	10	
Negotiation speed multiplier	21	All negotiations end the day they begin.
Negotiation Agenda Settings		
Trading price parameters	$\gamma = 0.9, Q_{-1}(p) = 50$	Control the evaluation of trading prices.
Price multiplier	$\kappa \sim U(1.5, 2.0)$	The negotiation price issue factor, multiplied/divided by current trading prices to set the limit.

B World Configurations

An SCML-OneShot tournament comprises multiple simulations, each one characterized by a **world configuration**, which in turn comprises the following:

1. A two-level supply chain, which consists of exactly three products and two corresponding manufacturing processes.
2. Some number of factories at both levels in the chain, with an assignment of agents (i.e., factory managers) to each factory.
3. Factory profiles such that each factory is characterized by some number of lines (λ, constant across all factories in OneShot 2021), a production cost m_a, an initial balance b_l, which is constant across all factories at the same level in the chain in OneShot 2021.
4. Exogenous buy contract parameters for the raw material and exogenous sell contract parameters for the final product.
5. Utility function parameters, including a mean and a variance, for all agents, of their disposal cost and shortfall penalty distributions.

All of the above configuration settings are determined by parameters that appear in Table 2, with the exception of the last (utility function parameters), which are described in Table 3.

The following is a simplified[13] sketch of the process used to generate a OneShot world configuration:

- Sample the simulator parameters listed in Tables 2 and 3.[14]
- Generate catalog prices for each product ρ (produced at level $l = \rho - 1$) as the sum of the input and production costs: for all $\rho \in \{1, \ldots, L+1\}$, $\text{cp}_\rho = (\text{cp}_l + \mu_\rho)(1 + \pi_l)$, where $\mu_\rho = \frac{1}{|\mathcal{A}|_l} \sum_{a=0}^{|\mathcal{A}|_l} m_a$.
- Set the total number of active lines per process/level on all days $d \in [0, S-1]$, given the production capability factor (productivity per process) $\eta_l(d)$, to be $A_l(d) = \lfloor \lambda_l \eta_l(d) \rfloor$. Consequently, the total production capacity for product ρ (produced at level $l = \rho - 1$) on day d is $Q_\rho(d) = \min\{Q_l(d-1), A_l(d)\}$, for all $\rho \in \{1, \ldots, L+1\}$, and $Q_0(d) = A_0(d)$.

[13] See the `SCML2020OneShotWorld.generate()` method for more details.
[14] U denotes the uniform distribution and $\mathcal{N}$, the normal distribution.

Table 2: Simulator parameters for SCML world generation. We write l for manufacturing process/production level, ρ for product, a for agents/factory managers, and d for days.

Setting	**Distribution**	**Notes**
Number of processes (levels)	$L = 2$	Number of processes/levels. The actual number will depend on the number of participants, and will vary between simulations.
Number of factories per process (level) l	$\lvert\mathcal{A}\rvert_l \geq 4$, for all $l \in \{0, \dots, L\}$	Number of factories that can execute each process. The actual number will depend on the number of participants, and will vary between simulations.
Production cost at process (level) l	$m_l \sim l \times U[1, 10]$	Production costs increase with level. They are higher for intermediate products closer to the finished product, and lower for those closer to the raw material.
Profit per process (level) l	$\pi_l \sim \mathcal{N}\left(U[0.1, 0.2], 0.05\right)$	The profit achievable if all factories exchanged products at catalog prices and produced at maximum capacity, assuming they all incurred the average production cost.
Productivity per process (level) l	$\eta_l(d) \sim U[0.8, 1.0]$, for all $d \in \{0, \dots, S-1\}$	The fraction of production lines per process that are assumed occupied when generating the configuration (sampled independently for each process and day).
Number of lines per factory a	$\lambda_a = 10$	Number of lines per level $\lambda_l = \sum_{a=0}^{\lvert\mathcal{A}\rvert_l} \lambda_a$.
Raw material catalog price	$\text{cp}_0 = 10$	Raw material catalog price, on which all other catalog prices depend.
Cash availability	$\xi \sim U[1.5, 2.5]$	When $\xi > 1$, the cash injected into the simulator beyond a base amount, which is what would be required for each factory to execute its manufacturing process to produce an average quantity assuming average production costs and catalog prices.
Price relative std. dev. per product ρ	$\sigma_\rho \sim U[0.1, 0.2]$	The fraction of the mean that yields the standard deviation from which exogenous contract prices are sampled.

Table 3: Parameters for OneShot utility function generation. All of these paramters are per agent.

Setting	Distribution	Notes
Production costs	$m_a \sim U[m_l, 4 \times m_l]$	Per-agent production costs
Disposal Cost Distributions†	$\mu_a^\alpha \sim U(0.0, 0.2)$ $\sigma_a^\alpha \sim U(0.0, 0.02)$	Per-agent disposal cost distributions
Shortfall Penalty Distributions†	$\mu_a^\beta \sim U(0.2, 1.0)$ $\sigma_a^\beta \sim U(0.0, 0.1)$	Per-agent shortfall penalty distributions
Day d's disposal costs and shortfall penalties	$\alpha_a(d) \sim \mathcal{N}(\mu_a^\alpha, \sigma_a^\alpha \mu_a^\alpha)$ $\beta_a(d) \sim \mathcal{N}(\mu_{a\beta}, \sigma_a^\beta \mu_a^\beta)$	Sampled per agent per day
† *If sampled disposal costs or shortfall penalties are negative, the corresponding absolute value is used instead, which amounts to slightly increasing the probability of small values for each.* • *Utility function parameters are subject to change.*		

- Compute the endowments for the factories at level l (producing product $\rho = l + 1$) as follows: for all $l \in \{0, \ldots, L-1\}$,

$$b_l = \xi \left(\frac{\text{cp}_l + \mu_\rho}{|\mathcal{A}|_l} \right) \sum_{d=0}^{S-1} Q_\rho(d)$$

  This initial balance is intended to be sufficient so that each factory can cover the cost of producing an average quantity of the product at its level for the duration of the simulation, even if it never sells anything. More specifically, in this calculation, each factory is assumed to buy $Q_\rho(d)/|\mathcal{A}|_l$ inputs at catalog prices cp_l, and produce that same quantity of outputs at an average cost μ_ρ, each and every day of the simulation. For $\xi > 1$, this design ensures that the average factory will not go bankrupt, although it does not guarantee the same for any particular agent.

- The total quantity of the raw material across all exogenous buy contracts with a delivery date on day d is $Q_0(d)$, and the total quantity of the finished product across all exogenous sell contracts with a delivery date on day d is $Q_L(d)$. These totals are divided among the corresponding factories randomly, but in such a way that the fraction of exogenous contracts assigned to one factory relative to others at the same level remains consistent throughout a simulation. For example, one agent may have quantities hovering around 4, and another around 6, across all days in the simulation.

  The price is sampled from a normal distribution with mean equal to the product's catalog price cp_ρ and standard deviation $\sigma_\rho \text{cp}_\rho$. With the current settings, in which $\sigma_\rho \sim U[0.1, 0.2]$, the standard deviation will be somewhere between 10% and 20% of the mean for each product.

C Tournament Generation

This section describes the process of running a tournament in more detail. Note, however, that these details are subject to change without notice.

Note: You can safely skip this section if you are not interested in these details. *The main takeaway is: when you design your agents, you should not make any assumptions about the other agents in the world.*

We start by differentiating between two concepts:

Basic Configuration A world configuration up to the assignment of agents to factories.

Assigned Configuration A basic configuration with all factories assigned to agents (i.e, factory managers)—that is, a world configuration.

We assume a set of C competitors denoted by $\mathcal{C} = \{0, \ldots, C-1\}$. In general, the number C is smaller than the number of submitted agents, as not all submitted agents will participate in all simulations. On the

contrary, the tournament will be run in a round-robin fashion, with, for example, $C = 3$, so that only three submitted agents partake in each simulation. The remaining agents, if any, will be default agents designed by the SCML organizing committee. These default agents are designed to facilitate trade, and are included in the round robin to create a fair competition among various combinations of submitted agents.

We denote by A_i the number of copies of competitors (i.e., agents) of type $i \in \mathcal{C}$ included in any given simulation. Note that A_i is 1 in the OneShot track.

A basic configuration is generated as follows:

1. Simulation parameters are set as described in Section 6.
2. If this is a standard competition, A_i is set to 1; otherwise, it is a collusion competition, and $A_i \sim U[2, 4]$.
3. Draw a number of levels/processes $L \sim U(2, 5)$.
4. For each process/production level (l), draw a number of factories/agents $|\mathcal{A}|_l \sim U(2, x)$, where x is selected such that $|\mathcal{A}|_l L \geq A_i C$.
5. Generate the rest of a basic configuration, given L and $|\mathcal{A}|_l$, as per Appendix B.
6. Select $A_i C$ of the factories and call them the assignable factories. Partition this set into C sets of factories, $B_0, \ldots, B_{C-1}$, each of cardinality A_i.

Now that we have a basic configuration and a partition of the assignable factories, we generate A_i copies of each agent type i. The factory sets in the partition are then randomly matched with the agent types: e.g., agent type i might be assigned to factory set B_i, for all agent types $i \in \mathcal{C}$. The result of this process is a world configuration, which is simulated until completion $K \geq 1$ times.

The assignment of agent types to factory sets is then rotated one step (i.e., factories assigned to agent type i are assigned to agent type $i + 1 \mod C$), and the world is simulated again with this new assignment another K times. This process is repeated C times ensuring that every agent type is assigned to every factory set in the partition, and that each such assignment is simulated K times.

Let's walk through an example to clarify this process. Assume three competing agents (i.e., agent types), A_0, A_1, and A_2, are participating in the collusion league, so let's create three copies of each type. Now assume a world of 10 factories, distributed over three levels, L_0, L_1, and L_2, with three factories at level 0; two at level 1; and five at level 2. We partition these 10 factories into factory sets at random. For example, $B_0 = \{0, 3, 5\}$, $B_1 = \{1, 2, 9\}$, and $B_2 = \{6, 7, 8\}$, with one factory (4) leftover, to be assigned to the organizing committee's agents. Now the first set of K simulations will assign B_0 to A_0, B_1 to A_1, and B_2 to A_2; the second will rotate this assignment, so that B_0 is assigned to A_1, B_1 to A_2, and B_2 to A_0; and the third will rotate this assignment again, so that B_0 is assigned to A_2, B_1 to A_0, and B_2 to A_1.

The number of basic configurations and the number of simulations of each assigned configuration will be determined based on available computational resources. However, as per the aforementioned process, the number of assigned configurations for each basic configuration will always be equal to the number of competitors, C. If the number of submitted agents is actually C, not 3, and the competition is run with M agent types present in each simulation, not 3, then this process will be repeated $\binom{C}{M}$ times, for each possible choice of M agents among C competitors, leading to $KM\binom{C}{M} = \frac{C!K}{(M-1)!(C-M)!}$ simulations for each basic configuration. [15]

References

[1] T. Baarslag, K. Hindriks, C. Jonker, S. Kraus, and R. Lin, "The first automated negotiating agents competition (ANAC 2010)," in *New Trends in agent-based complex automated negotiations*, pp. 113–135, 2012.

[15] The number of agent types per simulation (M) can range between 2 and C.

[2] Y. Mohammad, E. A. Viqueira, N. A. Ayerza, A. Greenwald, S. Nakadai, and S. Morinaga, "Supply chain management world," in *International Conference on Principles and Practice of Multi-Agent Systems*, pp. 153–169, Springer, 2019.

[3] R. Aydoğan, D. Festen, K. V. Hindriks, and C. M. Jonker, *Alternating Offers Protocols for Multilateral Negotiation*, pp. 153–167. Springer International Publishing, 2017.

[4] A. Rubinstein, "Perfect equilibrium in a bargaining model," *Econometrica*, vol. 50, no. 1, pp. 97–109, 1982.

[5] Y. Mohammad, S. Nakadai, and A. Greenwald, "NegMAS: A platform for situated negotiations," in *Twelfth International Workshop on Agent-based Complex Automated Negotiations (ACAN2019) in conjunction with IJCAI 2019*, August 2019.