Supply Chain Management League

Automated Negotiating Agents Competition

SCML Organizing Committee:

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

March 3, 2020

Contents

The purpose of this document is to provide an overview of the Automated Negotiation Agent Competition (ANAC) Supply Chain Management League (SCML).[1] We summarize the rules of the game, and then we explain how to play: i.e., the mechanics of the tournament.

1 Overview

The SCM world simulates a supply chain consisting of multiple factories that buy and sell products from one another. The factories are represented by autonomous agents that act as factory managers. Each agent decides which other agents to buy and sell from, and then negotiates with them. Their goal is to turn a profit, and the agent with the highest profit (averaged over multiple simulations) wins.

The simulation proceeds in discrete time steps, which we refer to as days. During each day, multiple simultaneous negotiations transpire, and outputs are manufactured from inputs. The game is intended to further research on agent negotiation; as such the design emphasizes negotiation and de-emphasizes operations (e.g., scheduling).

Factories **Factories** in the SCM world convert **products** into other products by running **manufacturing processes** on their **production lines**. All processes take one day to complete. Factories store the inputs and outputs of manufacturing processes in their **inventories**, and their funds in their **accounts**.

Each factory has multiple production lines, each of which is assigned a profile specifying the cost at which it can execute the various manufacturing processes. In general, these costs can vary from factory to factory, may vary from line to line. In SCML2020, however, each factory will have a set of identical production lines, each of which can run only a single manufacturing process (i.e., all other processes will have infinite cost).

Factory costs are private information: i.e., no factory knows any other factory's costs.

Production Graph The **production graph** is assumed to be directed and acyclic, with products and manufacturing processes as its nodes. An edge from a product to a process node indicates that this product is an *input* to this process. An edge from a process to a product node indicates that this product is an *output* of this process. (Note that there are no edges between product or between process nodes.)

Figure 1 depicts a sample production graph for SCML2020. Observe that it is a **chain**. Because each factory can run only one manufacturing process in SCML2020, each can likewise be assigned to only one production level, corresponding to their particular process assignment.

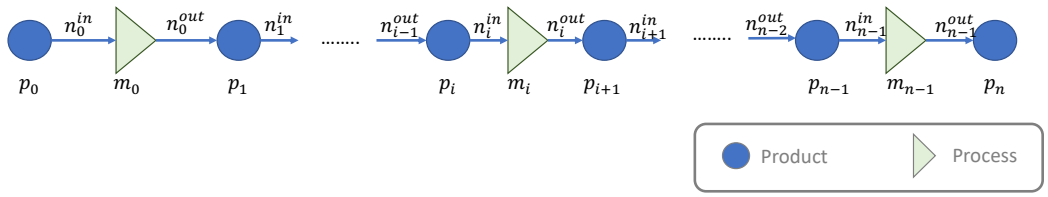


Figure 1: A sample production graph, with manufacturing processes m and products p. Numbers (denoted by n) on edges indicate the quantity consumed or produced by the corresponding manufacturing process.

Agents and Negotiation The agents in the SCM world function as **factory managers**. In addition to managing production, they negotiate with other agents to reach agreements to buy and sell products, which they can then sign as contracts. Such agreements are generated via bilateral negotiations using a variant of the **alternating offers protocol** typically used in ANAC competitions [1, 2]. Each offer specifies a buyer,

[1] We use SCM to refer to our simulation of a supply chain management world, SCML to refer to the league running at ANAC, and SCML20XX to refer to the league running during year 20XX.

a seller, a product, a quantity, a delivery time, and a unit price. The sequences of offers and counteroffers in a negotiation are private to the negotiating parties.

In SCML2020, agents negotiations are restricted such that they can only negotiate to buy inputs required for their own production, and to sell outputs produced by their own factory. No other negotiations are allowed; in particular, there are no "middle men" in the SCML2020 world.

The SCM world does not endow agents with utility functions. On the contrary, all utility functions are endogenous, meaning they are engendered by the simulation dynamics and agents' interactions with other agents. Endogenous utility functions are a distinguishing feature of SCML. It is an agent's job to assign utilities to potential contracts, given its unique production capabilities, and then to negotiate with other agents to secure those which are most favorable to them.

In SCML2020, agents consuming the raw material will be endowed with exogenous buy contracts but no exogenous sell contracts, while agents producing the finished product will be endowed with exogenous sell contracts but no exogenous buy contracts. No other agents will be endowed with any exogenous contracts. By design, *no agent can turn a profit without negotiating successfully*, since no agent is endowed with both exogenous buy and exogenous sell contracts.

Breach Processing When a contract comes due, the simulator tries to execute it (i.e., move products from the seller's inventory to the buyer's, and move money from the buyer's account to the seller's). If this execution fails, either because of insufficient funds on the part of the buyer, insufficient products on the part of the seller, a breach of contract occurs. Figure 2(a) depicts these two possible breach conditions. In both cases, the contract is executed to the extent possible, and the agent in breach of contract is penalized and reported to the breach list. (See Section 4.1 for details.)

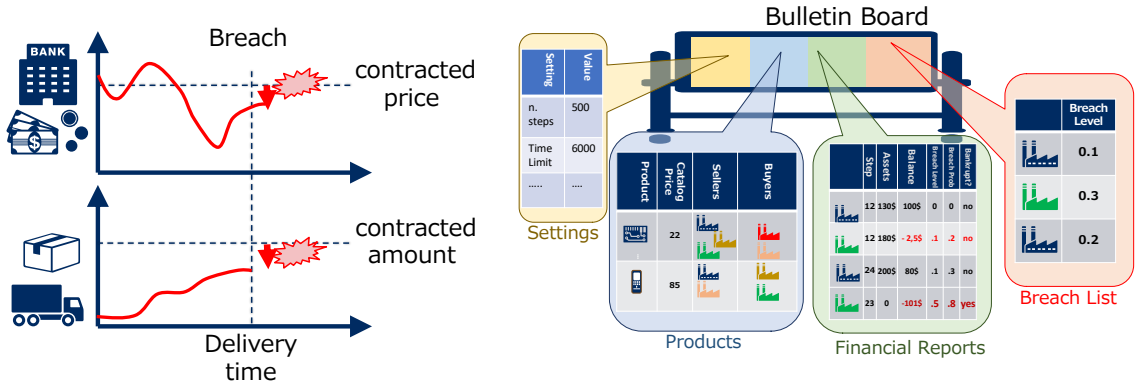
Bankruptcy Processing If an agent is unable to meet its financial obligations, it is declared bankrupt. The assets of bankrupt agents are liquidated, and their factories are closed (no further production can transpire). They can no longer participate in negotiations. The system takes over their outstanding contracts, and fulfills them to the extent possible. (See Section 4.2 for details.)

Spot Market The SCM world also simulates a spot market. Generally speaking, the spot market exists so that agents who would otherwise be in breach of contract for insufficient products (funds) can instead buy (sell) as necessary on the spot market at buy (sell) prices, which are always above (below) **trading prices**—an average over the historic prices at which products are traded. In SCML2020 specifically, sellers with insufficient products are forced to buy on the spot market, while buyers with insufficient funds are declared bankrupt immediately, in which case the system uses the spot market for liquidating the inventory of bankrupt agents. (See Section 4.3 for details.)

Bulletin Board The SCM world contains a world-readable **bulletin board** (see Figure 2(b)) that conveys both static and dynamic information about the game and all factories over the course of the simulation. The static information includes the game 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 **catalog prices**, one per product, which is a nominal price that represents the starting point of trading prices, and can also be used by agents to guide negotiations. The dynamic information includes a breach list, where breaches of contract are reported; and a financial news section, which is updated only periodically (except in the case of bankruptcy), that reports the financial standing of all factories.

Note that trading prices are not known to the agents. They are maintained by the system for use in breach and bankruptcy processing, and for valuing inventory at the end of a game.

The Simulation Each simulation of the SCM world runs for multiple (say, 1000) days. Before the first day, each agent is assigned a *private* manufacturing profile. In addition, the bulletin board is populated with the production graph information and catalog prices, an initial balance is deposited into each agent's account, and agents are endowed with exogenous contracts. Then, during each day:



(a) The two potential breach conditions: insufficient funds and insufficient products.

(b) The bulletin board includes Game Settings, Product Information, Financial Reports, and the Breach List.

Figure 2: Breach conditions and the bulletin board.

1. Agents can engage in multiple (say, 100) rounds of negotiations with their negotiating partners. They can also read the bulletin board, and request negotiations with other agents (for the next day).
2. All contracts that have come due 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.
3. The manufacturing processes on all lines in all factories are run: i.e., inputs are removed from inventory, outputs are stored in inventory, and production costs are subtracted from the factories' accounts.

2 Game Entities

In this section, we describe the components of the SCM world—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, what to sell, what to manufacture, and when to engage in all of these activities.

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. The specific manufacturing structure varies randomly from simulation to simulation, but each simulation's specific structure is posted on the bulletin board at the start.

Although the SCM world supports directed acyclic graphs,[2] the production graph used in SCML2020 is a **chain**, with but one raw material (p_0), but one finished product (p_{n+1}), and $n-1$ intermediate products, $p_1, \dots p_{n-1}$. Correspondingly, there is a set M of manufacturing processes, starting at process m_1 and ending at process m_n, with each intermediate process m_i for $i \in \{1, \dots, n\}$, taking as input one unit of product p_i, and producing as output one unit of product p_{i+1}.

In addition, there is a set of factories F, each of which is endowed with certain manufacturing capabilities. More specifically, each factory $f \in F$ is characterized by a manufacturing **profile** that consists of a set of lines L_f together with a cost function $C_f : L_f \times M \to \mathbf{Z}^+_\infty$, which indicates the cost of running process $m \in M$ on line $l \in L_f$.

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

Manufacturing processes consume their inputs at the *beginning* of the day on which they are executed, and produce their outputs at the end of this same day, making them available in the factory's inventory at the *beginning* of the next day.

In SCML2020, each factory is capable of running only one process in the production graph: i.e., the cost of running any processes other than that one is infinite. Moreover, the cost of running that process does not vary across a factory's multiple lines—although it does vary across factories. In addition, production becomes more and more costly, on average, the closer a product comes to being finished.

There is also a **spot market** in the SCM world, which in SCML2020 is used only by the system during breach and bankruptcy processing (see Sections 4.1 and 4.2). The buy (sell) price on the spot market is always above (below) the **trading price**, by an amount determined by the *spot market global penalty* and an agent's personalized *spot market penalty*, depending on its reliance on the spot market (see Section 4.3).

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 factory to manage.[3] The factory that the agent manages is announced publicly, but each factory's manufacturing profile is private information, known only to its factory manager. Throughout the simulation, an agent's state is described by the state of its factory (i.e., its scheduled jobs), its account balance, its inventory, a set of negotiated contracts, and a set of exogenous contracts. More specifically:

Factory state A list of manufacturing jobs J scheduled for execution in the agent's factory ($f \in F$). Each such job $j \in J$ is a tuple (t_j, l_j, m_j) indicating that the manufacturing process $m_j \in M$ should be initiated on line $l_j \in L_f$ on day $t_j \in \mathbf{Z}^+$. This list is maintained by the agent.

Inventory $S : P \to \mathbf{Z}_0^+$ The current inventory of all products at the agent's factory f. This quantity is maintained by the simulator.

Balance $B \in \Re$ The account balance accumulated by the agent's factory f. This quantity is maintained by the simulator.

Negotiated contracts C A list of contracts representing agreements with other agents. This list is updated by the simulator whenever two agents sign a contract.

Exogenous contracts E A list of exogenous buy and exogenous sell contracts. Exogenous contracts are private information, revealed h days in advance of a contract's delivery date (where $h \in \mathbf{Z}^+$ is the *exogenous contracts horizon*).

3 Negotiation

At a high-level, negotiating to reach an agreement, and then signing a contract, proceed as follows:

1. An agent contacts another, requesting a negotiation, and specifying the negotiation issues in an agenda.
2. If the request is accepted, negotiation proceeds via the alternating offers protocol (see Section 3).
3. If the negotiation successfully reaches an agreement, the agents are given the opportunity to make the agreement binding, by signing a contract.

Either party can cancel a successfully negotiated agreement without penalty by refusing to sign it. Giving agents the opportunity to either sign or not sign contracts is similar to the CONFIRM message of [3]. To reduce the load on the system, every agent is assigned a *negotiation quota, which is an upper bound on the number of negotiations it can request on any given day.*

[3] In SCML2020, each agent will manage only one factory. In future years, agents may manage multiple factories.

3.1 Mechanism

The negotiation mechanism adopted by SCML2020 [4] is a variant of Rubinstein's alternating offers protocol [5]. It involves two agents, who take turns making offers for a finite number of rounds and/or seconds. One agent open 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

This process repeats until either an agreement or a deadline is reached. To reach an agreement, both parties must accept the offer. If no agreement is reached by the deadline, the negotiation fails.

A key difference between the variant of the protocol used in the SCM world and other implementations of the alternating offers protocol is that in the first round of negotiations in the SCM world, *both* agents are asked to propose, and then one of these proposals is chosen at random to be the initial offer. Consequently, agents do not know whether they were the first to propose or not. This trick was designed to prevent the protocol from degenerating into an ultimatum game [6], which can happen in a simulation such as the SCM world, if the maximum number of rounds of negotiation is announced *a priori*. With this trick in place, however, the actual number of rounds can only ever be known up to a factor of ± 1.

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.

3.2 Data Structures

An agent invites other agents to negotiate with it by sending it a negotiation request, with a negotiation agenda. When one agent responds positively to another's request, a negotiation ensues.

A **negotiation agenda** ν is a tuple $(k_\nu, a_\nu, p_\nu, u_\nu, q_\nu, t_\nu)$, where:

Kind $k_\nu \in \{\textbf{buy}, \textbf{sell}\}$ Indicates whether the agent is looking to buy or sell.

Agent $a_\nu \in F$ The agent requesting the negotiation.

Product $p_\nu \in P$ The product.

Negotiation Issues The issues the agent wants to negotiate about, which can be any of the following:

Unit Price u_ν The unit price range.

Quantity q_ν The quantity range.

Delivery Time t_ν The delivery time range.

The unit price, quantity, delivery time are ranges of integers specifying a range of negotiable values. A single value (rather than a range) is used to indicating a non-negotiable issue.

An **offer** o is a triple (u_c, q_c, t_c) consisting of a unit price $u_c \in \mathbf{Z}^+$, a quantity $q_c \in \mathbf{Z}^+$, and a delivery time $t_c \in \mathbf{Z}^+$. Note that offers do not include any ranges corresponding to any issues; the precise value of each issue is fully specified.

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

Seller $s_c \in F$ The seller agent.

Buyer $b_c \in F$ The buyer agent.

Product $p_c \in P$ The product.

Offer o_c The agreed upon offer.

A contract also includes several bits indicating whether or not it has been signed, cancelled, or nullified. It is signed if both parties sign. It is cancelled if either party does not sign. It is nullified if either party goes bankrupt before it is executed. In case a contract has been signed, the signatories and the signing date are recorded. In case a contract has been cancelled or nullified, the relevant date is recorded.

An **agreement** is an unsigned contract which has not been cancelled or nullified. An **exogenous contract** (agreement) is a contract (agreement) in which one of the buyer or the seller is the system.

4 Contract Execution

When a contract becomes due, the simulator attempts to execute it. Successful execution involves transferring the agreed-upon quantity of products from the seller's inventory to the buyer's, and likewise transferring the agreed-upon value of the contract from the buyer's account to the seller's. In the event that a contract does not execute successfully, either because of insufficient funds or insufficient products, a **breach** occurs, and perhaps even a bankruptcy. In this section, we describe breach and bankruptcy processing.

4.1 Breach Processing

Buyers can commit a breach for *insufficient funds*, while sellers can commit a breach for *insufficient products*. The game is designed so that breaching a contract should lead to a financial loss.

Table 1: Breach Conditions and Levels. The maximum breach level is 1.

Cause	Criterion	Perpetrator	Breach Level
Insufficient funds	$b_c.f.W_f < u_c q_c$	Buyer b_c	$\frac{u_c q_c - b_c.f.W_f}{u_c q_c}$
Insufficient products	$s_c.f.S_f(p_c) < q_c$	Seller s_c	$\frac{q_c - s_c.f.S_f(p_c)}{q_c}$
$x.f$ is the factory associated with agent x. W_f is the balance in factory f's account (after any borrowing). S_f is factory f's inventory level. p_c, u_c, q_c are the product, unit price, quantity specified in contract c.			

Table 1 describes the circumstances of breaches and the levels that ensue. When an agent commits a breach (insufficient funds for a buyer, or insufficient products for a seller), a **breach report** is published on the bulletin board. Each breach report is a tuple (a, l, k), where $a \in A$ is the agent that caused the breach, $l \in (0, 1]$ is the breach level, with larger values corresponding to more severe breaches, and $k \in \{$*insufficient funds*, *insufficient products*$\}$ is the kind of breach committed.

In SCML2020, when an agent commits an insufficient products breach, it is forced to buy products on the spot market to make up for its shortfall. When an agent commits an insufficient funds breach, it is declared bankrupt immediately. But one insufficient funds breach by a buyer leads to bankruptcy. Likewise, but one insufficient products breach by a seller, which cannot be fulfilled at spot market prices—so leads to an insufficient funds breach—leads to bankruptcy.

4.2 Bankruptcy

If ever an agent commits an insufficient funds breach, so that its balance would become negative, it is declared bankrupt immediately. The bankrupt agent is prevented from engaging in any further negotiations, and its factory is closed (no further production can transpire).

A bankrupt agent's inventory is liquidated on the spot market at *sell* (i.e., relatively low) prices. Future contracts with a bankrupt agent are then transferred to the system, which uses the cash from the liquidation

to honor each of these contracts to the extent possible on the day they come due, buying products as necessary from the spot market at *buy* (i.e., relatively high) prices.

Finally, a **financial report** for a bankrupt agent is posted to the bulletin board immediately, and agents are informed through independent channels as well (see Section 7.2.1; **On Agent Bankrupt**).

4.3 Spot Market Prices

The spot market prices depend on catalog prices, past trading prices, and agents' individual spot market price penalties. The terms are intended to penalize those agents who rely on the spot market more often, more than those that don't.

The **trading price** (tp) for product p on day s is calculated as follows:

$$\text{trading price}(p,s) \equiv \text{tp}(p,s) = \frac{\beta^{s+1}\, Q_{-1}(p)\, \text{cat}(p) + \sum_{i=0}^{s} \beta^{s-i}\, Q_i(p)\, \mu_i(p)}{\beta^{s+1} + \sum_{i=0|Q_i(p)>0}^{s} \beta^{s-i}} \ , \tag{1}$$

where $\text{cat}(p)$ is the catalog price of product p, $\beta \in [0,1]$ is a discount factor (*trading price discount factor*), $Q_0(p)$ is a weight representing the *effective* quantity that is represented by the catalog price (*prior catalog price quantity*), $Q_i(p)$ is the total quantity of product p traded on day i (in contracts executed even partially on that day), and $\mu_i(p)$ is the average price per item at which product p traded at day i. More specifically,

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

where C^i is the set of all contracts executed (even partially) on day i, $c.p$ is the product traded via contract c, $c.u$ 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).

Agent a's *spot market price penalty* (ip; individual penalty) for product p on day s is computed as follows:

$$\text{spot market price penalty}_a(p,s) \equiv \text{ip}_a(p,s) = \lambda \sum_{i=0}^{s} \alpha^{s-i} q_a(p,i) \ , \tag{2}$$

where $\lambda \geq 0$ and $\alpha \in [0,1]$ are simulation parameters, and $q_a(p,i)$ is the total number of units of product p bought on the spot market by agent a on day i. Note that an agent's penalty for day s is calculated *before* it buys any products on the spot market. Consequently, the amount (about to be) bought impacts the spot market price. Further, buying a units incurs a higher penalty than buying b units, for $a > b$.

An agent a is forced to buy products on the spot market if it commits a product breach, at the price $\text{tp}(p,s) \times (1 + \text{gp}) \times (1 + \text{ip}_a(p,s))$. If agent a goes bankrupt, the system sells its inventory on the spot market, at the price $\text{tp}(p,s)/((1 + \text{gp}) \times (1 + \text{ip}_a(p,s))$, where gp is the *spot market global penalty*.

4.4 Examples

Breach Processing Suppose agent A's spot market price penalty for product p is 0.1, the spot market global penalty is 0.2, and p's trading price is 7 dollars. Further, assume a contract on day 3 that specifies that A should sell 10 units of product p to agent B at unit price of 5 on day 4. When day 4 arrives, A has only 6 units in inventory, while B has only 21 dollars in its account. As a result, A commits a $4/10 = 0.4$ level insufficient products breach and B commits a $29/50 = 0.58$ level insufficient funds breach. A is then *forced* to buy the missing 4 units on the spot market at a price of $\lceil 7 \times 1.2 \times 1.1 \rceil = 10$ dollars. As B can only pay 21 dollars, it is declared bankrupt with a final balance of $21 - 50 = -29$.

Bankruptcy Processing Consider once again the example above. In addition, suppose that B's inventory contains 100 units of product p (trading price 8.4), 110 units of product p' (trading price 14.4), and nothing more. Finally, assume B's spot market price penalty for p is 0 [4], and for p' is 0.5.

The bankruptcy procedure executes as follows.

First, B's spot market *selling* price for each product is calculated, which in our example is $8.4/1.2 = 7$ for p, and $14.4/(1.2\times1.5) = 8$ for p'. Then, everything in B's inventory is liquidated on the spot market (at a loss that depends on the agent's spot market price penalty), generating proceeds of $\lfloor 100 \times 7 + 110 \times 8 \rfloor = 1580$ dollars, leaving the system $1580 + 21 = 1601$ dollars with which to satisfy B's outstanding contracts—including the one that caused the bankruptcy—to the extent possible.

Next, this very contract—the one that caused the bankruptcy—is executed to the extent possible by the system on behalf of B. Now that A has in its possession 10 units, and the system holds 1601 dollars on behalf of B, their contract is executed normally, with 50 dollars transferred to A's account from "B's" (i.e., the system's account on behalf of B), and 10 units of p removed from A's inventory. **N.B.** The products received by the system in the execution of this contract are not used for future contracts. More generally, after liquidation, any further products or cash received by the system in the name of the bankrupt agent are destroyed; they are not be used to honor future contracts.

Third, the system fulfills B's other outstanding obligations to the extent possible, using the available cash to either buy or sell products, in the latter case only after first buying them on the spot market. Assume B has five other outstanding contracts when it goes bankrupt (day 4):

c_1 Sell 50 units of p' at a unit-price of 10 dollars to agent C on day 4.

c_2 Buy 70 units of p at a unit-price of 5 dollars to agent D on day 5.

c_3 Sell 50 units of p' at a unit-price of 10 dollars to agent C on day 5.

c_4 Buy 10 units of p at a unit-price of 20 dollars to agent D on day 6.

c_5 Buy 16 units of p at a unit-price of 4 dollars to agent D on day 7.

To value future sell contracts, the system uses the price for *buying* p' on the spot market on behalf of B, which is $\lceil 14.4 \times 1.2 \times 1.5 \rceil = 26$. The total value of B's future contracts is thus $50 \times 26 + 70 \times 5 + 50 \times 26 + 10 \times 20 + 16 \times 4 = 3214$ dollars.

The system cannot possibly honor all of these contracts with the 1551 dollars available. Contracts are honored in order of their delivery dates, which in this example is as follows. Contract c_1, which comes due on day 4, is executed normally, at a cost of $50 \times 26 = 1300$ dollars, leaving $1551 - 1300 = 251$ dollars. Contracts c_2 and c_3 (i.e., all the contracts that come due on day 5) cannot both be executed normally as they would require $70 \times 5 + 50 \times 26 = 1650$ dollars (> 251). As they cannot be fully executed, c_2 and c_3 are executed in order of their signing times, resolving ties randomly. Let's assume c_3 was signed before c_2. Contract c_3 can be partially executed with the quantity $\lfloor 251/26 = 9 \rfloor$, leaving $251 - 9 \times 26 = 17$ dollars. Contract c_2 can then be partially executed with the quantity $\lfloor 9/5 \rfloor = 1$, leaving $9 - 5 \times 1 = 4$ dollars. The remaining 4 dollars cannot be used to satisfy any part of contract c_4, so c_4 is nullified. Finally, contract c_5 is partially exectued with a quantity of $\lfloor 4/4 \rfloor = 1$. At this point, all the liquidation money has been consumed, so all remaining contracts, if any, are nullified.

Once this fulfillment schedule is determined, agent C is informed (immediately) that c_1 will execute normally, and that c_3 is nullified but has been replaced with a new contract c_3', which is identical to c_3, except for the reduced quantity. Likewise, agent D is informed (immediately) that contract c_4 is nullified, and that c_2 and c_5 are also nullified, but have been replaced with new contracts c_2' and c_5', which are identical to c_2 and c_5, except for the reduced quantities. All other agents (e.g., A) are informed (immediately) that B is bankrupt via their **On Agent Bankrupt** callbacks.

[4] Because p is the input product of B, it is guaranteed that it never bought it on the spot market before.

5 Financial Reports

Summaries of agents' financial status are published periodically (every *report-period* steps). These reports contain the following information about agents' cash, inventory, and past breach behavior:

Balance The agent's balance. (A negative balance indicates a bankrupt agent.)

Inventory The value of the agent's inventory, evaluated at catalog prices.

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

Breach Level The agent's average breach level.

6 Simulation Steps

Figure 3: Order of execution of events each simulation day.

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.[5] After initialization, the simulator repeats the following loop (Figure 3), which calls the agents' step functions, among other things:

1. Run all registered negotiations for *negotiation-speed-multiplier* rounds. Concluded negotiations are then registered for signing after *signing-delay* days. Moreover, exogenous contracts with delivery dates in the upcoming *exogenous contracts horizon* days are registered for signing.
2. Sign (or not) all contracts registered for signing today (including exogenous ones).

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

3. Execute all contracts due to be executed today, by delivering products from the sellers to the buyers, and transferring money from the buyers to the sellers. For those contracts that cannot be executed, handle any potential breaches and report any bankruptcies.

4. Call the step functions of all agents in an unspecified order. Doing so should trigger new negotiation requests, potentially leading to new negotiations the following day.

5. Update the trading prices of all products and the per-agent spot market price price penalties.

6. Simulate one day of production for all manufacturing processes running on all lines in all factories. Move all completed products to inventory.

7. Publish financial reports (every *trading period* days).

There are a few details pertaining to timing worth mentioning:[6]

- To remember the order of events, you can think of poor negotiators meeting in the early morning and working hard to make agreements (including exogenous ones). The CEOs then meet to sign only the agreements they like, which become contracts, while having breakfast (on their yacht). Trucks deliver products and money is transferred (i.e., contracts are executed) before noon (to avoid road congestion). The CEO then messages the negotiators (mid-golf game) with instructions about the next day's negotiations. Factories then run until late into the evening. Financial reports are published after the factories shut down for the night, weekly on Fridays.

- Since products are manufactured *after* contract execution, an agent can never sell its output products on the same day they are manufactured. Nevertheless, it *can* use input products in production on the same day they are bought. Similarly, it can use funds earned from sales for production on the same day products are sold.

7 The SCML Platform

Like SCML2019, SCML2020 will run on top of NegMAS [7], which is a Python framework for developing autonomous negotiation agents embedded in simulation environments. The SCML2020 simulation was developed from the ground up, which means that agents developed for SCML2019 will need to be adapted before they can participate in SCML2020. Going forward, we hope/expect future iterations of SCML to be interface-compatible with agents developed for SCML2020.

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.

[6] These design choices maintain a clean separation between the underlying NegMAS platform and the SCML implementation.

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.

7.2.1 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:

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

On Step Called in the simulation loop. Simulates one step of the agent's negotiation logic.

The next set of callbacks are event driven; they are triggered by the events their names suggest:

Respond to Negotiation Request Called whenever another agent requests a negotiation with this agent. This agent can agree to negotiate or not. If the agent agrees, a new negotiation is registered.

Sign All Contracts Called by the simulator to allow the agent to sign all negotiated agreements, so that they become binding contracts. In addition, potential exogenous agreements are revealed to the agent in this callback, which like negotiated agreements, the agent is free to sign or not.

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

On Contracts Finalized Called with the set of signed and cancelled contracts. A contract is signed if both parties sign. It is cancelled if either party does not sign.

On Agent Bankrupt Called when an agent is declared bankrupt to inform all other agents. Agents that are party to any future contracts with the bankrupt agent receive a list of contracts that will be nullified (i.e., cancelled and never executed), and those that will be partially executed (i.e., a smaller than originally agreed upon nonzero quantity will be traded at the delivery time).

On Failure Called when production fails due to lack of requisite inputs or funds.

7.2.2 Actions

Agents control their negotiations and finances using the following actions:

Request Negotiation Sends a negotiation request to another agent. The requesting agent must specify a **negotiator** to use for this negotiation.

Request Negotiations Sends a list of negotiation requests to a list of agents. The requesting agent must specify a list of negotiators, or a **controller**, to use for this negotiation.

Schedule Production Adds a manufacturing process to a specified line to start on a specified day. If a line is not given, the factory uses any available line. If the day is given as a range, the factory uses a day in this range according to the *scheduling method*, which either schedules production on the latest possible day, or on the earliest.

Cancel Production Cancels scheduled production.

Set Production Sets production for a given step (usually current) at all lines.

Agents can also gather information about their factory and other agents by accessing the Agent-World-Interface. Available methods are:

Get State Reads the factory state.

Available For Production Returns the production slots (day/line) available for production.

Bulletin Board Access the bulletin board to read product information, financial reports, the breach list, or simulation settings (e.g. number of days, current day, etc).

8 Tournament Mechanics

How to Participate To participate in the Supply Chain Management League (SCML), all you need to do is write and submit code for an autonomous agent that acts as a factory manager. While the production graph will be a chain in SCML2020, with agents managing but one factory with identical lines, your agent should be robust enough to manage any such factory with any manufacturing profile (i.e., any factory assignment and production cost), because its particular profile will vary from simulation to simulation.

How to Compete There will be two separate tracks in SCML2020. All agents will be run in both tracks. In the *standard* track, at most one instantiation of each team's agent will run in each simulation, together with an unknown mix of agents prepared by other participants and agents prepared by the organizing committee.

In the *collusion* track, multiple instantiations of the same team's agent will run during a single simulation. The exact number of instantiations of each will vary across simulations, and will not be announced in advance. In this track, it is perfectly legal for instances of the same agent to collude with one another to try to corner the market, or exhibit other collusive behaviors.

How to Win An agent's performance will be measured by its score. An agent's score will be the *median*[7] of the profits accrued by all the factories it is assigned to manage across all simulations.

The profit accrued by a factory during one simulation is calculated as follows:

$$\text{Profit} = \frac{B_N + 1/2\, I_N - B_0}{B_0}, \tag{3}$$

where B_0 and B_N are the agent's balances at the beginning and end of the simulation, respectively, and I_N is the value of the products in the agent's inventory at the end of the game. This value is based on the trading price (see Equation 1), but to incentive trade, inventory is valued at only *half* the trading price; that way, it is more profitable on average to sell products rather than hoard them.

The two tracks 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 in the final round.

The final results will be announced at IJCAI 2020. It is expected that finalists will send a representative to the ANAC workshop at IJCAI 2020, where they will be given the opportunity to give a brief presentation describing their agent. Two awards will be announced at IJCAI 2020 (with associated monetary rewards) corresponding to the two tracks (standard and collusion).

[7] In SCML2019, the agent's score was the mean.

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.

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

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

Exogenous contracts horizon (*exogenous_contracts_horizon*) Maximum number of days in advance of an exogenous contract's delivery date when it is revealed to an agent, given as a percentage of the total number of days in the simulation. For example, a value of 10% in a 500 day game means that all exogenous contracts that arrive on a given day have a delivery date at most 50 days in the future.

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

Table 2 lists the simulator parameters and their settings for SCML2020.

Table 2: Hyperparameter settings for SCML2020.

Setting	**Value**	**Notes**
Number of simulation days (S)	$50 < S < 200$	Based on available computational resources.
Total simulation time (in seconds)	7200	Two hours
Exogenous contracts horizon	$\lceil \sim U(0.1, 0.4)\, S \rceil$	Sampled uniformly, between 10% and 40% of the number of simulation days.
Reporting period	5	
Negotiation Settings		
Negotiation time limit (in second)	120	Two minutes
Negotiation rounds limit	20	
Negotiation quota	$> b \times h$	At least the number of potential partners (b) $\times$ the exogenous contract horizon (h).
Offer time limit (in seconds)	10	
Negotiation speed multiplier	21	All negotiations end the day they begin.
Spot Market Parameters		
Trading price parameters	$\beta = 0.9, Q_{-1}(p) = 50$	Equation 1
Global penalty	0.15	
Per-agent penalty parameters	$\lambda = 0.1, \alpha = 0.9$	Equation 2

B World Configurations

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

1. A supply chain (i.e., a production graph; see Figure 1) with nodes representing products and manufacturing processes/production levels. Product nodes are annotated with their catalog prices.

2. Some number of factories at each level 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 (ν, constant across all factories), a production cost c_f, and an initial balance b_l, which is constant across all factories at the same level in the chain. Each factory executes exactly one manufacturing process at one and only one level in the chain; at all other levels, c_f is infinite.

4. Exogenous buy contracts for the raw material and exogenous sell contracts for the final product.

The following is a simplified[8] sketch of the process used to generate an SCML2020 world configuration:

- Sample the hyperparameters listed in Table 3 from the specified distributions.[9]

- Generate catalog prices for each product p (produced at level $l = p - 1$) as the sum of the input and production costs: for all $p \in \{1, \ldots, L+1\}$, $g_p = (g_l + \mu_p)(1 + \pi_l)$, where $\mu_p = \frac{1}{F_l} \sum_{f=0}^{F_l} c_f$.

- Set the total number of active lines per process/level on all days $d \in \{0, \ldots, S-1\}$, given the production capacity factor $\eta_l(d)$, to be $A_l(d) = \lfloor \nu_l \, \eta_l(d) \rfloor$. Consequently, the total production capacity for product p (produced at level $l = p - 1$) on day d is $Q_p(d) = \min\{Q_l(d-1), A_l(d)\}$, for all $p \in \{1, \ldots, L+1\}$, and $Q_0(d) = A_0(d)$.

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

$$b_l = a \left(\frac{g_l + \mu_p}{F_l} \right) \sum_{d=0}^{S-1} Q_p(d)$$

  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 $Q_p(d)/F_l$ inputs at catalog prices g_l, and produce that same quantity of outputs at an average cost μ_p, each and every day of the simulation. For $a > 1$, 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 (i.e., not necessarily evenly).

  The quantity available to a factory f on day d is further divided into a set of n_f contracts, whose number increases with the exogenous contract controllability parameter (e). Each of these contracts is revealed to the relevant agent at most *exogenous-contracts-horizon* days before its delivery date.

  The unit price of all exogenous contracts for product p is set to the catalog price (g_p).

[8] See the `SCML2020World.generate()` method for details.

[9] U denotes the uniform distribution and $\mathcal{N}$, the normal distribution.

Table 3: Hyperparameters for SCML world generation. We write l for manufacturing process/production level, p for product, f for factories, 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	$F_l \geq 2$, 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	$c_l \sim l \times U[1, 10]$	Production costs increase with level, so they are higher for intermediate products closer to the finished product, and lower for those closer to the raw material.
Production cost per factory f	$c_f \sim U[c_l, 10 \times c_l]$	Production costs per factory range widely.
Number of lines per factory f	$\nu_f = 10$	Number of lines per level $\nu_l = \sum_{f=0}^{F_l} \nu_f$.
Raw material catalog price	$g_0 = 10$	Raw material catalog price, on which all other catalog prices depend
Productivity per process (level)	$\eta_l(d) \sim U[0.8, 1.0]$, 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)
Profit per process (level)	$\pi_l \sim \mathcal{N}(U[0.1, 0.2], 0.05)$	The profit achievable if all factories exchanged products at catalog prices and produced at maximum capacity, assuming they all incurred the average production cost.
Cash availability	$a \sim U[1.5, 2.5]$	When $a > 1$, the cash injected into the system 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.
Exogenous Controllability	$e \sim U[0.2, 0.8]$	Controls how many exogenous contracts per day will be received by factories at the ends of the chain. Larger values mean more contracts, giving the agent more fine-tuned control over the quantities it can sign for.

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 $\mathcal{C} = \{0, \dots, C-1\}$. 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 standard track, but exceeds 1 in the collusion 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 $F_l \sim U(2, x)$, where x is selected such that $F_l L \geq A_i C$.
5. Generate the rest of a basic configuration, given L and F_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, $B_0, \dots, B_{C-1}$, 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 generated 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 leading to $KM\binom{C}{M} = \frac{C!K}{(M-1)!(C-M)!}$ simulations for each basic configuration.[10].

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.

[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] C. R. Williams, V. Robu, E. H. Gerding, and N. R. Jennings, "Negotiating concurrently with unknown opponents in complex, real-time domains," in *Proc. of the Twentieth European Conference on Artificial Intelligence*, pp. 834—-839, 2012.

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

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

[6] W. Guth, R. Schmittberger, and B. Schwarze, "An experimental analysis of ultimatum bargaining," *Journal of Economic Behavior & Organization*, vol. 3, no. 4, pp. 367–388, 1982.

[7] 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.

[10]The number of agent types per simulation (M) can be anywhere between 2 and C.