SCM World

Automated Negotiating Agents Competition

SCM League Organizing Committee:

Yasser Mohammed
Amy Greenwald
Katsuhide Fujita
Satoshi Morinaga
Shinji Nakadai

April 1, 2019

Contents

Abstract

The SCM league models a real-world scenario characterized by profit-maximizing agents that inhabit a complex, dynamic, negotiation environment. It was created with the intent of increasing the relevance of research on autonomous agent negotiations, by going beyond **context-free** negotiation scenarios, where agents decide only how to act in a single static negotiation. Specifically, our aim to further research on the design of **context-aware, situated** agents, who must decide *about what*, *with whom*, and *when* to negotiate, as well as how to best coordinate their actions across multiple concurrent negotiations.

Another distinguishing feature of the SCM league is the fact that agents' utility functions are endogenous, meaning they are the product of the market's evolution, and hence, cannot be dictated to agents in advance of running the simulation. It is an agent's job to devise their utilities for various possible agreements, given their unique production capabilities, and then to negotiate with other agents to contract those that are most favorable to them. Under these conditions, a major determiner of an agent's wealth, and hence, their final score, will be their ability to position themselves well in the market and negotiate successfully.

Overview

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

The SCM World

The world modeled by the SCM league consists of five types of entities: factories, miners, consumers, a bank, and an insurance company. In more detail:

Factories Entities that convert raw materials and intermediate products into intermediate and final products by running their manufacturing processes for some time, assuming all inputs and enough time are available to run the processes. Factories have accompanying warehouses in which to store the inputs and outputs of their manufacturing processes.

Different factories are endowed with different capabilities, specified as *private* production profiles, known only to the factory itself (and its manager agent). For example, two factories may produce the same product using the same inputs but at different costs and at different time scales: e.g., one may produce them faster, but at a higher cost.

Factories negotiate as both buyers and sellers in the SCM world.

Miners Facilities capable of mining raw materials as needed to satisfy their negotiated contracts. Miners act only as sellers in the SCM world.

Consumers Companies interested in consuming a subset of the final products to satisfy some predefined consumption schedule. Consumers act only as buyers in the SCM world.

Bank A single loan provider that provides loans to potential buyers.[1]

Insurance Company A single insurance company that can insure buyers against **breaches of contract** committed by sellers (i.e., failure to deliver promised products on time), and vice versa (i.e., insufficient funds in the buyer's wallet at the time of delivery to pay the seller).

Note that there are no transportation companies. Consequently, the SCM league at present is ignoring logistics. Instead, it is assumed that all products can be transported between all entities free of charge, after a predefined constant delay. In addition, warehouse capacity is currently assumed to be infinite.

Agents In the SCM world, each type of entity is run by a **manager** agent. The organizing committee will provide manager agents for miners, consumers, the bank, and the insurance company. It is the job of the participating teams to develop a **factory manager** agent. The goal of each factory manager agent is to accrue as much wealth (i.e., profit) as possible. The organizing committee will also provide a default agent: i.e., a **greedy factory manager**, instances of which will participate in the competition to ensure sufficiently many trading opportunities. Additionally, participants can base the development of their own agents on this base agent. The buying and selling agents in the SCM world are depicted in Figure 1.

Negotiation Mechanism Figure 2 depicts the role of negotiations in the SCM world. Miner, factory manager, and consumer agents can all buy and sell products based on agreements they reach, and then sign as contracts. Such agreements are generated via bilateral negotiations using a variant of the **alternating offers protocol** typically used in ANAC competitions. The sequences of offers and counteroffers in these negotiations are private to the negotiators.

[1]The bank is disabled in the 2019 SCML competition.

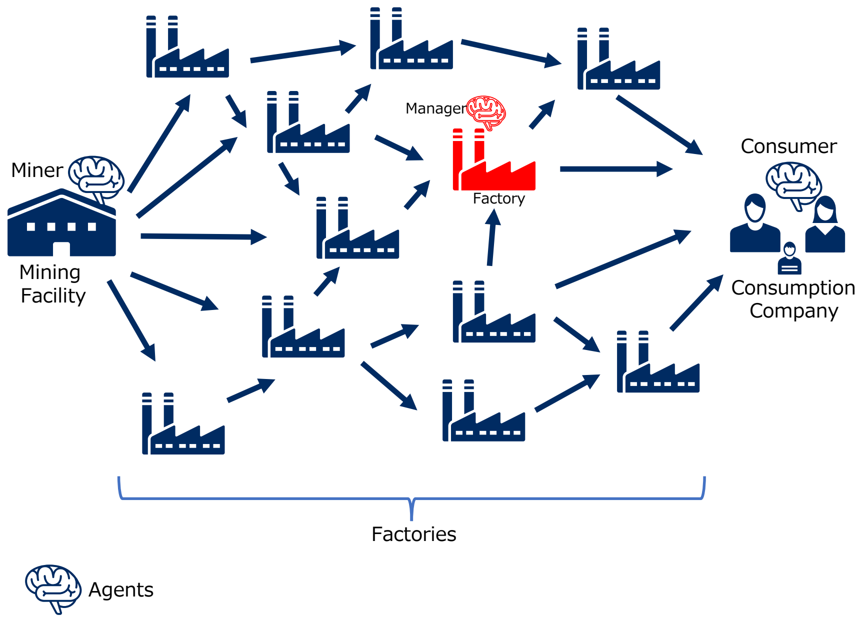


Figure 1: The SCM world: mining facilities (controlled by miners), consumption companies (controlled by consumers), and factories (controlled by factory manager agents). Participating ANAC teams develop agents that act as factory managers. Arrows represent the flow of products as they are bought and sold.

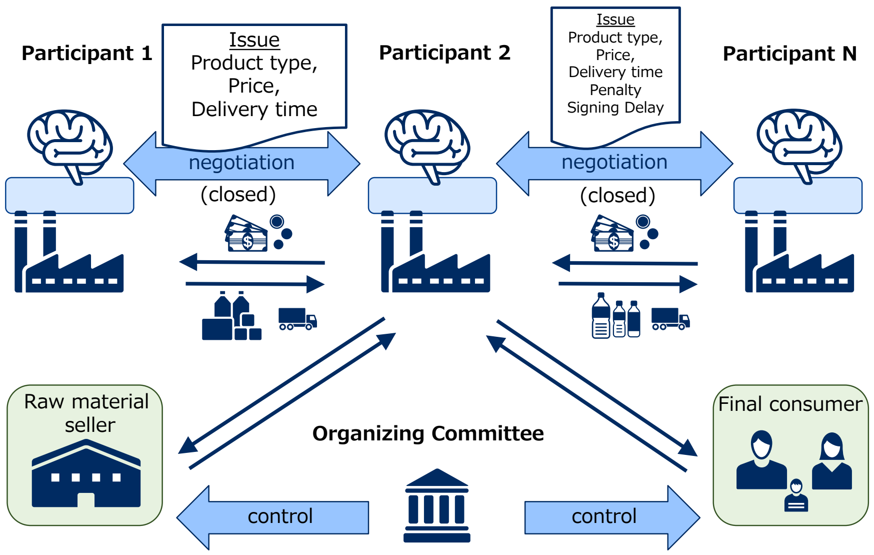


Figure 2: The role of negotiation in the SCM world. Agreements are only reachable through closed (i.e., private) bilateral negotiations using the alternating offers protocol.

An offer must specify a buyer, a seller, a product, a quantity, a delivery time, and a unit price. Optionally, an offer can also specify a grace period for converting the offer into a contract, during which time either party may opt out without penalty; and a penalty term, which would be incurred by the seller in the event they commit a breach of contract.

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 wallet to the seller's). If this execution fails, a breach of contract can occur. Breaches can also occur if either party decides not to honor the contract. (Figure 3 depicts the two potential breach conditions.) In all cases of potential breaches, the simulator offers the agents involved the opportunity to renegotiate.

To find negotiation partners, agents publish their interest in negotiating on a bulletin board that lists **call-for-proposals** (CFPs). Each such CFP specifies the publisher and the proposed negotiation issues. Interested agents then request a negotiation. Initiating such negotiation implies acceptance of the negotiation agenda (i.e., the proposed negotiable issues). If an agent instead does not accept the negotiation agenda, they can publish their own CFP listing alternative negotiation issues.

Public Information Contracts are private; they are not posted on the bulletin board. However, whenever renegotiation fails after a breach of contract, the breach is published on the bulletin board, on a **breach list**, which indicates the perpetrator and the level of the breach. If an agent goes bankrupt, this news is published on the bulletin board, on a **bankruptcy list**.

In addition to the breach list and the bankruptcy list, which may help agents decide who *not* to trade with, **quarterly reports** are also published on the bulletin board listing, for each factory manager agent, their assets, including their balance and the value of the products in their inventory (valued at catalog prices), and their liabilities, including the value of their outstanding loans, and their credit rating.

The Simulation Each simulation of the SCM world runs for multiple (say, 100) steps. Before the first step, each factory agent is assigned a (private) manufacturing profile. In addition, catalog prices for all products are posted, and an initial balance is deposited into each agent's wallet. Then, during each step:

1. Factories make their loan payments. All contracts that come due are executed, and any breaches that arise are handled.

2. Factory manager agents then engage in negotiations for multiple steps (say, 10). During this time, they are also free to read the bulletin board, post CFPs, and respond to CFPs.

3. Finally, all production lines in all factories advance one time step, meaning required inputs are removed from inventory, generated outputs are stored in inventory, and profile-specific production costs are subtracted from the factories' balances.

Utility Functions The SCM world does not endow agents with utility functions. On the contrary, all utility functions are endogenous, meaning they are engendered by the simulator's dynamics and agents' interactions with other agents. Endogenous utility functions that arise as the market evolves are a distinguishing feature of the SCM league. It is an agent's job to design their utilities for various possible agreements, given their unique production capabilities, and then to negotiate with other agents to contract those that are most favorable to them. Under these conditions, a major determiner of an agent's wealth, and hence, their final score, will be their ability to position themselves well in the market and negotiate successfully.

Demand-driven Markets The current rendition of the SCM world is demand driven, meaning proactive consumers drive demand by posting buy CFPs. Default factory manager agents react by responding to the consumers' buy CFPs (offering to sell), and then post their own buy CFPs further down the chain. Miners at the other end of the chain are similarly reactive.

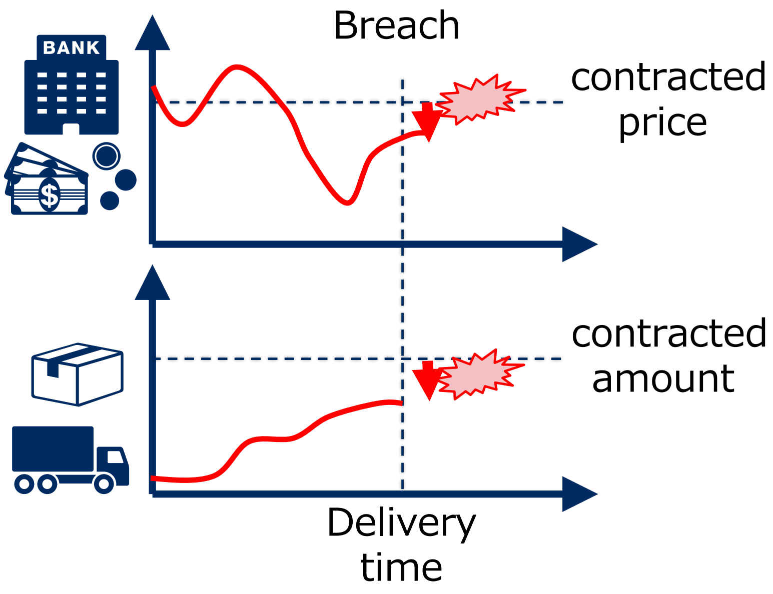
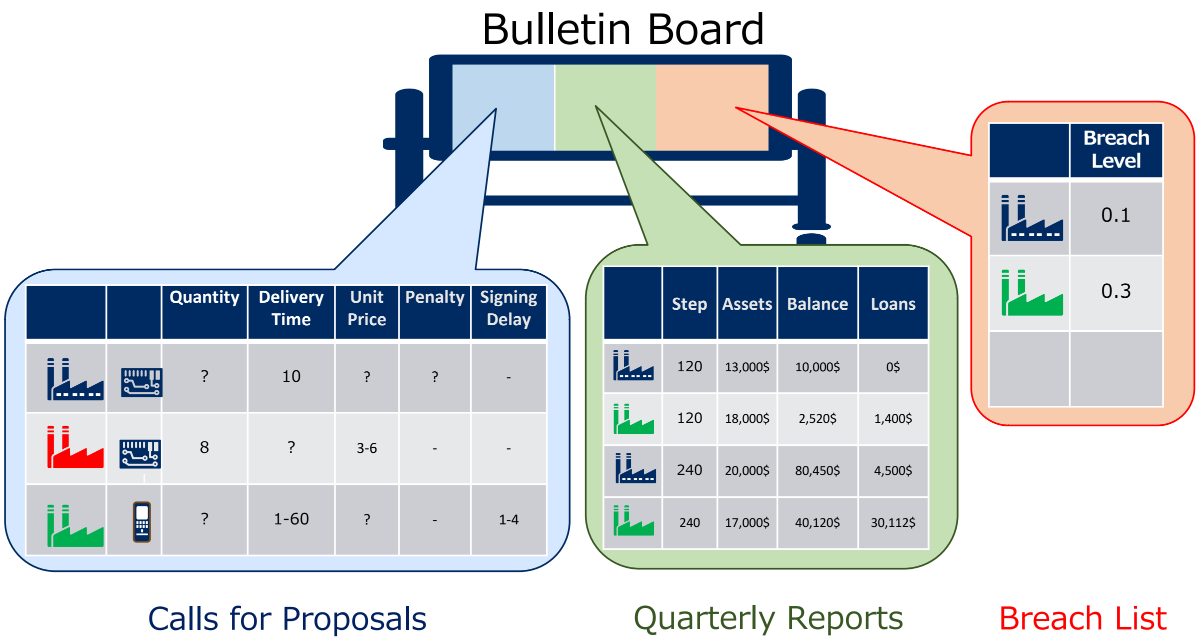


Figure 3: The two potential breach conditions: insufficiently many products and insufficient funds.



		Quantity	Delivery Time	Unit Price	Penalty	Signing Delay
🟦	📟	?	10	?	?	-
🟥	📟	8	?	3-6	-	-
🟩	📱	?	1-60	?	-	1-4

	Step	Assets	Balance	Loans
🟦	120	13,000$	10,000$	0$
🟩	120	18,000$	2,520$	1,400$
🟦	240	20,000$	80,450$	4,500$
🟩	240	17,000$	40,120$	30,112$

	Breach Level
🟦	0.1
🟩	0.3

Figure 4: The bulletin board: the Call for Proposals, the Quarterly Reports, the Breach List.

We call the current SCM world design a **pull economy**. The opposite extreme would involve proactive miners at one end of the chain, and reactive consumers at the other: i.e., a **push economy**. We expect future renditions of the SCM league to involve some combination of pulling and pushing.

ANAC 2019 SCM League The ANAC 2019 SCM league utilizes a simplified version of the SCM world in which there is a single raw material, a single final product, all factories have identical production lines, and no loans are allowed (i.e., the bank is disabled), among other simplifications. See Chapter 7 for details.

Tournament Mechanics

How to Participate All you need to do to participate in the SCM league is write and submit code for an autonomous agent that acts as a factory manager. While the structure of the production graph will be fixed for the 2019 SCM league, your agent should be robust enough to handle any manufacturing profile, because its particular profile will not be revealed until seconds before each simulation begins.

In addition to agents submitted by the community, the SCM world will be inhabited by miner, consumer, and default factory manager agents designed by the organizing committee. The organizing committee will provide a description of the behavior of these agents, including the miners' and consumers' (exact) utility functions, the factory managers' scheduling algorithm, and an estimation method for the factory managers' utility functions. The negotiation algorithms that the organizing committee's agents employ will not be announced, but will likely be standard choices, such as tit-for-tat or aspiration-level negotiators. Participants are free to use any or all components of the default factory manager agent, or they can develop their own.

How to Compete There will be three separate competitions in the 2019 SCM league. In the first, the basic competition, at most one instantiation of each team's agent will run in each simulation. In some of these simulations, all the other agents will be greedy factory manager agents. In others, agents submitted by other teams will also participate, but again at most one instantiation of each.

In the second, the collusion competition, multiple instantiations of the same team's agent may run during a single simulation (with multiple greedy factory manager agents as well). The exact number of instantiations of each will vary across simulations, and will not be announced in advance. In this competition, it is possible for instances of the same agent to try to collude with one another to corner the market, or exhibit other collusive behaviors.

The final, the sabotage competition, is intended to uncover fragile aspects of the SCML design. Teams who enter this competition should try to sabotage the market, for example, by preventing trades, or by negatively impacting the profits accrued by others. Sabotaging agents will not compete against one another directly; they will be evaluated independently in the presence of non-sabotaging agents only. Furthermore, they will be excluded from the other two competitions.

How to Win An agent's performance will be measured by its score. In the basic and collusion competitions, an agent's score will be the average profits accrued by all its factories in all its instantiations in all simulations. In the sabotage competition, agents' profits will not factor into their score; only their ability to sabotage the market/game will matter.

The three competitions 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 teams that built the top-scoring agents will be notified in June, with the final results and awards announced at IJCAI 2019. It is expected that finalists will send a representative to the ANAC workshop at IJCAI 2019, where they will be given the opportunity to give a brief presentation describing their agent. Three awards will be announced at IJCAI 2019 (with associated monetary rewards) corresponding to the three competitions (basic, collusion, and sabotage).

The organizing committee will determine the number of simulations needed in each round to ensure a fair comparison among all submitted agents. In addition to identifying the winner of each competition, all teams whose agents achieve scores that are not statistically different from the winner's will be inducted into the SCM league's **hall of fame**.

Chapter 0

Notation

The following general notation is used throughout this document:

$\mathbf{Z}$ The set of integers.

$\mathbf{R}$ The set of real numbers.

$\mathbf{X}^{+}$ The set of positive elements of the set X excluding zero.

$\mathbf{X}_{\mathbf{0}}^{+}$ The set of positive elements of the set X including zero.

$\mathbf{X}_{\infty}^{+}$ The set of positive elements of the set X including ∞.

x Any element of the set X, irrespective of any ordering.

x_i The ith element of the ordered set X.

Chapter 1

Game Entities

In this chapter, we describe the components of the SCM world. We first describe the **environment** that SCM agents inhabit, meaning the manufacturing structure—what can be manufactured and how. We then describe the autonomous **agents** themselves, meaning the active entities that make decisions about what to buy, what to sell, what to manufacture, and when.

1.1 Production Graph

The manufacturing structure of an SCM world—what can be manufactured and how—is represented by a **production graph**. This graph comprises two components, a set of products P and a set of manufacturing processes M. The former characterize what can be produced from what raw materials. The latter specify how those products can be produced. The manufacturing structure varies randomly from simulation to simulation, but the specific structure (i.e., the realizations of the random variables) is communicated to all agents at the start.

Products P A set of n_p **products**. This set is fixed for the duration of the simulation.

Each product has a catalog (i.e., nominal) price $c_p \in \mathbf{R}^+$. The actual price at which products are traded will depend on agent negotiations, so will not necessarily be this catalog price.

A product p is called a **raw material** if it is not the output of any manufacturing process (i.e., $\neg\exists m \in M$ s.t. $p \in \mathrm{Out}_m$). The set of all raw material products is called *Raw*.

A product p is called a **final product** if it is not the input to any manufacturing process (i.e., $\neg\exists m \in M$ s.t. $p \in \mathrm{In}_m$). The set of all final products is called *Final*.

A product p is called an **intermediate product** if it is not a raw material or a final product (i.e., $p \notin \mathit{Raw} \wedge p \notin \mathit{Final}$). The set of all intermediate products is called *Intermediate*.

Manufacuring Processes M A set of n_m **manufacturing processes**. This set is fixed for the duration of the simulation.

Each process m is a tuple $(\mathrm{In}_m, \mathrm{Out}_m)$.

In_m is a set of n_m^{in} triples representing inputs to the manufacturing process. In each such triple $\left(p_m^{\mathrm{in}}, q_m^{\mathrm{in}}, t_m^{\mathrm{in}}\right)$, $p_m^{\mathrm{in}} \in P$ is an input product type, $q_m^{\mathrm{in}} \in \mathbf{Z}^+$ is the requisite quantity of p_m^{in}, and $t_m^{\mathrm{in}} \in [0,1]$ is the relative time during production at which quantity q_m^{in} of product p_m^{in} is consumed.

Out_m is a single triple $(p_m^{\mathrm{out}}, q_m^{\mathrm{out}}, t_m^{\mathrm{out}})$, representing the output product type $p_m^{\mathrm{out}} \in P$, the generated quantity $q_m^{\mathrm{out}} \in \mathbf{Z}^+$ of p_m^{out}, and the relative time $t_m^{\mathrm{out}} \in [0,1]$ during production at which quantity q_m^{out} of product p_m^{out} is produced.

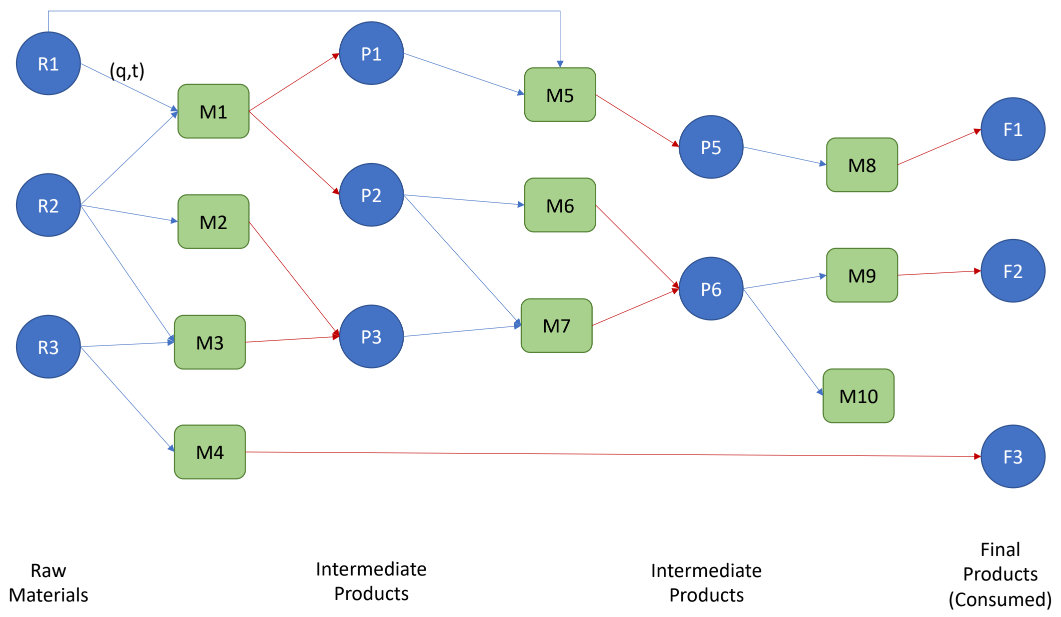


Figure 1.1: A sample production graph depicting raw materials, intermediate products, and final products. Each blue node represents a product. Each green node represents a manufacturing process. Blue edges correspond to inputs (from product to manufacturing process). Red edges correspond to outputs (from manufacturing process to product). Each edge is annotated with either an input or output quantity required or generated, and the relative time (from 0 to 1) at which the product is consumed or produced.

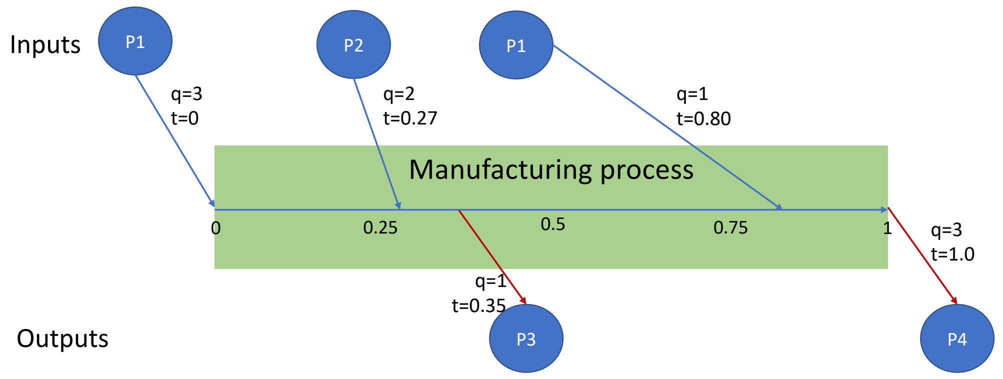


Figure 1.2: An example of a manufacturing process showing the quantities and times at which inputs are consumed and outputs are produced. This process starts by consuming 3 units of $P1$. Two units of $P2$ are then consumed, after 27% of the processing time. One unit of a by-product $P3$ is generated after 35% of the processing time. Another unit of $P1$ is consumed after 80% of the processing time. Finally, 3 units of $P4$ are generated at the end of the manufacturing process.

Together, the products and manufacturing processes define a production graph, which specify what can be produced, and how, in the SCM world. A sample production graph is depicted in Figure 1.1.

Figure 1.2 presents a detailed example of a manufacturing process, depicting the quantities and times at which inputs are consumed and outputs are produced, relative to the total time (i.e., number of time steps) during which the process is executed. In this example, all times are discretized, with all inputs assumed to be consumed at the beginning of a time step, and all outputs produced at the end. Assuming the total number of steps needed to execute this process is 10, and that production starts at step n, relative input and output times are calculated as follows:

- 3 units of $P1$ are consumed at the beginning of step n.
- 2 units of $P2$ are consumed at the beginning of step $n + 2$.
- 1 unit of $P3$ is produced at the end of step $n + 3$.
- 1 unit of $P1$ is consumed at the beginning of step $n + 8$.
- 3 units of $P3$ are produced at the end of step $n + 9$.

1.2 SCM Dynamic Entities: Factories, Miners, and Consumers

Section 1.1 described the manufacturing structure of the SCM world. Actual production, and trade, are accomplished by dynamic, decision-making entities called **autonomous agents**. This section describes these agents—miners, consumers, and factory managers.

Miners $\mathcal{M}$ A list of miners who generate raw materials. Each miner is represented by a tuple (π_m, U_m, C_m), where:

Miner Profile π_m See Section 6.1 for a description of the miners' profiles.

Miner Utilities U_m See Section 6.1 for a description of the miners' utility fuctions.

Miner Contracts C_m A list of contracts representing agreements between miners and factories. All contracts in this list are sell contracts. A miner always honors its contracts.

Each contract c is a tuple $(a_c, p_c, u_c, q_c, t_c, \rho_c, s_c)$, where $a_c \in F$ is the buyer, $p_c \in P$ is the product, $u_c \in \mathbf{R}$ is the unit price, $q_c \in \mathbf{Z}^+$ is the quantity, $t_c \in \mathbf{Z}^+$ is the delivery time, ρ_c is the penalty charged to the miner per unit of undelivered product, and $s_c \in \mathbf{Z}^+$ is the time at which the contract becomes binding (i.e., signing time). This list is updated by the simulator whenever an agreement is reached between a miner and a factory manager.

Consumers $\mathcal{C}$ A list of consumers representing retail companies. Each consumer is represented by a tuple (π_c, U_c, C_c) where:

Consumer Profile π_c See Section 6.2 for a description of the consumers' profiles.

Consumer Utilities U_c See Section 6.2 for a description of the consumers' utility functions.

Consumer Contracts C_c A list of contracts representing agreements between consumers and factories. All contracts in this list are buy contracts. A consumer always honors its contracts.

Each contract c is a tuple $(a_c, p_c, u_c, q_c, t_c, \rho_c, s_c)$, where $a_c \in F$ is the seller, $p_c \in P$ is the product, $u_c \in \mathbf{R}$ is the unit price, $q_c \in \mathbf{Z}^+$ is the quantity, $t_c \in \mathbf{Z}^+$ is the delivery time, ρ_c is the penalty on the seller per unit undelivered product, and $s_c \in \mathbf{Z}^+$ is the time at which the contract becomes binding (i.e., signing time).

This list is updated by the simulator whenever an agreement is reached between a consumer and a factory manager.

Factory Managers $\mathcal{F}$ A set of n_f agents that can manufacture products using their factories. Each factory manager $g \in \mathcal{F}$ is defined by a tuple (F_g, C_g), where:

Factory F_g Each factory manager controls a singleton set of factories F_g. The factory $f \in F_g$ is characterized by a tuple $\left(L_f, S_f^{\max}\right)$, which is fixed for the duration of the simulation.

Lines L_f A set of n_f^l lines running at factory f. Associated with each line l_f is a manufacturing **profile** $R_f : M \to \mathbf{Z}_\infty^+ \times \mathbf{R}_\infty^+$ that maps manufacturing processes $m \in M$ to tuples (t_{flm}, c_{flm}) representing the time t_{flm} it takes and the cost c_{flm} to factory f of running m.
During the simulation, each line l_f is described by its state $\left(m_{l_f}, t_{l_f}\right)$, where $t_{l_f} \in \mathbf{Z}_0^+$ is the number of time steps during which manufacturing process $m_{l_f} \in M$ has been running on l_f. This state is updated by the simulator.

Maximum Storage Capacity $S_f^{\max} \in \mathbf{Z}_\infty^+$ The storage (i.e., inventory) capacity at factory f. If $S_f^{\max}$ is finite, we assume that all products are stored in boxes of equal volumes, and that this volume evenly divides $S_f^{\max}$.

A factory f's state (J_f, S_f, W_f, B_f) is described by:

Scheduled Jobs J_f A list of tuples representing scheduled jobs. Each such job j_f is a tuple $\left(t_{j_f}, l_{j_f}, m_{j_f}\right)$ indicating that the manufacturing process $m_{j_f} \in M$ should be initiated on line $l_{j_f} \in L_f$ at time step $t_{j_f} \in \mathbf{Z}^+$. This list is updated by the factory manager.

Storage $S_f : P \to \mathbf{Z}_0^+$ The current inventory in storage at factory f. This is quantity is updated by the simulator.

Wallet $W_f \in \mathbf{R}$ The balance accumulated by factory f. Note that contents of a wallet can be negative: i.e., there is the possibility of overdraft. This quantity is updated by the simulator.

Loans $B_f \in \mathbf{R}_+$ The factory's outstanding loan value, which never falls below 0.

Contracts C_g A list of contracts representing agreements with other agents. The contracts can either be buy or sell contracts. Each contract c is a tuple $(k_c, a_c, p_c, u_c, q_c, t_c, \rho_c, s_c)$, where k_c is the contract kind (buy or sell), $a_c \in \mathcal{M} \cup \mathcal{C} \cup \mathcal{F}$ is the partner (seller or buyer), $p_c \in P$ is the product, $u_c \in \mathbf{R}$ is the unit price, $q_c \in \mathbf{Z}^+$ is the quantity, $t_c \in \mathbf{Z}^+$ is the delivery time, ρ_c is the penalty charged to the seller per unit of undelivered product, and $s_c \in \mathbf{Z}^+$ is the time at which the contract becomes binding (i.e., signing time). This list is updated by the simulator whenever an agreement is reached between a consumer and a factory manager.

1.3 SCM Static Entities: Banking and Insurance

In addition to the dynamic entities (i.e., **agents**) inhabiting the SCM world, there are additional entities which we refer to as **static** entities, because they are not active decision makers. Not only is their behavior predefined, it does not vary with the state of the simulation.

Bank The SCM world contains a single bank from which money can be borrowed when needed (and only when needed).[1] Lending works as follows:

1. If a contract is due, but the buying party does not have sufficient funds, any amount up to the buyer's shortfall can be borrowed from the bank at an interest rate that depends on the financial standing of the agent (i.e., its credit rating; see Section 6.3 for details). The buyer is then free to decide whether to borrow at this rate, or to risk committing a breach.

[1]The bank is disabled in the 2019 SCML competition. Nonetheless, we include a description of its functionality, to provide participants with a complete picture of future competitions.

2. Agents pay back their loans in installments, one payment per time step. If they cannot make a payment, their credit rating suffers. Still, they are automatically offered another loan to make the payment, but this loan may be at a very high interest rate (see Section 6.3 for details). An agent can refuse this loan, but only by declaring bankruptcy.

3. If an agent goes bankrupt, they are reported to the bankruptcy list with a time stamp, and all agents are informed of this addition to the bankruptcy list. The other agents can then choose to adjust their negotiation strategy, in any ongoing negotations with the bankrupt agent.

   The bankrupt agent's inventory is then liquidated, at some fraction *liquidation rate* of catalog prices. Its cash is then used (to the extent possible) to make the missed payment, either to the bank in case it was a loan installment, or to a trading partner in case of a breach (see Section 2.3).

   Additionally, all future contracts with the bankrupt agent are voided, and any remaining funds (up to the total value of these contracts) are paid to its trading partners in proportion to the contracts' values. Finally, its credit rating is adjusted based on the *bankruptcy missed payment value*.

Insurance Company The SCM world also contains an insurance company that can insure contracts against future potential breaches. Given a contract between two agents, either agent can buy an insurance. If an agent insures a contract, and their trading partner commits a breach, the insurance company steps in immediately, and accomplishes the following:

1. If it is the seller who is insured, and the buyer does not have the requisite funds, the buyer buys what they can, and then the insurance company buys the rest of the agreed upon quantity (up to the amount the seller has to sell) at the agreed upon unit price.

2. If it is the buyer who is insured, and the seller does not have the requisite products, the seller sells what they can, and then the insurance company sells the rest of the agreed upon quantity (up to the amount the buyer can buy) at the agreed upon unit price.

From the point of view of the insured, it is as if the trading partner had not committed a breach. However, the perpetrator is still reported to the breach list, unless they can manage to renegotiate with their victim.

The cost of an insurance policy depends on the premium, which is some fraction of the contract's value. Insurance policies are only available *before* a contract's execution time; premiums are cheapest when the contract is signed, and increase in cost as the delivery time approaches. See Section 6.4 for details about how the insurance policy premium is determined.

Chapter 2

Game Rules

In this chapter, we describe the rules of the SCM game, meaning the negotiation protocol (alternating offers), as well as the rules pertaining to signing contracts, executing contracts, and resolving breaches. Also relevant to this discussion is what information is public (e.g., past breaches) vs. what is private (successful contracts).

At a high-level, negotiation, contract signing and execution, and breach resolution proceed as follows:

1. An agent posts a call-for-proposals (CFP) on the SCM world's bulletin board.
2. Another agent responds to that call, indicating that they are interested in negotiating.
3. Negotiation between these two agents proceeds via the alternating offers protocol (see Section 2.2).
4. If the negotiation is successful, the agents are given the opportunity to sign a contract.
5. If the agents sign the contract, they are asked if they are willing to execute the contract. If they are unwilling, a breach is recorded on the bulletin board.
6. If they are willing, and if they have sufficient funds and products, funds and products are exchanged. If they do not have sufficient funds and products, they are given an opportunity to renegotiate.

2.1 Calls for Proposals

An agent invites other agents to negotiate with it by posting a **call for proposals** (CFP) on the bulletin board. When one agent responds positively to another's CFP, a negotiation ensues.

A call for proposals is a tuple $(k_c, a_c, p_c, u_c, q_c, t_c, s_c)$, where:

Kind $k_c \in \{\mathbf{buy}, \mathbf{sell}\}$ Indicates whether the agent is looking to buy or sell.

Agent $a_c \in F$ The agent publishing the call.

Product $p_c \in P$ The product of interest.

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

Unit Price u_c The unit price.
Quantity q_c The quantity.
Delivery Time t_c The delivery time: i.e., the contract's execution time.
Signing Time s_c The signing time: i.e., the time at which the contract becomes binding.
Penalty ρ_c A per-unit penalty on the seller if the product is not available in full at delivery time.

The unit price and the penalty can be a real value, or a list of real values, or a range of real values. The quantity, delivery time, and signing time can be an integer, a list of integers, or a range of integers. In case any of these quantities are not a single number, they specify a range of negotiable values for that quantity.

2.2 Negotiation Mechanism

The negotiation mechanism adopted by the SCM world is a variant of Rubinstein's alternating offers protocol [AFHJ17]. Rubinstein's protocol involves two agents, who take turns making offers. One agent starts the negotiation with an opening bid, after which the other agent takes one of the following actions:

1. Accept the offer
2. Make a counter offer, thus rejecting and overriding the previous offer
3. Walk away, thus declaring an end to the negotiation, without having reached an agreement

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

The only difference between the variant of the protocol used in the SCM world and Rubinstein's 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 [GSS82], which can happened 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., 10) 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 offers or propose new ones.

2.3 Contract Signing and Execution & Breach Resolution

If an agreement is reached, then, when the time comes, the simulator asks both parties to sign a contract. If they both sign, the contract becomes binding. Either party can cancel a successfully negotiated contract without penalty by refusing to sign it before it becomes binding.

When a contract's delivery date arrives, 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 wallet to the seller's. In the event that a contract does not execute successfully, a **breach** is reported.

Table 2.1 describes the causes of breaches, and the levels that ensue. The maximum breach level is 1, which occurs when either agent outright refuses to execute a contract. In addition, buyers can commit a breach for *insufficient funds*, while sellers can commit a breach for *insufficient products*. When a seller commits a breach, they incur penalties, which are paid both to the victim (the **negotiated penalty**) and to "society" (the **global penalty**). These penalties are calculated as the product of the unit price, the seller's inventory shortfall, and the negotiated and global penalty rates, respectively. If a seller cannot pay the negotiated penalty, they commit a further breach, *insufficient funds for penalty*. Note that the breach list only records breaches against other agents; failure to pay a global penalty is not recorded here.[1]

Before any breaches are reported to the breach list, there is an opportunity for the agents involved to renegotiate. The renegotation mechanism works as follows:

1. If a contract cannot be executed due to insufficient funds, then the bank offers the buyer a loan, after which the following logic is employed:
   (a) If the buyer borrows enough to cover their shortfall, the contract is executed, and there is no breach due to *insufficient funds*.
   (b) If the buyer does not borrow enough, then there is a potential breach due to *insufficient funds*. There is an opportunity to renegotiate (see Step 3) to avoid this breach.

[1]In 2019, failure to pay a global penalty automatically causes bankruptcy, as there is no bank nor are there any loans.

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	$\frac{q_c - s_c.f.S_f(p_c)}{q_c}$
Insufficient funds for penalty	$s_c.f.W_f < \gamma$	Seller s_c	$\frac{\gamma - s_c.f.W_f}{\gamma}$
Refusal to execute	Refusal to execute	The refusing agent	1
$x.f$ is the factory associated with agent x. W_f is the balance in factory f's wallet (after any borrowing). S_f is factory f's inventory level. p_c, u_c, q_c, ρ_c are the product, unit price, quantity, and negotiated penalty specified in contract c. ρ_g is the global penalty paid to "society" by sellers who do not honor their contracts. $\gamma \equiv (\rho_c + \rho_g)(q_c - s_c.f.S_f(p_c))$ is the sum of the negotiated and global penalties (if any).			

Table 2.1: Breach Conditions and Levels: The insufficient funds breach level depends on the shortfall in the buyer's account (after any borrowing) that precludes them from paying the contract price. The insufficient products breach level depends on the shortfall in the seller's inventory. The insufficient funds for penalty breach level depends on the shortfall in the seller's account (after any borrowing) that precludes them from paying their penalties. The refusal to execute breach level is always 1.

2. If a contract cannot be executed due to insufficient products, then the following logic is employed:
   (a) If the contract does not include a negotiated penalty, then there is no possibility of an *insufficient funds for penalty* breach, because the breach list only records breaches against other agents, not against "society." Nonetheless, there is a potential breach due to *insufficient products*. Agents can renegotiate (see Step 3) to avoid this breach.
   (b) If the contract includes a negotiated penalty, then there may be an *insufficient funds for penalty* breach:
      i. If the seller can pay all the ensuing penalties (possibly with a loan from the bank) they pay them, and there is no *insufficient products* breach nor *insufficient funds for penalty* breach.
      ii. If the seller can pay the negotiated penalties only (possibly with a loan from the bank) but they cannot pay the global penalty, there is no *insufficient products* breach. There is also no *insufficient funds for penalty* breach, because the breach list only records breaches against other agents, not against "society."
      iii. If the seller cannot pay the negotiated penalties in full which implies that they also cannot pay the global penalty, there are potentially two breaches, due to both *insufficient products* and *insufficient funds for penalty*. There is an opportunity to renegotiate (see Step 3) to avoid both these breaches.
3. If the contract has not yet been executed, either because of a potential buyer breach (Case 1) or a potential seller breach (Case 2), or both, or because the agents were prompted by the simulator to proceed with the contract and one of them outright refused, the following steps are taken:
   (a) There is a possibility for renegotiation, as follows:
      i. Each agent's *total breach level* for this contract is calculated by adding up all their *potential* breach levels. In the next step, we refer to the agent with the *lower total breach level* for this contract as A (breaking ties randomly), and to the other as B.
      ii. The agents are then offered the opportunity to set a *renegotiation agenda*, in turn. Agent A is asked first. The offer informs agent A about all breaches committed while trying to execute this contract. When the renegotiation agenda is passed on to agent B, the breach information is passed along as well.

iii. Agent B can then either accept the opportunity to renegotiate, or reject it. If agent B agrees to renegotiate, a new negotiation is run to completion, immediately. If this negotiation leads to a new contract, all breaches of the original contract that failed to execute are voided, meaning they are not reported to the breach list.

iv. If agent B does not agree to renegotiate, Steps 3(a)ii and 3(a)iii are repeated, with the agents' roles reversed. If the agents again fail to agree to renegotiate, the opportunity is lost.

v. The following issues comprise a *renegotiation agenda*:

A. Quantity to be delivered immediately. Cannot exceed the available quantity in the seller's inventory.

B. Unit price for the quantity to be transferred immediately. Cannot exceed the available balance in the buyer's wallet.

C. Quantity to be delivered later.

D. Unit price for quantity to be delivered later.

E. Penalty on the quantity to be delivered later.

F. Later delivery date.

(b) If the opportunity to renegotiate fails, all breaches are reported to the breach list. Specifically:

i. If the buyer was a perpetrator, an *insufficient funds* breach is reported. The breach level is determined by the shortfall in the buyer's wallet.

ii. If the seller was a perpetrator, an *insufficient products* breach is reported. The breach level is determined by the shortfall in the seller's inventory.
To the extent possible, penalties (if any) are then paid to first to the buyer (at the negotiated penalty rate) and then to "society" (at the global penalty rate). If the negotiated penalty is not paid in full, a further breach is reported, *insufficient funds for penalty*. If the global penalty is also not paid in full, the agent is offered a loan to pay this penalty. An agent can refuse this loan, but only by declaring bankruptcy.[2]

iii. Note that both the buyer and seller could be perpetrators.

(c) The contract is then executed to the extent possible, after confirming that partial execution is acceptable to any party not in breach of contract. If both parties are in breach of contract, partial execution is obligatory. Partial execution entails:

- If a buyer committed the breach, then the available balance in its wallet is transferred to seller's wallet, and a corresponding proportional amount of the product (given the unit price) is moved from the seller's inventory to the buyer's.
- If a seller committed the breach, then all available supply of the product is moved from the seller's inventory to the buyer's, and a corresponding proportional amount of money (given the unit price) is moved from the seller's wallet to the buyer's.

(d) If a contract is renegotiated, or if it is partially executed, any insurance claims pertaining to the original contract are void.

2.4 Information Revelation

The final piece of the game rules that remains to be specified is the simulator's information-revelation policy, meaning what information is public, and what is private.

Private Information All agents' manufacturing profiles are private. Additionally, anything pertaining to negotiations (and re-negotiations) between agents is private. No other agents except those involved see any intermediate proposals and responses, nor the contents of any final, agreed-upon contracts.

[2] In 2019, failure to pay a global penalty automatically causes bankruptcy, as there is no bank nor are there any loans.

Public Information The manufacturing structure of the SCM world (the products and the production graph) is public. This information is broadcast to all registered agents at the start of each game.

In addition, the SCM world maintains a bulletin board, where CFPs, breaches, bankruptcies, and quarterly reports are published. The contents of CFPs are described in Section 2.1.

Each breach record 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 funds for penalty*, *insufficient product*, *refusal to execute*$\}$, is the kind of breach committed.

Whenever an agent goes bankrupt, this agent is added to the bankruptcy list, and news of their bankruptcy is immediately broadcast to all agents.

Reports are published periodically about all agents' financial status. These reports contain the following information about agents' assets (cash and inventory) and liabilities (outstanding loans and credit rating):

Cash The balance in the agent's wallet. (This balance is infinite for consumers.)

Inventory The value of the agent's inventory, evaluated at catalog prices. (This value is infinite for miners.)

Liabilities The value of the agent's outstanding loans and their credit rating. (The former value is zero for consumers and miners, while the latter is the *maximum credit rating* for both.)

Chapter 3

Simulation

All SCM agents implement an initialization function and a step function. The former is called by the simulator to initialize their behavior. After initialization, the simulator repeats the following loop, which calls the agents' step functions, among other things:

1. Offer agents the opportunity to sign all contracts due to become binding now.

2. Run all registered negotiations for *negotiation speed multiplier* rounds. If an agreement is reached, then if it is due to become binding now and its signing delay is zero, offer agents the opportunity to sign the contract now. Concluded negotiations, signed or not, may trigger new negotiation registrations. If *immediate negotiations* is enabled, start running these negotiations during this time step as well (instead of during the next time step).

3. Call the bank and insurance company's step functions. The bank's step function collects interest and loan payments. The insurance company's step function is a noop.

4. Call the step functions of all decision-making agents—miners, consumers, and factory managers—in an unspecified order. Doing so registers new negotiations, which, if *immediate negotiations* is enabled, start running immediately; otherwise, they start running during the next time step.

5. Execute all contracts due to be executed at this time step. For those contracts that can be executed, begin delivering products from the sellers to the buyers.[1] For those contracts that cannot, offer loans as necessary, process insurance claims, and handle any potential breaches. Note that any renegotiations are carried out immediately, meaning they conclude and are signed (or not) before proceeding.

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

There are a few details pertaining to timing worth mentioning:

- When contracts are executed, funds are deducted from the buyer's wallet at the *beginning* of the time step, and deposited in the seller's wallet at the *end* of the time step. This means that a factory manager cannot use funds it receives from sales at time t to buy products, or run a manufacturing process, before time $t + 1$.

- When products are manufactured, inputs are removed from inventory at the *beginning* of the time step, and outputs are stored in inventory at the *end* of the time step. This means that a factory cannot sell products it manufactures at time t, or use those products as inputs to another manufacturing process, before time $t + 1$.

[1]In 2019, the *transportation delay* is 0, so products scheduled for delivery arrive by the end of the time step (but not before).

- When products are scheduled to be delivered, they are removed from the seller's inventory at the *beginning* of a time step, and stored in the buyer's inventory at the *end* of that time step plus the *transportation delay*, say d. This means that a factory cannot sell products it buys at time t, or use them as inputs to a manufacturing process, before time $t + d + 1$.

Chapter 4

Agents

An SCM agent controls an entity in the SCM world. Recall that there are three types of SCM agents:

Miners Agents that control the generation of one or more raw materials.

Consumers Agents that control consumption of one or more final products.

Factory Managers Agents that control factories (one agent per factory).

This chapter describes the functionality (i.e., operations) common to all SCM agents, as well as details specific to the miners and consumers. Discussion of the factory manager agents is deferred until Chapter 5.

Before describing SCM agents, however, we describe negotiators, which are themselves agents that are spawned by SCM agents when they are ready to negotiate.

4.1 Negotiators

A negotiator is an entity that conducts negotiations on behalf of an agent (miner, consumer, or factory manager). For backward compatibility with other ANAC negotiators (e.g., Genius agents, aspiration-level negotiators, etc.), 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 agents themselves, created by an SCM agent. 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 propose trades and respond to proposals. They do not interact further with the agents that create them, after they are assigned their utility function.

2. Pass-through negotiators just pass trade proposals and responses through to the agent that created them. Decision making is therefore the responsibility of the creating agent. These negotiators can be used to implement a centralized negotiation strategy.

4.2 SCM Agents

This section describes the callbacks and actions common to all SCM agents.

4.2.1 SCM Agents' Callbacks

SCM 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., sign a contract, confirm contract execution, 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:

On New CFP Called whenever a new CFP is published on the bulletin board. The agent can specify a condition (e.g., buy CFPs only) such that only those CFPs that satisfy this condition will trigger this callback.

On CFP Removal Called whenever a CFP is removed from the bulletin board. The agent can specify a condition (e.g., buy CFPs only) such that only those CFPs that satisfy this condition will trigger this callback.

On Negotiation Request Accepted/Rejected Called when a negotiation request initiated by the agent is accepted/rejected.

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

On Contract Signed/Cancelled/Nullified Called when a contract the agent is party to is 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.

Sign Contract Called by the simulator when it is time to sign a contract. Agents can refuse to sign, but by default, agents sign all contracts.

Confirm Contract Execution Called by the simulator when it is time for one of the agent's contracts to be executed. The agent can reject the execution: i.e., refuse to honor the contract. By default, agents accept contract execution.

Confirm Loan Called by the simulator when the agent is a buyer, but the balance in the agent's wallet is less than the value of a contract that is about to be executed. The agent should respond with a value in $[0, 1]$, indicating what fraction of the loan it is willing to take on.

Respond to Negotiation Request Called whenever another agent responds to one of this agent's CFPs. This agent can agree to negotiate or not. If the agent agrees, a new negotiation is registered.

Set Renegotiation Issues Called when the agent's trading partner is in breach of contract. It can set the renegotiation agenda, or refuse to renegotiate. By default, agents refuse to renegotiate.

Respond to Renegotiation Request Called in case another agent requests renegotiation of a contract. The agent may reject the negotiation or accept it. By default, agents reject renegotiation requests.

Confirm Partial Contract Execution Called when the agent's trading partner is in breach of contract but the agent is not. They can allow or refuse partial execution of the contract. By default, agents accept partial contract execution.

On Agent Bankrupt Sent to all agents whenever any agent goes bankrupt.

4.2.2 SCM Agents' Actions

SCM entities are controlled by agents that manage their day-to-day activities, negotiations, and finances:

Activity Control Mine, consume, or schedule manufacturing jobs, as appropriate.

Negotiation Control Publish CFPs, accept or reject negotiation requests, and spawn negotiators.

Financial Control Manage loans and insurance.

More specifically, SCM agents control their negotiations and finances using the following actions:

Publish CFP Publishes a CFP on the bulletin board.

Remove CFP Removes a previously published CFP from the bulletin board.

Request Negotiation Sends a negotiation request to the publisher of a CFP listed on the bulletin board.

Create Utility Function Creates a utility function.

Spawn Negotiator Spawns a negotiator.

Register/Unregister Interest Registers (un-registers) interest in a set of products. Agents receive **On New CFP** and **On CFP Removal** callbacks for products they are registered as interested in.

Evaluate Insurance Cost Given a contract (either hypothetical or already agreed upon), query the insurance company for the cost of policy to insure this contract.

Buy Insurance Allows an agent to buy an insurance, transferring funds immediately from its wallet to the insurance company. This action must be preceded by **Evaluate Insurance Cost**.

Hide Inventory (or Funds) An agent can choose to hide some of its inventory (or funds), so that that inventory (or funds) is not used to execute any contracts. This action gives agents the flexibility to save their inventory (or funds) for later contracts.

4.3 Demand-driven Markets

Demand-driven markets are driven by demand. In the SCM world, this amounts to proactive consumers driving demand by posting buy CFPs. Miners at the other end of the chain merely react to these buy CFPs by offering to sell. Their supply is never exhausted.

4.3.1 Reactive Miner Agents

Recall that in demand-driven markets, miners are reactive. In particular, miners never issue sell CFPs; they merely respond to other agents' buy CFPs.

This section describes how the miners' reactive behavior is implemented, by describing their implementations of the various callbacks and actions common to all SCM agents.

On Init No op.

On Step No op. (Reactive miners are just that: reactive.)

On New CFP Reactive miners respond to all buy CFPs for products that they can mine, given that they are initiated by non-bankrupt agents whose total breach level is than a predefined *max breach level*. To do so, they first generate a utility function to guide the negotiation. Then, they spawn a negotiator with this utility function (and reservation value zero). Finally, they request a negotiation with the agent who publised the CFP using this negotiator.

The miner's **OnNewCFP** function is described in Algorithm 1.

Algorithm 1

```
1: procedure Miner OnNewCFP
2:     if the miner can mine the CFP's product then
3:         Create a utility function (See Section 6.1, Algorithm 4).
4:         Spawn a negotiator with this utility function and reservation value zero.
5:         Request a negotiation with the agent who published the CFP using this negotiator.
```

Confirm Loan Miners never need loans, as they never buy anything.

Respond to Negotiation Request Reactive miners do not need to respond to negotiation requests as they never post CFPs, so they are never approached by agents to negotiate with them.

4.3.2 Proactive Consumer Agents

Recall that in demand-driven markets, consumers are proactive. They drive demand by posting buy CFPs.

This section describes how the consumers' proactive behavior is implemented, by describing their implementations of the various callbacks and actions common to all SCM agents.

On Init No op.

On Step The consumers' step function, which determines how and when they publish CFPs, is described in Algorithm 3. This function depends on the consumer's profile (See Section 6.2). Specifically, it depends on their consumption schedule S_c and how dynamic that schedule is D_c.

On New CFP Proactive consumers always ignore new CFPs, as they never request negotiations.

Confirm Loan Consumers never need loans, as their balances are infinite.

Respond to Negotiation Request Proactive consumers accept negotiation requests from non-bankrupt agents with a breach level less than a predefined *max breach level*. Whenever a consumer accepts a negotiation request about some product p, it spawns a negotiator using Algorithm 2.

Algorithm 2

```
1: procedure Consumer RespondToNegRequest
2:     if the consumer can consume the CFP's product then
3:         Create a utility function (See Section 6.2, Algorithm 5).
4:         Spawn a negotiator with this utility function and reservation value $\infty$.
5:         Accept the negotiation request using this negotiator.
```

Algorithm 3

```
1:  procedure CONSUMER STEP($S_c, D_c, n$)
2:      if $n = 0$ then
3:          $S \leftarrow S_c$
4:      else
5:          Delete all my existing CFPs from the bulletin board.
6:          Calculate average quantity across existing contracts at each time step.
7:          Average these averages to calculate the average contract quantity per time step.
8:          for $p \in P$ do                                          ▷ For all products
9:              $C(p) \leftarrow$ Total quantity of product $p$ already contracted to be delivered at time $n$.
10:             $S(p) \leftarrow (1 - D_c)\,(S_c - C(p)) + D_c\,\bar{q}$
11:     PUBLISHCFPS($S, n$)
12: procedure PUBLISHCFPS($S, n$)
13:     for $p \in P$ do                                              ▷ For all products
14:         unit price $\leftarrow$ negotiable
15:         product $\leftarrow p$
16:         time $\leftarrow n$
17:         quantity $\leftarrow [1, S(p)]$
18:         Publish a CFP with these parameters.
```

Chapter 5

Factory Manager Agents

Every factory is run by a factory manager agent. This chapter describes the base factory manager agent, and an extension of it, the greedy factory manager agent. The latter is of particular interest for two reasons: first, it serves as a concrete example of how to implement a factory manager agent; and second, this agent will manage all factories run by the organizing committee during the tournament simulations.

Factory manager agents store two key data structures, a **current factory state** and a **predicted factory state**. The former includes the factory's current inventory levels, its current wallet balance, its current outstanding loans, and its current job schedule. The latter includes the same, but predicted from the next time step through the end of the simulation.

5.1 Factory Manager Agents

This section describes additional callbacks and actions specific to factory manager agents.

5.1.1 Factory Manager Agents' Callbacks

In addition to the standard callbacks common to all SCM agents (Section 4.2.1), factory manager agents have two additional callbacks related to their factories:

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

On Storage Overflow Called when products cannot be transferred to inventory due to insufficient space.

5.1.2 Factory Manager Agents' Actions

In addition to the standard actions common to all SCM agents (Section 4.2.2), factory manager agents have several additional actions related to production:

Get State Reads the factory state.

Schedule Job Adds a manufacturing process to the specified line to start at the specified time step. (Once a manufacturing process has been scheduled, it is called a **job**.)

Cancel Job Cancels a scheduled job. If the job is already underway, cancellation costs may apply.

5.2 Helpers

Factory manager agents maintain the current and predicted factory states using two helpers, a job scheduler and a factory simulator.

5.2.1 Job Scheduler

The **(job) scheduler** schedules production on behalf of a factory manager agent. The input to a scheduler is the factory's profile, its state, a time horizon T, a set of contracts C. The scheduler then attempts to schedule the input contracts within the given time horizon. It outputs the following:

Validity The set of contracts it is able to honor: i.e., some $C^* \subseteq C$.

Scheduled Jobs If $C^* \neq \emptyset$, a set of scheduled jobs for the duration of the time horizon T, meaning manufacturing processes along with their lines and times to be run, to honor the contracts in C^*.

Production Demands If $C^* \neq \emptyset$, a list of production demands for the duration of the time horizon T, each specifying a product p, a quantity q, and a time at which q units of p must become delivered to the factory to honor the contracts in C^*.

5.2.2 Factory Simulator

The **factory simulator** can be used to *simulate* the results of the factory taking many of its available actions (e.g., buy (sell) products, start (stop) jobs; see Sections 4.2.2 and 5.1.2 for details), as well as some actions which arise only via callbacks (e.g., Confirm Loan), but which affect the factory's predicted state.

Given an initial factory state, and a sequence of operations to be simulated at specified times (in the present or in the future), the simulator can be used to *predict* inventory levels, the factory's wallet balance, outstanding loans, and the manufacturing schedule on all the factory's lines at any point in the future.

The following operations are key to the simulator's functionality.

Start (stop) job Start (stop) a job on a specific line.

Buy (Sell) products Used to buy (sell) specific quantities of a product at a specific unit price.

Scheduling operations Mark certain quantities of certain products in inventory as *reserved.* The scheduler ignores these products during scheduling, to ensure they are not used more than once.

Ignore shortages in inventory, balance, or space (in the finite maximum storage case).

Make payments and take out loans Used to buy insurance, make loan payments, and take out loans.

The simulator is optimistic: it works under the assumption that no breaches will be committed in the future, no contracts will be nullified, and no production failures will occur. Of course, these assumptions will often be violated in the SCM world. For that reason, the simulator provides a **Fix** operation that takes as input a factory state and a time step. *Fixing* the specified factory state at the specified time step, it then proceeds to update all its predictions.

Of note, the simulator supports **transaction control** using **bookmarks** that can be either **committed** or **rolled-back**. A transaction starts by setting a new bookmark. All simulated operations are then tentatively recorded. If the caller *rolls back* the transaction, these operations are all cancelled, returning the simulator to its bookmarked state. If, on the other hand, the caller *commits* the transaction, these operations are committed. In both cases, the bookmark is deleted.

5.3 Base Factory Manager Agent

The base factory manager agent, from which all other factory managers inherit, does nothing but maintain an internal factory simulator, and update its simulator's internal factory state every time step.

Init Initialize its internal simulator.

Step Call the simulator's **Fix** function to reset its internal factory state to the current factory state.

5.4 Greedy Factory Manager Agent

This section describes the implementation of the default factory manager agent, hereafter called the **greedy factory manager**, because it relies on a greedy scheduler.

The greedy factory manager is reactive. It responds only to buy CFPs, and after successful negotiations, it publishes only buy CFPs, further down the supply chain.

On Init Initialize a greedy scheduler (see Section 5.5) and an internal consumer agent.

On Step No op.

On New CFP Greedy factory managers do not respond to sell CFPs.

Greedy factory managers consider responding to all buy CFPs for products that they can produce,[1] given that they are initiated by non-bankrupt agents whose total breach level is than a predefined *max breach level*. They do so by trying to schedule the products requested in the CFP using their scheduler, given the current state of their factory. If production is not possible (e.g., because all the required inputs cannot be acquired in time), the CFP is ignored. If the CFP is not ignored, a negotiation is requested with the publisher of the CFP, using a negotiator that uses a default utility function which calculates the **marginal utility** of any hypothetical contract as follows:

1. Estimate the total expected utility of the current set of contracts.
   (This information is stored in the factory's simulator.)
2. Estimate the total expected utility of the current set of contracts *and* the hypothetical contract.
   (This value is calculated by first calling the scheduler to obtain a job schedule, then calling the simulator to predict the factory's final wallet balance based on that job schedule, and then rolling back those transactions, since this contract is as yet hypothetical.)
3. The difference between these two values is the marginal utility of the hypothetical contract.

This implementation does not take into account any ongoing negotiations, nor does it consider future potential contracts nor future potential breaches.

Sign Contract Greedy factory managers run the greedy scheduler with each hypothetical contract before signing. If the contract is successfully scheduled, they agree to sign.

Respond to Negotiation Request The internal consumer object responds as a consumer would.

On Negotiation Failure If the factory manager was not the publisher of the CFP that led to this failed negotiation, it retries the negotiation with the same negotiator and utility function at most five times.

On Contract Signed If the factory manager is the seller in this contract, it runs its scheduler,[2] which returns jobs to schedule and production demands. It then calls **Schedule Job** to schedule all the returned jobs. It also updates its internal consumer object's consumption schedule with the production demands, so that the next time it calls **On Step**, it publishes CFPs accordingly.

If the factory manager is the buyer in this contract, it tells the simulator to update its predicted factory state—specifically, its predicted inventory and wallet balance—according to contract's terms.

[1]Greedy factory managers never act solely as a "middle man," buying products only to resell them as is.

[2]In reality, the scheduler is not run twice, both in **Sign Contract** and **On Contract Signed**. The output of the call to the scheduler in **Sign Contract** is saved internally, and read again in **On Contract Signed**.

5.5 Greedy Scheduler

The greedy scheduling algorithm hinges on the assumption that factory managers are *reactive*: i.e., they never post sell CFPs; they sell only by responding to others' buy CFPs, in which case successful negotiations lead them to post their own buy CFPs further down the chain. Consequently, the more difficult decisions faced by the greedy scheduler pertain to sell contracts: what outputs can be produced in a timely manner at a reasonable cost? In contrast, buy contract decisions are easy: they are scheduled as long as they are feasible, because they always stem from an earlier decision by the factory manager agent to post a buy CFP.[3]

Recall that a scheduler takes as input the factory's profile, its state, a time horizon T, and a set of contracts C. The greedy scheduler attempts to schedule each input contract in ascending order of delivery time (breaking ties randomly), as follows:

Buy contracts: Buy contracts are scheduled as long as they are feasible. Feasibility requires that sufficient funds are expected to be on hand at delivery time, sufficient funds are on hand right now to insure this contract, and that sufficient space is available in inventory. If these conditions are met, the contract is scheduled, meaning inventories and funds are updated accordingly.

Sell contracts: A **production schedule** is a map from manufacturing lines and slots (i.e., time steps) to manufacturing processes (i.e., jobs). The greedy scheduler schedules sell contracts such that jobs are never fragmented, meaning manufacturing processes are always scheduled across a sequence of consecutive manufacturing slots. We call such a sequence a **chunk**.

The greedy scheduler can allocate manufacturing processes to chunks in many orders, as determined by the *scheduling strategy*, including shortest-first, longest-first, earliest-first, or latest-first. The 2019 implementation uses latest-first.

Assume a sell contract c with product p_c, quantity q_c, and delivery time t_c. Scheduling works as follows:

1. Find the quantity $\hat{q}$ of product p_c predicted to be available at time t_c. Calculate the quantity needed, namely $q = q_c - \hat{q}$. If $q \leq 0$, there is no need for any production. Inventory is decreased by q at time t_c, and processing of this contract ends successfully.
2. Find all lines whose profiles can support a manufacturing processes that produces p_c. Discard those with insufficiently many chunks to produce one unit of p_c.
3. Sort the remaining line-profile combinations in ascending order of their production cost per unit, assuming all inputs are acquired at catalog prices.
4. Sort the chunks in each line-profile combination according to the *scheduling strategy*.
5. Create tentative jobs to produce p_c in the designated chunk order, keeping track of the quantity produced, until q units of p_c has been produced, or until no further chunks are available. In the former case, convert the tentative jobs to scheduled ones, and mark this contract as successful. In the latter case, delete the tentative jobs, and mark this contract as unsuccessful.
6. Calculate the production demands corresponding to all scheduled jobs by comparing the necessary inputs to the expected inventory across the relevant time horizon T.

Once all contracts have been processed, the greedy scheduler returns the (possibly empty) subset of C that were successfully scheduled, as well as the scheduled jobs and their corresponding production demands.

[3]Note that if factory managers were instead *proactive*, meaning they post only sell CFPs but no buy CFPs, the more difficult decisions faced by the greedy scheduler would pertain to buy contracts: what inputs should be acquired because they can be transformed into products that can be sold in a timely manner for a profit? In contrast, sell contract decisions would be easy: they would be scheduled as long as they are feasible, because they would always stem from an earlier decision by the factory manager agent to post a sell CFP.

Chapter 6

Simulator Functions

In this chapter, we describe the utility functions of the miners and consumers, as well as the pricing functions of the bank and the insurance company. The throughput of the SCM simulation (i.e., the number of successful negotiations and trades) is highly dependent on these choices. Optimization of these hyper-parameters is ongoing. Feedback and suggestions from the community are welcome.

6.1 Miners' Characterization

Miners are characterized by their utilities, which in turn depend on their profiles. Hence, this section starts with a detailed description of miners' profiles, and then describes their utilities.

Miners' Profiles A miner m's profile is a tuple π_m whose entries are used to determine the utility of mining (i.e., generating) each raw material:

$P_m \subseteq P$ The products that the miner can generate/mine.

$B_m \in R_0^+$ Level of *insufficient funds* and *insufficient funds for penalties* breaches accumulated by an agent above which the miner will not negotiate with that agent. This level is calculated as the sum of all such breach levels.

$\hat{\alpha}_{um} : P_m \to (0, 1)$ A measure of how much the utility of a contract for a product is affected by the unit price.

$\hat{\alpha}_{qm} : P_m \to (0, 1)$ A measure of how much the utility of a contract for a product is affected by the quantity.

$\hat{\alpha}_{tm} : P_m \to (0, 1)$ A measure of how much the utility of a contract for a product is affected by a time delay.

$\hat{\tau}_{um} : P_m \to (0, 1)$ **and** $\hat{\beta}_{um} : P_m \to \mathbf{R}^+$ Control the shape of the function representing the contribution of the unit price to the utility of a contract for a product.

$\hat{\tau}_{qm} : P_m \to (0, 1)$ **and** $\hat{\beta}_{qm} : P_m \to \mathbf{R}^+$ Control the shape of the function representing the contribution of the quantity to the utility of a contract for a product.

$\hat{\tau}_{tm} : P_m \to (0, 1)$ **and** $\hat{\beta}_{tm} : P_m \to \mathbf{R}^+$ Control the shape of the function representing the contribution of the time delay to the utility of a contract for a product.

$\mathbf{cv}_m \in \mathbf{R}_0^+$ The coefficient of variation used for random sampling. A measure of how much the utility function varies between negotiations for the same product.

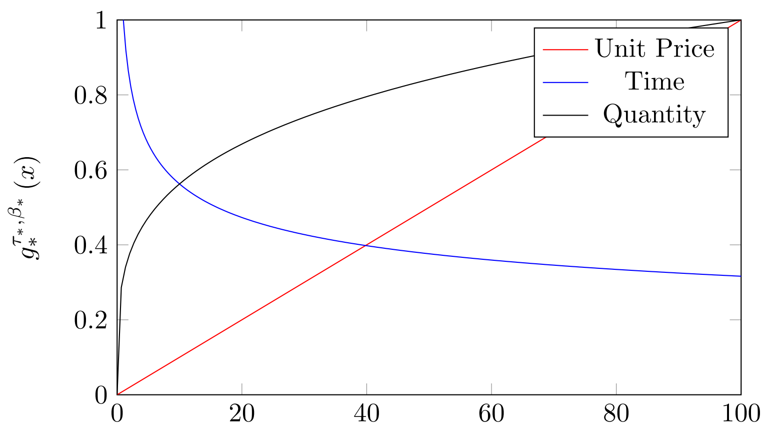


Figure 6.1: The three components g_* of the miners' utility functions—unit price, time, and quantity—plotted using the 2019 SCM league settings; in particular, $\beta_* = \hat{\beta}_*$ and $\tau_* = \hat{\tau}_*$.

Miners' Utilities At a high-level, miners should prefer to mine fewer raw materials, as late as possible, charging the highest possible prices. In fact, to increase the simulation throughout, miners prefer to mine more, rather than fewer, raw materials. These assumptions lead to the form of the miner's utility function, described in Equation 6.1, and generated via Algorithm 4.

To simplify notation as we describe this function (in both the text and in the algorithm), we drop the miner subscript m from all entries in the profile tuple (e.g., $\hat{\alpha}_p \equiv \hat{\alpha}_{pm}$). Given contract parameters (u, q, t) denoting unit price, quantity, and execution time, respectively, for all products $p \in P$,

$$U^{\alpha_*,\tau_*,\beta_*}(u,q,t) = \begin{cases} 0, & u < 0 \text{ or } q < 0 \text{ or } t < 0 \\ \alpha_u(p)\, g_u^{\tau_*,\beta_*}(u) + \alpha_q(p)\, g_q^{\tau_*,\beta_*}(q) + \alpha_t(p)\, g_t^{\tau_*,\beta_*}(t), & \text{otherwise} \end{cases} \tag{6.1}$$

The parameters α_*, where $*$ is the issue name (i.e., $* \in \{u, q, t\}$), are values in $(0, 1)$ drawn from a Dirichlet distribution that varies with the miner and the product. The parameters τ_* and β_*, where $*$ is again the issue name, are drawn from a normal distribution that likewise varies with the miner and the product. The functions g_* are monotonic in the issue value, $x \in \mathbf{R}_0^+$:

$$g_*^{\tau_*,\beta_*}(x) = \left(\frac{x}{\beta_*}\right)^{\tau_*}, \tag{6.2}$$

Algorithm 4

```
1: procedure MINER UFUNGENERATOR($p, \hat{\alpha}_*, \hat{\tau}_*, \hat{\beta}_*, \mathrm{cv}$)
2:     $\alpha_*(p) \sim$ Dirichlet $(\hat{\alpha}_*(p))$        ▷ Sample issue weights (unit price, quantity, delivery time)
3:     for $* \in \{u, q, t\}$ do        ▷ Generate the control parameters of $g_*$
4:         $\tau_*(p) \sim \mathcal{N}(\hat{\tau}_*(p), \mathrm{cv} \cdot \hat{\tau}_*(p))$
5:         $\beta_*(p) \sim \mathcal{N}\left(\hat{\beta}_*(p), \mathrm{cv} \cdot \hat{\beta}_*(p)\right)$
       return $U^{\alpha_*,\tau_*,\beta_*}(u, q, t)$        ▷ see Equation 6.1
```

N.B. When $\mathrm{cv} > 0$, Algorithm 4 generates a fresh set of utility functions at every time step. Consequently, even if a miner already negotiated with another agent about an existing CFP without success, it will behave differently the next time, so their negotiation may succeed.

6.2 Consumers' Characterization

Consumers are characterized by their utilities, which in turn depend on their profiles. Hence, this section starts with a detailed description of consumers' profiles, and then describes their utilities.

Consumers' Profiles A consumer c's profile is a tuple π_c whose entries are used to determine the utility of consuming each final product:

$P_c \subseteq P$ The products that the consumer can consume.

$B_c \in R_0^+$ Level of *insufficient products* breaches accumulated by an agent above which the consumer will not negotiate with that agent. This level is calculated as the sum of all such breach levels.

Consumption Schedule $S_c : P_c \times N \to \mathbf{Z}_0^+$ The consumption schedule of each product. Note that this schedule varies with time.

Dynamicity $D_c \in [0, 1]$ How dynamic is the consumption schedule the consumer. Zero means that the customer's schedule is fixed (given by S_c), and will not change over the course of the simulation. The larger D_c is, the more frequently will the consumer re-evaluate its consumption schedule based on the history of its consumption.

Underconsumption Flexibility $\hat{U}_c : P_c \to [0, 1]$ How flexible is the consumer in the face of underconsumption.

Overconsumption Flexibility $\hat{O}_c : P_c \to [0, 1]$ How flexible is the consumer in the face overconsumption.

$\hat{\alpha}_{uc} : P_c \to (0, 1)$ A measure of how much the utility of a contract for a product is affected by the unit price.

$\hat{\alpha}_{qc} : P_c \to (0, 1)$ A measure of how much the utility of a contract for a product is affected by the quantity.

$\hat{\tau}_{uc} : P_c \to (0, 1)$ **and** $\hat{\beta}_{uc} : P_c \to \mathbf{R}^+$ Control the shape of the function representing the contribution of the unit price to the utility of a contract for a product.

$\hat{\tau}_{qc} : P_c \to (0, 1)$ Controls the shape of the function representing the contribution of the quantity to the utility of a contract for a product.

$\mathbf{cv}_c \in \mathbf{R}_0^+$ The coefficient of variation used for random sampling. A measure of how much the utility function varies between negotiations for the same product.

Consumers' Utilities Each consumer is assigned a consumption schedule S based on its profile. The utility functions reward consumers who follow their schedules closely, and penalize deviations from them. These assumptions lead to the form of the consumer's utility function, described in Equation 6.3, and generated via Algorithm 5.

To simplify notation, we drop the consumer subscript c from all entries in the profile tuple (e.g., $\hat{\alpha}_p \equiv \hat{\alpha}_{pc}$). Given contract parameters (u, q, t) denoting unit price, quantity, and execution time, respectively, for all products $p \in P$,

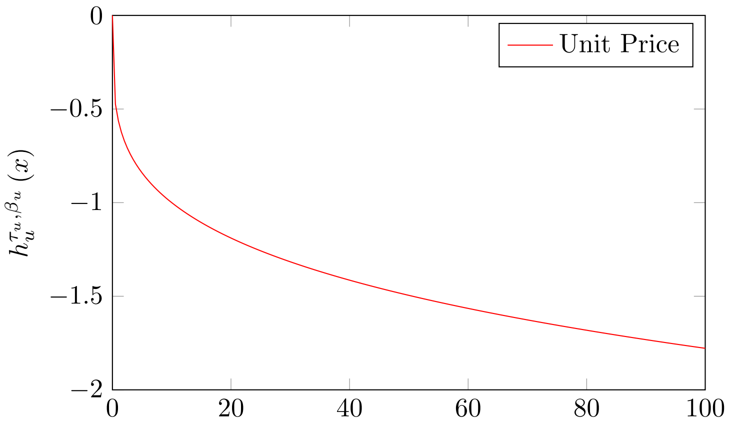
$$U_S^{\alpha_*,\tau_*,\beta_u,U,O}(u, q, t) = \begin{cases} 0, & u < 0 \text{ or } q < 0 \text{ or } t < 0 \\ \alpha_u(p)\, h_u^{\tau_u,\beta_u}(u) + \alpha_t(p)\, h_q^{\tau_q,U,O}(q, S(t)), & \text{otherwise} \end{cases} \tag{6.3}$$

The parameters α_*, where $*$ is the issue name (i.e., $* \in \{u, q, t\}$), are values in $(0, 1)$ drawn from a Dirichlet distribution that varies with the consumer and the product. The parameters $\beta_u, \tau_u, \tau_q, U$, and O, are drawn from a normal distribution that likewise varies with the consumer and the product. The function $h_u^{\tau_u,\beta_u}$ is monotonic in the unit price, $x \in \mathbf{R}_0^+$:

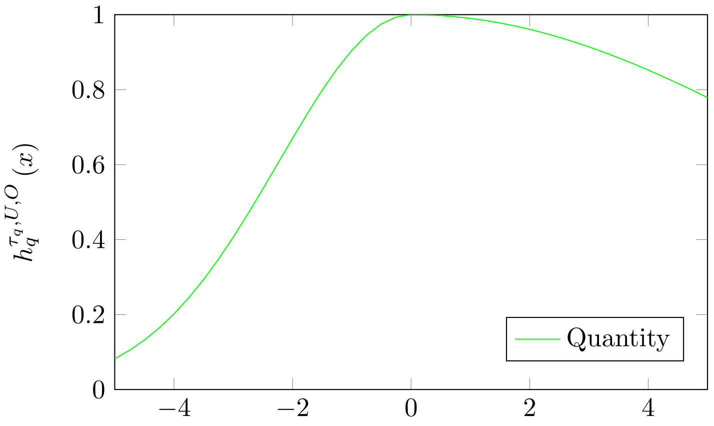
$$h_u^{\tau_u,\beta_u}(x) = -\left(\frac{x}{\beta_u}\right)^{\tau_u} \tag{6.4}$$

The function $h_q^{\tau_q,U,O}$ takes as input two quantities, the first given by the contract, and the second, by the consumer's schedule at time t. The function has the following form:

$$h_q^{\tau_q,U,O}(x,y) = \begin{cases} e^{-U\left(\frac{y-x}{y}\right)^{\tau_q}} & x \leq y \wedge y \neq 0 \\ e^{-O\left(\frac{x-y}{y}\right)^{\tau_q}} & x \geq y \wedge y \neq 0 \end{cases} \tag{6.5}$$



(a) The component h_u (unit price) of the consumers' utility functions, plotted using the 2019 SCM league settings; in particular, $\beta_u = \hat{\beta}_u$ and $\tau_u = \hat{\tau}_u$.



(b) The component h_q of the consumers' utility functions, plotted using the 2019 SCM league settings; in particular, $O_q = \hat{O}_q$, $U_q = \hat{U}_q$, and $\tau_q = \hat{\tau}_q$.

Figure 6.2: Components of the consumers' utility functions.

N.B. When cv > 0, Algorithm 5 generates a fresh set of utility functions at every time step. Consequently, even if a consumer already negotiated with another agent about an existing CFP without success, it will behave differently the next time, so their negotiation may succeed.

6.3 Banking

At present (i.e., for the 2019 SCM league), agents do not negotiate with the bank. The bank simply offers agents non-negotiable loans (assuming they have not gone bankrupt). The agent can accept any fraction of an offer, or reject it outright.

The bank offers loans to agents in two particular circumstances:

Algorithm 5

```
1: procedure CONSUMER UFUNGENERATOR($p, \hat{\alpha}_*, \hat{\tau}_*, \hat{\beta}_u, \hat{U}, \hat{O}, \text{cv}$)
2:     $\alpha_*(p) \sim$ Dirichlet $(\hat{\alpha}_*(p))$        ▷ Sample issue weights (unit price, quantity, delivery time)
3:     $\beta_u(p) \sim \mathcal{N}\left(\hat{\beta}_u(p), \text{cv} \cdot \hat{\beta}_u(p)\right)$
4:     for $\hat{x} \in \left\{\hat{\tau}_u, \hat{\tau}_q, \hat{U}, \hat{O}\right\}$ do        ▷ Generate the control parameters of $h_*$
5:         $x(p) \sim \mathcal{N}(\hat{x}(p), \text{cv} \cdot \hat{x}(p))$
       return $U_S^{\alpha_*,\tau_*,\beta_u,U,O}(u, q, t)$        ▷ see Equation 6.3
```

1. When a contract is about to be executed, and the agent is a buyer, but the balance in their wallet is less than the full value of the contract.

   Assume a contract c with unit price u_c, quantity q_c, and time t_c is due to be executed. If the buyer's available balance b is insufficient (i.e., if $b < u_c q_c$), then the bank automatically offers the buyer a loan of any amount up to $l_{\max} = u_c q_c - \max(0, b)$.

2. When the balance in the agent's wallet is insufficient to make a payment on an outstanding loan.

   Assume that an installment of value v is due on a loan. If the agent's balance $b < v$, then the bank automatically offers the agent a new loan in the (exact) amount of $l_{\max} = v - \max(0, b)$.

All loans must be paid back in a fixed number of installments k, starting after a fixed grace period g, at an interest rate η. As such, the total amount of a loan is $l(1+\eta)^k$, where $l \leq l_{\max}$ is the amount borrowed. This total is divided into k equal installments to be paid back one per time step, starting after g time steps.

The number of installments and the grace period are simulator parameters. The interest rate is given by:

$$\eta = \eta_0 + \left(\frac{\eta_0 - \eta_{\max}}{L_{\max}}\right) L + \delta_t\, g + \delta_r\,(r_0 - r)\,,$$

where η_0 is the base interest rate, $\eta_{\max}$ is the maximum interest rate, $L_{\max}$ is the loan amount at which the maximum interest rate is charged, L is what the agent owes the bank from previous loans (if anything), δ_t is an interest increment per step during the grace period, δ_r is an interest increment per unit of *drop* in credit rating, r_0 is the agent's initial credit rating, and r is their current credit rating. Values for η_0, $\eta_{\max}$, $L_{\max}$, δ_t, δ_r, and r_0 are all simulator parameters. The credit rating r is reduced from its initial value r_0 whenever the agent fails to make a payment on a loan, but recovers over time (see below). The base rate η_0 varies with the two circumstances under which loans are offered. It is generally higher for loans needed to repay earlier loans than for those needed to pay for contracted products.

Credit Rating Calculation A low credit rating is indicative of an agent that does not pay back its loans. An agent's credit rating is thus a function of the agent's missed payments.

All agents start with an initial credit rating r_0. Whenever an agent misses a payment, its credit rating is updated as $r = r_0 \exp^{-\lambda_{\text{cr}} x}$, where x denotes the agent's missed payments, and λ_{cr} controls how fast the credit rating decreases with this amount.

Note that the input to this formula—missed payments—is not a mere sum; rather, it is a discounted sum. For discount factor $\gamma_{\text{cr}} \in (0, 1]$, missed payments $= \sum_{\tau=0}^{t} \gamma_{\text{cr}}^{t-\tau} x_\tau$, where x_t denotes a missed payment at time step t. Discounting allow agents to recover, with time, from a bad credit rating.

If an agent goes bankrupt, the missed payment at that time step is the *bankruptcy missed payment value*.

6.4 Insurance

An agent can query the insurance company to learn the cost of an insurance policy for any (hypothetical or) real contract c (expected to be or) already agreed upon at time e_c, if contracted at any time step $t \geq e_c$.

A policy's premium is a fraction of the contract's value, $u_c q_c$. This fraction is a function of three simulator parameter settings: α, the base insurance premium; α_t, how the insurance premium increases over time; and δ_α, how the insurance premium varies with breaches.

More specifically, the insurance premium associated with contract c at time step t is given by:

$$\text{policy premium} = \begin{cases} \alpha_c \left(1 + \alpha_t \left(t_c - t\right)\right) & e_c \leq t < t_c \\ \infty & \text{otherwise} \end{cases}$$

where

$$\alpha_c = \alpha + \delta_\alpha \times \text{sum\_total\_of\_partner's\_breach\_levels}$$

Insurance policies become more expensive over time, up until the contract's execution time, at which point their cost is infinite. Insurance policies also cost more when the trading partner is more prone to breaches.

Chapter 7

2019 ANAC SCM League

This chapter describes the SCM league as it will be run during its 2019 incarnation as part of ANAC 2019. The general SCM world design described earlier in this document was simplified in a number of ways to reduce the barriers to entry for participants. These simplifications were carefully designed so as to keep the SCM league relevant as a research challenge.

7.1 World Design

Production Graph The production graph is a chain, with a single raw material and a single final product. All processes take a single input at the beginning of a time step, and produce a single output at the end. The number of manufacturing processes is therefore the number of intermediate products plus one, the final product. The number of intermediate products is sampled uniformly from $\mathcal{U}\{1, 4\}$.

[**Catalog Prices**] The single raw material (p_0) is assigned a catalog price of 1. The catalog prices of all other products are calculated as follows:

$$p_i = 1.15 \times (p_{i-1} + \mu_i)$$

where p_{i-1} is the single input to the single process that generates p_i, and μ_i is the average cost for running this process (which depends on the factories' profiles).

Factories The maximum storage capacity of all factories is infinite. Each factory's storage is initially empty, and the initial balance in their wallet is $1,000$ units of currency.

All factories have 10 lines, each with the same profile, but different factories can have different profiles.

[**Line Profiles**] A Line can run only a single manufacturing process. All manufacturing processes complete in a single time step. The cost associated with a process is sampled uniformly from $\mathcal{U}\{1, 4\}$.

Miners The coefficient of variation cv_m for all miners is 0.05. This setting leads to miners with similar – but not the same – utility functions. The number of miners is fixed at 5.

[**Miner Profiles**] All miners share the following profile:

$$\hat{\alpha}_u = \hat{\alpha}_t = \hat{\alpha}_q = 1.0$$

$$\hat{\beta}_u = \hat{\beta}_q = 100.0, \hat{\beta}_t = 1.0$$

$$\hat{\tau}_u = 1.0, \hat{\tau}_t = -0.25, \hat{\tau}_q = 0.25$$

Consumers The coefficient of variation cv_c for all consumers is 0.1. This setting leads to consumers with similar utility functions. The number of consumers that can consume the single final product is fixed at 5.

[**Consumer Profiles**] All consumers share the following profile:

$$\hat{U}_c = 0.1, \hat{O}_c = 0.01$$

$$\hat{\alpha}_u = 1.0, \hat{\alpha}_q = 0.5$$

$$\hat{\tau}_u = 0.25, \hat{\tau}_q = 2.0$$

$$\hat{\beta}_u = 10.0$$

Furthermore, at all time steps t, the consumption schedule for all consumers c and for all products p (i.e., $S_c(p,t)$) is sampled uniformly from $\mathcal{U}\{0,5\}$, and dynamicity is set to zero (i.e., $D_c = 0$).

Default Factory Manager The greedy factory manager this year will use a greedy scheduler with a *latest* scheduling strategy and a scheduling time-horizon that covers one step after the last contract signed or being considered. It will always buy insurance, and will not consider other running negotiations when calculating the utility on any negotiation but will consider all signed contracts.

7.2 Simulator Parameters

The behavior of the simulator is controlled by the following global parameters which control the simulator itself, the bank, and the insurance company. The name of the corresponding parameter in the league's platform (NegMAS) is given in parentheses in the following subsections.

Global Parameters

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

Total simulation time (*time_limit*) $\in \mathbf{R}^{+}_{\infty}$ The maximum number of seconds in a single run of the SCM world.

Negotiation Time Limit (*neg_time_limit*) $\in \mathbf{R}^{+}_{\infty}$ The maximum number of seconds of the alternating offers protocol in any simulated negotiation.

Negotiation Number of Steps Limit (*neg_n_steps*) $\in \mathbf{Z}^{+}_{\infty}$ The maximum number of steps (i.e., offers) of the alternating offers protocol in any simulated negotiation.

Negotiation Time Per Step Limit (*neg_step_time_limit*) $\in \mathbf{Z}^{+}_{\infty}$ The maximum time in seconds per step (i.e., offers) of the alternating offers protocol in any simulated negotiation.

Negotiation Speed Multiplier (*negotiation_speed*) $\in \mathbf{Z}^{+}_{\infty}$ The number of rounds of the alternating offers protocol per simulation time step.

Immediate Negotiations (*start_negotiations_immediately*) $\in \{\mathbf{yes}, \mathbf{no}\}$ If set to *yes* (*no*), after a new negotiation is registered, negotiations start immediately (during the next time step).

Default Signing Delay (*default_signing_delay*) Default signing delay, if not negotiated in a contract.

Transportation Delay (*transportation_delay*) The number of steps it takes to move products from the seller to the buyer.

Negotiable Penalties $\in \{\mathbf{yes}, \mathbf{no}\}$ Whether penalties can be negotiated.

Allow Breach Renegotiations (*breach_processing*) $\in \{\mathbf{yes}, \mathbf{no}\}$ Whether or not agents are allowed to renegotiate in case of a breach.

Global Breach Penalty (*breach_penalty_society*)$\in \Re_0^+$ The penalty to "society" for committing a breach as a fraction (or multiple) of the contract's value. This penalty is disabled when set to zero.

Bank Parameters

The bank is characterized by the following parameters:[1]

Initial Credit Rating (*initial_credit_rating*) $r_0 \in \Re^+$ The value at which all agents' credit ratings are initialized.

Credit Rating Adjustment Factor (*cr_adjustment*) $\lambda_{\mathbf{cr}} \in \Re^+$ The factor by which all agents' credit ratings are adjusted.

Credit Rating Discount Factor (*cr_discount*) $\gamma_{\mathbf{cr}} \in (0, 1]$ The factor by which all agents' missed payments are discounted.

Base Interest Rate For Manufacturing Loans (*base_ir_manfacturing*) $\eta_0^m \in (0, 1) \bigcup \infty$ The base interest rate for borrowing to pay for contracted products. Setting this rate to ∞ disables the possibility of overdraft.

Base Interest Rate For Loan Installments (*base_ir_installment*) $\eta_0^l \in (0, 1) \bigcup \infty$ The base interest rate for borrowing to make loan payments. Setting this rate to ∞ disables the possibility of overdraft.

Maximum Interest Rate (*interest_max*) $\eta^{\max} \in [1, \infty)$ Maximum interest rate for with no grace period and at maximum credit rating.

Loan at Maximum Interest Rate (*loan_at_max_interest*) $L_{\max} \in \Re^+$ The loan value at which maximum interest rate is charged.

GP Interest Rate Increment (*interest_time_increase*) $\delta_t \in \Re^+$ Interest rate increase per unit during the grace period.

CR Rate Increment (*interest_cr_increase*) $\delta_r \in \Re^+$ Interest rate increase per unit drop in credit rating.

Bankruptcy Missed Payment Value (*bankruptch_penalty*) $\in \Re^+$ The missed payment value equivalent to going bankrupt.

Insurance Company Parameters

The insurance company is characterized by the following parameters:

Base Insurance Premium (*premium*) $\alpha \in (0, 1) \bigcup \infty$ The base cost of an insurance policy, which is a fraction of the contract's value.

Insurance Premium Time Increment (*premium_time_increment*) $\alpha_t \in \mathbf{R}_0^+$ Specifies how much the insurance premium increases over time.

Insurance Premium Breach Increment (*premium_breach_increment*) $\delta_\alpha \in \mathbf{R}_0^+$ Specifies how much the insurance premium increases with each additional breach covered by an insurance policy.

Table 7.1 lists the simulator settings for the 2019 SCM league.

[1]The bank is disabled in the ANAC 2019 SCM league.

Table 7.1: Simulator settings for the 2019 SCM league.

Setting	**Value**	**Notes**
Number of simulation time steps	50 to 100	
Total simulation time	$7,200$	Two hours
Negotiation Settings		
Negotiation time limit	120	Two minutes
Negotiation number of steps limit	20	All negotiations end after 20 steps (offers).
Negotiation time per step limit	10	All negotiations time out after 10 seconds without an offer or counteroffer.
Negotiation speed multiplier	21	Ensures that all negotiations end during the same time step in which they begin.
Immediate negotiations	no	Whether a registered negotiation starts immediately or at the next time step.
Default Signing delay	1	Agreements are signed as contracts after one step.
Production Settings		
Transportation delay	0	No transportation delay.
Penalty and Breach Settings		
Negotiable penalties	yes	
Allow breach renegotiations	yes	Enabled, but built-in agents do not renegotiate.
Global breach penalty	0.02	A penalty of 2% is imposed on all breaches that are not successfully renegotiated (In an earlier version of this document this was incorrectly written as 200%).
Insurance Parameters		
Base insurance premium	0.1	
Insurance premium time increment	0.1	
Insurance premium breach increment	0.1	
Bank Parameters		
The bank is disabled. No loans are allowed		

Chapter 8

Agent Evaluation

This chapter describes how agents will be evaluated in the 2019 SCM league. In short, their performance will averaged across multiple world executions, with winners determined by statistical significance testing.

8.1 Tournament Design

There will be three separate competitions in the 2019 SCM league. In the first, the basic competition, at most one instantiation of each team's agent will run in each simulation. In some of these simulations, all the other agents will be greedy factory manager agents. In others, agents submitted by other teams will also participate, but again at most one instantiation of each.

In the second, the collusion competition, multiple instantiations of the same team's agent may run during a single simulation (with multiple greedy factory manager agents as well). The exact number of instantiations of each will vary across simulations, and will not be announced in advance. In this competition, it is possible for instances of the same agent to try to collude with one another to corner the market, or exhibit other collusive behaviors.

The final, the sabotage competition, is intended to uncover fragile aspects of the SCML design. Teams who enter this competition should try to sabotage the market, for example, by preventing trades, or by negatively impacting the profits accrued by others. Sabotaging agents will not compete against one another directly; they will be evaluated independently in the presence of non-sabotaging agents only. Furthermore, they will be excluded from the other two competitions.

8.2 Evaluation Criteria

An agent's performance will be measured by its score. In the sabotage competition, agents' profits will not factor into their score; only their ability to sabotage the market/game will matter. The exact metrics by which sabotage will be scored will not be revealed in advance, as we do not want saboteurs to tailor their strategies towards any particular forms of sabotage; we are interested in eliciting their creativity.

In the basic and collusion competitions, an agent's score will be the average profits accrued by all its factories in all its instantiations in all simulations. The profit accrued by a factory will be calculated as follows:

$$\text{Profit} = \frac{(B_N - B_0) + (f(S_N) - f(S_0))}{B_0 + f(S_0)}, \tag{8.1}$$

where B_0 and B_N are the agent's wallet balances at the beginning and end of the simulation, respectively, S_0 and S_N are the storage states at the beginning and end of the simulation, respectively, and f is the **storage valuation method**.

There are several possible storage valuation methods, including valuing products at their *catalog prices*, their *average selling price in the world*, their *average selling price per agent*, or simply at zero. In 2019, all products in storage will be valued at zero. This will encourage agents to try to sell everything they produce before the end of the simulation.

8.3 Winners and the SCML Hall of Fame

Given the scores of the agents in each competition, as well as the scores of the default factory manager agents provided by the organizing committee, the **winner** of each competition will be the team whose agent that achieves the (absolute) highest score (i.e., the highest average profit). Moreover, the winner will receive one of the following badges based on its relative score, as compared to that of the other agents:

A: Competition Champion The winner's score is higher than that of all other agents, including the default factory manager agents, as evaluated by a one-sided equal-variance t-test.

In this case, the difference between the winner's score and all other agents' is statistically significant.

B: Team Champion The winner is not a competition champion, but the winner's score is higher than that of all but the default factory manager agents, as evaluated by a one-sided equal-variance t-test.

In this case, the winner's score *is* statistically different from all other teams', but is *not* statistically different from one or more of the default agents'.

C: Winner The winner is not a team champion and the winner's score is not higher than that of all other teams', as evaluated by a one-sided equal-variance t-test.

In this case, the winner's score is *not* statistically different from at least one of the other agents'. All teams whose agents achieve scores that are not statistically different from the winner's will be inducted into the SCM league's **hall of fame**.

Appendix A

Supply-driven Markets

Supply-driven markets are driven by supply. In the SCM world, this amounts to proactive miners driving supply by posting sell CFPs. Consumers at the other end of the chain merely react to these sell CFPs by offering to buy. Their demand is never exhausted.

A.1 Proactive Miner Agents

Recall that in supply-driven markets, miners are proactive. They drive supply by posting sell CFPs.

This section describes how the miners' proactive behavior is implemented, by describing their implementations of the various callbacks and actions common to all SCM agents.

On Init No op.

On Step The proactive miners' step function, which determines how and when they publish CFPs, is described in Algorithm 7. This function depends on the miner's profile (See Section 6.1). Specifically, it depends on their mining schedule S_m and how dynamic that schedule is D_m.

On New CFP Proactive miners always ignore new CFPs, as they never request negotiations.

Confirm Loan Miners never need loans, as they never buy anything.

Respond to Negotiation Request Proactive miners accept negotiation requests from non-bankrupt agents with a breach level less than a predefined *max breach level*. Whenever a proactive miner accepts a negotiation request about some product p, it spawns a negotiator using Algorithm 6.

Algorithm 6

```
1: procedure Miner RespondToNegRequest
2:     if the miner can mine the CFP's product then
3:         Create a utility function (See Section 6.1, Algorithm 4).
4:         Spawn a negotiator with this utility function and reservation value zero.
5:         Accept the negotiation request using this negotiator.
```

A.2 Reactive Consumer Agents

Recall that in supply-driven markets, consumers are reactive. In particular, consumers never issue buy CFPs; they merely respond to other agents' sell CFPs.

This section describes how the consumers' reactive behavior is implemented, by describing their implementations of the various callbacks and actions common to all SCM agents.

Algorithm 7

```
1: procedure MINER STEP($S_m, D_m, n$)
2:     if $n = 0$ then
3:         $S \leftarrow S_m$
4:     else
5:         Delete all my existing CFPs from the bulletin board.
6:         Calculate average quantity across existing contracts at each time step.
7:         Average these averages to calculate the average contract quantity per time step.
8:         for $p \in P$ do                                                    ▷ For all products
9:             $C(p) \leftarrow$ Total quantity of product $p$ already contracted to be delivered at time $n$.
10:            $S(p) \leftarrow (1 - D_m)\,(S_m - C(p)) + D_m\,\bar{q}$
11:    PUBLISHCFPS($S, n$)
12: procedure PUBLISHCFPS($S, n$)
13:    for $p \in P$ do                                                        ▷ For all products
14:        unit price $\leftarrow$ negotiable
15:        product $\leftarrow p$
16:        time $\leftarrow n$
17:        quantity $\leftarrow [1, S(p)]$
18:        Publish a CFP with these parameters.
```

On Init No op.

On Step No op. (Reactive consumers are just that: reactive.)

On New CFP Reactive consumers respond to all sell CFPs for products they can consume, given that they are initiated by non-bankrupt agents whose total breach level is than a predefined *max breach level*. To do so, they first generate a utility function to guide the negotiation. Then, they spawn a negotiator with this utility function (and reservation value ∞). Finally, they request a negotiation with the agent who publised the CFP using this negotiator.

The reactive consumer's **OnNewCFP** function is described in Algorithm 8.

Algorithm 8

```
1: procedure REACTIVE CONSUMER ONNEWCFP
2:     if the consumer can consume the CFP's product then
3:         Create a utility function (See Section 6.2, Algorithm 5).
4:         Spawn a negotiator with this utility function and reservation value $\infty$.
5:         Request a negotiation with the agent who published the CFP using this negotiator.
```

Confirm Loan Consumers never need loans, as their balances are infinite.

Respond to Negotiation Request Reactive consumers do not need to respond to negotiation requests as they never post CFPs, so they are never approached by agents to negotiate with them.

Bibliography

[AFHJ17] Reyhan Aydoğan, David Festen, Koen V. Hindriks, and Catholijn M. Jonker, *Alternating offers protocols for multilateral negotiation*, pp. 153–167, Springer International Publishing, Cham, 2017.

[GSS82] Werner Güth, Rolf Schmittberger, and Bernd Schwarze, *An experimental analysis of ultimatum bargaining*, Journal of Economic Behavior & Organization **3** (1982), no. 4, 367 – 388.