CMieux: Adaptive Strategies for Competitive Supply Chain Trading

Michael Benisch
School of Computer Science,
Carnegie Mellon University
mbenisch@cs.cmu.edu

Alberto Sardinha
School of Computer Science,
Carnegie Mellon University
alberto@cs.cmu.edu

James Andrews
School of Computer Science,
Carnegie Mellon University
jandrews@andrew.cmu.edu

Norman Sadeh
School of Computer Science,
Carnegie Mellon University
sadeh@cs.cmu.edu

Abstract

Supply chains are a central element of today's global economy. Existing management practices consist primarily of static interactions between established partners. Global competition, shorter product life cycles and the emergence of Internet-mediated business solutions create an incentive for exploring more dynamic supply chain practices. The Supply Chain Trading Agent Competition (TAC SCM) was designed to explore approaches to dynamic supply chain trading between automated software agents. TAC SCM pits against one another trading agents developed by teams from around the world. Each agent is responsible for running the procurement, planning and bidding operations of a PC assembly company, while competing with others for both customer orders and supplies under varying market conditions. This paper presents Carnegie Mellon University's 2005 TAC SCM entry, the CMieux supply chain trading agent. CMieux implements a novel approach to coordinating supply chain bidding, procurement and planning, with an emphasis on the ability to rapidly adapt to changing market conditions. We present empirical results based on 200 games involving agents entered by 25 different teams during what can be seen as the most competitive phase of the 2005 tournament. Not only did CMieux perform among the top five agents, it significantly outperformed these agents in procurement while matching their bidding performance. We also simulated 40 games against the best publically available agent binaries. Our results show CMieux has significantly better average overall performance than any of these agents in this setting.

1 Introduction

Existing supply chain management practices consist primarily of static interactions between established partners [4]. As the Internet helps mediate an increasing number of supply chain transactions, there is a growing interest in investigating the potential for exploring the benefits of more dynamic supply chain practices [1, 12] that involve automated software agents that negotiate many of the trades. The Supply Chain Trading Agent Competition (TAC SCM) was designed to explore such approaches to dynamic supply chain trading. TAC SCM pits against one another trading agents developed by teams from around the world. Each agent is responsible for running the procurement, planning and bidding operations of a PC assembly company, while competing with others for both customer orders and supplies under varying market conditions. Specifically, the game features a number of different types of computers, each requiring different sets of components that can be procured from multiple suppliers. Agents make money by selling and delivering finished PCs to customers. Supplier and customer market conditions stochastically change over time and from one game to another to ensure that agents are tested across a broad range of representative situations.

This paper presents Carnegie Mellon University's 2005 TAC SCM entry, the CMieux automated supply chain trading agent. CMieux's architecture departs markedly from traditional Enterprise Resource Planning architectures and commercially-available supply chain management solutions through its emphasis on tight coordination between supply chain bidding, procurement and planning. Through this coordination, our trading agent is capable of adapting rapdily to changing market conditions and outperform its competitors. In particular, we present empirical results based on 200 games involving agents entered by 25 different teams during what can be seen as the most competitive phase of the 2005 tournament. Not only did CMieux perform among the top five agents, it significantly outperformed these agents in procurement while matching their bidding performance. We also present results from 40 simulated exhibition games against the top five publically available agent binaries. The results of these games show that CMieux has significantly better average overall performance than any of the public agents in this setting.

The remainder of this paper is organized as follows. Section 2 summarizes the TAC SCM environment, and highlights the features and challenges from a planning and scheduling perspective. Section 3 presents an overview of CMieux and a detailed description of its underlying modules. Sections 4 and 5 present empirical results and concluding remarks.

2 TAC Supply Chain Management

This section provides a summary of the TAC Supply Chain Management game. The full description can be found in the official specification document [5].

The TAC SCM game is a simulation of a supply chain where six computer man-

ufacturer agents compete with each other for both customer orders and components from suppliers. A server simulates the customers and suppliers, and provides banking, production, and warehousing services to the individual agents. Every game has 220 simulated days, and each day lasts 15 seconds of real time. The agents receive messages from the server on a daily basis informing the state of the game, such as the current inventory of components, and must send responses to the same server until the end of the day indicating their actions, such as requests for quotes to the suppliers. At the end of the game, the agent with the highest sum of money is declared the winner.

Normally, each manufacturer agent tackles separately important sub-problems of a supply chain: procurement of components, production and delivery of computers, and computer sales. Figure 1 summarizes the high level interactions between the various entities in the game.

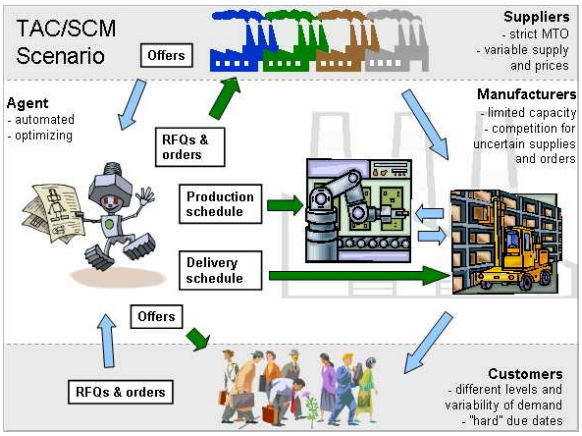


Figure 1: Summary of the TAC SCM Scenario

2.1 Procurement of Components

Each agent is able to produce and store 16 different computer configurations in their own production facility, by using different combinations of components. These computers are made from four basic components: CPUs, motherboards, memories, and hard drives. There are a total of 10 different components: two brands and speeds of CPUs, two brands of motherboards, and two sizes of hard disks and memories. The game includes 8 distinct suppliers, and each component has a base price that is used as reference for suppliers making offers. Each PC type also has a base price equal to the sum of the base prices of its components.

Every day, agents can send requests for quotes (RFQs) to suppliers with a given reserve price, quantity, type and delivery date. A supplier receives all RFQs on a given day, and processes them together at the end of the day to find a combination of offers

that approximately maximizes its revenue. On the following day, the suppliers send back to each agent an offer for each RFQ with a price, a possibly adjusted quantity, and a due date. Due to capacity restrictions, the supplier may not be able to supply the entire quantity requested in the RFQ by the due date. Thus, it responds by issuing up to two modified offers, each of which relaxes one of the two constraints:

- **Quantity**, in which case offers are referred to as *partial offers*.
- **Due date**, in which case offers are referred to as *earliest offers*.

The suppliers have a limited capacity for producing a component, and this limit varies throughout the game according to a mean reverting random walk. Moreover, suppliers also limit their long-term commitments by reserving some capacity for future business. The pricing of components is based on the ratio of demand to supply, and higher ratios result in higher prices. Each day the suppliers estimate their free capacity by scheduling production of components ordered in the past and components requested that day as late as possible. The price offered in response to an RFQ is equal to the requested components base price discounted by a function proportionate to the supplier's free capacity before the RFQ due date. The manufacturer agents normally face an important trade-off in the procurement process: pre-order components for the future where customer demand is difficult to predict, or wait to purchase components and risk being unsuccessful due to high prices or availability.

A reputation rating is also used by the suppliers to discourage agents from driving up prices by sending RFQs with no intention of buying. Each supplier keeps track of its interaction with each agent, and calculates the reputation rating based on the ratio of the quantity purchased to quantity offered. If the reputation falls bellow a minimum value, then the prices and availability of components are affected for that specific agent. Therefore, agents must carefully plan the RFQs sent to suppliers.

2.2 Computer Sales

The server simulates customer demand by sending customer requests for quotes (RFQ) to the manufacturer agents. Each customer RFQ contains a product type, a quantity, a due date, a reserve price, and a daily late penalty. Moreover, these customer requests are classified into three market segments: high range, mid range, and low range. Every day, the server sends a number of RFQs for each segment according to a Poisson distribution, with an average that is updated on a daily basis by a random walk. The total number of RFQs per day ranges between 80 and 320, and demand levels can change rapidly throughout the game. Thus, agents are limited in their ability to plan sales, production and procurement.

The manufacturer agents respond to the customer RFQs by bidding in a first price sealed bid reverse auction: agent's cannot see competitors bids, and the lowest offer price wins the order. Agents do receive market reports each day that inform the highest and lowest winning bid prices on the previous day.

2.3 Production and Delivery

Each manufacturer agent manages an identical factory, where it can produce any type of computer. The factory is simulated by the game server, and also includes a warehouse for storing components and finished computers. Each computer type requires a specified number of processing cycles, and the factory is also limited to produce 2000 cycles (approx. 360 units) per day.

Each day the agent sends a production schedule to the game server, and the simulated factory produces computers in the schedule for which the required components are available. A delivery schedule is also sent to the server on a daily basis, and it must specify the products and quantities of computers to be shipped to each customer order on the following day. Only computers available in inventory can be shipped to customers.

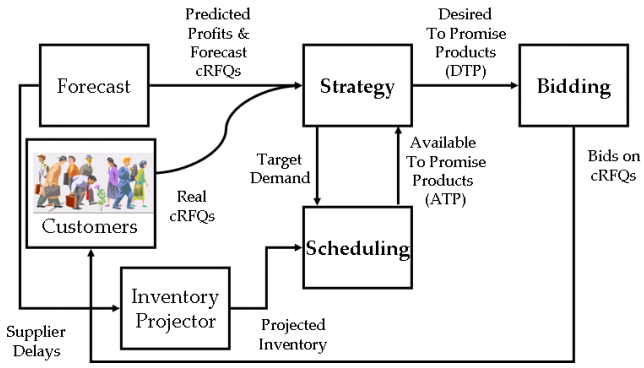
2.4 Related Work

Development teams of TAC SCM agents have proposed several different approaches for tackling important sub-problems in dynamic supply chains. Deep Maize [6] uses game theoretic analysis to factor out the strategic aspects of the environment, and to define an expected profitable zone of operation. The agent uses market feedback [8] to dynamically coordinate sales, procurement and production strategies in an attempt to stay in the profitable zone. SouthamptonSCM [7] presents a strategy for using fuzzy reasoning to compute bid prices on RFQs. RedAgent [11] presents an internal market architecture with simple heuristic-based agents that individually handle different aspects of the supply chain process. TacTex [9] presents machine learning techniques that were extended to form the customer bid price probability distributions in CMieux. The TacTex-05 team also offers considerable insight into the overall strategy behind their first-place agent in [10]. The Botticelli team [3] shows how the problems faced by TAC SCM agents can be modeled as mathematical programming problems, and offers heuristic algorithms for bidding on RFQs and scheduling orders.

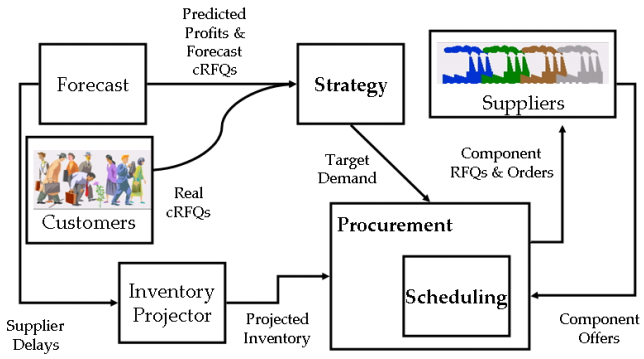
3 CMieux

A typical supply chain [4] may involve a variety of participants, such as: customers, retailers, wholesalers/distributors, manufacturers, and component/raw material suppliers. The objective of a supply chain is to maximize the overall value it generates, which is typically measured through profitability.

In a direct sales model [4], such as the one used by Dell Inc., a leading PC distributor, manufacturers fill customer orders directly. Retailers, wholesalers and distributors are bypassed, leaving only three participants - customers, manufacturers and suppliers. This is the most dynamic supply chain framework presently in use, which is the main reason that TAC is built around this SCM model. However, TAC SCM goes beyond



(a) B2C Interaction Overview



(b) B2B Interaction Overview

Figure 2: Primary interactions between modules for B2C and B2B operations in CMieux.

the limits of present practices by providing manufacturers with the opportunity to simultaneously search daily for the best supply prices, while concurrently adjusting asking prices based on changing market conditions.

Competitiveness in dynamic supply chain scenarios, such as those considered in TAC SCM, require significantly tighter integration of procurement, bidding and planning functionality than implemented in today's systems [1]. CMieux is dynamic supply chain trading agent that implements novel adaptive strategies to support this type of integration. In contrast to many other TAC SCM entries, CMieux *continuously re-evaluates* both low-level strategies, such as its current procurement plan, and high-level strategies, such as its current target market share.

3.1 Overview

Figure 2 shows the architecture of our CMieux supply chain trading agent, highlighting key interactions between its five main modules. The *bidding module* is responsible for

responding to customer requests with price quotes. The *procurement module* sends RFQs to suppliers and decides which offers to accept. The *scheduling module* produces a tentative assembly schedule for several days based on available and incoming resources (i.e. capacity and components). The *strategy module* makes all high-level strategic decisions, such as what fraction of the assembly schedule should be promised to new customers and what part of the demand to focus on. The *forecasting module* is responsible for predicting the prices of components and the future demand.

Figure 3 gives a general overview of CMieux's main daily execution path. The agent begins by collecting any new information from the server, such as the new set of supplier offers, and customer requests. This information is fed to the forecast module, which updates its predictions of future demand and pricing trends accordingly. The forecast demand is given to the strategy module to determine what part of it our agent should target. From the set of forecast future RFQs the strategy module chooses a subset as the target demand. The procurement module then determines whether or not to accept each newly acquired supplier offer. All offers from suppliers are accepted unless they are too late to be useful, or too expensive to remain profitable. The scheduling module builds a tentative tardiness minimizing production schedule for up to twenty days in the future. The schedule includes the agent's actual orders, and the future orders composing the target demand. The target demand orders are used to determine how many finished PCs the agent has Available to Promise (ATP).

On the Business to Consumer (B2C) side, the strategy module uses the tentative ATP and the forecast selling conditions from the forecasting module to determine what the agent Desires to Promise (DTP). The DTP is used by the bidding module, along with learned probabilistic models of competitor pricing. The bidding module chooses prices to maximize the agent's expected profit, while offering the amount of products specified by the DTP in expectation.

The procurement module determines how many components are needed to reach the level of inventory specified by the strategy module. It compares the desired levels to the projected levels, and determines what additional components are needed. Each day the procurement module attempts to procure a fraction of the needed components based on the prices and availability predicted by the forecasting module.

3.2 Forecast Module

The forecast module is an important part of the pro-active planning strategies employed by CMieux. It helps inform a number of key decisions, such as the planning of RFQs sent to suppliers and the setting of target market shares for different end products, about the current and future market landscape. A formal description of the main inputs and outputs of the forecast module are provided in Figure 4, the following outlines the module's two primary functions.

1. Update daily data structures with server information.
2. **Forecast Module** → update forecasts.
   - Predict future orders and prices using regressions
   - Predict component arrivals based on observed delays
3. **Strategy Module** → compute target demand.
4. **Procurement Module** → accept supplier offers.
   - Accepts offers that are reasonably priced.
   - Accepts partial offers that are sufficiently large.
   - Accepts earliest offers that are not excessively late.
5. **Scheduling Module** → make production schedule.
   - Uses dispatch scheduling and minimizes tardiness.
   - Available to Promise (ATP) products come from scheduled forecast orders.
6. **Strategy Module** → compute target sales.

7a. **Bidding Module** → compute customer offers.
   – Probability models of competitor pricing are used to maximize expected profit and sell DTP in expectation.

7b. **Procurement Module** → send supplier requests.
   – Target demand is broken into requests to minimize expected offer cost.

Figure 3: Overview of CMieux's daily main loop.

Forecast Inputs and Constants:

- $\boldsymbol{R}$, the set of observed customer RFQs.
- $\boldsymbol{O}^{\mathtt{C}}$, the set of customer orders received by the agent.
- $\boldsymbol{O}^{\mathtt{S}}$, the set of supplier orders received by the agent.
- $\boldsymbol{D}^{\mathtt{F}}$, the number of days to forecast into the future.

Forecast Outputs:

- $\hat{\boldsymbol{R}}$, a set of RFQs representative of those the agent will see up to $\boldsymbol{D}^{\mathtt{F}}$ days in the future.
- $\boldsymbol{f}^{\mathtt{C}} : \boldsymbol{j}, \boldsymbol{d} \rightarrow \mathbb{R}$, a function predicting the selling price of SKU $\boldsymbol{j}$ on day $\boldsymbol{d}$.
- $\boldsymbol{f}^{\mathtt{S}} : \boldsymbol{k}, \boldsymbol{d} \rightarrow \mathbb{R}$, a function predicting the purchase price of component $\boldsymbol{k}$ on day $\boldsymbol{d}$.

Figure 4: Forecast module inputs and outputs.

3.2.1 Customer Demand Forecasting

The first responsibility of the forecast module is to predict a set of RFQs representative of those our agent expects to see in the future. These RFQs are used by the strategy

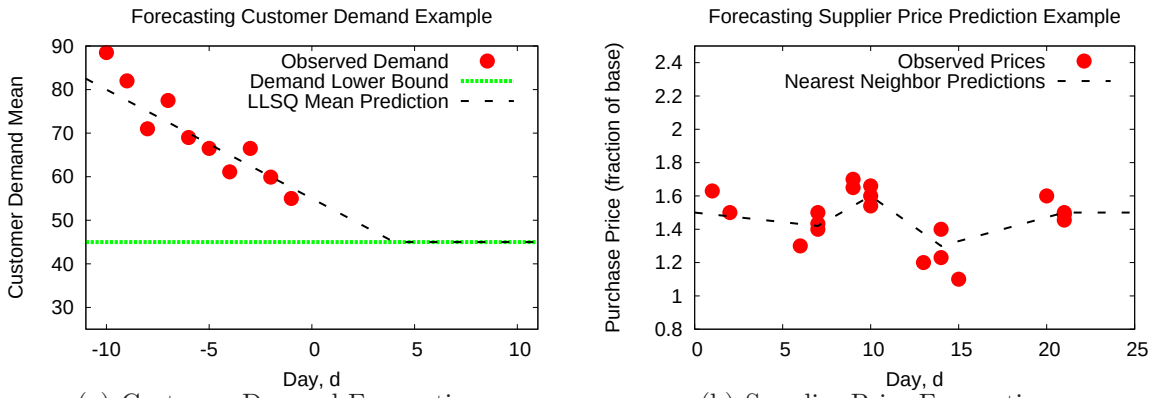
module to determine the agent's target demand. The forecast module generates a representative set of RFQs by predicting the mean and trend of the customer demand from past observations. Each of the different product grades (high, medium and low) in TAC SCM is governed by its own mean and trend. The actual number of RFQs of each type received each day is drawn from a Poisson distribution with the mean of that type. The mean for each product type changes geometrically each day based on the trend (the trend is multiplied by the mean and the result is added to the subsequent day's mean), and the trend is changed by a small amount each day according to a random walk. The forecast module attempts to predict each of the changing mean and trend of the Poisson distribution governing demand separately, using a linear least squares fit of observations from the past several game days. The predicted trends along with parameters given in the game specification are used to generate an appropriate set of RFQs.

3.2.2 Price Forecasting

The second responsibility of the forecast module is predicting the selling price of each product, and the purchasing price of each component up to $D^{\mathtt{F}}$ days into the future. This information is useful to several of the other modules in the agent, such as the procurement module, that base decisions on current market conditions. The product selling prices are predicted in the same fashion as the demand trends. A linear least squares (LLSQ) fit is computed for the selling prices of each product over the past several game days (additionally, we enforce lower and upper bounds on the predictions to ensure they remain relatively conservative).

The purchasing prices from a particular supplier are predicted using a nearest-neighbor (NN) technique based on historical prices quoted from that supplier. The forecast module predicts supplier prices on a particular day in the future by averaging observed quotes with nearby due-dates. Figure 5 shows examples of these two prediction techniques. On any given day in TAC SCM the agent is limited to a maximum of 5 requests per supplier and component type. Any of the requests that are unused by the procurement module are used as probes to aide the NN prediction of the forecast module. The probe dates are chosen to provide the most information, by picking days that are farthest from existing observations.

An additional responsibility of the forecast module is predicting the delays that the agent can expect on outstanding supplier orders. Suppliers delay the shipment of orders when their capacity stochastically descends below the level they had previously promised. The forecast module predicts the delays on outstanding orders based on the delays observed previously for each supplier and component type. For each product line it determines the delay on the most recent order and propagates it as the expected delay on all other outstanding orders. This relatively simple technique helps the planning aspects of the agent react early to a potential back-log in supplies.



(a) Customer Demand Forecasting (b) Supplier Price Forecasting

Figure 5: Examples of the techniques used by the forecast module to predict prices.

3.3 Strategy Module

The strategy module continuously re-evaluates and coordinates strategic decisions, including setting market share targets and selling quotas. These targets are continously tweaked to reflect both present and forecast market conditions.

More specifically, the strategy module determines what subset of the forecast customer RFQs the agent should aim to win (the "target demand") and what fraction of the its finished products the agent should plan on selling on any given day (the "desired to promise" products, or DTP). In other words, the strategy module modulates how the output of the forecast module impacts the procurement, scheduling and bidding modules (as illustrated in Figure 2). The primary inputs and outputs of the strategy module are summarized formally in Figure 6.

3.3.1 Computing Target Demand

On any particular day in the game, the strategy module must first determine the agent's target demand from the forecast demand. The goal of the strategy module is to target a fraction of the forecast demand that will lead to the highest overall profit (this is the agent's ultimate goal). In TAC SCM each agent competes with only five other agents. The agents can significantly impact their own profit margins by flooding or starving a market. Thus, targeting a larger percentage of the forecast will push profit margins down. On the other hand, agents have a limited factory capacity each day. If products are selling for a profit and factory capacity goes un-utilized, the un-used capacity is lost earning potential. This creates the need for a balance between decreasing target demand to increase profit margins, and still targeting enough demand to maintain high factory utilization.

The strategy module uses a heuristic to address this problem. When products are

Strategy Inputs and Constants:

- $\boldsymbol{O}$, the set of pending orders.
- $\hat{\boldsymbol{R}}$, future customer RFQs from *forecast module*.
- $\boldsymbol{f}^{\mathtt{C}}$, customer price function from *forecast module*.
- $\boldsymbol{f}^{\mathtt{S}}$, supplier price function from *forecast module*.
- $\boldsymbol{S}^{\mathtt{ATP}}$, the component of the production schedule from the *scheduling module* allocated to future orders.

Strategy Outputs:

- $\hat{\boldsymbol{O}}$, the set of orders representing a target demand, generated from actual orders and forecast future RFQs.
- $\hat{\boldsymbol{S}}$, quantities of PC that the agent currently desires to promise each day (DTP).

Figure 6: Strategy module inputs and outputs.

selling for a profit, it always targets exactly enough demand to stay at full utilization. The relative percentage of each product, or the *product mixture*, used to fill the target to full capacity is slightly adjusted each day based on the profit margin change of each product type. When the profit margin of a product increases (decreases), its relative percentage in the product mixture increases (decreases) slightly. Figure 7 summarizes this adjustment process for a single product.

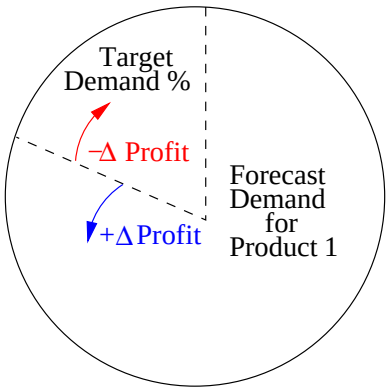


Figure 7: The strategy module adjusts the percentage of forecast for *each product* that enters target demand based on change in profit margin.

When a product is no longer being sold for a profit, the strategy module calculates the product mixture in the same way. However, the mixture is post-processed so that

the contribution of the unprofitable product is significantly decreased[1]. This may cause the total target demand to fall below full factory utilization[2].

3.3.2 Computing Desired to Promise (DTP)

After the target demand is computed by the strategy module, it is used by the scheduling module to develop a tentative production schedule for several days into the future (the *scheduling window*). The scheduling module uses information about incoming and available components, as well as previously committed orders. Using this information it determines when, if at all, each of the target orders will be produced (this process is described in Section 3.4). The part of this schedule assigned to filling target demand orders (as opposed to *actual* orders) indicates production that is not yet allocated to filling existing customer orders, or the available to promise (ATP) production.

As we have already explained, even at times when selling is profitable, an agent does not desire to sell more than it has to in order to maintain full factory utilization. Doing so can result in a flooded market and diminished profit margins. To avoid this, the strategy module uses the ATP schedule to determine what the agent actually desires to promise each day (the DTP). In an effort to sell as little as possible and still maintain full factory utilization, the DTP consists of PCs appearing only in the first two day of the ATP schedule. The first two days of the ATP include un-promised finished products and un-promised products being produced that day (see Figure 8). This technique for computing the DTP ensures that the agent never sells more than

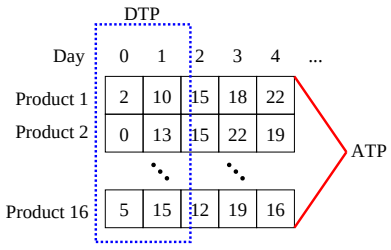


Figure 8: The strategy module instructs the agent to promise the first two days of available PCs.

its available capacity and left over inventory for a single day. It also guarantees that the products being sold are as flexible as possible with respect to satisfying customer requests. Since all of the DTP is available to ship on the very next day, it allows the

[1]In practice we found that completely removing unprofitable products from the product mixture provided too much of an advantage to competing agents. This motivated our decision to allow the agent to occasionally sell a small percentage of products at a loss.

[2]During the tail of a game the agent revises this heuristic to ensure it completes the game with as little inventory as possible

bidding module to safely bid on any customer request without worrying about late penalties.

3.4 Scheduling Module

The scheduling module continuously maintains a production schedule over a horizon of several days. This schedule reflects current contracts, forecast contracts and projected component inventory levels. It helps drive other planning decisions including which customer RFQs to bid on and which RFQs to send to suppliers.

More specifically, the scheduling module makes a tentative production schedule for D^{S} days into the future. The primary inputs and outputs of the module are summarized formally in Figure 9. The inputs include a set of orders, $\hat{O}$, from the strategy module and the projected component inventory, I, for the remainder of the game. The orders in $\hat{O}$ represent the target demand of the agent and include both actual and forecast future orders.

Scheduling Inputs and Constants:

- $\boldsymbol{\hat{O}}$, a set of orders representing target demand from *strategy module*, each order $\boldsymbol{i}$ includes the following information:
  - $\boldsymbol{d_i}$, the due date of the $\boldsymbol{i}$'th order.
  - $\boldsymbol{p_i}$, the daily late penalty associated with the $\boldsymbol{i}$'th order (the contractual penalty for actual orders and a small constant for forecast orders).
  - $\boldsymbol{s_i}$, the SKU for the product type associated with the $\boldsymbol{i}$'th order.
  - $\boldsymbol{q_i}$, the quantity of products associated with the $\boldsymbol{i}$'th order.
  - $\boldsymbol{b_i} \in \boldsymbol{\{0, 1\}}$, a flag indicating whether or not the $\boldsymbol{i}$'th order is an actual order or a forecast order.
- $\boldsymbol{I}$, the projected component inventory for all remaining days. $\boldsymbol{I_{dk}}$ is the projected inventory level of component $\boldsymbol{k}$ on day $\boldsymbol{d}$.
- $\boldsymbol{D^{\mathsf{S}}}$, number of days in the schedule (scheduling window)
- $\boldsymbol{\alpha}$, the slack weighting parameter for ATC priorities.

Scheduling Outputs:

- $\boldsymbol{S}$, a production schedule for $\boldsymbol{D^{\mathsf{S}}}$ days, $\boldsymbol{S_d}$ is the set of orders scheduled for production on day $\boldsymbol{d}$.

Figure 9: Scheduling module inputs and outputs.

3.4.1 Production Scheduling

The scheduling module uses a heuristic to sort orders according to "slack" (time before due date) and penalty, and a greedy dispatch technique to fill the production schedule.

The dispatch technique (presented in pseudo-code in Figure 11) proceeds as follows. It iterates through each day in the scheduling window and computes the priority of each unscheduled order during each iteration. The priorities are computed according to the Vepsalainen's apparent tardiness cost (ATC) dispatch rule [13]. The ATC priority favors orders with large penalties and little time to complete, since these are likely to be orders that require the most immediate attention. The slack weighting parameter, α, dictates the exact trade off in priority between slack and tardiness. An example of the ATC priorities for different orders with $\alpha = 1$ (the value used in our agent) is graphed in Figure 10 as the scheduling day increases.

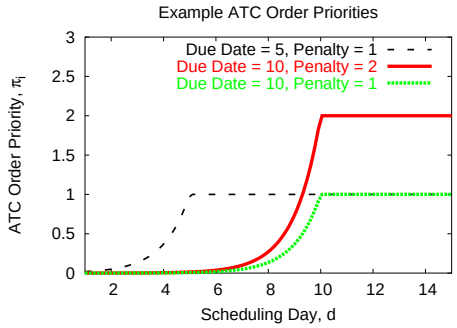


Figure 10: Example of ATC priorities for different orders on different scheduling days ($\alpha = 1$). Notice that ATC prioritizes by penalty once an order's due date is reached, and by slack for orders with the same penalty.

While building a particular day's segment of the production schedule, the dispatch scheduler attempts to add each order to the production schedule according to its priority (orders with larger priorities are considered first). When an order is considered, the scheduler determines whether or not there are enough available resources (i.e. capacity and components) on the current day of the iteration through the scheduling window. If there are enough unclaimed resources, the order is scheduled for production and the necessary components and production cycles are allocated. If there are not enough resources available on the day in question, the order is removed from the queue and is considered again the next day. The scheduler proceeds to the following day when all orders have been considered and either scheduled or delayed. This entire process repeats until the scheduling window is exceeded.

3.5 Bidding Module

The bidding module is responsible for responding to a subset of the current customer RFQs. Its goal is to sell the resources specified in the DTP at the highest prices possible. The inputs and outputs of the bidding module are formally summarized in Figure 12.

procedure `dispatch`$(O, I, D^{\mathtt{S}}, \alpha)$
  $O' \leftarrow O$
  for $d = 0$ **to** $D^{\mathtt{S}}$ **do**
    $S_d \leftarrow \emptyset$
    sort O' according to apparent tardiness priority. The priority of the i'th order, π_i, is calculated as:

$$\pi_i = p_i \left[\exp\left(-\frac{1}{\alpha} \max(0, d_i - d) \right) \right]$$

    while $O' \neq \emptyset$ **do**
      pop the highest priority order o_i from O'
      if `has_inventory`(o_i, I_d) **and** `has_capacity`(o_i, S_d) **then**
        `schedule_production`(o_i, I_d, S_d)
      end if
    end while
    $O' \leftarrow O \setminus S$
  end for

Figure 11: A summary of the Apparent Tardiness Cost (ATC) dispatch scheduling technique used by the scheduling module.

Bidding Inputs and Constants:

- $\boldsymbol{R}$, the set of current RFQs.
- $\boldsymbol{G : r, p \rightarrow (0, 1)}$, a cumulative density function that takes an RFQ, $\boldsymbol{r}$, and a unit price, $\boldsymbol{p}$, and provides the probability that the winning price for $\boldsymbol{r}$ will be greater than $\boldsymbol{p}$.
- $\boldsymbol{G^{-1} : r, (0, 1) \rightarrow p}$, the inverse of $\boldsymbol{G}$, takes an RFQ and a probability and returns the corresponding price.
- $\hat{\boldsymbol{S}}$, the DTP from *strategy module*.

Bidding Outputs:

- $\boldsymbol{F}^{\mathtt{C}}$, a set of offers for customers. Each offer corresponds to an RFQ in $\boldsymbol{R}$, and includes a unit price.

Figure 12: Bidding module inputs and outputs (G and G^{-1} are maintained internally).

3.5.1 Bidding for Customer Orders

The bidding module maintains a probability distribution, G, for each product type that specifies the likelihood of winning a particular RFQ for that product type at any price. The distributions are learned off line using RFQs from previously played games to

build a modified regression tree called a *distribution tree*. The tree provides a Normal distribution over winning bid prices for each RFQ based on its features (such as due date, penalty, and reserve price) and the features of the market at the time it was sent (such as the previous day's high and low winning bid prices). The distributions are used to select offers that maximize expected revenue, subject to the restriction that the expected amount of products sold is less than or equal to the DTP.

The bidding module addresses the bidding problem separately for each product type, and reduces its task to a continuous knapsack problem (CKP) instance. The CKP is a variant of the knapsack problem classically studied in operations research. The CKP asks: given a knapsack with a weight limit and a set of weighted items – each with its value defined as a function of the fraction possessed – fill the knapsack with fractions of those items to maximize its value. In the CKP instance reduced from the bidding problem for a specific product type, the items are RFQs for that product with weights equal to their quantities. The weight limit in the CKP is the quantity of the product appearing in the DTP. The value of a fraction, x, of an RFQ, r, is the expected unit revenue that yields a winning probability of x. The expected unit revenue is defined as the probability with which the customer is expected to accept the offer (as specified by the bidding module's probability distribution) times the offer price, $G^{-1}(r, x) \times x$.

CMieux uses a binary search algorithm to solve the CKP instance for each product that is guaranteed to provide a solution within ϵ of optimal expected revenue. The search algorithm operates on the derivatives of the expected unit revenue functions. It finds the largest derivative value corresponding to a solution that does not violate the weight limit of the knapsack. Since the distributions are Normal the expected unit revenue functions are strictly concave, and the solution corresponding to the largest feasible derivative value is optimal. For full descriptions of the reduction to a CKP, the ϵ-optimal algorithm, and the probability distributions used in CMieux the reader is directed to [2].

3.6 Procurement Module

The procurement module handles all aspects of requesting and purchasing components. It is designed to rapidly adapt to changing market conditions. Each day, it considers sending requests with widely varying quantities and lead times in an effort to exploit gaps in current supplier contracts. By finding such gaps, or slow days for the suppliers, the agent ensures that its procurement prices tend to fall below its competitors. The flexibility gained by considering so many different procurement strategies in this way sets CMieux apart from most existing supply chain practices, as well as those of other agents designed for TAC SCM.

Each day, the procurement module performs two tasks: i.) it attempts to identify a particularly promising subset of current supplier offers, and ii.) it constructs a combination of RFQs to be sent to suppliers that balances the agent's component

needs with identified gaps in current supplier contracts.

The procurement module takes as input the set of recent supplier offers, the projected inventory, the target demand and the forecast pricing functions (see Figure 13).

Procurement Inputs and Constants:

- $\boldsymbol{F^{\mathsf{S}}}$, the set of offers from suppliers.
- $\boldsymbol{I}$, the projected component inventory for all remaining days. $\boldsymbol{I_{kd}}$ is the projected inventory level of component $\boldsymbol{k}$ on day $\boldsymbol{d}$.
- $\boldsymbol{\hat{O}}$, target demand from *strategy module*.
- $\boldsymbol{f^{\mathsf{C}}}$, customer price function from *forecast module*.
- $\boldsymbol{f^{\mathsf{S}}}$, supplier price function from *forecast module*.
- $\langle \boldsymbol{D^-, D^+, D^{\mathsf{G}}} \rangle$, the earliest, and latest days to consider requesting for, and the granularity to discretize search.
- $\boldsymbol{K^{\mathsf{S}}}$, the number of requests allowed per supplier.

Procurement Outputs:

- $\boldsymbol{\hat{F}}$, the set of supplier offers to accept.
- $\boldsymbol{Z}$, the procurement requests for each supplier. $\boldsymbol{Z_{lk}} = \{\boldsymbol{z_1, \ldots, z_{K^{\mathsf{S}}}}\}$ is the set of requests for supplier $\boldsymbol{l}$ and component $\boldsymbol{k}$. Each request includes the following information:
  - $\boldsymbol{q_i}$, the quantity of the request.
  - $\boldsymbol{d_i}$, the due date of the request.
  - $\boldsymbol{r_i}$, the reserve price of the request.

Figure 13: Procurement module inputs and outputs.

3.6.1 Accepting Supplier Offers

The module accepts supplier offers using a rule-based decision process. The agent begins by selecting offers that are satisfactory based on price, quantity and due date using historical data. In an effort to keep the agent's reputation as high as possible[3], the agent first accepts offers that satisfy the quantity and due date requirements of the corresponding RFQ ("full offers"). Next, if still needed, satisfactory offers with relatively large quantities ("partial offers"), or early due dates ("earliest complete offers") are also accepted.

[3]Maintaining a perfect reputation was identified as an important strategic goal for the 2005 competition.

3.6.2 Sending Supplier Requests

Since offer prices, due dates and quantities are dictated by the specific requests they are offers for, the primary responsibility of the procurement module is requisitioning. The requisitioning procedure used in CMieux attempts to request some of the components it needs (that it has not already purchased) to maintain its target production levels, each day. Its main goal is to ensure that the prices offered in response to the requests are as low as possible. The requisitioning procedure chooses between many different lead times and quantities, based on the forecast supplier market landscape.

---

procedure `request`$(I, \hat{O}, f^{\mathtt{C}}, f^{\mathtt{S}}, \langle D^-, D^+, D^{\mathtt{G}}\rangle, K^{\mathtt{S}})$

  let $\hat{I}$ be a fraction of the difference between inventory maintaining production levels of $\hat{O}$, and inventory available in I.
  for each component, k **do**
    $d^* \leftarrow \{\}$
    $u^* \leftarrow 0^{K^{\mathtt{S}}-1}$
    for each set of $K^{\mathtt{S}}+1$ dates, $\{d_1, \ldots, d_{K^{\mathtt{S}}+1}\}$, between D^- and D^+, discretized by $D^{\mathtt{G}}$ **do**
      $u \leftarrow$ `approx_utility`$(\{d_1, \ldots, d_{K^{\mathtt{S}}+1}\}, \hat{I}, k, f^{\mathtt{C}}, f^{\mathtt{S}})$
      if `sum`$(u) >$ `sum`(u^*) **then**
        $d^* \leftarrow \{d_1, \ldots, d_{K^{\mathtt{S}}+1}\}$
        $u^* \leftarrow u$
      end if
    end for
    for $i = 1$ **to** $K^{\mathtt{S}}$ **do**
      let l be the supplier of k with the lowest price on day d_i
      $q_i \leftarrow \sum_{d=d_i}^{d_{i+1}} \hat{I}_{kd}$
      $z_i \leftarrow \langle d_i, q_i, u_i^*/q_i\rangle$
      $Z_{lk} \leftarrow Z_{lk} \cup \{z_i\}$
    end for
  end for
  return Z

procedure `approx_utility`$(\{d_1, \ldots, d_{K^{\mathtt{S}}+1}\}, \hat{I}, k, f^{\mathtt{C}}, f^{\mathtt{S}})$

  $u \leftarrow 0^{K^{\mathtt{S}}}$
  let J be the set of products that component k is used in
  for $j \in J$ **do**
    let β_j and β_k be base prices of product j and component k from the game specification
    for $i = 2$ **to** $K^{\mathtt{S}}+1$ **do**
      $d \leftarrow d_{i-1}$
      while $d < d_i$ **do**
        $\hat{u} \leftarrow \left(\frac{\beta_k}{\beta_j} f^{\mathtt{C}}(j,d)\right) - f^{\mathtt{S}}(k,d)$ */* unit profit from j */*
        $u_{i-1} \leftarrow u_{i-1} + \hat{I}_{kd}\frac{\hat{u}}{|J|}$
        $d \leftarrow d+1$
      end while
    end for
  end for
  return u

Figure 14: Pseudo-code for the requisitioning procedure used by the procurement module.

In order to determine what requests to send to suppliers, the procurement module computes, $\hat{I}$, the difference between the inventory required to maintain production levels specified by the target demand, and the projected inventory for the remainder of the game (i.e. the components that it needs but has not yet purchased). However, our agent does not *need* to procure this entire difference each day. The components are not needed immediately, thus it can divide the purchasing of components in $\hat{I}$ across several days. To that end, the quantities specified in $\hat{I}$ beyond $D^{\mathtt{S}}$ days in the future (the scheduling window) are linearly depleted. This enables the agent to aggressively procure components within its scheduling window, so that late penalties are not incurred on existing contracts. In addition, it allows the agent to buy some of

the components it needs well in advance, when they are likely to be cheapest.

The process of computing what specific requests to send to suppliers is then decomposed by component type. For each component type, the procurement module generates several sets of $K^{\mathtt{S}}$(the limit on RFQs sent each day) lead times and searches for the best set.

More specifically, the module uses brute force to enumerate all ways to choose a tuple of $K^{\mathtt{S}}+1$ dates between D^{-} and D^{+}, discretized by $D^{\mathtt{G}}$. The first $K^{\mathtt{S}}$ dates in the tuple specify the RFQ lead times. Each of the RFQs requests the components needed by the agent between its lead time, and the next date in the tuple (this is why there must be one more than $K^{\mathtt{S}}$ dates in the tuple).

For example, if $D^{-}=5$, $D^{+}=20$, $D^{\mathtt{G}}=5$, and $K^{\mathtt{S}}=2$, then the procurement module would consider the following tuples of dates $\langle 5,10,15\rangle$, $\langle 5,10,20\rangle$, $\langle 5,15,20\rangle$ and $\langle 10,15,20\rangle$. Each RFQ is used to procure the parts specified in $\hat{I}$ between its due date and the subsequent due date in the tuple. Consider the tuple $\langle 5,10,15\rangle$, which involves two RFQs. The first has a lead time of 5 and a quantity equal to the sum of the parts specified in $\hat{I}$ between 5 and 10 days in the future. The second RFQ has a lead time of 10 and a quantity equal to the sum of the parts specified in $\hat{I}$ between 10 and 15 days in the future.

The utility of the RFQs generated by each tuple is computed by approximating the sum of the utility of the components they request and subtracting their forecast prices. In order to approximate the utility of a component, k, we compute the ratio between its base price, β_k, and the base price of each product, j, it is included in, β_j. The base price ratio, $\frac{\beta_k}{\beta_j}$, provides an approximation of the fraction of product j's revenue attributable to component k. Thus, the utility of a component is approximated as the average selling price, weighted by the base price ratio, of each of the products it is included in. The cost of each RFQ is given by the supplier pricing forecast function, $f^{\mathtt{S}}$.

The RFQs with the greatest utility for each component are sent to the appropriate suppliers. The reserve price of each RFQ is set to be the average utility of the components it includes. Figure 14 provides pseudo-code outlining this requisitioning process.

In addition, we augmented this flexible and dynamic requisitioning procedure with the following improvements.

Increased bottleneck component utility: The utility of a component can be further refined by taking into account situations where the agent has all but one of the components required to assemble a particular type of PC (making it a *bottleneck component*). This situation can become more severe toward the end of the game as the agent faces the prospect of being stuck with mis-matched components. For example, our agent can have hundreds of motherboards, memory, and CPUs to make a specific product, and be missing only the hard drives. To address this issue, the procurement module artificially inflates the base price ratio of bottleneck components (such as the

hard drives in the example), and decreases the base price ratio of all other components[4]. The inflation factor is increased as the agent nears the end of the game.

Dynamically refined search granularity: An additional observation was that, for short lead times, supplier pricing was often drastically different even between lead times as little as 1 day apart. In practice, our agent used a search granularity of about $D^{\mathtt{G}} = 5$ days, which caused it to frequently miss promising early lead times. To address this issue, after finding the most promising lead time tuple at a particular granularity our agent generated new sets of tuples using finer and finer granularity around previously identified promising tuples. This helped the agent more effectively cover the space without drastically effecting its runtime.

Parallelization across components: The requisitioning technique described above decomposes its search through lead time tuples by component type. In order to give our agent the ability to perform a finer search we parallelized the requisitioning process across multiple CPUs, each of which was responsible for considering a subset of components. Due to the natural decomposition of our problem formulation, the parallel processes had no need to interact (other than to aggregate their final solutions) making this a relatively simple refinement to implement.

4 Empirical Evaluation

To validate the adaptive and dynamic techniques utilized in our agent we present three sets of empirical results. The first are taken from the 2005 TAC SCM seeding rounds[5], and summarize CMieux's bidding and procurement performance over 200 games involving agents entered by 25 different teams. A second set of results is also presented showing that CMieux significantly outperforms five other publically available versions of top agent binaries over several repeated games. A third set of results examines the accuracy of our forecast module when it comes to predicting supplier prices and customer demand. This includes looking at how well a top performing agent such as CMieux is able to predict supplier prices using the limited number of RFQs allowed by the game specifications as well as how accuracy is affected by the forecast horizon in different game phases.

4.1 Procurement and Bidding Performance

Evaluating the performance of a supply chain trading agent is challenging even in the context of TAC SCM. The competition effectively consists of two different tournaments:

[4]This can be thought of as a coarse approximation of a component's marginal utility

[5]Competition data is available at `sics.se/tac/scmserver`

1. a seeding round tournament featuring a large number of agents competing over a period of 2 weeks in about 400 games
2. a set of final rounds, where small sets of agents are pitted against one another in a relatively limited number of games (ranging from 8 to 16 per round).

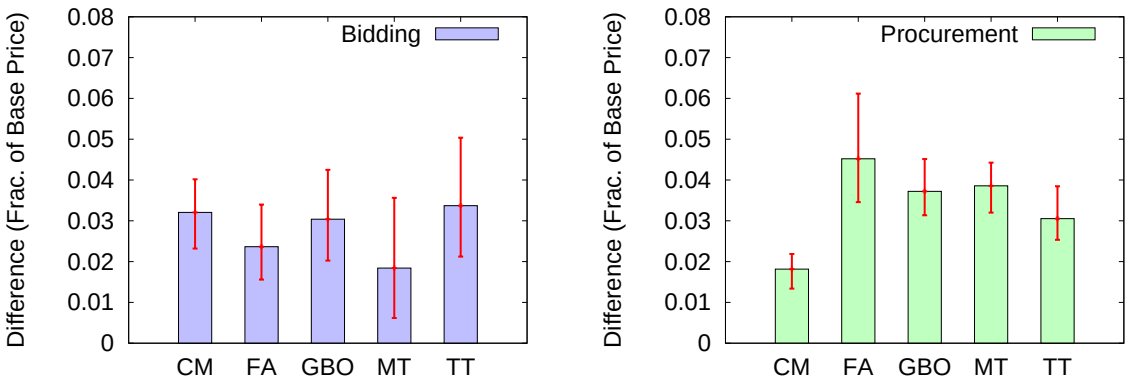
Not only do they feature a small number of games but, because they repeatedly pit the same agents against one another, final rounds also potentially reward destructive strategies that may not be representative of real world competition (e.g. an agent disrupting competitors at the expense of its own bottom line). In 2005, CMieux finished 4th in the seeding rounds and reached the tournament's semi-finals. While encouraging, these results only provide a partial picture of CMieux's performance. In this section, we provide a more in-depth analysis of our agent during what can be viewed as the most competitive phase of the competition, namely the 200 games played by the 25 agents participating in the second week of the seeding rounds. All agents at that stage have already been fine tuned over the course of about 600 games (two weeks of qualifying rounds, and one week of seeding).

Specifically, our results provide a statistical comparison between the performance of the agents with the top 5 mean overall scores during the second week of the seeding rounds, namely CMieux (abbreviated CM), FreeAgent (FA), GoBlueOval (GBO), MinnieTAC (MT) and TacTex-05 (TT).

Performance was measured so as to identify those agents that were able to extract the highest sale price and lowest purchasing price in each game they played. Specifically, for each of the top 5 agents in each game it played in we computed how far it was from paying the least for its components and obtaining the most for its end products among the agents playing in that particular game. This was measured as the relative difference from the best average procurement price[6] and the best average selling price. For each of the top 5 agents we report the mean (with 95% confidence intervals) of these values across all of the games they each played in (see Figure 15).

The bidding results for all 5 agents are relatively similar. As can be seen, each of the top 5 agents is on average within about 3% of the base price from being the best in its games. However, while MinnieTAC (MT) was the closest to the best agent in its games, with an average difference of about 2% of the base price, there is no statistically significant difference between any of the top 5 agents (as evidenced by their overlapping confidence intervals). On the other hand, the procurement results show that our agent, CMieux (CM), is significantly closer to being the best than all 4 of the other top 5 agents. These results seem to validate CMieux's approach to tightly coordinating its bidding, planning and procurement operations. They also suggest that the agent's approach to optimizing the RFQs it sends to suppliers (*requisition process*) was significantly more effective than the procurement strategies implemented by its competitors.

[6] All prices are considered as fractions of the corresponding product or component's base price.



(a) Bidding Analysis

(b) Procurement Analysis

Figure 15: The mean (with 95% confidence intervals) difference between each of the top 5 agents' average game unit price and the best unit price in the game, during the second week of the 2005 TAC SCM seeding rounds.

4.2 Results Against Public Binaries

In order to further validate the overall performance of our agent in a statistically significant fashion we simulated 40 games against the same mix of the best agents drawn from a pool of publically available binaries[7]. The binaries used in our experiments included TacTex "TT" (the 2005 champion agent), Mertacor "MC", MinneTac "MT" (both finalists in the 2005 competition), Phantagent "PH" and GoBlueOval "GBO" (both in the top 10 agents from the seeding rounds).

To achieve statistically significant results with relatively few samples compared to the parameter space of the game we used the following analysis technique. First, for each game, we shifted all scores by adding a constant value that left the worst performing agent with a score of zero. Next, we computed the sum of all the shifted scores in each game and the fraction of the sum represented by each agent's shifted score. This provided us with a more stable normailzed value representing the fraction of total profit accumulated by each agent in each game that is comparable accross games. We computed the mean and 95% confidence interval of this value for each agent in our experiments accross the 40 games. The results in Figure 16 show that CMieux makes significantly more average relative profit per game than any of the other five agents.

4.3 Forecast Accuracy

In this section, we report additional results investigating how well a top performing agent such as CMieux can predict supplier prices and customer demand, given the

[7]Agents are available at `sics.se/tac/showagents.php`

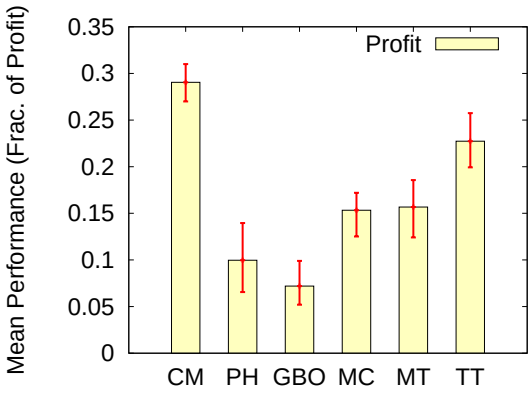
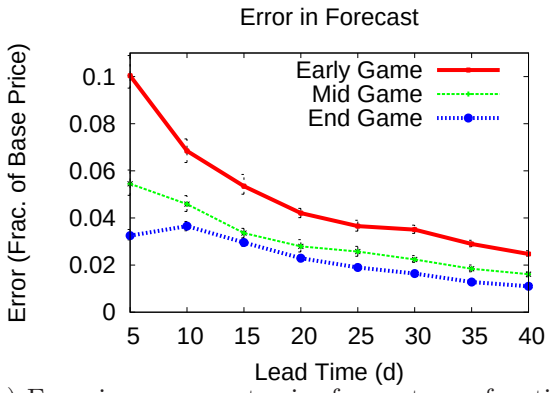


Figure 16: Mean (with 95% confidence interval) fraction of game profit over 40 games.

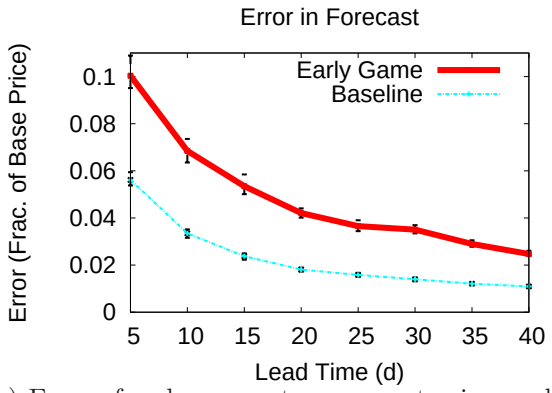
high degree of stochasticity associated with these markets. This includes looking at how well the agent is able to predict supplier prices, given the limited number of RFQs allowed by the game specifications as well as how accuracy is affected by the forecast horizon in different game phases.

Results reported below were obtained by pitting CMieux against 5 publicly available agents, namely TacTex-05, Phantagent, Mertacor, CrocodileSCM, GoBlueOval. All 5 of these agents were among those qualifying for the final rounds of the 2005 competition.

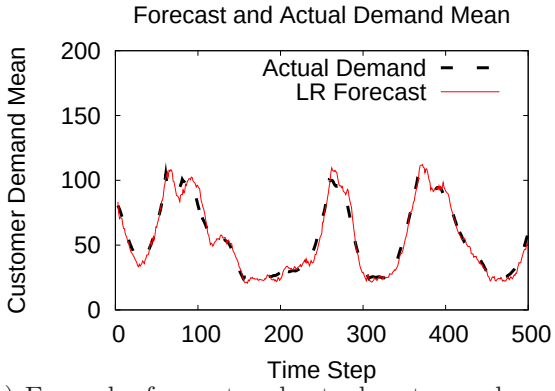
Figure 17(a) plots CMieux's error in predicting supplier prices during the early, mid, and late segments of the games as a function of lead time. For comparison purposes, the results in Figure 17(b) show the error of the early segment predictions of both CMieux's forecasting technique and a baseline variant that relaxes the game's restriction on the number of probes that can be sent by an agent - results for the mid and end game phases are similar. The plots provide the mean error (with 95% confidence intervals) as a fraction of component base price of all component price forecasts made during those segments. The prediction error is measured for each possible lead time between 5 and 40 days at 5 day intervals. The results show that the early segment of the game is the most difficult segment for CMieux's forecast module to accurately predict supplier pricing, for all lead times. Even the baseline variant with unlimited number of probes is unable to achieve high accuracy during this segment. This is not surprising considering that agents are not likely to have settled into an equilibrium yet and are reacting to start up effects (effects introduced by the fact that all agents begin the game with no components). Additionally, we can see that both our technique and the baseline variant have more error when predicting prices on orders with shorter lead times during all game segments. The difficulty of predicting prices with short lead times is exaggerated during the early segment due to the previously mentioned instability. Despite the instability we see that the greatest error in the supplier price forecasting is only about 10% of the base price, resulting from the prediction of short lead time prices during



(a) Error in component price forecast as a function of lead time in different game segments.



(b) Error of early segment component price prediction with baseline that used unlimited probes.



(c) Example: forecast and actual customer demand mean.

Figure 17: Empirical evaluation of the forecast module.

the early segment of the game. Forecasting of orders with longer lead times, and short lead times later in the game, is generally accurate within 95% of the base price.

Figure 17(c) shows an example of a changing customer demand mean, and the predictions of the forecast module based on observations of draws from a Poisson distribution with that mean. The results on this particular example show that the forecast module is relatively effective at predicting the mean and following its trend. To gain a better understanding of the effectiveness of our forecast module for predicting customer demand, we compared it to a naive technique that assumed the current mean was the most recently observed sample from the Poisson distribution. A more detailed analysis of our technique and the naive technique across multiple games revealed that on average our forecast module was within 7% (plus or minus 1% with 95% confidence) and the naive technique was within 12% (plus or minus 1% with 95% confidence) of

properly predicting the mean of each product type's demand distribution. While this result does not show a largely significant difference between our forecast module and the naive technique, our forecast module was much better at predicting the trends of the means. Our technique was within 2% (plus or minus less than 1% with 95% confidence) of predicting the trends on average, whereas the naive technique had an average of about 18% (plus or minus 2% with 95% confidence) error when predicting the trends.

5 Conclusions

This paper presented a high level view of the interactions between the different modules composing CMieux, Carnegie Mellon University's 2005 TAC SCM entry, as well as detailed descriptions of its decision making processes. CMieux's architecture departs markedly from traditional Enterprise Resource Planning architectures and commercially-available supply chain management solutions through its emphasis on tight coordination between supply chain bidding, procurement and planning.

CMieux finished 4th in the 2005 seeding rounds of the TAC SCM tournament and reached the competition's semifinals. In this paper, we presented a more in-depth analysis of the agent's performance based on 200 games involving agents entered by 25 different teams during what can be seen as the most competitive phase of the 2005 tournament and 40 games against the same 5 top performing public binaries. The results show that our agent performed on par with the best in its bidding while significantly outperforming these agents in terms of procurement during the 2005 seeding rounds, and made more relative profit per game on average than any of the other public binaries tested. These results seem to validate CMieux's approach to tightly coordinating its bidding, planning and procurement operations. They also suggest that the agent's approach to optimizing the RFQs it sends to suppliers (*requisition process*) was significantly more effective than the procurement strategies implemented by its competitors.

6 Acknowledgments

The research reported in this paper has been funded by the National Science Foundation under ITR Grant 0205435.

References

[1] R. Arunachalam and N. Sadeh. The supply chain trading agent competition. *Electronic Commerce Research Applications*, 4(1), 2005.

[2] M. Benisch, J. Andrews, and N. Sadeh. Pricing for customers with probabilistic valuations as a continuous knapsack problem. In *Proceedings of ICEC'06*, 2006.

[3] M. Benisch, A. Greenwald, I. Grypari, R. Lederman, V. Naroditsky, and M. Tschantz. Botticelli: A supply chain management agent. In *Third International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2004.

[4] S. Chopra and P. Meindl. *Supply Chain Management*. Pearson Prentice Hall, New Jersey, 2004.

[5] J. Collins, R. Arunachalam, N. Sadeh, J. Eriksson, N. Finne, and S. Janson. The supply chain management game for 2005 trading agent competition, 2005.

[6] J. Estelle, Y. Vorobeychik, M. P. W. S. Singh, C. Kiekintveld, and V. Soni. Strategic interactions in a supply chain game, 2003.

[7] M. He, A. Rogers, X. Luo, and N. R. Jennings. Designing a successful trading agent for supply chain management. In *Proceedings of AAMAS'06*, 2006.

[8] C. Kiekintveld, M. P. Wellman, S. Singh, J. Estelle, Y. Vorobeychik, V. Soni, and M. Rudary. Distributed feedback control for decision making on supply chains. In *Fourteenth International Conference on Automated Planning and Scheduling*, 2004.

[9] D. Pardoe and P. Stone. Bidding for customer orders in tac scm. In *AAMAS-04 Workshop on Agent-Mediated Electronic Commerce*, 2004.

[10] D. Pardoe and P. Stone. Predictive planning for supply chain management. In *Proceedings of Automated Planning and Scheduling'06*, 2006.

[11] D. P. Philipp W. Keller, Felix-Olivier Duguay. Redagent-2003: An autonomous, market-based supply-chain management agent. In *Third International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2004.

[12] N. Sadeh, D. Hildum, D. Kjenstad, and A. Tseng. Mascot: an agent-based architecture for coordinated mixed-initiative supply chain planning and scheduling. In *Proceedings of Workshop on Agent-Based Decision Support in Managing the Internet-Enabled Supply-Chain at Agents'99*, 1999.

[13] A. Vepsalainen and T. Morton. Priority rules for job shops with weighted tardiness costs. *Management Science*, 33(8):1035–1047, 1987.