Botticelli: A Supply Chain Management Agent

M. Benisch, A. Greenwald, I. Grypari, R. Lederman, V. Naroditskiy, and M. Tschantz
Department of Computer Science, Brown University, Box 1910, Providence, RI 02912
{mbenisch,amygreen,vnarodit,mtschant}@cs.brown.edu
{Ioanna_Grypari}@brown.edu, {rogerl}@alumni.brown.edu

Abstract

The paper describes the architecture of Brown University's agent, BOTTICELLI*, a finalist in the 2003 Trading Agent Competition in Supply Chain Management (TAC SCM). In TAC SCM, a simulated computer manufacturing scenario,* BOTTICELLI *competes with other agents to win customer orders and negotiates with suppliers to procure the components necessary to complete its orders.*

In this paper, two subproblems that dictate BOTTICELLI*'s behavior are formalized: bidding and scheduling. Mathematical programming approaches are applied in attempt to solve these problems optimally. In addition, greedy methods that yield useful approximations are described. Test results compare the performance and computational efficiency of these alternative techniques.*

1. Introduction

A supply chain is a network of autonomous entities, or agents, engaged in *procurement* of raw materials, *manufacturing*—converting raw materials into finished products—and *distribution* of finished products. The Trading Agent Competition in Supply Chain Management (TAC SCM) is a simulated computer manufacturing scenario in which software agents tackle complex problems in supply chain management. This paper describes the structure of Brown University's agent BOTTICELLI, a finalist in TAC SCM 2003.

TAC SCM agents face uncertainty about the future, but they must make decisions before the uncertainty is resolved: e.g., agents must procure raw materials and manufacture finished products before customer orders arrive. BOTTICELLI handles the uncertainty in manufacturing and distribution using stochastic programming techniques (see Benisch *et al.* [1]). Here, we focus on our approach to the *bidding* problem: find an optimal set of bids to place on customer RFQs, balancing the tradeoff between maximizing profits—by placing high bids—and maximizing the likelihood of winning multiple customer orders—by placing low bids.

This paper is organized as follows. In Section 2, we give an overview of TAC SCM. Next we describe the architecture of our agent, BOTTICELLI. This architecture emphasizes three problems—bidding, production scheduling, and delivery scheduling. Section 4 describes a heuristic approach to these problems: greedy scheduling and bidding via hill-climbing. Section 5 details solutions that approximate optimal stochastic programming solutions. Section 6 presents experimental results.

2. TAC SCM

In TAC SCM, six software agents compete in a simulated sector of a market economy, specifically the personal computer (PC) manufacturing sector. Each agent can manufacture 16 different types of computers, characterized by different *stock keeping units* (SKUs). Building each SKU requires a different combination of components, of which there are 10 different types. These components are acquired from a common pool of suppliers at costs that vary as a function of demand. After assembly, each agent can sell its PCs to a common pool of customers by underbidding the other agents. The agents are ranked based on their profits over 220 days, each of which lasts 15 seconds.

Each day in the TAC SCM simulation customers send a set of *requests for quotes* (RFQs) to the agents. Each RFQ contains a SKU, a quantity, a due date, a penalty rate, and a reserve price—the highest price the customer is willing to pay. Each agent sends an *offer* to each customer for each RFQ, representing the price at which it is willing to satisfy that RFQ. After the customer receives all its offers, it selects the agent with the lowest-priced offer and awards that agent with an *order*. Either: the winning agent delivers the entire order by its due date, in which case it is paid in full; it delivers the entire order within five days of its due date, in which case it is paid the amount of its offer less a penalty based on the number of late days; or, it cannot deliver the entire order within five days of its due date, in which case the order is canceled, no revenues are accrued, and the maximum penalty is incurred.

Meanwhile, the agents themselves are sending RFQs to suppliers, requesting a specific quantity of a component to arrive on a particular day. The suppliers respond to these requests the next day with either partial or full offers, indicating the price per unit at which the RFQ can be satisfied. If an agent receives a partial offer, the supplier cannot deliver the requested quantity of the component on the day on which it was requested, but it can deliver a lesser quantity on that day. Full offers either have a delivery date on the day requested, or a delivery date later than the one requested, in which case they are often accompanied by partial offers. Among these offers, an agent can choose to accept at most one, in which case agent and supplier enter into a contract agreeing that the agent will be charged for the components upon their arrival.

At the end of each day, each agent converts the components it acquired from suppliers into SKUs according to a production schedule it generates for its factory. It also reports a delivery schedule assigning the SKUs in its inventory to customer orders.

3. Agent Architecture

Each simulated TAC day represents a decision cycle for an agent, during which time the agents must solve four problems: procurement, bidding, production scheduling, and delivery scheduling. The *procurement* problem involves communicating with suppliers via RFQs, and selecting supplier offers to accept among those which are received in response to these RFQs. The *bidding* problem is to decide how to assign offer prices to each customer RFQ. The *production scheduling* problem is to decide how many of each SKU to assemble each day. The *delivery scheduling* problem is to decide which orders to ship to which customers, using product inventory. The objective in all of these problems is to maximize *expected* profits, given some probabilistic model that captures the uncertainty in the game. A high-level description of the TAC SCM decision problem is presented in Figure 1.

An artifact in the design of TAC SCM 2003 (namely, negligible component prices on day 1), resulted in us placing little emphasis on *procurement*. Rather, we focused on the development of solutions to the *bidding*, *scheduling*, and *delivery* problems. High-level descriptions of the three problems are given in Figures 4, 6, and 7. These three problems are highly interconnected. Indeed, an optimal solution to the production scheduling problem yields an optimal solution to the delivery scheduling problem, since ultimately revenues depend on which orders are successfully delivered to their respective customers. Moreover, an optimal solution to the bidding problem yields an optimal solution to both scheduling problems, since bidding decisions depend on manufacturing and distribution constraints: too few winning bids lead to missed revenue opportunities; too many winning bids lead to late penalties.

TAC SCM Decision Problem
Objective:
  Maximize Expected Profits
Inputs:
  Product Pricing Model
  Component Cost Model
  Set of Supplier Offers
  Set of Customer RFQs
  Set of Customer Orders
  Procurement Schedule
  Component Inventory
  Product Inventory
Outputs:
  Procurement Schedule: set of Supplier RFQs and Orders
  Bidding Policy: map from Customer RFQs to Prices
  Production Schedule: map from Cycles to SKUs
  Delivery Schedule: map from SKUs to Customer Orders

Figure 1. TAC SCM Decision Problem

The architecture of BOTTICELLI was designed with these relationships in mind, and thus the bidding module envelops the scheduling module, which in turn envelops the delivery module as shown in Figure 2. Once a bidding policy is determined by the bidding module, the scheduling module finds a production schedule, and the delivery module ships products to customers.

The flow of information through the agent is as follows: Each day the modeling module receives information about other agents' actions on the previous day as well as information about the offers the bidding module submitted and the orders that resulted from those offers. The modeling module uses this information to update its models and passes an updated model to the bidding module. The bidding module uses the new model to produce an offer for each of the day's RFQs. The offer prices are determined with the aid of the scheduling module. When invoked, the scheduling module learns from the procurement module the quantity of each component that is expected to be in inventory on any particular day. It then determines how to allocate machine cycles to make products for existing orders and likely future orders. The scheduling module relies on the delivery module to determine how to allocate product inventory to existing orders and likely

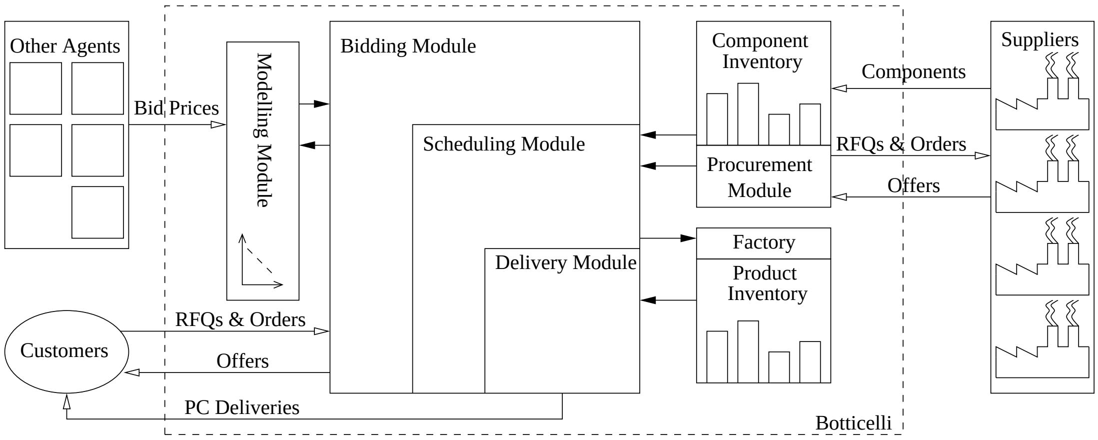

Figure 2. Botticelli: A Modular Design

future orders. After the bidding, scheduling, and delivery modules finalize their decisions, the procurement module sends to suppliers RFQs for additional components and orders based on the current offers.

4. Bidding: A Hill-Climbing Approach

In the preliminary rounds, BOTTICELLI relied on a hill-climbing bidder, which successively adjusts bid prices according to the results of a scheduler. At a high-level, the bidder is initialized with some set of bid prices; given these prices, a production and delivery schedule is computed; and, based on the results of the scheduler, bid prices are tweaked. The goal of this hill-climbing algorithm is to fill our production schedule, which we assume is positively correlated with maximizing expected profits. TacTex utilizes a similar solution to the bidding problem [5].

In a preprocessing step, we schedule only orders, no offers. As long as all orders can be scheduled for delivery, we proceed with the hill-climbing bidder.

It is crucial to our approach that the scheduler make use of the probabilities of winning each offer: the scheduler must schedule offers based on *expected quantities*.

We initialize bids to prices at which, according to our pricing model, we will win every RFQ with certainty. At these initial prices, if the scheduler cannot fit every order and RFQ into the schedule, then those RFQs which are not deemed profitable enough to include in the schedule at their current prices form a natural set of RFQs for which to raise prices. Indeed, we increase the prices of these RFQs, thereby decreasing their winning probabilities. In the next iteration, the scheduler, which schedules according to expected quantities, may be able to schedule these RFQs for production. Prices are increased (i.e., probabilities are decreased) until all RFQs can be scheduled. This process is guaranteed to converge, since the winning probability of RFQs above their reserve prices is zero, yielding a corresponding expected quantity of zero.

4.1. Scheduling: A Greedy Approach

Our greedy scheduler is passed both orders and offers, which it sorts as follows:

- Orders are placed before offers, since offers might not be won.
  - Orders are sorted by ascending due date, then by descending penalty.
  - Offers are sorted by descending profit per cycle (p_ι/c_j, where $j = f_\iota$), then by ascending due date, and lastly by descending penalty.

Note that offers are not sorted by probability. We experimented with this ordering, but profitability proved to be more important than probability.

Let o be the current order or offer and let j be o's SKU. The greedy scheduler addresses the orders and offers in sorted order as follows:

1. Schedule backwards from o's due date. That is, start by scheduling as much as possible of SKU j on the day o is due. If more needs to be scheduled, then schedule as much as possible on each successively earlier day until either no more is needed or the current day is reached.
2. If more of SKU j still needs to be produced, allocate as much as possible from product inventory.
3. If still more of SKU j is needed, schedule forwards from o's due date until either all of order o is scheduled or the cancellation date is reached.
4. If the cancellation date is reached, then cancel all scheduled production of SKU j for o.

Note that if o's due date is the current day, then there is no time to produce any more of SKU j. In this case, the greedy scheduler begins at step 2.

5. Bidding: A Mathematical Programming Approach

We now formulate mathematical programs to solve the delivery scheduling, production scheduling, and bidding problems. Our proposed solution to the bidding problem relies on a solution to the production scheduling problem. Similarly, our proposed solution to the production scheduling problem relies on a solution to the delivery scheduling problem. In our exposition, we distinguish between *simple* optimization problems, in which there is no uncertainty, and *stochastic* optimization problems. We present optimal solutions to the simple subproblems before describing our approximate solutions to the stochastic optimization problems. All solutions are described in terms of the variables, constants, and abbreviations listed in Figure 3.

5.1. Simple Scheduling

In simple scheduling, there is no uncertainty because there are no customer RFQs. The sole purpose of simple scheduling is to fill standing customer orders.

5.1.1. Simple Delivery Scheduling The (simple) delivery scheduling problem is one of allocating SKUs in product inventory to customer orders, given a $(D+E)$-day production schedule (see Figure 4). The following integer program solves the delivery scheduling problem. (Note: y_{jl} is constant in this formulation.)

$$\max_z \sum_{i=1}^{O} \left[\left(\sum_{l=1}^{d_i+E} z_{il}\pi_{il} \right) - \zeta_i \rho_{i(d_i+E)} \right] \tag{1}$$

subject to:

$$z_{il} \in \{0,1\}, \quad \forall i,l \tag{2}$$

$$\sum_{l=1}^{d_i+E} z_{il} \le 1, \quad \forall i \tag{3}$$

$$\sum_{l=1}^{t} \sum_{\{i \mid f_{ij}=1\}} q_i z_{il} \le b_j + \sum_{l=1}^{t-1} y_{jl}, \quad \forall j, t = 1,\ldots,D+E \tag{4}$$

The objective (Equation 1) is to maximize revenue and minimize penalties; but, no order can be delivered more than once (Equation 3); and, the total quantity of SKU j associated with orders delivered by day t cannot exceed the total inventory of SKU j produced by day $t-1$ plus any initial inventory (Equation 4).

5.1.2. Simple Production Scheduling The simple production scheduling problem is one of allocating cycles to SKUs, given a set of customer orders, initial component and product inventory, and a $(D+E)$-day procurement schedule (see Figure 5).

Variables

x_ι	bidding policy: bid price for RFQ ι
y_{jl}	production schedule: quantity of SKU j scheduled for production on day l
z_{il}	delivery schedule: 1 if order i is delivered on day l; 0 otherwise
$z'_{\iota l}$	delivery schedule: 1 if RFQ ι is delivered on day l; 0 otherwise

Constants in the Objective Functions

R	number of RFQs
O	number of orders
D	latest due date among all orders
E	number of days before a late order is canceled
q_i	quantity of order i
d_i	due date of order i
p_i	revenue for delivering order i on or before d_i+E
ρ_{il}	penalty incurred if order i is delivered on day l
q'_ι	quantity of RFQ ι
d'_ι	due date of RFQ ι
$\rho'_{\iota l}$	penalty incurred if RFQ ι is delivered on day l

Abbreviations in the Objective Functions

π_{il} revenue earned by delivering order i on day l

$$\pi_{il} = \begin{cases} q_i p_i & l \le d_i \\ q_i p_i - \rho_{il} & d_i < l \le d_i + E \end{cases}$$

$\pi'_{\iota l}(p)$ revenue earned by delivering RFQ ι on day l at price p

$$\pi'_{\iota l}(p) = \begin{cases} q'_\iota p & l \le d'_\iota \\ q'_\iota p - \rho'_{\iota l} & d'_\iota < l \le d'_\iota + E \end{cases}$$

ζ_i 1 if order i is not delivered at all; 0 otherwise

$$\zeta_i = 1 - \sum_{l=1}^{d_i+E} z_{il}$$

ζ'_ι 1 if RFQ ι is not delivered at all; 0 otherwise

$$\zeta'_\iota = 1 - \sum_{l=2}^{d'_\iota+E} z'_{\iota l}$$

Additional Constants in the Constraints

a_k	components of type k in initial inventory
α_{kl}	components of type k delivered on day $l+1$
b_j	number of PCs of SKU j in initial inventory
c_j	cycles expended to produce one PC of SKU j
e_{jk}	1 if SKU j contains component type k; 0 otherwise
f_{ij}	1 if order i is for SKU type j; 0 otherwise
$f'_{\iota j}$	1 if RFQ ι is for SKU type j; 0 otherwise
C_d	number of cycles on day d

Probabilistic Pricing Model

$P_\iota(p)$	probability of winning RFQ ι at price p

Figure 3. Mathematical Programming Variables, Constants, and Abbreviations

Delivery Scheduling
Inputs:
  Production Schedule
  Set of Customer Orders
  Product Inventory
Output:
  Delivery Schedule: map from SKUs to Customer Orders

Figure 4. Simple Delivery Scheduling

Simple Production Scheduling
Inputs:
  Set of Customer Orders
  Procurement Schedule
  Component Inventory
  Product Inventory
Outputs:
  Production Schedule: map from Cycles to SKUs
  Delivery Schedule: map from SKUs to Customer Orders

Figure 5. Simple Production Scheduling

The following integer program solves the simple production scheduling problem.

$$\max_{y,z} \sum_{i=1}^{O} \left[\left(\sum_{l=1}^{d_i+E} z_{il}\pi_{il} \right) - \zeta_i \rho_{i(d_i+E)} \right] \tag{5}$$

subject to Constraints 2, 3, 4, and the following:

$$y_{jl} \in \mathbb{Z}_{\geq 0}, \quad \forall j, l \tag{6}$$

$$\sum_{l=1}^{t} \sum_{\{j \mid e_{jk}=1\}} y_{jl} \leq a_k + \sum_{l=1}^{t-1} \alpha_{kl} \quad \forall k, t \tag{7}$$

$$\sum_{j} c_j y_{jl} \leq C_l, \quad \forall l \tag{8}$$

As in delivery scheduling, the objective (Equation 5) is to maximize revenue and minimize penalties; and, all of the delivery scheduling constraints still apply. In addition, Equation 7 expresses the resource constraint on components: The total quantity of component k used through day t cannot exceed the total quantity of component k delivered by day $t-1$ plus any initial inventory of component k. Equation 8 enforces the capacity constraint: The total number of production cycles used to produce all SKU types on day l cannot exceed the machine's capacity on day l.

5.2. Stochastic Scheduling and Bidding

Allowing for customer RFQs as well as standing customer orders introduces uncertainty into the scheduling problems. This uncertainty also arises in the bidding problem, where its exact nature depends on bids.

To handle this uncertainty, the scheduling problem can be formulated as a stochastic program (see Benisch *et. al.* [1]). In solving this stochastic program, we show that the sample average approximation method (SAA) [4] outperforms the expected value method [2] on this problem. Nonetheless, we relied on the expected value method in our implementation of BOTTICELLI-2003 because it readily applies to the bidding problem, whereas SAA does not.

5.2.1. Expected Production Scheduling In the production scheduling problem, the objective is to allocate cycles to SKUs not only to fill existing customer orders, but in addition to fill offers—customer RFQs equipped with bid prices—which may or may not become orders. We model this uncertainty by associating probabilities with offers: offers with low bid prices are assigned high probabilities, whereas offers with high bid prices are assigned low probabilities.

Production Scheduling
Additional Inputs:
  Bidding Policy
  Product Pricing Model
  Set of Customer Orders
  Procurement Schedule
  Component Inventory
  Product Inventory
Outputs:
  Production Schedule: map from Cycles to SKUs
  Delivery Schedule: map from SKUs to Customer Orders

Figure 6. Production Scheduling

The following integer program approximates the production scheduling problem. (Note: x_ι is constant in this formulation.)

$$\max_{y,z,z'} \sum_{i=1}^{O} \left[\left(\sum_{l=1}^{d_i+E} z_{il}\pi_{il} \right) - \zeta_i \rho_{i(d_i+E)} \right] + \sum_{\iota=1}^{R} P_\iota(x_\iota) \left[\left(\sum_{l=2}^{d'_\iota+E} z'_{\iota l}\pi'_{\iota l}(x_\iota) \right) - \zeta'_\iota \rho'_{\iota(d'_\iota+E)} \right] \tag{9}$$

subject to Constraints 2, 3, 6, 7, 8, and the following:

$$z'_{\iota l} \in \{0,1\}, \quad \forall \iota, l \tag{10}$$

$$\sum_{l=2}^{d'_\iota+E} z'_{\iota l} \leq 1, \quad \forall \iota \tag{11}$$

$$\sum_{l=1}^{t} \sum_{\{i \mid f_{ij}=1\}} q_i z_{il} + \sum_{l=2}^{t} \sum_{\{\iota \mid f'_{\iota j}=1\}} P_\iota(x_\iota) q'_\iota z'_{\iota l} \leq b_j + \sum_{l=1}^{t-1} y_{jl}, \quad \forall j, t \tag{12}$$

The objective function (Equation 9) maximizes profits from orders and *expected* profits from RFQs. Equation 11 states that no RFQ can be delivered more than once. Equation 12, which considers RFQs as well as orders, replaces Equation 4. Note the use of *expected* quantity $P_\iota(x_\iota)q'_\iota$ regarding RFQs in Equation 12.

Bidding
Inputs:
    Product Pricing Model
    Set of Customer RFQs
    Set of Customer Orders
    Procurement Schedule
    Component Inventory
    Product Inventory
Outputs:
    Bidding Policy: map from Customer RFQs to Prices
    Production Schedule: map from Cycles to SKUs
    Delivery Schedule: map from SKUs to Customer Orders

Figure 7. Bidding

5.2.2. Bidding The objective in the bidding problem is to find an optimal bidding policy. We solve this problem by extending the solution to the production scheduling problem based on the expected value method. In production scheduling, all RFQs are equipped with bid prices, which are constants. In the bidding problem, the prices at which to offer to fill RFQs are variables. Once prices become variables rather than constants, the objective function is no longer linear. (In fact, in our formulation, it is not even quadratic.) Thus, in our implementation we discretize prices to recover a linear formulation:

M number of prices
$\mu_{m\iota}$ price of RFQ ι with index m
$z'_{\iota lm}$ 1 if RFQ ι is delivered on day l at price indexed by m; 0 otherwise

Now the following integer program approximates the bidding problem.

$$\max_{y,z,z'} \sum_{i=1}^{O}\left[\left(\sum_{l=1}^{d_i+E} z_{il}\pi_{il}\right) - \zeta_i\rho_{i(d_i+E)}\right] + \sum_{m=1}^{M}\sum_{\iota=1}^{R} P_\iota(\mu_{m\iota})\left[\left(\sum_{l=2}^{d'_\iota+E} z'_{\iota lm}\pi'_{\iota l}(\mu_{m\iota})\right) - \zeta'_\iota\rho'_{\iota(d'_\iota+E)}\right] \quad (13)$$

subject to Constraints 2, 3, 6, 7, 8, and the following:

$$z'_{\iota lm} \in \{0,1\}, \quad \forall \iota, l, m \quad (14)$$

$$\sum_{m=1}^{M}\sum_{l=2}^{d'_\iota+E} z'_{\iota lm} \le 1, \quad \forall \iota \quad (15)$$

$$\sum_{l=1}^{t}\sum_{\{i \mid f_{ij}=1\}} q_i z_{il} + \sum_{m=1}^{M}\sum_{l=2}^{t}\sum_{\{\iota \mid f'_{\iota j}=1\}} P_\iota(z'_{\iota lm})q'_\iota z'_{\iota lm} \le b_j + \sum_{l=1}^{t-1} y_{jl}, \quad \forall j,t \quad (16)$$

6. Experiments

In this section we report on experiments designed to compare the performance of three bidding algorithms, one based on our mathematical programming solution, one hill-climbing bidder, and one blend of the two.

6.1. Heuristics

To bid optimally in TAC SCM, an agent would have to optimize with respect to (i) each of the other agent's individual strategies; and (ii) all possible future scenarios, weighted by their likelihoods. Agent modeling is not feasible in TAC, since the behavior of individual agents is observed only by the server. Thus, we collapse all agents' behaviors into one model (see Section 6.1.1). Furthermore, since it would be intractable to consider all possible futures, we rely on an heuristic that stands in the place of simulating the future—specifically, future orders (see Section 6.1.2).

6.1.1. Modeling The modeling module predicts the relationship between the bid price of an offer and the probability of winning that offer. There are several sources of information available for modeling this relationship. In our implementation, we utilize two: the first is a report provided by the server each day with the maximum and minimum closing prices for each SKU on the previous day; the second is BOTTICELLI's past offer prices and the orders that resulted. Our modeling module is concerned only with price and probability relationships for each SKU, rather than for each RFQ, since maximum and minimum prices are SKU-specific.

For each SKU, the modeler plots the minimum and maximum prices from the previous day at probabilities 1 and 0, respectively. Intuitively, low prices are likely to be winning prices, while high prices are likely to be losing prices. In addition, for each of the previous d days, BOTTICELLI's average offer prices are plotted against the ratio of the number of offers won to the number of offers issued. In total, our modeling module is provided with $d + 2$ points, which it fits using a least-squares linear regression. This linear *cdf* (price vs. probability graph) is adopted as the model that is input to the bidding module. (See Figure 8.)

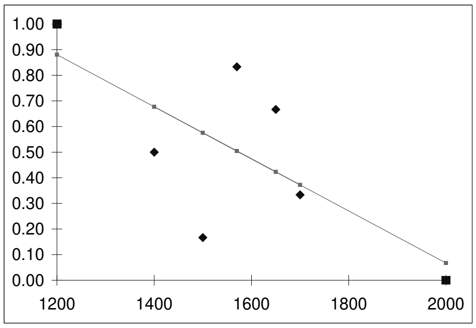

Figure 8. Price vs. Probability for a SKU. Diamonds are data points from offers sent during the past d days. Squares are data points from the previous day's minimum and maximum prices.

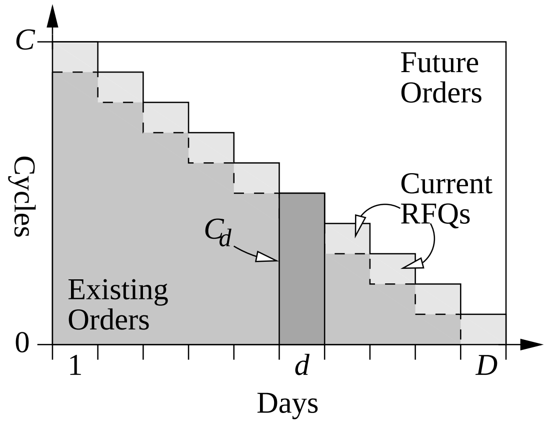

Figure 9. On day d, only $C_d = \frac{C((D-d)+1)}{D}$ cycles are made available to the scheduler. Cycles outside the triangle are reserved for future orders. D is the number of days of production in the schedule. C is the daily production capacity.

By experimentation, we found the value of 5 to be a good choice for d. This value allowed BOTTICELLI to be responsive enough to the changes in price that often accompanied another agent receiving a shipment of supplies, but prevented any drastic overreactions. We experimented with using additional information to create more stable models, such as providing weights for points based on the number of offers they represented, and maintaining the average of the d previous days' minimum and maximum prices. These methods, however, did not respond well to price jumps that were typical of the 2003 TAC SCM competition.

6.1.2. The Triangle Method In scheduling for multiple days of production, BOTTICELLI's scheduling module relies on the following heuristic: do not use all cycles on all days, but rather save production cycles on future days for future RFQs (see Figure 9). This heuristic is motivated by two assumptions. First, higher revenues can be earned by winning the same quantity of RFQs over multiple days, rather than winning a large quantity of RFQs on one day, since, according to our model, an agent can only win a large quantity on one day by bidding low prices. Second, the "character" RFQs of tomorrow will not differ significantly from the RFQs of today, since all RFQs are drawn from a uniform distribution. In particular, future RFQs will not be significantly better or worse than today's RFQs in terms of quantity, due date, etc. If, however, a change in the *number* of RFQs is predicted,[1] BOTTICELLI saves more (less) cycles if the number of RFQs is predicted to increase (decrease), since prices tend to increase (decrease) accordingly.

1 BOTTICELLI predicts the level of demand using a particle filter. Details of this approach are beyond the scope of this paper.

6.2. Experimental Setup

Our experiments consisted of 20 day trials, which proceeded as follows: On each day, the algorithms received a randomly generated set of RFQs drawn from a distribution similar to that of the TAC SCM game specification. Specifically, 300 RFQs were generated at random, with parameters uniformly distributed in the ranges shown in Table 1. Given these RFQs, the algorithms produced a bidding policy as well as production and delivery schedules for $D = 10$ days. Based on its bid prices and the corresponding probabilities, an algorithm won orders for some of the RFQs. The algorithms were then responsible for producing and delivering the products for these RFQs before their due dates or they were penalized according to the rate specified in the RFQ. The tests continued in this fashion for 20 days; this number was long enough to allow the algorithms to distinguish themselves, but short enough to allow several hundred iterations.

In order to mitigate any start effects in our experiments, the algorithms were initialized with the same set of 150 customer orders (thus, the first day looked like all other days). We made the simplifying assumption that all algorithms had an infinite component inventory, which, as alluded to earlier, is an artifact of the TAC SCM game design in 2003. Finally, to isolate the effects of the bidding algorithms, we relied on models that could perfectly predict the likelihood of winning any RFQ at any price.

Parameter	Range
Price	[$1600, $2300]
Quantity	[1, 20]
SKU	[1, 16]
Penalty	[5%, 15%] of Price

Table 1. Uniform Distribution Ranges

	Profits	Deliveries	$\overline{\text{Price}}$	Penalty
HG	$7,781,100	6,847	$1,193	$505,610
HE	$8,019,600	7,286	$1,095	$285,950
EB	$9,600,900	7,860	$1,222	$113,660

Table 2. Experimental Results

6.3. Experimental Results

The algorithms included in our experiments were the hill-climbing bidder with a greedy production scheduler (HG), the hill-climbing bidder with an expected production scheduler (HE), and the expected bidder (EB), which used its own schedule for production. Both of the hill-climbing bidders utilized a greedy scheduler to evaluate candidate bidding policies, as such policies needed to be evaluated hundreds of times. (The greedy scheduler completed in .01 seconds, on average, whereas the expected production scheduler completed in 1 second.) However, we allowed one of the hill-climbing bidders to utilize an expected scheduler for production scheduling only. Our hypothesis was that the expected bidder with built in scheduling and delivery modules would out perform all of the others, as it would be capable of performing a more global optimization while solving the bidding problem.

Relevant statistics of the 500 trials are given in Table 2. The mean profits of each algorithm over 20 days with 95% confidence intervals are shown in Table 3. These results validated our hypothesis. The expected bidder outperformed both instances of the the hill-climbing bidders in every category in Table 2. The 95% confidence intervals shown in Table 3 reveal that the difference in profits is statistically significant. The addition of the expected scheduling algorithm to the hill-climbing bidder helped it to achieve fewer penalties by improving the production scheduling solutions; however, the lack of a global bidding strategy still crippled its abilities. It seems that the expected bidder produced results that were close to optimal, since its total penalty was relatively small and it managed to utilize its factory at nearly full capacity each day without wasting many finished products.

	Low	High
HG	$7,756k	$7,804k
HE	$7,988k	$8,050k
EB	$9,585k	$9,617k

Table 3. Mean Profits—95% Confidence Intervals

7. Conclusion

Following Kiekintveld [3], we identify three key issues in supply chain management that are modeled in TAC SCM: (i) *uncertainty* about the future; (ii) *strategic behavior* among the entities; and (iii) *dynamism*: i.e., the temporal nature of the chain. BOTTICELLI adequately handles uncertainty (in the bidding problem), but makes simplifying assumptions to handle the strategic and dynamic components of the game. Rather than model each competing agent's strategic behavior individually, we collapse all agents' behaviors into one model, and optimize with respect to this model. In essence, we use decision-theoretic optimization techniques to approximate solutions to game-theoretic problems. Dynamic optimization models and techniques (e.g. MDPs) might be applicable in TAC SCM, but to optimize with respect to all possible future scenarios is clearly intractable. Instead, we rely on an heuristic we call the *triangle method*, by which we save production cycles on future days for future RFQs, particularly if prices are predicted to increase. In future versions of BOTTICELLI, we plan to build more powerful models of the agents' strategic environment, and to incorporate more sophisticated methods of dynamic optimization, particularly in the procurement problem.

References

[1] M. Benisch, R. Bent, A. Greenwald, V. Naroditskiy, and M. Tschantz. A stochastic programming approach to tac scm. Submitted for publication, November 2003.

[2] J. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer, New York, NY, 1997.

[3] C. Kiekintveld, M. 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. To appear.

[4] A. Kleywegt, A. Shapiro, and T. Homen-De-Mello. The sample average approximation method for stochastic discrete optimization. *SIAM Journal of Optimization*, 12:479–502, 2001.

[5] D. Pardoe and P. Stone. Tactex-03: A supply chain management agent. Submitted for publication, Jan 2004.