Optimization Under Uncertainty in Online Trading Agents

Michael Benisch*
Department of Computer Science
Brown University, Box 1910
Providence, RI 02912
mbenisch@cs.brown.edu

Abstract

Reasoning about uncertainty is an increasingly important aspect of automated decision making in domains such as airline crew scheduling, vehicle routing, and supply chain management. In this thesis, I examine the impact of various types of uncertainty on automated reasoning in such domains, as well as effective methods for addressing the uncertainty. The specific problem facing agents in the Trading Agent Competition in Supply Chain Management (TAC SCM) provides a rich setting for the discussion of uncertainty, and is the application domain of primary focus.

The production scheduling component of TAC SCM is examined, which concerns the optimization of a single finite capacity machine to satisfy pending contracts, and possible future ones. This problem is formulated as a stochastic program and is solved using the *sample average approximation* (SAA) [1] in an online setting to find today's optimal schedule, given probabilistic models of the future.

In addition, the architecture of Brown University's agent, Botticelli, (a finalist in the 2003 TAC SCM) is discussed, and the bidding component of TAC SCM is formulated as an extension of production scheduling. Mathematical programming approaches are applied in attempt to solve the problems optimally, and greedy methods that yield useful approximations are described. Test results compare the performance and computational efficiency of these alternative techniques.

1 Introduction

For many years, researchers in artificial intelligence and operations research have studied difficult problems in combinatorial optimization such as supply chain management, vehicle routing, and airline crew scheduling. The majority of this research has focused on solving deterministic problems; however, in many applications there is inherent uncertainty that is not captured by deterministic models. Moreover, in many optimization settings, stochastic information about the shape of the future is readily available in the form of probabilistic models built from historical data. Recently, computational and technological advances have made it feasible to reason about this stochasticity.

In general, two strategies are adopted when dealing with uncertainty in combinatorial optimization. The first strategy treats problems in an online fashion: algorithms are forced to make decisions in the face of incomplete information and accommodate new information only as it becomes available. Such algorithms typically fall into two categories. The first category includes simple, greedy heuristics for handling new information as it unfolds. The second category includes algorithms for finding optimal solutions given what is known; then, as new information becomes available, the solutions are re-optimized with the new information, respecting any unalterable prior decisions.

The second strategy for dealing with uncertainty in combinatorial optimization focuses on determining ahead of time a solution that is optimal in the expected sense. Stochastic programming is one example of

*This thesis includes joint work by Amy Greenwald, Victor Naroditsky, Michael Tschantz, Roger Lederman, and Ioanna Grypari [4] [3].

this strategy [7]. At a high level, stochastic programming considers problems in two stages. Decisions must be made in the first stage before pertinent information about the second stage is revealed, but the objectives in the second stage are dependent on the first stage decisions. Given stochastic information available about the second stage outcomes, the goal is to find the first stage decisions that maximize the profits of the first stage plus the expected profits of the second stage.

One computational bottleneck to solving stochastic programs is the calculation of expected profits in the second stage. This calculation typically involves enumerating all possible outcomes of the second stage (also known as *scenarios*). In many problems there are combinatorially many scenarios, making it prohibitively expensive to calculate the expected profits of the second stage. One common means of approximating this calculation is the so-called *expected value method* [7] (defined below).

Unfortunately, the expected value method ignores large portions of the given stochastic information. It has been shown that using additional stochastic information can improve the quality of solutions in dynamic vehicle routing [5, 6], packet scheduling [8], and elevator dispatching [20]. In these sample applications, stochastic information is exploited in widely different ways; however, the unifying theme seen throughout this research is that there are considerable advantages to taking account of stochastic information.

Shapiro, *et al.* [1, 16, 23] recently proposed an alternative approximation technique called *Sample Average Approximation* (SAA), which reduces the number of scenarios in consideration. They suggest that reasoning about only a subset of the scenarios, randomly sampled according to the scenario distribution, can effectively approximate the full scenario space. An important theoretical justification for this method is that as the sample size increases, the solution converges to an optimal solution in the expected sense. Indeed, the convergence rate is exponentially fast.

In the first part of this thesis, these two strategies to handling uncertainty are combined: *techniques for finding optimal solutions in the expected sense are used to solve combinatorial problems in an online setting.* The problem addressed is the scheduling component of the Trading Agent Competition in Supply Chain Management (TAC SCM), a classic combinatorial optimization problem with uncertainty (see `www.sics.se/tac/`). The problem is formulated as a stochastic program (SP) and SAA is used in an online setting to find today's optimal schedule, given predictive information about the future. This optimization procedure forms the heart of Botticelli, one of the finalists in the TAC SCM 2003 competition.

Two sets of experiments are described, using either one or two days of information about the future. In our two day experiments (using one day of information about the future), we show that SAA outperforms the *expected value method*, which solves a deterministic variant of the problem assuming all stochastic inputs have deterministic values equal to their expected values. In our three day experiments (using two days of information about the future), we show that SAA with lookahead outperforms greedy SAA.

The second part of this thesis focuses on the architecture of our agent, Botticelli. This architecture emphasizes three problems—bidding, production scheduling, and delivery scheduling. Section 8 describes a heuristic approach to these problems: greedy scheduling and bidding via hill-climbing. Section 9 details solutions that approximate optimal stochastic programming solutions. Section 10 presents experimental results.

2 TAC SCM

In recent years, the amount of time available for making complex managerial decisions in commercial settings has decreased dramatically [18]. Assuming this trend continues, it will become increasingly more important to develop tools that automate the decision making process. TAC SCM is a simulated market economy in which software agents tackle complex optimization problems in dynamic supply chain management.

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.

The TAC SCM simulation proceeds as follows: Each day, 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.[1] 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.

In the meantime, 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 components acquired from suppliers into SKUs according to a production schedule it generates for its finite-capacity, single-machine factory. In addition, it reports a delivery schedule assigning the SKUs in its inventory to customer orders.

Each simulated day represents a decision cycle for an agent, during which time the agents must solve the following four problems: bidding, scheduling, procurement, and allocation.

- The *bidding* problem determines the offer price for each RFQ.
- The *scheduling* problem determines the production schedule for each day.
- The *procurement* problem determines which components to buy from suppliers.
- The *allocation* problem matches SKUs in inventory to orders.

These four problems are highly interconnected. Indeed, an optimal solution to the scheduling problem yields an optimal solution to the procurement and allocation problems, since revenue maximization and cost minimization, which guide scheduling decisions, depend on how inventory is allocated to orders and on what supplies are procured. Moreover, an optimal solution to the bidding problem yields an optimal solution to the scheduling problem, since bidding decisions depend on manufacturing capacity constraints: too few winning bids lead to missed revenue opportunities, while too many winning bids lead to late penalties.

All of these problems involve decisions that must be made today with only stochastic information about tomorrow. *TAC SCM agents face combinatorial, online optimization problems with inherent uncertainty.* Due to an artifact in the design of TAC SCM 2003—namely, negligible component prices on day 1, which led to the placement of essentially infinite orders on day 1 for supplies to be delivered throughout the game—BOTTICELLIfocused on solving only three of these four problems: bidding, scheduling, and allocation.

To solve the scheduling problem, an agent must choose a schedule that accounts for outstanding orders, possible orders (among the existing RFQs), future RFQs, outstanding component orders, future component costs, and current component and SKU inventory. Which orders will materialize among the existing RFQs, the shape of future RFQs, possible supplier defaults on outstanding component orders, and future component costs are all stochastic elements of the scheduling problem. The following sections are concerned with solving simplified yet representative formulations of this scheduling problem.

[1] An agent may select to not send an offer for an RFQ, but this is equivalent to issuing an offer price above the reserve price.

3 Simple Scheduling

The *simple scheduling* problem is defined as follows: *Given* a set of orders, characterized by SKU, quantity, due date, penalty, and price; initial component inventory; a procurement schedule for components on each day; initial product inventory; one machine of finite capacity; the number of production cycles required to produce each product; and product specifications, namely which components comprise which products, *find* a production schedule that optimizes profit, or revenue less costs. This problem is dubbed *simple* (relative to the TAC SCM scheduling problem), since revenues and costs are deterministic.

3.1 Integer Linear Programming Solution

In this section, an integer linear programming (ILP) solution to the simple scheduling problem is presented.

3.1.1 Constants and Variables

Let O denote the set of orders. Each order $i \in O$ is characterized by the following information: SKU s_i, price p_i, quantity q_i, due date d_i, penalty ρ_i, and reserve price r_i. Let D denote the maximum due date among all orders and E denote the maximum acceptable overdue date. Let l range over days $1, 2, \ldots, D + E \equiv N$. Now, ρ_{il} is the penalty incurred if order i is filled on day l. For notational simplicity, let π_{il} represent the profit for filling order i on day l. The constant π_{il} is formally defined as follows:

$$\pi_{il} = \begin{cases} p_i & l \leq d_i \\ p_i - \rho_{il} & d_i < l \leq d_i + E \\ -\rho_{i(d_i+E)} & l > d_i + E \end{cases}$$

Let a_k denote the quantity of component k in initial inventory and b_j denote the quantity of SKU j in initial inventory. According to the procurement schedule, let a_{kl} denote the quantity of component k to be delivered on day l. Let C denote the capacity of the machine in terms of production cycles, and let c_j denote the number of production cycles required to manufacture SKU j. If component k is part of SKU j, then $e_{jk} = 1$; otherwise, $e_{jk} = 0$. Similarly, if order i is for SKU j, then $f_{ij} = 1$; otherwise, $f_{ij} = 0$.

In addition to these constants, this solution relies on the following variables:

- $z_{il} \in \{0, 1\}$, which indicates whether or not order i is filled on day l. (For notational simplicity, allow an order i to be filled on days $l > d_i + E$; however, conceptually, orders filled on day $d_i + E + 1$ are in fact unfilled orders and are treated as such in the formalization.)[2]
- $y_{jl} \in \mathbb{Z}_+$, which denotes the amount of SKU j scheduled for production on day l.

3.1.2 Objective Function and Constraints

The simple scheduling problem can be stated as follows:

$$\max \sum_{i \in O} \sum_{l=1}^{d_i+E+1} z_{il}\pi_{il} \tag{1}$$

[2]We introduce these variables for ease of exposition of the ILP, but in the implementation $z_{id_i+E+1} = 1 - \left(\sum_{l=1}^{d_i+E} z_{il}\right)$.

$$\text{subject to:} \quad \sum_{l=1}^{d_i+E+1} z_{il} = 1, \quad \forall i \tag{2}$$

$$\sum_{\{i \mid f_{ij}=1\}} \sum_{l=1}^{L} q_i z_{il} \leq b_j + \sum_{l=1}^{n-1} y_{jl}, \quad \forall j, n \in \{1, \ldots, N\}, L = \min(n, d_i + E) \tag{3}$$

$$\sum_{\{j \mid e_{jk}=1\}} \sum_{l=1}^{n} y_{jl} \leq a_k + \sum_{l=1}^{n-1} a_{kl} \quad \forall k, n \in \{1, \ldots, N\} \tag{4}$$

$$\sum_{j} c_j y_{jl} \leq C, \quad \forall l \tag{5}$$

$$z_{il} \in \{0, 1\}, \quad \forall i, l \tag{6}$$

$$y_{jl} \in \mathbb{Z}_+, \quad \forall j, l \tag{7}$$

- **Equation 1** is the objective function, namely to maximize profits, where the quantity $\sum_{l=1}^{d_i+E+1} z_{il}$ indicates whether or not order i is filled on day l.
- **Equation 2** states that an order must be filled exactly once. (Every order is either filled on some day $l \leq d_i + E$, or it is filled on day $d_i + E + 1$, meaning it is not filled.)
- **Equation 3** states that the total quantity of SKU j associated with all orders filled by day n does not exceed the total inventory produced by day $n - 1$ plus any initial inventory of SKU j.
- **Equation 4** expresses the resource constraints on components: The total quantity of component k used through day n must not exceed the total quantity of component k ordered by day $n - 1$ from all suppliers plus any initial inventory of component k.
- **Equation 5** enforces the capacity constraint: The total number of production cycles used to produce all SKU types on day l must not exceed the machine's daily capacity C.

4 Probabilistic Scheduling

The simple scheduling problem is extended to a *probabilistic* scheduling problem with an additional input. We add a set of RFQs, characterized like orders, but with an additional parameter α_i that represents i's likelihood of becoming an order. Implicitly, this formulation of the problem assumes that all likelihoods are independent. In probabilistic scheduling, the objective is to find a production schedule that maximizes *expected* profit. The following stochastic program (SP) achieves this objective.

4.1 Stochastic Programming Solution

Given a set of orders, and a set of RFQs *today* only a fraction of which will be realized *tomorrow*, we seek to produce an "optimal" set of SKUs *today* s.t. *tomorrow*'s profits will be maximized. More specifically, we seek to produce some set of SKUs, trading off production of those SKUs that can be used to fill the most profitable RFQs with those that can be used to fill those RFQs that are most likely to become orders.

Let $w_i \in \{0, 1\}$ indicate whether or not order i is filled on day 1,[3] and let $v_j \in \mathbb{Z}_+$ denotes the amount of SKU j scheduled for production on day 1. Let Ω_m denote the set of RFQs that are realized in the mth scenario (σ_m). Now let $z_{ilm} \in \{0, 1\}$ indicate whether or not order $i \in O$ or RFQ $i \in \Omega_m$ is filled on day l in scenario m, and let $y_{jlm} \in \mathbb{Z}_+$ denote the amount of SKU j scheduled for production on day l in scenario m.

$$\max \sum_{i \in O} w_i \pi_{i1} + \sum_{m} P(\sigma_m) \left[\sum_{i \in O \cup \Omega_m} \sum_{l=2}^{d_i+E+1} z_{ilm} \pi_{il} \right] \tag{8}$$

[3]Note: $w_i = 0$ for all RFQs i.

$$\text{subject to:} \quad w_i + \sum_{l=2}^{d_i+E+1} z_{ilm} = 1, \quad \forall m, i \in O \cup \Omega_m \tag{9}$$

$$\text{Stage 1 :} \quad \sum_{\{i \mid f_{ij}=1\}} q_i w_i \leq b_j, \quad \forall j \tag{10}$$

$$\sum_{\{j \mid e_{jk}=1\}} v_j \leq a_k, \quad \forall k \tag{11}$$

$$\sum_j c_j v_j \leq C \tag{12}$$

$$w_i \in \{0,1\}, \quad \forall i \qquad \text{and} \qquad v_j \in \mathbb{Z}_+, \quad \forall j \tag{13}$$

$$\text{Stage 2:} \quad \sum_{\{i \mid f_{ij}=1\}} q_i \left(w_i + \sum_{l=2}^{L} z_{ilm} \right) \leq b_j + v_j + \sum_{l=2}^{n-1} y_{jlm}, \quad \forall j, m, n \in \{2, \ldots, N\}, L = \min(n, d_i + 1) \tag{14}$$

$$\sum_{\{j \mid e_{jk}=1\}} \left(v_j + \sum_{l=2}^{n} y_{jlm} \right) \leq a_k + \sum_{l=1}^{n-1} a_{klm} \quad \forall k, m, n \in \{2, \ldots, N\} \tag{15}$$

$$\sum_j c_j y_{jlm} \leq C, \quad \forall m, l \in \{2, \ldots, N\} \tag{16}$$

$$z_{ilm} \in \{0,1\}, \quad \forall i, l, m \qquad \text{and} \qquad y_{jlm} \in \mathbb{Z}_+, \quad \forall j, l, m \tag{17}$$

- **Equation 8** is the objective function, namely to maximize profits, where the quantity $\sum_{l=1}^{d_i+E+1} z_{ilm}$ indicates whether or not order i is filled on day l in scenario Ω_m.
- **Equation 9** states that orders and RFQs must be filled exactly once. In particular, an order can be filled on day 1, or it can be filled at some later date in the scenarios. An RFQ can only be filled at some later date in the scenarios.
- **Equations 10, 11, and 12** pertain to the w_i and v_j variables: i.e., production and the allocation of inventory to orders on day 1. The total quantity of SKU j allocated to orders on day 1 cannot exceed the initial inventory of SKU j. The total quantity of component k used in production on day 1 cannot exceed the initial inventory of component k. The total number of production cycles used to produce all SKU types on day 1 must not exceed the machine's daily capacity C.

The final set of constraints pertains to production and the allocation of inventory to orders and RFQs on days $2, \ldots, N$ in the various scenarios.

- **Equation 14** expresses the resource constraints on inventory. In all scenarios, the total quantity of SKU j associated with all orders filled by day n (either on day 1 or on some later date in the scenarios) cannot exceed the total inventory produced by day $n-1$ plus and the initial inventory.
- **Equation 15** expresses the resource constraints on components. In all scenarios, the total quantity of component k used through day n cannot exceed the total quantity of component k procured by day $n-1$ and any initial inventory.
- **Equation 16** enforces the capacity constraint. In all scenarios, the total number of production cycles used to produce all SKU types on day l cannot exceed the machine's daily capacity C.

Lastly, let us the compute the probabilities of the various scenarios σ_m. Viewing σ_m as a bit vector, $\sigma_{mi} \in \{0,1\}$ indicates whether or not RFQ i is realized in scenario m. Now, the probability of the mth scenario is given by:

$$P(\sigma_m) = \prod_i \alpha_i^{\sigma_{mi}} (1-\alpha_i)^{1-\sigma_{mi}} \tag{18}$$

Algorithm	Output
Expected-Value (EV)	ILP with expected profits and quantities
Expected-Profit (EP)	ILP with expected profits
Expected-Quantity (EQ)	ILP with expected quantities
SAA	SP using sample average approximation with current RFQs
SAA-Average (SAAA)	SP using average future RFQs
SAA-Sampling (SAAS)	SP using sampled future RFQs
Not-in-time Production (NTP)	ILP ignoring RFQs

Table 1: Approximation Algorithms

5 Approximation Algorithms

Table 1 summarizes the seven algorithms that are featured in the experiments. The first three algorithms approximate the SP solution by solving variants of the simple scheduling problem. The *expected-value* algorithm solves the simple scheduling problem using expected profits and expected quantities. The *expected-profit* (respectively, *expected-quantity*) algorithm solves the simple scheduling problem using only expected profits (respectively, quantities).

The next three algorithms approximate the SP solution using *sample average approximation* (SAA), whereby they sample a subset of the scenario space according to its distribution, and optimize only with respect to those samples. *SAA-greedy* samples scenarios only consisting of one day's worth of actual RFQs. This algorithm makes no attempt to reason about future RFQs. *SAA-average* samples scenarios consisting of N days' worth of RFQs, assuming that all future RFQs look like an average RFQ. *SAA-sampling* samples scenarios consisting of N days' worth of RFQs; but, SAA-sampling generates sample future RFQs from an RFQ distribution, rather than assume that all future RFQs look like an average RFQ.

Finally, *not-in-time* production ignores stochastic information entirely. It only schedules orders—i.e., RFQs that have been realized. As its name suggests, this strategy can often lead to late penalties, since production does not begin until one day after RFQs are received.

6 Scheduling Experiments

The experiments we performed modeled the scheduling problem faced by an agent competing in the TAC SCM game, and similar problems faced by dynamic supply chain management systems. These experiments tested two hypotheses: (i) algorithms that utilize more stochastic information outperform those that do not; and (ii) algorithms that look ahead into the future outperform greedy algorithms.

Each N day trial of our experiments proceeded as follows. On each day, the algorithms received randomly generated RFQs drawn from a distribution similar to that of the TAC SCM game specification. Specifically, 200 RFQs were generated at random, with parameters uniformly distributed in the ranges shown in Table 5. Unlike in TAC SCM, (i) each RFQ was assigned some probability of becoming an order, and (ii) each RFQ was due on the day *immediately* after it was issued. Given a set of outstanding orders and new RFQs, the algorithms generated schedules and produced inventory. (Note: Based on the above distributions, 100 RFQs were expected to be converted to orders each day. Since each RFQ takes an average of 55 production cycles, the production of all orders requires more than the 2000 cycle capacity granted—the numbers 55 and 2000 are based on the TAC SCM game specification.) The next day after some more of the RFQs became orders, the algorithms allocated (i.e., delivered) product inventory resulting from production on previous days to current orders. Each order that was filled yielded some revenue, orders filled after their due dates also yielded revenue but incurred a penalty, and orders that were not filled at all incurred the maximum penalty of 5 times the RFQ's daily penalty value.

In our experiments, we made the following simplifying assumptions: no initial orders, no initial product inventory, and infinite component inventory. The third simplification, as alluded to earlier, is an artifact of

Parameter	Range
SKU	[1, 16]
Price	[$1600, $2300]
Quantity	[1, 20]
Penalty	[5%, 15%] of Price
Probability	[0, 1]

(a) Ranges of Uniform Distributions

Metric	Description
P	mean profit per order
C	% cycles used to fill orders
P/C	mean profit per cycle
EVPI	expected value of perfect information
VSI	value of stochastic information

(b) Description of Metrics

Table 2: Distribution Ranges and Metric Descriptions

the TAC SCM game design in 2003. These first two simplifications were designed to isolate the effects being tested by avoiding unnecessary complexity.

6.1 Metrics

Table 2(b) describes the metrics computed during each trial that were used to evaluate the approximation algorithms. The first metric, mean profit per order, was the primary measure of an algorithm's performance. Secondly, the percentage of cycles used to fill orders, indicates that percentage of the 2000 available cycles which were used by an algorithm to produce PCs that were actually sold. Perhaps more informatively, the next metric, profit per cycle, measures how well the algorithms filled more profitable, rather than less profitable, orders.

The *expected value of perfect information* is calculated by subtracting the mean profit an algorithm achieved from the maximum possible, which it could have achieved had it had perfect foresight: i.e., if it knew exactly which RFQs would become orders. The maximum possible mean profit was calculated using the ILP described in Section 3 after the fact. The *value of stochastic information* is the difference between an algorithm's mean profit and that of the Expected Value algorithm. This metric describes how much an algorithm gained or lost by utilizing stochastic information beyond simple expected values.

6.2 Two Day Experiments

In the two day experiments, algorithms received one set of RFQs and scheduled one day of production. The resulting product inventory was allocated to orders on the day 2. Any orders that were not filled incurred the maximum late penalty.

These experiments tested the ability of the algorithms to schedule production relying on only stochastic information. After day 1, there was no opportunity for production; thus, there was no opportunity to satisfy any orders that could not be filled from production on day 1.

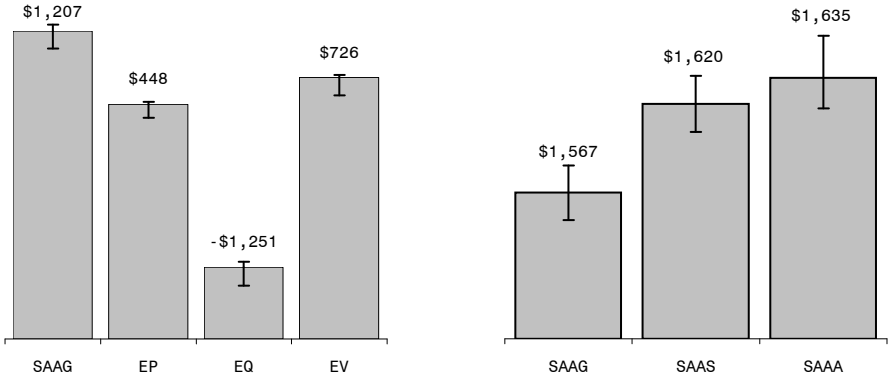
The metric values described in Table 2(b) for the two day experiments are shown in Table 3. In addition, the 95% confidence intervals of each algorithm's mean profit are shown in Figure 1(a).[4] Since SAAA and SAAS are identical to SAAG when there is only one day of production; these lookahead algorithms were excluded from the two day experiments. The NTP algorithm does not have a chance to schedule any production at all in these experiments, and was also excluded.

In the two day experiments, SAAG outperformed the other algorithms under all metrics. Figure 1(a) shows with 95% confidence that SAAG was significantly better in terms of mean profit. In second place (in terms of mean profit) was the EV algorithm. Despite selling fewer cycles, the EV algorithm outperformed the EP algorithm in terms of mean profit. These results suggest that the EV algorithm was filling fewer orders, but choosing some of the more profitable ones (as evidenced by the P/C values). The EP algorithm uses a more risky technique when scheduling production, since it attempts to fill every RFQ in its entirety.

[4]These confidence were intervals calculated using the bootstrap percentile-t method (see, for example, [9])

Algorithm	P	C	P/C	EVPI	VSI
SAA-Greedy	**$1,207**	**95.7%**	**$63.59**	**78,550**	**48,105**
Expected Profit	$448	93.9%	$24.40	154,450	-27,800
Expected Quantity	$-1,251	81.5%	$-77.71	324,390	-197,740
Expected Value	$726	90.8%	$47.65	126,650	0

Table 3: Two Day Experiments: Metric Values



(a) Two Day Experiments

(b) Three Day Experiments

Figure 1: Mean Profits with 95% Confidence Intervals

When it chose to fill an RFQ with a large expected profit and the RFQ did not become an order, at best the products that were made could be given to other less profitable RFQs. The EV algorithm subverts this problem by only producing RFQs in proportion to their likelihood of becoming an order. EQ performed relatively poorly on all computed metrics because it was scheduling production to fill expected quantities of what were sometimes unlikely realizations.

By design, SAAG uses more stochastic information than the other algorithms; therefore, the results from these experiments confirm our hypothesis that using more stochastic information leads to better performance.

6.3 Three Day Experiments

In the three day experiments, algorithms received two sets of RFQs and scheduled two days of production. The optimal solution to this scheduling problem in the expected sense is described by the stochastic program in Section 9, when the set of scenarios includes all combinations of realizations over both days of RFQs.

More specifically, the three day experiments proceeded as follows: The first set of RFQs, all of which were due on day 2, was received on day 1. The algorithms then scheduled production and built up their product inventory. On day 2, a subset of day 1's RFQs was selected at random to become orders. In addition, a second set of RFQs was received, all of which were due on day 3. The algorithms again scheduled production and built up their product inventory. In addition, any orders due on day 2 that could be filled were shipped, and revenues were recorded. On day 3, a subset of day 2's RFQs was selected at random to become orders. At this point, any outstanding orders due on day 2 that could be filled were shipped, and revenues were recorded, less late penalties; any orders due on day 3 that could be fulfilled were shipped, and revenues were recorded; and, penalties were recorded for any unfilled orders.

The purpose of these experiments was (i) to show that using more stochastic information is at least as useful across multiple days as it was in the 2 day experiment, and (ii) to test the ability of the algorithms

Algorithm	P	C	P/C	EVPI	VSI
SAA-Greedy	$1,567	98.6%	$79.5	87,350	34,310
SAA-Sample	$1,620	98.1%	$82.6	76,810	44,848
SAA-Average	**$1,635**	98.1%	**$83.4**	**73,670**	**47,990**
Expected Profit	$1,294	**98.7%**	$65.69	142,000	-20,200
Expected Quantity	$593	95.8%	$31.34	282,100	-160,300
Expected Value	$1,395	96.8%	$72.34	121,800	0
Not-In-Time	$-4,557	49.3%	$-462.53	1,312,100	-1,190,400

Table 4: Three Day Experiments: Metric Values

that made use of stochastic information to plan for the future given stochastic knowledge about the shape of future RFQs. To an extent, these experiments also tested the ability of the algorithms to recover from possible misuse of stochastic information in the two day experiments; but, such affects would be better uncovered by multiple day experiments.

As shown in Table 6.3, the stochastic programs outperformed all of the other schedulers in all but one calculated metric. Once again, these results confirm our hypothesis that using more stochastic information leads to better performance. All other results are consistent with results from the two day experiments.

Figure 1(b) shows that the stochastic algorithms that rely on forecasts about future RFQs outperformed SAAG. Unlike the greedy algorithm, the algorithms with lookahead use stochastic information about future RFQs to make scheduling decisions. These experiments confirmed our second hypothesis that utilizing more stochastic information about the future also lead to better performance.

The improvement seen was the result of day 1's with low-priced RFQs. The greedy algorithm was forced to cope with these poor RFQs because it did not utilize any stochastic information about the future. On the other hand, the algorithms with lookahead chose to schedule production that filled predicted future RFQs with higher prices, rather than waste production cycles on RFQs with low prices. SAAS and SAAA perform comparably in these experiments (see Figure 1(b)) because the RFQs sampled by SAAS were drawn from a uniform distribution, and thus tended to reflect mean RFQs. Given a non-uniform distribution, we conjecture that the sampling algorithm would make better use of stochastic information about future RFQs than an algorithm that relies on only the mean.

7 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 2.

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. 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 2: 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 3. 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.

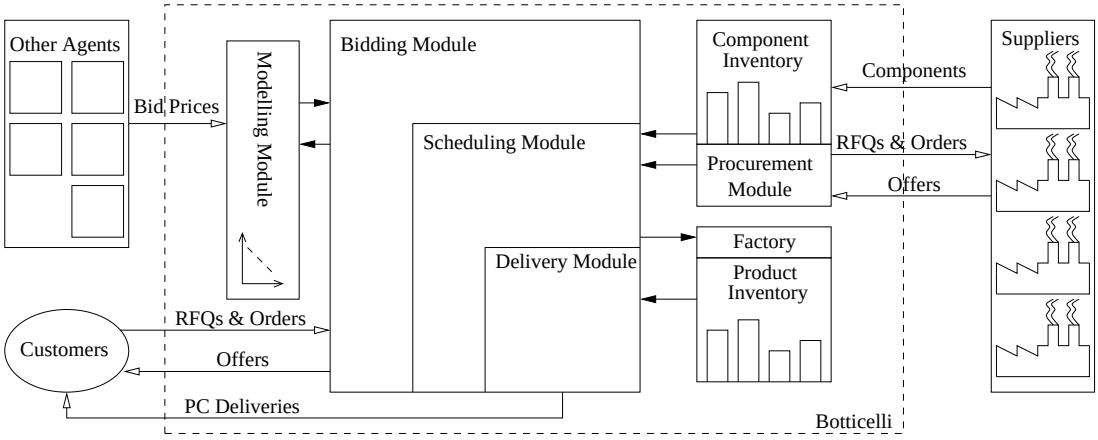


Figure 3: Botticelli: A Modular Design

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 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.

8 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 [21].

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.

8.1 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.

9 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 Sections 3, and 4.

9.1 Stochastic Scheduling and Bidding

Allowing for customer RFQs as well as standing customer orders introduces uncertainty into the scheduling problems, as discussed in Section 4. 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 Section 4). In solving this stochastic program, we show that the sample average approximation method (SAA) [16] outperforms the expected value method [7] 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.

9.1.1 Bidding

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 4: Bidding

The objective in the bidding problem is to find an optimal bidding policy, a high level description of the bidding problem is given in Figure 4. 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.

$$\max_{y,z,z'} \sum_{i=1}^{O} \left[\left(\sum_{l=1}^{d_i+E+1} z_{il}\pi_{il} \right) \right] + \sum_{m=1}^{M} \sum_{\iota=1}^{R} P_\iota(\mu_{m\iota}) \left[\sum_{l=2}^{d'_\iota+E+1} z'_{\iota lm}\pi'_{\iota l}(\mu_{m\iota}) \right] \tag{19}$$

subject to Constraints 2, 4, 5, 6, 7, and the following:

$$z'_{\iota lm} \in \{0,1\}, \quad \forall \iota, l, m \tag{20}$$

$$\sum_{m=1}^{M} \sum_{l=2}^{d'_\iota + E + 1} z'_{\iota lm} = 1, \quad \forall \iota \tag{21}$$

$$\sum_{l=1}^{t} \sum_{\{i \mid f_{ij}=1\}} q_i z_{il} + \sum_{m=1}^{M} \sum_{l=2}^{t} \sum_{\{\iota \mid f'_{\iota j}=1\}} P_\iota(m) q'_\iota z'_{\iota lm} \leq b_j + \sum_{l=1}^{t-1} y_{jl}, \quad \forall j, t \tag{22}$$

10 Bidding 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.

10.1 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 10.1.1). 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 10.1.2).

10.1.1 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 5.)

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.

10.1.2 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 6). 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.

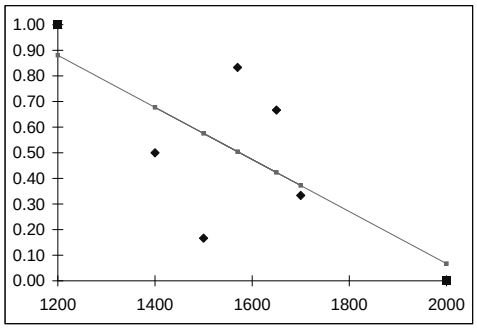


Figure 5: Price vs. Probability for a SKU. Diamonds are data points from offers sent during the past d days. Squares are data points from the previous day's minimum and maximum prices.

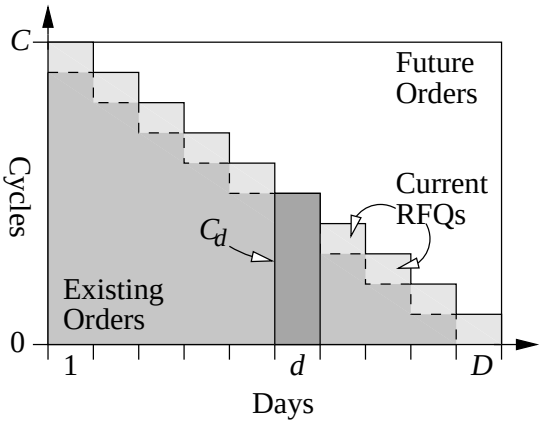


Figure 6: On day d, only $C_d = \frac{C((D-d)+1)}{D}$ cycles are made available to the scheduler. Cycles outside the triangle are reserved for future orders. D is the number of days of production in the schedule. C is the daily production capacity.

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,[5] BOTTICELLI saves more (less) cycles if the number of RFQs is predicted to increase (decrease), since prices tend to increase (decrease) accordingly.

10.2 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 5. 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.

[5] BOTTICELLI predicts the level of demand using a particle filter. Details of this approach are beyond our scope here.

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

Table 5: 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 6: Experimental Results

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.

10.3 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 6. The mean profits of each algorithm over 20 days with 95% confidence intervals are shown in Table 7. These results validated our hypothesis. The expected bidder outperformed both instances of the the hill-climbing bidders in every category in Table 6. The 95% confidence intervals shown in Table 7 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 7: Mean Profits—95% Confidence Intervals

11 Related Work

The Trading Agent Competition has been held annually since it was first introduced in 2000 [28]. It has fostered a community of researchers studying trading agent dynamics, and has produced extensive literature on related topics. Wellman *et al.* [29] provide a detailed summary of the entrants in the 2001 Classic variant of the Trading Agent Competition, a competitive Travel Agent simulation. In addition, Reeves and Wellman [22] use the framework of TAC Classic to develop an approach to automating business contract negotiation. Literature referring to TAC SCM is beginning to appear as well. For example, Kiekintveld *et al.* [15] describe an effective distributed feedback mechanism which was utilized in the Univeristy of Michigan's TAC SCM 2003 entry, DeepMaize.

Studies of supply chain management problems are prevalent in the combinatorial optimization and operations research literature. Generally, this research has advocated a decentralized approach to optimizing the necessary managerial decisions. An overview of models with such coordinated decentralized approaches to supply chain production scheduling and resource procurement is presented by Thomas and Griffin [27]. Swaminathan *et al.* provide an early description of dynamic supply chain practices, and explore computational mechanisms for partially automating them [26]. They also explore the benefits of modelling supply chain management practices in a multi-agent fashion, which provides early motivation for TAC SCM [25]. Bassek and Akella [30] argue for a more centralized approach to optimizing production scheduling and procurement planning of a single raw material, and Sun and Sadeh [12] extend this model to multiple raw materials. Members of the electronic commerce community have considered bidding aspect of supply chain management in isolation [2].

The deterministic TAC SCM production scheduling problem is closely related to the Discrete Lotsizing and Scheduling Problem (DLSP), first presented by Lasdon and Terjung [17] in 1971. The DLSP concerns optimizing several different sized batch scheduling jobs over discrete time periods on one or more finite capacity machines. This is analogous in TAC SCM to optimizing delivery scheduling decisions for orders of varying quantities, given finite daily factory production capacity. However, in TAC SCM, customer orders can be filled by factory production that is broken across multiple days, as unused PCs from one day are saved for the next. Although this will rarely occur in the deterministic TAC SCM production scheduling problem, batch scheduling cannot span multiple time periods at all in the DLSP. Fleischmann [10] presents a "generic" formulation of the DLSP as an integer linear program, which is very similar to our formulation of the TAC SCM Simple Scheduling problem, described in Section 3. The main differences, in addition to the important difference explained above, result from a lack of startup costs in TAC SCM, and the ability to violate constraints concerning demand fulfillment by paying late penalties in TAC SCM. Furthermore, the TAC SCM production scheduling problem involves varying rewards associated with different units of customer demand, due to different prices associated with each order. Finally, the TAC SCM production scheduling problem allows for a varied number of items to be scheduled in each time period, since each order takes a different amount of factory capacity, which is constrained on a daily basis. Salomon *et al.* [19] examine the computational complexity of various DLSP instances, and show that even very simple cases (single machine problems with absent setup costs) are NP-Hard.

Recent work has been done to extend deterministic formulations of DLSP to account for uncertainty about future events. For example, Haugen *et al.* [13] suggest using a progressive hedging algorithm to solve a stochastic variant of the problem, dubbed the Stochastic Lotsizing and Scheduling Problem (SLSP). The SLSP is very closely related to the TAC SCM probabilistic scheduling problem presented in Section 4, again the main differences being the same as those found in the deterministic counterparts.

12 Conclusion

The research problems in the Trading Agent Competitions are typically approached using clever heuristics and optimization techniques. With a few notable exceptions [11, 24], these methods have tended to ignore some of the information that characterizes the uncertainty in the problems. We have seen that it is possible to substantially improve the performance of algorithms by incorporating stochastic information about the

future. More importantly, we have seen the precise methodology for including stochastic information is an important indicator of an algorithm's performance. Indeed, the scheduling decisions determined by a stochastic programming approach that aims to characterize all of the uncertainly outperforms methods that make no use or partial use of uncertainty.

Research on dynamic supply chain management can proceed in a number of future directions. By itself, the probabilistic scheduling approach makes worthwhile decisions given a fixed distribution for future RFQs. However, in the bidding problem, there is an opportunity to alter the distributions of the RFQs that could become orders, namely by raising or lowering bids. In BOTTICELLI, the probabilistic scheduling algorithm serves as a critical component for evaluating various bidding strategies. The algorithms presented here rely heavily on stochastic programming to handle uncertainty; however, there are other techniques, such as consensus [5] and POMDPs [14], for coping with uncertainty, which may prove useful when the full TAC SCM problem (including procurement) is considered.

Following Kiekintveld [15], 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] AHMED, S., AND SHAPIRO, A. The sample average approximation method for stochastic programs with integer recourse. *Submitted for publication* (2002).

[2] ARUNACHALAM, R., AND SADEH, N. The supply chain trading agent competition. *Workshop on Trading Agent Design and Analysis* (2004). To appear.

[3] BENISCH, M., GREENWALD, A., GRYPARI, I., LEDERMAN, R., NARODITSKIY, V., AND TSCHANTZ, M. Botticelli: A supply chain management agent. In *Autonomous Agents and Multi Agent Systems* (2004). To appear.

[4] BENISCH, M., GREENWALD, A., NARODITSKIY, V., AND TSCHANTZ, M. A stochastic programming approach to tac scm. In *ACM Conference on Electronic Commerce* (2004). To appear.

[5] BENT, R., AND HENTENRYCK, P. V. Scenario Based Planning for Partially Dynamic Vehicle Routing Problems with Stochastic Customers. *Operations Research, To appear* (2001).

[6] BENT, R., AND HENTENRYCK, P. V. Dynamic Vehicle Routing with Stochastic requests. In *International Joint Conference on Artificial Intelligence (IJCAI)* (Acapulco, Mexico, 2003).

[7] BIRGE, J., AND LOUVEAUX, F. *Introduction to Stochastic Programming*. Springer, New York, NY, 1997.

[8] CHANG, H., GIVAN, R., AND CHONG, E. On-line Scheduling Via Sampling. In *Artificial Intelligence Planning and Scheduling (AIPS)* (Breckenridge, Colorado, 2000), pp. 62–71.

[9] EFRON, B., AND TIBSHIRANI, R. *An Introduction to the Bootstrap*. Chapman and Hall, 1993.

[10] FLEISCHMANN, B. The discrete lotsizing and scheduling problem. *European Journal of Operations Research 44(3)* (1990), 337–348.

[11] GREENWALD, A. Bidding marginal utility in simultaneous auctions. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence* (August 2003), pp. 1463–1464.

[12] J.SUN, AND SADEH, N. Coordinating multi-attribute reverse auctions subject to finite capacity considerations.

[13] K. HAUGEN, A. LKKETANGEN, D. W. Progressive hedging as a meta-heuristic applied to stochastic lot-sizing. *European Journal of Operations Research 132(1)* (2001), 116–122.

[14] KAELBLING, L., LITTMAN, M., AND CASSANDRA, A. Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence 101 (1-2)* (1998), 99–134.

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

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

[17] Lasdon, L., and Terjung, R. An efficient algorithm for multi-item scheduling. *Operations Research 19* (1971), 946–969.

[18] Lev, B., and Zarowin, P. The boundaries of financial reporting and how to extend them. *Journal of Accounting Research 37* (1999), 353–385.

[19] M. Salomon, L. Kroon, R. K., and van Wassenhove, L. Some extensions of the discrete lotsizing and scheduling problem. *Management Science 37(7)* (July 1991), 801–812.

[20] Nikovski, D., and Branch, M. Marginalizing Out Future Passengers in Group Elevator Control. *Uncertainity in Artificial Intelligence (UAI)* (2003).

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

[22] Reeves, D. M., Wellman, M. P., and Grosof, B. N. Automated negotiation from declarative contract descriptions. In *Proceedings of the Fifth International Conference on Autonomous Agents* (Montreal, Canada, 2001), J. P. Müller, E. Andre, S. Sen, and C. Frasson, Eds., ACM Press, pp. 51–58.

[23] Shapiro, A., and Homen-De-Mello, T. On rate convergence of monte carlo approximations of stochastic programs. *SIAM Journal on Optimization 11* (2001), 70–86.

[24] Stone, P., Schapire, R., Littman, M., Csirik, J., and McAllester, D. Decision-theoretic bidding based on learned density models in simultaneous, interacting auctions. *Journal of Artificial Intelligence Research 19* (2003), 209–242.

[25] Swaminathan, J., Smith, S., and Sadeh, N. Modeling supply chain dynamics: A multiagent approach, 1998.

[26] Swaminathan, J., Smith, S. F., and Sadeh, N. M. Modeling the dynamics of supply chains. In *Proceedings of the AAAI-94 SIGMAN Workshop* (Seattle, WA, 1994), pp. 113–122.

[27] Thomas, D., and Griffin, P. Coordinated supply chain management. *European Journal of Operations Research 94* (1996), 1–15.

[28] Wellman, M., and Wurman, P. A trading agent competition for the research community, 1999.

[29] Wellman, M. P., Greenwald, A., Stone, P., and Wurman, P. R. The 2001 trading agent competition. In *Fourteenth Conference on Innovative Applications of Artificial Intelligence* (Edmonton, 2002), pp. 935–941.

[30] Y.Bassok, and Akella, R. Multiplan co-ordination: Combined production and ordering with demand and supply uncertainity. *Management Science 37(12)* (1991), 12.