A Stochastic Programming Approach to Scheduling in TAC SCM

Michael Benisch, Amy Greenwald, Victor Naroditskiy, Michael Tschantz
Department of Computer Science
Brown University, Box 1910
Providence, RI 02912
{mbenisch,amy,vnarodit,mtschant}@cs.brown.edu

Abstract

In this paper, we combine two approaches to handling uncertainty: *we use techniques for finding optimal solutions in the expected sense to solve combinatorial optimization problems in an online setting.* The problem we address is the scheduling component of the Trading Agent Competition in Supply Chain Management (TAC SCM) problem, a combinatorial optimization problem with inherent uncertainty (see `www.sics.se/tac/`). This problem is formulated as a stochastic program, and is solved using the *sample average approximation* (SAA) method [1, 9, 12] in an online setting to find today's optimal schedule, given probabilistic models of 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 one and two days' worth of information about the future. In the two day experiments (using one day's worth of information about the future), it is shown that SAA outperforms the *expected value method* [4], which solves a deterministic variant of the problem assuming all stochastic inputs have deterministic values equal to their expected values. In the three day experiments (using two days' worth of information about the future), it is shown that SAA with lookahead outperforms greedy SAA. This approach generalizes to N days of lookahead, and since the problem setting is one of online optimization, the benefits of two day lookahead accrue rapidly. It remains to show that our approach improves the performance of agents in TAC SCM.

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 strategy [4]. 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* [4] (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 [2, 3], packet scheduling [5], and elevator dispatching [11]. 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, 9, 12] recently proposed an alternative approximation technique called *Sample Average Approximation* (SAA), which reduces the number of scenarios. They suggest using only a subset of the scenarios, randomly sampled according to the scenario distribution, to represent 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 this paper, we combine these two strategies to handling uncertainty: *we use techniques for finding optimal solutions in the expected sense to solve combinatorial problems in an online setting.* The problem we address is the scheduling component of the Trading Agent Competition in Supply Chain Management (TAC SCM) problem a classic combinatorial optimization problem with uncertainty (see `www.sics.se/tac/`). We formulate this problem as a stochastic program and we use SAA 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.

We describe two sets of experiments, 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. Our approach generalizes to N days of lookahead, and since our problem setting is one of online optimization, the benefits of two day lookahead accrue rapidly.

2 TAC SCM

In recent years, the amount of time available for making complex managerial decisions in commercial settings has decreased dramatically [10]. 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

[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.

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—our agent, BOTTICELLI, focused on solving only three of these four problems: bidding, scheduling, and allocation. Here, we present our solution to the scheduling (and allocation) problem.

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 remainder of this paper is concerned with solving simplified yet representative formulations of this scheduling problem.

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, we present an integer linear programming (ILP) solution to the simple scheduling problem.

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, we 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, our 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, we 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:

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

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

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

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

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

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

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

- **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.

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

- **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. (For orders $i \in O$, $\alpha_i = 1$.) 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.

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

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

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

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

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

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

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

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

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

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

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

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-Greedy (SAAG)	SP using only 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

- **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:

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

5 Approximation Algorithms

Table 1 summarizes the seven ILPs that are featured in our experiments. All ILPs were solved using CPLEX version 7.5, terminating with the first feasible solution. The first three 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. Expected profits are computed by multiplying π_{il} by α_i in Equation 1.[4] Expected quantities are computed by multiplying q_i by α_i in Equation 3. The *expected-profit* (respectively, *expected-quantity*) algorithm solves the simple scheduling problem using only expected profits (respectively, quantities).

[4] Recall that $\alpha_i = 1$ for all orders i.

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.[5]

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 Empirical Results

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 2(a). 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 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.

[5] All of our SAA algorithms sampled 30 scenarios, since larger sample sizes showed no significant advantage.

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

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

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).[6] 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.

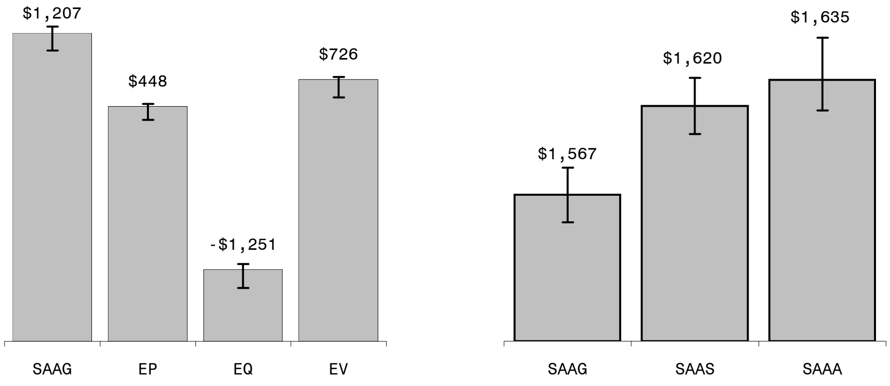
In the two day experiments, SAAG outperformed the other algorithms under all metrics. Figure 1(a) shows with 95% confidence that SAAG was significantly better in terms of mean profit. In second place (in terms of mean profit) was the EV algorithm. Despite selling fewer cycles, the EV algorithm outperformed 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. 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

[6] These confidence were intervals calculated using the bootstrap percentile-t method (see, for example, [6])



(a) Two Day Experiments

(b) Three Day Experiments

Figure 1: Mean Profits with 95% Confidence Intervals

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

in Section 4.1, 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 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 leads 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.

This paper only considers experiments of 2 and 3 days in duration. In future work, we plan to assess the performance of these algorithms over many days, such as the typical 220 days of the TAC SCM game.

7 Conclusion

The research problems in the Trading Agent Competitions are typically approached using clever heuristics and optimization techniques. With a few notable exceptions [7, 13], these methods have tended to ignore some of the information that characterizes the uncertainty in the problems. This paper suggests that it is possible to substantially improve the performance of algorithms by incorporating stochastic information about the future. More importantly, this paper shows that 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 [2] and POMDPs [8], for coping with uncertainty, which may prove useful when the full TAC SCM problem (including procurement) is considered.

References

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

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

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

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

[5] 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.

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

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

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

[9] 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.

[10] LEV, B., AND ZAROWIN, P. The boundaries of financial reporting and how to extend them. *Journal of Accounting Research 37* (1999), 353–385.

[11] NIKOVSKI, D., AND BRANCH, M. Marginalizing Out Future Passengers in Group Elevator Control. *Uncertainity in Artificial Intelligence (UAI)* (2003).

[12] 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.

[13] 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.