Empirically Testing Decision Making in TAC SCM

Erik Peter Zawadzki and **Kevin Leyton-Brown**
University of British Columbia
{epz, kevinlb}@cs.ubc.ca

Abstract

Design decisions for TAC SCM agents are usually evaluated empirically by running complete agents against each other. While this approach is sufficient for many purposes, it be difficult to use it for running large-scale, controlled experiments to evaluate particular aspects of an agent's design. This is true both for technical reasons (availability of other agent code, the trouble of setting up a TAC server, etc.) and especially because results can depend heavily on the experimenter's choice of opponent agents. This paper introduces a novel model of the TAC SCM scheduling problem for use in such empirical evaluations. The model aims to reduce the experimental variability caused by the experimenter's choice of opponent agents, replacing markets with stochastic processes that simulate them. These stochastic processes are designed by using machine learning to distill typical agent behaviors from real game logs taken from the TAC SCM finals. After describing the operation of our model, we validate it by showing that its predictions of opponent behavior are highly consistent with further game logs that were not used for building the model. Finally, we apply our model to investigate the performance of several integer/linear programming approaches for solving the delivery and scheduling subproblems in TAC SCM.

Introduction

Supply Chain Management (SCM) is a problem with great industrial importance. Supply chains describe the multi-stage process of converting raw materials into finished products for end users, usually with many manufacturing phases along the way. These different phases involve a variety of different stakeholders, such as suppliers, manufacturers, warehousers, transportation companies and retailers. Each stakeholder faces a variety of difficult problems, ranging from the computational (How could I best use a given set of goods if I were able to procure them? If I won a given set of contracts from my customers, how would I satisfy them using the inventory I have at hand?) to the game-theoretic (How can I expect other members of the supply chain to behave? How should I respond to these beliefs? How will others' future beliefs be shaped by *my* actions?)

The Trading Agent Challenge SCM (TAC SCM) game was established to promote study of automated approaches to solving these sorts of supply chain management problems (Collins *et al.* 2005). TAC SCM consists of a stochastic simulation of a relatively simple multi-period business environment that is nevertheless rich enough to give rise to some of the complexity inherent in more elaborate settings.

Designing an agent for TAC SCM can be understood as providing answers to two broad questions—how to model the environment (in particular, other agents' behavior, since other parameters of the simulation are given) and what decisions to take based on this model. A decision (made "daily" in the simulation) can be decomposed into four parts: component ordering, customer bidding, production scheduling, and delivery scheduling (Collins *et al.* 2005). All four problems are interrelated, with changes to each problem's solution affecting the solutions of the others. However, because TAC SCM agents face strict time constraints, a dominant approach from the literature has been to treat these decision subproblems as largely separate, allowing the interdependence of these subproblems to be expressed primarily through summary statistics (Benisch *et al.* 2004a; Pardoe & Stone 2006b; Dahlgren & Wurman 2004; Keller, Duguay, & Precup 2004).

In the early days of TAC, researchers tended to focus much of their energy on building good agents and explaining their designs. Recently, agent designs have begun to stabilize, and indeed there are signs of convergence to agreement about certain high-level ingredients of a successful TAC design. This has led to an increased focus on a more "scientific" study of TAC, with researchers identifying an abstract issue that arises in the competition, attempting to isolate that issue experimentally, and reporting the results along with some general conclusions (see, among others, Wellman *et al.* (2006), Benisch *et al.* (2004b), Kiekintveld *et al.* (2004), Pardoe & Stone (2006a), Vetsikas & Selman (2003)).

Our paper attempts to be guided by this scientific sensibility in studying the scheduling components of a TAC agent's design. We decided to focus on scheduling for several reasons. First, this is one of the key computational problems faced by an agent designer; for example, in our own (limited) past efforts to design a TAC agent, we found that addressing it consumed much of our time. Second, scheduling problems have widely been recognized as important in the artificial intelligence and operations research literatures, and have been studied extensively outside the TAC context.

Finally, while it is possible to identify a small set of "state-of-the-art" approaches for solving the problem, we are not aware of any work that compares these approaches directly in the context of the TAC SCM problem. Indeed, it is easy to see why such work has not been done: this would involve a great deal of implementation effort, and in the end its conclusions would depend on other elements of the agent design and on parts of the environment (e.g., the designs of the other competing agents) that had not varied.

We have attempted to get around these problems by designing a reusable test suite that can be used to empirically test scheduling approaches for TAC SCM. In what follows, we describe our modeling choices, validate our model using data from the logs of actual TAC SCM competitions, and then apply the model to gather some preliminary results comparing several approaches based on linear programming and stochastic programming.

Model

Evaluations of candidate solutions to the problems posed by TAC SCM have usually consisted of implementing a particular agent design. The resulting agents are then pitted against a variety of other agents to assess the suitability of the implemented design (Pardoe & Stone 2006b; Benisch *et al.* 2006b; Dahlgren & Wurman 2004; Keller, Duguay, & Precup 2004). While this approach is useful for drawing conclusions about complete agents, it is difficult to use it to draw more general results about particular approaches to the decision subproblems outside of the particular agent context that it was tested in, due to the complicated multi-agent nature of the competition.

One way to reduce the variability caused by multi-agent interactions in the customer and component markets is to simulate the majority of the TAC SCM server, but to replace any agent interaction with a stochastic process learned from log data. In TAC SCM, all the agents communicate back and forth with the two markets, as shown in Figure 1. The actions of an agent, such as bidding on a customer RFQ, will impact the other agents. In our model, drawn in Figure 2, the markets are replaced with processes that are independent of agent actions. Since the processes are not effected by the agents, there is no way for one agent's actions to impact any other agent, and so given these processes a single agent may be separated and studied in isolation.

This model attempts to retain "typical" market characteristics without the need to run a large set of individual agents. This retains some of the uncertainty of the complex multi-agent system, but makes the response of the decision module easier to experiment with and reduces variance. This can be especially useful if we want to study particular situations that infrequently occur, such as the impact of component scarcity on manufacturing scheduling.

TAC SCM is uncharacteristic of practical supply chain scenarios in that all agents abruptly begin at the same moment without any inventory, and similarly because that inventory abruptly loses all value at a commonly-known time. These two discontinuities have a significant impact on agent strategies and payoffs, and have been extensively studied, e.g., (Pardoe & Stone 2004; 2006b; Ketter *et al.* 2004; Benisch *et al.* 2004a; Eriksson, Finne, & Janson 2006). In our work, however, we are interested in modeling and studying a system's steady-state behavior. When using game logs, we therefore study only data from days 40–200, an interval that we observed to exhibit a qualitatively steady mean-winning price.

Motivation

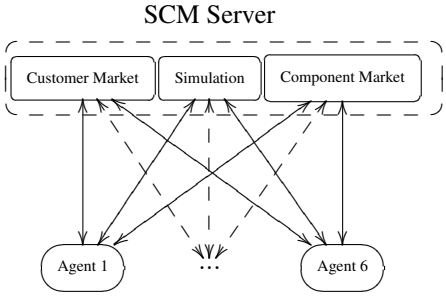

Figure 1: TAC SCM

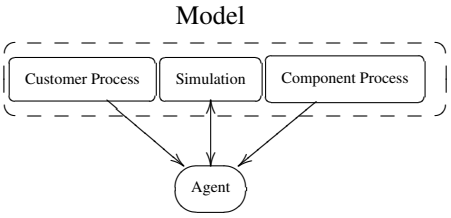

Figure 2: Proposed Model

Our model, summarized by the pseudo-code in Figure 3, generates RFQs in exactly the same way as the TAC SCM server, using the same distributions for determining due dates, late penalties, quantities, and reserve prices. It also mimics the server in considering delivery instructions: when the agent instructs that a particular customer order should be delivered, the model checks to make sure that delivering it is feasible with respect to the goods' shopkeeping unit (SKU) inventories, then evaluates the profits that the agent receives, appropriately subtracting late penalties. Orders that exceed the maximum allowable number of late days are removed from the list of outstanding orders, and the appropriate defaulting fee is removed from the agent's balance.

There is another area of multi-agent interaction that still needs to be modeled—the component market. This paper focuses on the customer market, which is described in the next section. Afterward, we also give a simple component market model. A more sophisticated approach would probably do better here; we intend to elaborate our model in this direction in future work.

Modeling Bidding

As identified above, one of the areas of multi-agent interactions that we will try to replace with a stochastic model is the

```
Initialize order = ∅
Initialize the PC inventory, skuInv = {0, ..., 0}
Initialize the component inventory,
 compInv = {0, ...0}
Initialize balance = 0
FOR t UNTIL T
  Get new set of customer RFQs Rfq
  Deliver any ordered component ready on
   day t − 1, determined with the component
   ordering model
  Give agent Rfq, Order, SkuInv, CompInv
  Ask agent for deliveries Del
  Ask agent for manufacturing decisions
   Man
  Ask agent for component orders CompOrd
  Ask agent for RFQ bids bid
  FOREACH d ∈ Del
    IF d is feasible
     Deliver d
     Subtract goods from skuInv
     Add profit to balance
    END IF
  END FOR
  FOREACH m ∈ Man
    IF m is feasible
     Add goods to skuInv
     Subtract components from compInv
    END IF
  END FOR
  FOREACH b ∈ Bid
    Use a model of customer bidding to
    determine winning probability
    IF b is the winning bid for r
     Order = Order ∪ {r}
    END IF
  END FOR
  FOREACH o ∈ Order
    IF o is canceled on day t
     Order = Order \ {o}
     Subtract defaulting penalty from balance
    END IF
  END FOR
END FOR
```

Figure 3: Model Pseudo Code

customer market. Ultimately, we wish to construct a principled way of determining the probability that an agent will win with a given bid amount for a given RFQ. In this section we describe the method that we propose for modeling that probability.

This model is called a Customer Market Process (CMP). We define this as the process that controls the distribution of winning RFQs for a particular day, and we wish to find one that is consistent with log data. More formally, we are attempting to model $p(B|\theta_t)$, where B is a random variable that represents the price of a winning order and θ_t is our distribution parameters for day t. A bid for an RFQ is more likely to win if more of the mass of the probability density function is at a higher price than the bid. (Remember, in an RFQ market, the lowest bid wins.) Let w be a binary variable such that $w = 1$ iff the agent wins the RFQ. Then the chance of winning an RFQ with a bid of b per unit on day t is

$$p(w|b, \theta_t) = \int_b^\infty p(x|\theta_t)dx. \quad (1)$$

This is the amount of probability mass higher than the placed bid. We will model this probability distribution as a linear dynamic system (LDS) and learn it from data using a Kalman filter.

Before explaining the details of our model, we remark that learning the CMP is similar to the problem of offer acceptance prediction (determining how likely it is for an bid to win an order) that all TAC SCM agents have to face with the additional demand of being a generative model. For instance, CMieux's price forecasting component fills this role (Benisch *et al.* 2006a). However this approach differs from ours in that it attempts to predict the future price of a product using a least-squares regression approach. This amounts to learning a distribution that has its mean at the predicted price and a constant variance, whereas we propose to learn a probability distribution in which both the mean and variance vary with time. TAC SCM 2006 winner TacTex-06's offer acceptance predictor is closer to our LDS model as it attempts to learn the function that maps the current bid to winning distribution using a particle filtering approach (Pardoe & Stone). In this paper we use the simpler and computationally more efficient Kalman filtering approach; below, we show that it is sufficient to achieve excellent performance.

We now justify and describe our model of the customer bidding problem as an LDS, after which we will describe how these models can be learned. We begin from the assumption that winning bid amounts are normally distributed, which is justified by the log data being tightly clustered around a single mean. This means that Equation (1) involves the normal cumulative distribution function which is easy to approximate using Taylor expansions. Assuming that the distribution is normal also allows us to work with simple summary statistics for each day.

In order to simplify the problem we will make the following assumptions:

1. θ_t is a linear function of θ_{t-1} with additive, unbiased, normal noise;

2. the distribution of winning bids for a given day and SKU is normal.

These assumptions mean that the problem can now be modeled as an LDS. LDS models may be seen as an extension of Hidden Markov Models (HMMs) with continuous latent variables. In this case, the latent variables, also called hidden states, are the parameters θ. Such models are widely used because their linearity and normality assumptions make it simple to perform parameter estimation, inference and prediction.

The process may be summarized in the following equations, following (Roweis & Ghahramani 1997), with notation defined below.

$$\theta_{t+1} = A\theta_t + w_t \tag{2}$$
$$\mathbf{y}_t = C\theta_t + v_t \tag{3}$$
$$w_t \sim \mathcal{N}(0, Q) \tag{4}$$
$$v_t \sim \mathcal{N}(0, R) \tag{5}$$

Let $\mathbf{y}_t$ be a vector of observations, in our case the sample means and sample standard deviations from TAC SCM logs. Let $\langle A, C, Q, R, \theta_0, V_0 \rangle$ be the parameters of the LDS. A is the state transition matrix, summarizing how the mean of the winning bid distribution changes between time steps. The matrix C is the observation matrix, or emission matrix, and governs how the latent variable affects our observations. The Q and R matrices govern how noisy state transitions and observations are respectively.

For a given LDS, we can perform inference (also known as filtering) on θ_t with a Kalman filter (Kalman 1960). For a given set of observations and known system dynamics (including the prior latent variable and covariance θ_0 and V_0), we can estimate the hidden state by the following set of equations.

$$\hat{\theta}_t = A\theta_{t-1} \tag{6}$$
$$\hat{V}_t = AV_tA^* + Q \tag{7}$$
$$K_t = \hat{V}_tC^*(C\hat{V}_tC^* + R)^{-1} \tag{8}$$
$$\theta_t = \hat{x}_t + K_t(y_t - C\hat{\theta}_t) \tag{9}$$
$$V_t = \hat{V}_t - K_tC\hat{V}_t \tag{10}$$

V_t represents the covariance of the state, $\hat{\theta}_t$ refers to the estimation of latent variables at time t given observations $y_{1:t-1}$, and $*$ is transposition (used instead of $\cdot^T$ to avoid notational confusion). Essentially, lines 6 and 7 represent prediction of the new state based on the old state and the dynamics, and lines 8, 9, and 10 represent the correction based on the actual observations proportional to the prediction error, $(y_t - C\hat{\theta}_t)$. With the ability to infer the hidden state variables, we can calculate the likelihood of a particular model

$$p(\theta_0)\prod_{t=1}^{T} p(\theta_t|\theta_{t-1})\prod_{t=1}^{T} p(y_t|\theta_t). \tag{11}$$

Because the likelihood for a long time series is usually small, we use the log likelihood as a measure of model 'goodness'.

The estimation of the system dynamics was performed using an EM algorithm (as was done, for example, by Ghahramani & Hinton (1996); EM is due to Dempster, Laird, & Rubin (1977)). The EM algorithm needs to be given an initial model, and this is one of the ways that we can make design choices about how we construct the initial model.

	Independent	Full
16	-35000	-50000
32	-34000	-94000
48	-34000	-144000

Table 1: Test Set Log Likelihoods

Model Validation

Once we have learned a model using EM, we need to validate it against game logs in order to be confident that we are accurately modeling the interactions that we seek to replace. The LDS models are validated by comparing both training and test set[1] log-likelihoods. Kalman filters are used to calculate the log likelihoods of several different LDS models. The first set of LDS models that assume that the processes for different SKUs are independent, with one, two or three hidden state variables per SKU. The resulting block-diagonal matrices, with slight perturbations, are used as initial points for a second learning phase that relaxes the independence assumption, leading to the models that this paper will refer to as 'full'. In the latter case the hidden state for the processes are 16, 32, and 48 dimensional, respectively.

We observed experimentally that dropping the SKU independence assumption leads to over-fitting, and using a hidden state with dimensionality higher than two does not increase test set likelihood (Table 1). Figure 4 shows the performance of a Kalman filter predicting the next day's winning price for a single the independent SKU model on a single game, and Figure 6 shows the same using the full SKU model. We see that the independent model leads to much more accurate predictions, where as the full model leads to predictions that are less responsive to local trends. When we zoom in on days 100 to 140 in Figure 5 and Figure 7, we can see that the winning price predictions for the full model has a much larger error than the independent model.

Therefore, for the remainder of this work, we use 16 independent LDSs with two-dimensional hidden states to generate the distribution of winning bids. Observe that this model assumes (contrary to intuition) that there is not a link between bidding behavior across different SKUs. The fact that this is our best model is probably due to the relatively small amount of training data that we used. We expect that if we had trained on a larger dataset, the full model would have outperformed the independent model.

The independent model has a mean winning price prediction error (the mean difference between the predicted mean

[1]The training set was drawn from the 2006 finals and semi-finals: games 5605, 6167, 5561, 8589, 5123, 6168, 5606, 8590, 5124, 5562, 5079, 6169, 8591, 5563, 5607, 6170, 5125, 5080, 6171, 8592, 5564, 5608, 5126, 6172, 8593, 5609 and 6173. The test set was also drawn from the 2006 finals: games 8594, 6174, 5610, 6175, 5611 and 5612. Learning a model from this training set took roughly 5 hours on a 100 CPU cluster

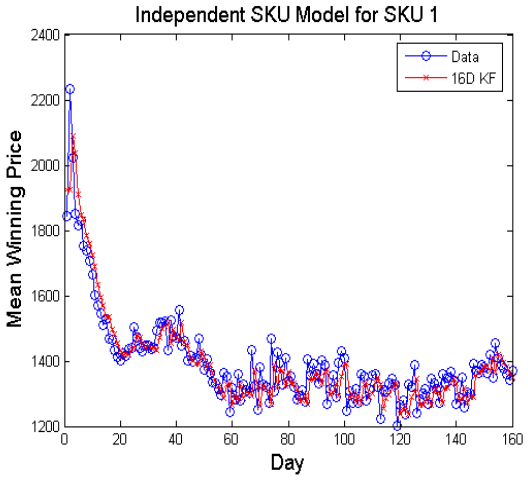

Figure 4: Independent SKU Mean Prediction

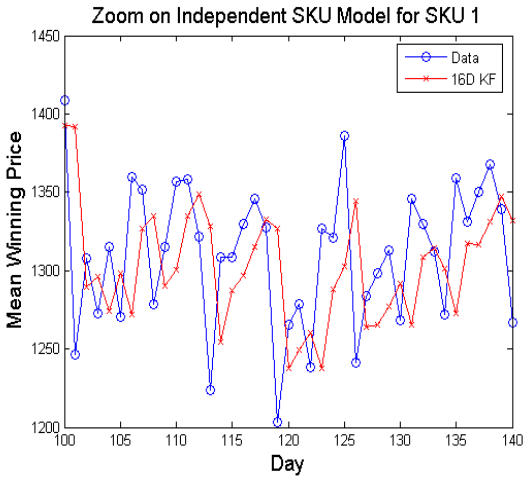

Figure 5: Zoom on Independent Mean Prediction from Figure 4

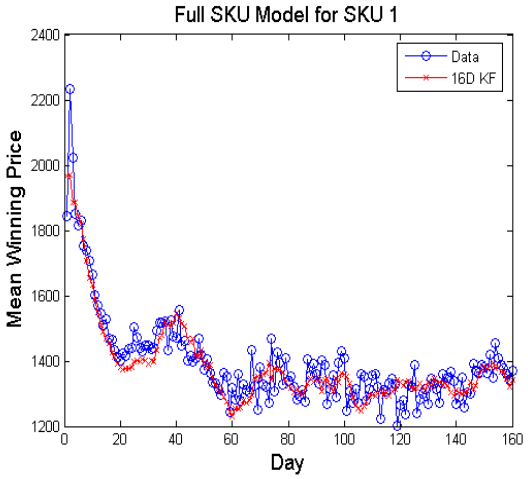

Figure 6: Mean Prediction

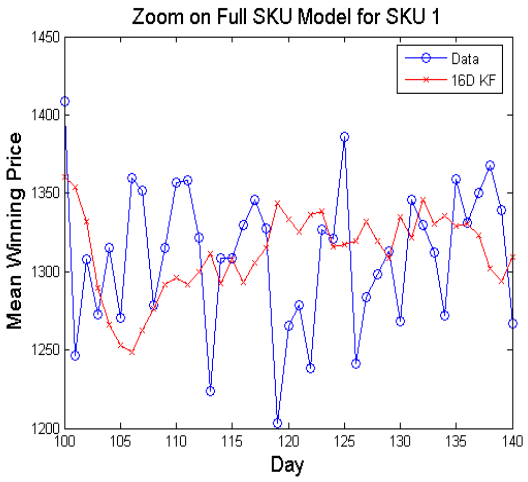

Figure 7: Zoom on Mean Prediction from Figure 6

winning price and the actual mean winning price) of 57.0 on the test set, and this is a error low enough that the model seems to be faithful to the underlying market data.

Component procurement

As described above, modeling the component market was not a focus of our work; nevertheless, we have included a simple model here. This model was motivated by structural similarity to TAC SCM (rather than functional similarity as determined by validating against log data). A future direction we intend to pursue is designing a time-series model for component requests that is representative of data drawn from TAC SCM game logs.

The available component capacity is simulated by a random walk similar to the one governing daily capacity in TAC SCM. To account for the effects of other agents, the nominal capacity is one sixth of the size of the nominal capacity in TAC (92 components per day as opposed to 550 components), and the size of the random steps in the random walk has been increased.

The supplier modules receive a set of component requests from the customer. Each day, a supplier module faces a 1-0 knapsack problem to determine which orders it should deliver, where it is trying to maximize the number of components delivered subject to component inventory constraints. If there is daily unused manufacturing capacity, then components will be manufactured and added to the inventory if there are current outstanding component orders, and insufficient inventory to satisfy them. This model does not favor late orders, and so large numbers of small orders will cause large orders to starve.

As our model is designed to investigate scheduling questions, it is unnecessary to model component prices; therefore, we have not done so.

Application: Comparing Scheduling Techniques

Of course, a model is only useful if it can be used to answer interesting experimental questions. Our main interest in building this model was to use it ourselves in future work, to investigate different existing optimization techniques for solving TAC's scheduling problems, and ultimately to build novel optimization algorithms tailored to TAC. We imagine that our model will also be useful to other researchers in the field for various other purposes. So far, we have only performed some initial experiments, and so we do not wish to make conclusive claims about the merits of different optimization approaches. Even so, we find the results that follow suggestive and thought-provoking; they certainly indicate the sorts of uses to which our model may be put.

In this section we will describe experiments that compared the following three algorithms, based on integer linear programming (ILP) and stochastic integer linear programming (SILP).

Myopic is an ILP, summarized in Figure 8, that maximizes the revenue that it can immediately obtain by deliveries for the current day. The myopic program solves sixteen 1-0 knapsack problems, one for each of the SKUs, where the

```
MAXIMIZE delivery profit
SUBJECT TO
Deliveries must all have sufficient
 inventory to satisfy them
```

Figure 8: Myopic pseudocode

```
MAXIMIZE ∑^Horizon delivery profit + expected
 RFQ profit - storage costs
SUBJECT TO
Each day, deliveries must all have
 sufficient inventory to satisfy them
Each day, manufacturing must require fewer
 cycles than the cycle cap
Each day, manufacturing must require fewer
 components than inventory
Each order and RFQ must be delivered once
 in the horizon, or defaulted on
```

Figure 9: SILP pseudocode

knapsack capacities are the SKU inventories and the items are orders. Because the optimization problem only considers deliveries that are possible on the current day, manufacturing decisions have no impact on the optimization. This means that the delivery problem for each of the 16 SKUs become independent. Production scheduling is decided greedily in a second phase by determining how many of each SKU will need to be manufactured to satisfy the current outstanding orders, and then trying to build as many of each as possible, subject to component and cycle constraints.

SILP optimizes profit for an n-day rolling horizon expected-profit problem, sketched in Figure 9, for delivery and production scheduling (Benisch *et al.* 2004b). SILPs are a well known approach to optimization when some of the parameters of the problem are random according to some known distribution. In our particular formulation the coefficients for any terms in the objective function involving uncertain quantities, namely RFQs, are replaced with their expectations. The constraints are written by assuming that any uncertain quantities are in fact certain.

SAA is a sample average approximation (SAA) SILP, summarized in Figure 10, that has k samples and an n-day horizon. Rather than taking expectations in the objective function or the constraints, a set of samples are taken from the probability distribution, and each has an associated set of decision variables to optimize over. In order to have a coherent plan for the current day, they all have the same set of decision variables for the first day. This approach has been well studied in operations research (see Shapiro (2001) for examples), and has also been implemented in the TAC SCM context (Benisch *et al.* 2004b).

Time limitations are an important practical constraint in many real problems, particularly those that involve trading in markets. (TAC SCM has a 15 second limit per day on

MAXIMIZE delivery profit for first day - storage costs for first day + $\sum^{Samples}\sum^{Horizon}$delivery profit + expected RFQ profit - storage costs

SUBJECT TO

Each day and sample, deliveries must all have sufficient inventory to satisfy them

Each day and sample, manufacturing must require fewer cycles than the cycle cap

Each day and sample, manufacturing must require fewer components than inventory

Each order and RFQ must be delivered once in the horizon, or defaulted on, in every sample

Figure 10: SAA pseudocode

calculations.) Thus, an important element of an algorithm's empirical performance is the quality of solutions it is able to find within a limited amount of time. We therefore studied this property of the three algorithms' anytime behavior, though we varied the time limit in some experiments below. All three algorithms were implemented in Java with CPLEX 10.1. CPLEX was told to emphasize feasibility over optimality, and used time as a termination condition (rather than relative or absolute optimality gap). Upon termination, the best feasible solution found during the ILP search was returned and used as a decision.

None of these algorithms deal with customer bidding or with component ordering. In order to be able to compare the three scheduling approaches in a more controlled manner, we used the same simple heuristic algorithms for bidding and ordering for each approach.

First we describe the heuristic algorithm used for customer bidding. We used 16 independent Kalman filters to track the SKU LDSs from the model. These filters were allowed to fully observe the sample mean and standard deviation of the winning bids from the previous day, and predict the bid for each SKU that achieves a winning probability of 0.90 (which might not be possible due to RFQ reserve prices). This policy was set to bid aggressively for RFQs determined to be valuable. The set of RFQs was sorted in ascending order according to late penalty, and then bids were greedily placed, for the previously determined amount, on each RFQ subject to a capacity constraint based on the daily manufacturing cycle capacity. Specifically, the bidder bids until the total aggregate quantity offered exceeds 250 PCs. This capacity constraint was established to avoid overcommitting to orders, and the particular threshold was determined by comparing the number of orders that the myopic algorithm defaulted on during a small set of tests.

For the component ordering algorithm we used a simple heuristic that orders a constant number of components each day. This amount is 75 for each CPU and 150 for the other parts. This orders components uniformly for each PC SKU, and is designed provide parts for 300 PCs, which is smaller than the RFQ bidding limit. In order to prevent inventories from bloating a maximum inventory size of 3000, or roughly ten days of production, is maintained.

The delivery schedule optimization for myopic only considers deliveries possible on the current day. The decision for customer bidding, therefore, has no effect on the current day's delivery scheduling decision. However, production scheduling is effected by the delivery schedule, as the number of order outstanding is dependent on which orders were already scheduled to be delivered. Figure 11 summarizes the precedence order, showing that the delivery schedule must be solidified before deciding on the production schedule, and that customer bidding and component ordering have no precedence restrictions.

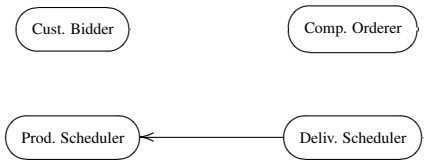

Figure 11: Subproblem Precedence Order for Myopic

In both SILP and SAA, production scheduling and delivery scheduling are decided upon in a unified optimization program, and so are solved simultaneously. However, because delivery and production schedules extend into the future, the customer bidding does impact production and delivery planning, and so customer bidding must precede the schedule optimization. The precedence ordering is summarized in Figure 12, and we can see that component ordering has no precedence restrictions.

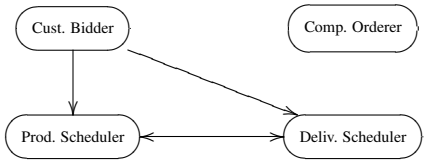

Figure 12: Subproblem Precedence Order for SAA and SILP

Results

The first set of experiments investigate the relationship between time and number of samples in three-day-horizon SAA and the quality of the plan returned. Experiments were run for each combination of using 2, 4, 6, and 8 samples, and given 10, 14, and 18 seconds for each sample. 100 simulation runs of 30 days were performed for each. In each case we used the same 100 scenarios; this blocked experiment design reduces variance. In all experiments, we measured the quality of a plan as the amount of profit that it made. Profit is defined as total revenue achieved minus late penalties, default penalties, and storage costs.

The results shown in Figure 13 and Table 2 indicate that solution quality, given a sample number, remained constant. This, along with the standard deviation data found in Table 3, suggests that time-truncated approximations are quite sensible, and that small increases of the time limit do not

Samples	10s	14s	18s
2	23.04	23.04	23.04
4	23.04	23.04	23.05
6	23.04	23.05	23.04
8	23.03	23.03	23.03

Table 2: Mean Profit, Number of Samples used by SAA vs. Time (Millions of Currency Units)

Samples	10s	14s	18s
2	1.76	1.76	1.76
4	1.75	1.75	1.75
6	1.74	1.75	1.75
8	1.74	1.74	1.74

Table 3: Standard Deviation, Number of Samples used by SAA vs. Time (Millions of Currency Units)

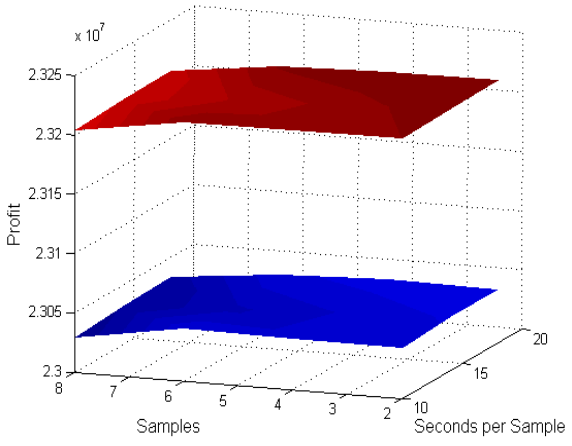

Figure 13: Bottom surface: Mean Profit
Top surface: Mean Profits plus one Standard Deviation

Algorithm	μ	σ
Myopic	1.5E7	2.2E6
SILP, 10s	2.3E7	1.8E7
SILP, 50s	2.3E7	1.8E7
SAA, 1 samples	2.3E7	1.8E6
SAA, 5 samples	2.3E7	1.7E6

Table 4: Profit for 3 day horizon planning

significantly impact solution quality. Table 2 also shows that the number of samples considered had no significant impact on solution quality. We feel confident in asserting that for the number of samples that can be reasonably solved in TAC SCM, the number of samples seems not to be a major factor impacting the solution quality for SAA.

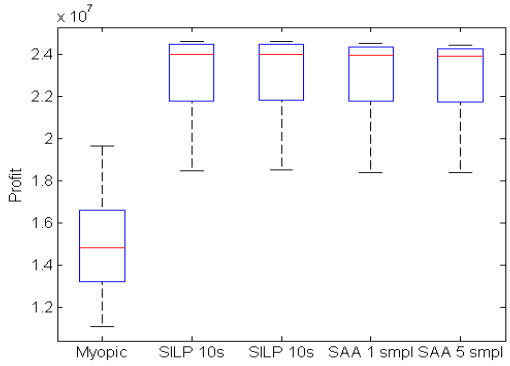

Figure 14: Algorithm Quality Comparison

Figure 14 shows the total profit achieved by our three different algorithms, including various time/sample limits for SILP and SAA (SAA was given 10 seconds per sample). While both the SAA and SILP approaches perform better than myopic, we see that there is no significant advantage to giving the SILP algorithm more time, or picking SAA or SILP over the other.

A more detailed breakdown of where each algorithm incurs loss, found in Table 5, shows that the myopic algorithm is completely unaffected by additional time, and this is because the myopic optimization is simple enough to be solved in under 10 seconds. The SILP displays minor fluctuations in mean late penalties, defaulting penalties, and storage costs, when more time is given but these are not significant. CPLEX logs indicate that good integer solutions (close to the objective value achieved in the linear program formed by relaxing integer constraints) are usually found before 10 seconds, and the remainder of the time is spend on relatively minor objective improvements. This accounts for why truncating the optimization process early yields a good approximation. SAA exhibits a more pronounced improvement when given more time. Table 5 also shows an increase in the mean late penalty and a decrease in both the storage costs and defaulting penalty. This suggests that, given more samples, orders that were previously decided to be undeliv-

Algorithm	Late Pen.	Default Pen.	Storage
Myopic	36000	948000	52000
SILP, 10s	3300	495000	24000
SILP, 50s	3300	495000	24000
SAA, 1 samples	2400	489000	24000
SAA, 5 samples	2800	486000	25000

Table 5: Mean Loss for 3 day horizon planning

erable in the one sample SAA program were instead delivered late in the 5 sample SAA program.

In the version of this work that was originally submitted for review, we used a different component ordering strategy that maintained a minimum inventory level, as opposed to the constant ordering policy used in this paper. This approach produced a counterintuitive result in which the myopic algorithm significantly outperformed both the SAA and SILP approaches. This demonstrates the importance of reasonable choices for the customer bidding and component ordering components. As mentioned above, in future work we intend to learn a component procurement model from data.

Conclusion

This paper introduced a novel model of the TAC SCM scheduling problem for use in empirical evaluations. This was done by replacing areas of multi-agent interaction with stochastic processes in order to give more control to the experimenter. The stochastic processes were learned from, and validated against, game logs taken from the 2006 TAC SCM finals using time-series machine learning techniques. This model was used to investigate ILP and SILP approaches in a time-constrained setting, investigating the trade off between optimization complexity and approximation quality. Some preliminary experimental results collected using our scheduling model showed that very early truncation of the optimization process usually resulted in good quality solutions. These results also indicated that considering many samples in an SAA approach in a time-limited setting had no effect on solution quality.

Future directions of work include designing and implementing an improved optimization algorithm for scheduling in TAC SCM. This will include testing alternate formulations of the objective function and constraints, and performing more detailed investigations into approximation quality in a time-constrained setting. In addition, we plan to incorporate component ordering and customer bidding into the optimization framework, and develop a more sophisticated and integrated approach. Another area of work is implementing a model for component procurement that can be validated by comparing its results to those found in game logs. Furthermore, we intend to do more work on the bidding model, investigating the idea that using priors on parameters during estimation for the LDSs could lead to more accurate models with higher likelihoods.

References

Benisch, M.; Greenwald, A.; Grypari, I.; Lederman, R.; Naroditskiy, V.; and Tschantz, M. 2004a. Botticelli: a supply chain management agent designed to optimize under uncertainty. *SIGecom Exch.* 4(3):29–37.

Benisch, M.; Greenwald, A.; Naroditskiy, V.; and Tschantz, M. C. 2004b. A stochastic programming approach to scheduling in TAC SCM. In *EC '04: Proceedings of the 5th ACM conference on Electronic commerce*, 152–159. New York, NY, USA: ACM Press.

Benisch, M.; Sardinha, A.; Andrews, J.; and Sadeh, N. 2006a. CMieux: adaptive strategies for competitive supply chain trading. *Proceedings of the 8th international conference on Electronic commerce: The new e-commerce: innovations for conquering current barriers, obstacles and limitations to conducting successful business on the internet* 47–58.

Benisch, M.; Sardinha, A.; Andrews, J.; and Sadeh, N. 2006b. CMieux: adaptive strategies for competitive supply chain trading. In *ICEC '06: Proceedings of the 8th international conference on Electronic commerce*, 47–58. New York, NY, USA: ACM Press.

Collins, J.; Arunachalam, R.; Sadeh, N.; Ericsson, J.; Finne, N.; and Janson, S. 2005. The supply chain management game for the 2006 trading agent competition. Technical Report CMU-ISRI-05-132, Carnegie Mellon University, Pittsburgh, PA 15213.

Dahlgren, E., and Wurman, P. R. 2004. PackaTAC: a conservative trading agent. *SIGecom Exch.* 4(3):38–45.

Dempster, A. P.; Laird, N. M.; and Rubin, D. B. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)* 39(1):1–38.

Eriksson, J.; Finne, N.; and Janson, S. 2006. Evolution of a supply chain management game for the trading agent competition. *AI Commun.* 19(1):1–12.

Ghahramani, Z., and Hinton, G. E. 1996. Parameter estimation for linear dynamical systems. Technical Report CRG-TR-96-2, University of Toronto.

Kalman, R. E. 1960. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering* 82(Series D):35–45.

Keller, P. W.; Duguay, F.-O.; and Precup, D. 2004. RedAgent-2003: An autonomous market-based supply-chain management agent. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, 1182–1189. Washington, DC, USA: IEEE Computer Society.

Ketter, W.; Kryzhnyaya, E.; Damer, S.; McMillen, C.; Agovic, A.; Collins, J.; and Gini, M. 2004. Analysis and design of supply-driven strategies in TAC-SCM. In *AAMAS04TADA*, 44–51.

Kiekintveld, C.; Wellman, M.; Singh, S.; Estelle, J.; Vorobeychik, Y.; Soni, V.; and Rudary, M. 2004. Distributed feedback control for decision making on supply chains.

Fourteenth International Conference on Automated Planning and Scheduling.

Pardoe, D., and Stone, P. An Autonomous Agent for Supply Chain Management.

Pardoe, D., and Stone, P. 2004. TacTex-03: A supply chain management agent. *SIGecom Exchanges: Special Issue on Trading Agent Design and Analysis* 4(3):19–28.

Pardoe, D., and Stone, P. 2006a. Predictive planning for supply chain management. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

Pardoe, D., and Stone, P. 2006b. TacTex-2005: A champion supply chain management agent. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, 1489–94.

Roweis, S., and Ghahramani, Z. 1997. A unifying review of linear Gaussian models. Technical report, University of Toronto, 6 King's College Road, Toronto M5S 3H5, Canada.

Shapiro, A. 2001. Monte Carlo simulation approach to stochastic programming. In *WSC '01: Proceedings of the 33nd conference on Winter simulation*, 428–431. Washington, DC, USA: IEEE Computer Society.

Vetsikas, I., and Selman, B. 2003. A principled study of the design tradeoffs for autonomous trading agents. *Proceedings of the second international joint conference on Autonomous agents and multiagent systems* 473–480.

Wellman, M. P.; Jordan, P. R.; Kiekintveld, C.; Miller, J.; and Reeves, D. M. 2006. Empirical game-theoretic analysis of the TAC market games. In *AMAS-06 Workshop on Game-Theoretic and Decision-Theoretic Agents*.