Multi-period Online Optimization in TAC SCM:
The Supplier Offer Acceptance Problem

Sarah Bell, Michael Benisch, Margaret Benthall, Amy Greenwald, and Michael Carl Tschantz
Department of Computer Science
Brown University, Box 1910
Providence, RI 02912
{sjb,mbenisch,mbenthal,amy,mtschant}@cs.brown.edu

Abstract

We formalize the supplier offer acceptance problem in TAC SCM as a multi-stage stochastic program. In addition, we suggest a heuristic for solving this problem using the rollout method, following one or two stage approximations of the multi-stage stochastic program as the base policy during rollouts. We also describe a heuristic based on the notion of marginal utility which is designed to scale our approach to problems with numerous decision variables.

1. Introduction

Many combinatorial optimization problems, such as job scheduling, facility location, and vehicle routing, involve making decisions under uncertainty. Not only are the effects of such decisions not known ahead of time, often they are not revealed until some time after decisions are made.

Increasingly, industrial applications are coming to rely on automated agents to make complex decisions. In the domain of supply chain management, for example, manufacturers are automating the process of procuring raw materials by moving these negotiations online [8]. In spite of any inherent uncertainty in decision-making (e.g., whether or not ordered materials will arrive on time), automated agents often make decisions based on (inaccurate) deterministic models, which serve to simplify the decision-making process in time-critical applications. An agent's failure to properly address uncertainty, however, limits its ability to avoid risk and detracts from its overall performance [3].

Furthermore, information that is not yet available may be revealed over discrete time periods. For example, a procurement agent may know that purchased materials will arrive within N days but not learn the actual arrival date until it occurs. In such cases, an agent faces the additional challenge of reasoning about future decisions without complete information about the effects of its current decisions. In this paper, we study strategies for making decisions that involve looking ahead to handle multiple periods of uncertainty. Typically, an optimization problem's complexity scales exponentially with the number of future periods of uncertainty; thus, we focus on approximation schemes.

A straightforward means of approximating a solution to an optimization problem with uncertainty extending over multiple periods of decision-making is as follows: formulate a simplified version of the control problem and solve this simplified problem repeatedly as each new set of information is revealed. To simplify the control problem, the (exponential) set of all possible future outcomes, or *scenarios*, is collapsed into a (relatively) small number of scenarios. One of the goals of this research is to show that this approach can be used to compute near-optimal solutions, given an exemplary choice of scenarios. We now describe three examples of this approach.

An extreme example of this approach is the *just-in-time* policy, in which the simplified control problem is deterministic: probability zero is attributed to all uncertain outcomes, which effectively ignores all future uncertainty. In manufacturing, just-in-time production dictates the ordering of components and the building of finished products only after orders for those products are in hand. Benisch *et al.* studied a manufacturing production problem in which the number of scenarios in the problem formulation was reduced by ignoring all uncertainty *beyond the second period*. In particular, in each period, the simplified control problem is a two stage stochastic program (i.e., two periods of lookahead).

In this paper, we suggest yet another approach, which is to formulate the simplified control problem as a discrete time Markov Decision Process (MDP), so that a decision-maker can consider the impact of its current decisions in future periods well beyond the second period. To solve this MDP, we generate future scenarios via Monte Carlo simulations, or *rollouts*, so that current decisions are based on the likelihood of future scenarios. In these simulations, we must rely on some policy to dictate future decisions: e.g., the just-in-time policy. Here, we use approximate solutions of the two stage stochastic program as rollout policies.

We apply these ideas to part of the component procurement problem in the Trading Agent Competition in Supply Chain Management (TAC SCM). Broadly speaking, component procurement refers to the securing of resources by a self-interested entity in an economic environment. In TAC SCM, a manufacturing agent acquires component parts by negotiating with suppliers. The agent orders materials, the supplier delivers them, and the agent transforms those components into finished products that it sells to its customers. The 2003 version of TAC SCM failed to emphasize the question of component procurement due to an artifact in the game design.[1] The rule changes instituted in 2004 address this issue, and as a result, we expect the procurement problem to be central to the upcoming competition. The procurement problem encompasses two subproblems: what requests to place with suppliers and which supplier offers to accept. We will focus on the latter, referred to herein as the Supplier Offer Acceptance Problem (SOAP).

2. The Supplier Offer Acceptance Problem

The Supplier Offer Acceptance Problem involves deciding which supplier offers to accept, given a fixed set of customer orders, taking into account the possibility that each component ordered may actually arrive any day on or after its contractual due date (which suggests ordering early) and the holding costs incurred for storing component inventory (which suggests ordering late). Supplier defaults pose a significant challenge to TAC SCM agents because they must balance the cost associated with ordering extra components against the risk of incurring late penalties if components are out of stock.

A supplier offer is characterized by a supplier, a component, a quantity, a price, and a due date. When deciding upon the set of supplier offers to accept, in addition to the tradeoff between supplier defaults and holding costs, a TAC SCM agent also factors into this decision the following:

- the set of customer orders: are there outstanding orders for high-margin products whose production relies on these components?
- the set of supplier orders: is sufficient stock of these components already on order?
- current component inventory: is there sufficient stock of these components already in inventory?
- current product inventory: can the order to which these components would be applied be filled from inventory?
- future supplier offers: might offers for the same component, quantity, and due date arrive tomorrow at cheaper prices from more reliable suppliers?

1 Some might argue that the 2003 version of TAC SCM emphasized *only* component procurement due to an artifact in the game design.

Supplier Offer Acceptance Problem
Inputs:
- Set of Customer Orders
- Set of Supplier Orders
- Set of Supplier Offers
- Supplier Offer Model
- Supplier Default Model
- Component Inventory
- Product Inventory
- Holding Costs

Outputs:
- Production Schedule: map from Cycles to Products
- Delivery Schedule: map from Products to Customer Orders
- Acceptance Policy: map from Supplier Offers to {YES,NO}

Objective:
- Maximize Revenue
- Minimize Penalties, Holding Costs, and Component Costs

Figure 1. SOAP

Ultimately, the output of SOAP is not only an acceptance policy—a decision about each supplier offer; rather, it also includes a production and delivery schedule, since the TAC SCM production and delivery optimization problems (see Benisch, *et al.* [2]) are nested inside SOAP. The objective is to compute an acceptance policy, together with production and delivery schedules, that maximize revenue, minimize penalties, and minimize holding and component costs. See Figure 1.

2.1. An MDP Formulation

Figure 2 illustrates SOAP formulated as a Markov decision process (MDP). The states in the MDP are characterized by customer orders, product and component inventory, past supplier orders, current supplier offers, and time. The actions in the MDP correspond to the different subsets of supplier offers that the agent may choose to accept and the different possible production and delivery schedules that it can execute. Each action is associated with various rewards and transitions, the latter of which some are deterministic (1, 2, and 3) and some are not (4 and 5).

1. build products scheduled for production, subtract the associated components from inventory, and add the products to inventory
2. ship products to customers, subtract them from inventory, and add revenue to the reward function
3. subtract holding costs from the reward function for all components and products in inventory
4. receive component orders from suppliers, add them to inventory, and subtract the associated component costs from the reward function
5. receive a new set of supplier offers

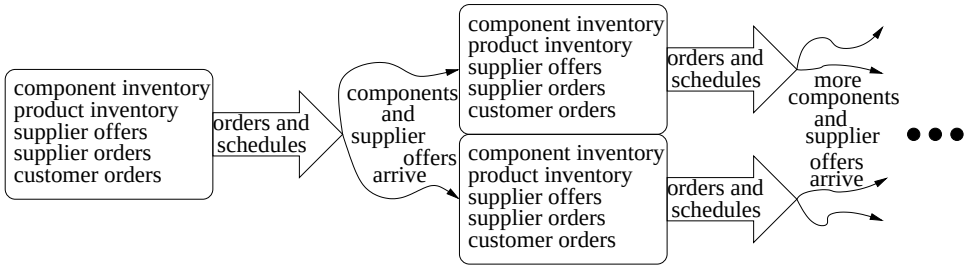

Figure 2. An MDP formulation of SOAP

2.2. An SP Formulation

In this section we describe SOAP formally, and in full generality, as a recursive, stochastic program (SP). The purpose of this description is to provide intuition into the problem structure, and to present a sense of its complexity. In the sections that follow, we describe tractable approximations of this formulation of SOAP that can be used to solve time-constrained variants of this problem with discrete and bounded inputs, as in TAC SCM.

At a high level, stochastic programs view problems in multiple stages. Decisions must be made in stage n before pertinent information about stage $n+1$ is revealed, but the objectives in stage $n+1$ are dependent on the decisions in stage n. Given stochastic information available about the outcomes in stage $n+1$, the goal is to find the stage n decisions that maximize the profits of stage n plus the expected profits of stage $n+1$.

Variables

$W_{l\mu}$ 1 if the μth supplier offer of day l is accepted; 0 otherwise
Y_{lj} quantity of product j scheduled for production on day l
Z_{li} 1 if order i is delivered on day l; 0 otherwise

Constants Re: Supplier Offers

$\mathcal{M}$ the set of all possible supplier offers
$M_{l\mu}$ the μth supplier offer of day l; note that the number of offers received on day l is $|M_l|$
$\vec{S}_{tl\mu}$ 1 if the μth supplier offer of day l is delivered on day t where $l < t$ and $\mu \leq |M_l|$; 0 otherwise
$q'_{l\mu k}$ quantity of component k in the μth supplier offer of day l
$p'_{l\mu}$ price of the μth supplier offer of day l
h_k holding cost for components of type k

Constants Re: Customer Orders

O number of customer orders
D total number of days in the game
E number of days before a late order is canceled
q_i quantity of products in order i
d_i due date of products in order i
p_i revenue for delivering order i on or before $d_i + E$
ρ_{li} penalty incurred if order i is delivered on day l
$\hat{\rho}_i$ penalty incurred if order i is canceled entirely
π_{li} revenue earned by delivering order i on day l

$$\pi_{li} = \begin{cases} q_i p_i & l \leq d_i \\ q_i p_i - \rho_{li} & d_i < l \leq d_i + E \\ 0 & l > d_i + E \end{cases}$$

a_k number of components of type k in initial inventory
b_j number of products of type j in initial inventory
c_j cycles expended to produce one product of type j
f_{ij} 1 if order i is for products of type j; 0 otherwise
g_{jk} 1 if products of type j are built using components of type k; 0 otherwise
C_l number of cycles on day l
J number of different product types
K number of different component types

Abbreviations

h'_j daily holding cost for products of type j; in TAC SCM 2004,

$$h'_j = \sum_{\{k \mid g_{jk}=1\}} h_k$$

γ_{kn} amount of component k in inventory on day n

$$\gamma_{kn} = a_k + \sum_{l=1}^{n} \sum_{\mu=1}^{|M_l|} \sum_{t=l}^{n-1} \vec{S}_{tl\mu} q'_{l\mu k} - \sum_{l=1}^{n} \sum_{\{j \mid g_{jk}=1\}} Y_{lj}$$

γ'_{jn} amount of product j in inventory on day n

$$\gamma'_{jn} = b_j + \sum_{l=1}^{n} \left(Y_{lj} - \sum_{\{i \mid f_{ij}=1\}} Z_{li} \right)$$

$\zeta_i(Z)$ 1 if order i is not delivered before being canceled; 0 otherwise

$$\zeta_i(Z) = 1 - \sum_{l=1}^{\min(d_i+E, D)} Z_{li}$$

Let $X : x$, where X is a matrix with n rows, denote the matrix whose first n rows are those of X and the last, the $(n+1)$th, row is x. Let $\langle\rangle$ denote the empty vector and $\langle\langle\rangle\rangle$ be a matrix with a single empty column. The notation $X \sqsubseteq Y$ means X has dimensions equal to Y's, with each entry of X either 1 or 0.

Equations Given the notation described above, the following mathematical program describes SOAP recursively:

$$Q(n, \vec{S}, M, W, Y, Z) = \max_{wyz} \sum_{i=1}^{O} (Z:z)_{ni}\pi_{ni} - \sum_{l=1}^{n}\sum_{\mu=1}^{|M_l|} \vec{S}_{nl\mu}p'_{l\mu} - \left(\sum_{k=1}^{K} h_k\gamma_{kn} + \sum_{j=1}^{J} h'_j\gamma'_{jn}\right) + \sum_{S \sqsubseteq M} \Pr(S|\vec{S}, M, W) \quad \times \int_{2^{\mathcal{M}}} \Pr(m) Q(n+1, \vec{S}:S, M:m, W:w, Y:y, Z:z)\, dm \quad (1)$$

$$Q(D+1, \vec{S}, M, W, Y, Z) = \sum_{i=1}^{O} \zeta_i(Z)\hat{\rho}_i \quad (2)$$

subject to the following constraints:

$$\sum_{t=1}^{n} (Z:z)_{ti} \leq 1 \qquad \forall i \in \{1..O\} \quad (3)$$

$$\sum_{j=1}^{J} c_j (Y:y)_{nj} \leq C_n \quad (4)$$

$$\sum_{\{i \,|\, f_{ij}=1\}} \sum_{t=1}^{n} q_i (Z:z)_{ti} \leq b_j + \sum_{t=1}^{n-1} (Y:y)_{tj} \quad \forall j \in \{1..J\} \quad (5)$$

$$\sum_{\{j \,|\, g_{jk}=1\}} \sum_{t=1}^{n} (Y:y)_{tj} \leq a_k + \sum_{l=1}^{n-1}\sum_{\mu=1}^{|M_l|}\sum_{t=l}^{n-1} \vec{S}_{tl\mu}q'_{l\mu k} \quad \forall k \in \{1..K\} \quad (6)$$

For $1 \leq n \leq D$, the objective (Equation 1) is to maximize revenue and minimize penalties, holding costs, and component costs not only in the current stage, but also the expected value of these quantities in future stages. For $n = D + 1$, the objective function (Equation 2) simply calculates the penalties for canceled orders.

The objective functions are subject to the following constraints: No customer order can be delivered more than once (Equation 3). The total number of production cycles used to produce all product types on day n cannot exceed the machine's capacity on day n (Equation 4). The total quantity of product j associated with orders delivered by day n cannot exceed the total inventory of product j produced by day $n-1$ plus any initial inventory (Equation 5). The total quantity of component k used through day n cannot exceed the total quantity of component k delivered by day $n-1$ plus any initial inventory of component k (Equation 6).

The goal on day n is to maximize $Q(n, \vec{S}, M, W, Y, Z)$ where $\vec{S}$, M, W, Y, and Z, which are all of length n, describe the agent's past actions and observations.

3. Approximation Techniques

In this section we present a class of approximation techniques that can be used to solve problems like SOAP with a reasonable amount of computation. The algorithms in this class vary according to their degrees of lookahead, from 0 days to $c \ll D$ days to C closer to D days. Regardless of the degree of lookahead, these techniques can be applied in an online fashion, so that solutions are repeatedly reoptimized as more and more uncertainty is resolved.

3.1. 0-Day Lookahead

The first approximation method we propose is straightforward: solve a deterministic simplification of the control problem in each period, by assigning probability 0 or probability 1 to any uncertain events. In the TAC SCM scheduling problem [2], assigning probability 0 to uncertain events yields to a just-in-time policy, where components are ordered from suppliers and manufactured into finished products only after customer orders are in hand.

A deterministic simplification of SOAP might ignore the possibility of supplier defaults as well as any future supplier offers: i.e., optimize with respect to the supplier offers in hand, assuming (i) the probability of default on any of these offers is zero and (ii) any orders on which the suppliers have already defaulted will arrive tomorrow with probability 1. This version of SOAP incorporates both optimistic and pessimistic viewpoints.

This problem is a special case of the stochastic program presented in Section 2.2:

- In every iteration beyond the first, set $\Pr(m)$ equal to one for $m = \langle\rangle$ and zero for all $m \neq \langle\rangle$; that is, we assume no supplier offers will arrive after today.
- In every iteration beyond the *first*, set $\Pr(S|\vec{S}, M, W)$ equal to zero for all S except for the one whose arrival dates match the due dates of all supplier offers, except those offers that were converted to orders and had due dates of today but were not delivered today. These untimely orders will have an arrival date of tomorrow in S. That is, we assume all components are delivered on time after today.

An algorithm that solves this deterministic version of SOAP obtains scheduling decisions far into the future without reasoning about any of future uncertainty.

3.2. 1-Day Lookahead

A more accurate solution technique is to formulate and (at least approximately) solve the two-stage stochastic program, with 1 day of lookahead, rather than 0, ignoring uncertainty about the future *beyond the second period*.

Benisch *et al.* tackle the production and delivery scheduling problems in TAC SCM with this technique, and show that it outperforms the aforementioned just-in-time policy [2].

In the SP that solves SOAP, the first stage represents the current day, when the information available to the agent includes customer orders, product and component inventory, past supplier orders, and current supplier offers. The second stage represents the next day, when the agent learns on which orders the suppliers have defaulted, and receives a new set of supplier offers. In the proposed simplification of SOAP, we assume there is zero probability of default on these second stage supplier offers, and that no supplier offers are due to arrive beyond stage two. Moreover, we assume that any orders on which the suppliers have already defaulted are certain to arrive by the day after tomorrow.

This problem is a special case of the stochastic program presented in Section 2.2:

- In every iteration beyond the *second*, set $\Pr(m)$ equal to one for $m = \langle\rangle$ and zero for all $m \neq \langle\rangle$, that is, assume no supplier offers will arrive after tomorrow.
- In every iteration beyond the *second*, set $\Pr(S|\vec{S}, M, W)$ equal to zero for all S except for the one whose arrival dates match the due dates of all supplier offers, except those offers that were converted to orders and had due dates of either today or tomorrow but were not delivered today or in the scenario describing tomorrow. These untimely orders will have an arrival date of the the day after tomorrow in S. That is, we assume all components are delivered on time, after tomorrow.

A solution to this simplification of SOAP would obtain long-term scheduling decisions by reasoning about only a very small part of the massive space of future scenarios.

One computational bottleneck to solving stochastic programs is the calculation of expected profits in the second stage. This calculation typically involves enumerating all possible scenarios (second stage outcomes); however, 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* (EV), which is to solve a deterministic variant of the problem assuming all stochastic inputs have deterministic values equal to their expected values (see, for example, Birge and Louveaux [6]). Shapiro, *et al.* [1, 11, 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 [12]. Benisch *et al.* [2] show that SAA outperforms EV in TAC SCM scheduling.

In summary, Benisch *et al.* [2] show that in TAC SCM production scheduling, SAA outperforms EV, which in turn outperforms just-in-time production. In particular, higher objective values can be achieved with 1 day of lookahead, rather than 0 days of lookahead. In this paper, we hypothesize that algorithms which rely on even more extensive lookahead can outperform SAA on SOAP. Intuitively, this hypothesis seems plausible; however, in practice, with such massive scenario spaces, approximate solutions could be so coarse that it would be better to outright ignore the future.

3.3. D-day Lookahead

Using the two-stage stochastic programming approach, we lack the ability to reason about uncertainty beyond one day (or, more generally, some small number of days, say $c \ll D$) into the future. Thus, we cannot consider the possibility that orders on which the supplier defaults might arrive an arbitrary number of days after their designated arrival dates. Towards this end, we propose an approximation scheme with C closer to D days of lookahead. Our technique combines rollout methods (see Bertsekas [5]) with marginal utility calculations (see, for example, [10]). Towards this end, we propose an approximation scheme with C closer to D days of lookahead.

The technique we propose is designed to approximate the optimal set of decisions today by *simulating* future scenarios. To picture this, recall the MDP formulation in Figure 2. In simpler problems, such as packet routing [7] and vehicle dispatching [9], where future uncertainty is independent of current actions, one can simply *sample* the future: e.g., assume packets or customers arrive according to some exogenous stochastic process, such as a Poisson process. But in SOAP, and MDPs in general, we cannot simulate future scenarios without relying on some policy that describes our actions at all states.

Given such a policy, say π, we can "rollout" each action a at each state s as follows: simulate a at s, its immediate rewards, the likely future states that ensue, and repeat, abiding by π at all future states. Now, we can choose the best action a^* among all actions: i.e., that which accrues the greatest long-term rewards. The policy improvement theorem [4] states that the policy π^*, where $\pi^*(s) = a^*$ *for all states* s, is an improvement over π in expectation.[2] We propose to rollout, and thereby improve the policy of, only those states that are visited. Via experimentation, we can determine the effectiveness of rollouts in SOAP.

2 The policy improvement theorem holds for suffi ciently many simulations as well, with high probability.

In SOAP (and in more general problems), there are a number of reasonable candidates for the base policy π in this rollout method. For example, we could choose π to be the policy that is obtained by solving the deterministic simplification of SOAP with 0-days of lookahead. Alternatively, and time-permitting, we could choose π to be a policy that is obtained with 1-day of lookahead, in which case we might approximate the solution to the two-stage stochastic program using the EV or SAA. The appropriate choice of rollout policy is an empirical question, which depends on time-critical factors, and to which the success of the overall procedure is ultimately tied.

The rollout method, which relies on repeated simulations, can yield an effective approximation (in small enough problems) at states with few actions. However, in SOAP, given N supplier offers, there are 2^N actions (even without taking into account scheduling decisions). Thus, we have developed an additional heuristic to solve instances of SOAP with large numbers of supplier offers.

Our heuristic is based on the idea of marginal utility. The marginal utility of a supplier offer is the difference between the utility of accepting and rejecting the offer. The marginal utility calculation can be approximated with a series of rollouts as described above: i.e., evaluate the action accept, evaluate the action reject, and compute the difference between the values of these two actions.

One simple heuristic is the following: include all offers with positive marginal utility. However, because several components are substitutes for one another, this heuristic can accept too many supplier offers. It is more accurate to compute the marginal utility of all supplier offers, accept the one with the highest marginal utility, and repeat until none of the remaining offers have positive marginal utility.

But this heuristic fails in SOAP because components are not only substitutes, they also complement one another. Starting from an empty inventory, no supplier offer would ever be accepted using this heuristic, since no supplier offer viewed in isolation yields positive marginal utility: all supplier offers incur costs, but revenues can only be accrued by selling products built using multiple components.

As a heuristic for SOAP, it is more effective to calculate the marginal *cost* of eliminating each supplier offer from an initial collection that includes the entire set of offers.[3] Offers without positive marginal utility are those which contain components enough of which cannot be assembled into products because complementary components or machine cycles are unavailable.

Finally, our heuristic: remove from the initial collection of supplier offers that with the highest marginal cost, recalculate the marginal costs of all the remaining supplier offers with respect to this new set of offers, and repeat until none of the remaining offers have positive marginal cost: i.e., all of the remaining offers have positive marginal utility. This procedure is detailed in Figure 3.

```
SELECTOFFERS(O)
 1  while O ≠ ∅
 2  do o* ← NIL
 3     m* ← ∞
 4     for o in O
 5     do m ← MarginalUtility(o)
 6        if m < m*
 7          then o* ← o
 8               m* ← m
 9     if m* > 0
10       then return O
11       else  O ← O \ {o*}
12  return ∅
```

Figure 3. O denotes the set of supplier offers, o^* denotes the worst offer evaluated thus far, and m^* denotes the marginal utility of o^*.

Using the proposed marginal utility heuristic, solving SOAP requires only $O(N^2)$ rollouts, rather than $O(2^N)$ rollouts, where $N = |O|$.

4. Conclusion

We have formalized the supplier offer acceptance problem (SOAP) in TAC SCM as a multi-stage stochastic program. In addition, we have suggested a heuristic for solving this problem using the rollout method, following one or two stage approximations of the multi-stage stochastic program as the base policy during rollouts. The main idea of the rollout method is look ahead into the future by simulating the long-term effects of each of the possible immediate actions, and to choose the best one. Not surprisingly, this method is only applicable to problems with small action sets, such as blackjack [13], in practice. Thus, we have also designed a heuristic based on the notion of marginal utility which should allow our approach to scale to problems with large numbers of 0/1 decision variables.

While the rollout method with lookahead requires a significant increase in computation over the stochastic programming formulations with little or no lookahead, enhanced with the marginal utility heuristic it lends itself well to parallelization. What is more, it is conceivable that it would be rather obvious whether to accept or to reject some fraction of the supplier offers, so that a preprocessing step could handle these easy cases, thereby limiting the num-

3 Whereas marginal utility is the difference between the value of accepting and rejecting a supplier offer, marginal cost is the negation of this quantity: i.e., the difference between the value of rejecting and accepting a supplier offer.

ber of (hard) decisions the rollout algorithm would make. With these improvements, our method could be effective in TAC SCM and other real-time decision-making environments. Indeed the proposed ideas are not limited to SOAP; we also plan to experiment with them in the framework of TAC Classic [14] in the near future.

In TAC SCM, additional work must be done to tie together SOAP with the bidding problem and the problem of deciding from which suppliers to request which supplies.

References

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

[2] BENISCH, M., GREENWALD, A., NARODITSKIY, V., AND TSCHANTZ, M. A stochastic programming approach to TAC SCM. In *ACM Electronic Commerce Conference* (New York, May 2004), pp. 152–160.

[3] BERRY, P. Uncertainty in scheduling: Probability, problem reduction, abstractions and the user, 1993.

[4] BERTSEKAS, D. P. Differential training of rollout policies. In *Proc. of the 35th Allerton Conference on Communication, Control, and Computing* (Allerton Park, Ill., October 1997).

[5] BERTSEKAS, D. P., AND CASTANON, D. A. Rollout algorithms for stochastic scheduling problems. *Journal of Heuristics 5*, 1 (April 1999), 89–108.

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

[7] BRODER, A., FRIEZE, A., AND UPFAL, E. A general approach to dynamic packet routing with bounded buffers, 2001.

[8] ESSMEYER, H. E-transaction enablers business models, trends and issues. Available at `http://citeseer.ist.psu.edu/401398.html`.

[9] GENDREAU, M., AND POTVIN, J.-Y. Dynamic vehicle routing and dispatching.

[10] GREENWALD, A., AND BOYAN, J. Bidding under uncertainty: Theory and experiments. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence* (July 2004), p. To Appear.

[11] 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.

[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] SUTTON, R., AND BARTO, A. *Reinforcement Learning: An Introduction*. MIT Press, Massachusetts, 1998.

[14] WELLMAN, M., WURMAN, P., O'MALLEY, K., BANGERA, R., LIN, S., REEVES, D., AND WALSH, W. A trading agent competition. *IEEE Internet Computing* (April 2001).