Scaling Up the Sample Average Approximation Method for Stochastic Optimization with Applications to Trading Agents

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

Abstract

Sample Average Approximation (SAA) is a well-known method for solving stochastic programs. Here, we attempt to scale up the SAA method to harder problems than those previously studied. We argue that to apply the SAA method effectively, there are three parameters to optimize: the number of evaluations, the number of scenarios, and the number of candidate solutions. We propose an experimental methodology for finding the optimal values of these parameters within fixed time and space constraints. We apply this methodology to solve two large-scale stochastic optimization problems that arise in the context of the annual Trading Agent Competition (see http://www.sics.se/tac). Both problems are expressed as integer programs and solved using CPLEX. Runtime increases linearly with the number of scenarios in one of the problems, and exponentially in the other. We find that, in the former problem, maximizing the number of scenarios yields the best solution, while in the latter problem, it is necessary to evaluate multiple candidate solutions to find the best solution, since increasing the number of scenarios becomes very expensive very quickly.

1 Introduction

Stochastic programming is a natural method for solving optimization problems under uncertainty. This approach considers a problem in two stages. Decisions are 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. A solution is sought that is "optimal in the expected sense."

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 (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* (e.g., [4]). Here, the available stochastic information is collapsed into a deterministic statistic (e.g., the mean), and a deterministic variant of the optimization problem is solved. But ignoring large portions of the available stochastic information has been shown to be detrimental to solution quality (e.g., [5]).

Shapiro, *et al.* [8; 1] proposed an alternative approximation technique called *sample average approximation* (SAA) to reduce the number of scenarios. They suggest using only a subset of the scenarios, randomly sampled according to the distribution over scenarios, to represent the full scenario space. An important theoretical justification for this method is that as the number of scenarios sampled increases, the solution to the approximate problem converges to an optimal solution in the expected sense. Indeed, the convergence rate is exponentially fast.

In this paper, we attempt to scale up the SAA method to harder problems than those previously studied. We tackle two stochastic optimization problems that arise naturally in the context of the annual Trading Agent Competition (TAC) (see http://www.sics.se/tac). The first problem is a bidding problem inspired by the TAC Classic game; the second problem is a scheduling problem inspired by the TAC Supply Chain Management game. Nested inside each of these problems is an NP-hard optimization problem.

We find that runtime increases linearly with the number of scenarios in the bidding problem, whereas it increases exponentially in the scheduling problem. Indeed, within reasonable time constraints we cannot re-

liably solve the scheduling problem with a sample size of more than 8 out of 2^{40} scenarios. Hence, the theory that justifies the SAA method is inapplicable here. Instead, we experiment with optimizing the tradeoff between the number of scenarios and the number of policies (i.e., candidate solutions). Multiple policies can be generated by sampling from the set of scenarios multiple times, and solving the ensuing approximation problems. In the bidding problem, maximizing the number of scenarios yields the best solution, while in the scheduling problem, it is necessary to evaluate multiple policies to find the best solution, since increasing the number of scenarios becomes very expensive very quickly.

2 Sample Average Approximation

Following [8], we are interested in solving optimization problems of the form:

$$v^* = \max_{x \in \mathcal{X}} g(x) \tag{1}$$

where

$$g(x) = \mathbb{E}_Q G(x, Y) \tag{2}$$

Here, $x \in \mathcal{X}$ is a vector of decision variables that take on values in the finite set $\mathcal{X}$; Y is a vector of discrete random variables with joint probability distribution Q and domain values $\omega \in \Omega$; $G(x, \omega)$ is a real-valued function of x and ω; $\mathbb{E}_Q G(x, Y)$ is the expected value of G at x: i.e.,

$$\mathbb{E}_Q G(x, Y) = \sum_{\omega \in \Omega} Q(\omega) G(x, \omega) \tag{3}$$

Each realization ω of Y, drawn according the distribution Q, is called a *scenario*. Note that the number of scenarios is exponential in the number of random variables. (For example, if there are n random variables, all of which can take on binary values, there are 2^n scenarios.) Consequently, it is prohibitively expensive to compute $\mathbb{E}_Q G(x, Y)$. On the other hand, it is relatively less expensive to compute $G(x, \omega)$.[1]

The *sample average approximation* (SAA) method is a numerical means of approximating a solution to Equation 1 via Monte Carlo simulation. The main idea is simple: (i) generate a set of scenarios $\mathcal{S}$ of size S by sampling values of Y from the distribution Q, and (ii) approximate the expected value in Equation 3, based on only the scenarios in $\mathcal{S}$. More specifically, generate a set of scenarios $\mathcal{S} = \{y_1, \ldots, y_S\}$, and solve the optimization problem:

$$\hat{v}_{\mathcal{S}} = \max_{x \in \mathcal{X}} \hat{g}_{\mathcal{S}}(x) \tag{4}$$

where for a set of scenarios $\mathcal{N} = \{a_1, \ldots, a_N\}$,

$$\hat{g}_{\mathcal{N}}(x) = \frac{1}{N} \sum_{i=1}^{N} G(x, a_i) \tag{5}$$

[1] In fact, in our application domains, the exact computation of $G(x, \omega)$ is NP-hard.

The SAA method, as applied in this paper, is shown in Algorithm 1. First, P candidate solutions, or policies, $\tilde{x}_1, \ldots, \tilde{x}_P$, are generated, by solving Equation 4 for P distinct scenario sets $\mathcal{S}_1, \ldots, \mathcal{S}_P$, each of size S. Second, these candidate policies are evaluated by computing $\hat{g}_{\mathcal{E}}(\tilde{x}_1), \ldots, \hat{g}_{\mathcal{E}}(\tilde{x}_P)$ for a fixed set of scenarios $\mathcal{E} = \{z_1, \ldots, z_E\}$ of size E. Third, the best candidate policy according to the evaluation phase is output.

Algorithm 1 Sample Average Approximation (E, S, P)

```
bestval ⇐ −∞
sample a set E of scenarios from Q
for all j = 1 to P do
    sample a set S of scenarios from Q
    x̂ ⇐ arg max_{x∈X} ĝ_S(x)
    calculate ĝ_E(x̂)
    if ĝ_E(x̂) > bestval then
        bestval ⇐ ĝ_E(x̂)
        bestsol ⇐ x̂
    end if
end for
return bestsol
```

Note the following:

- for all $\mathcal{X}' \subseteq \mathcal{X}$, $\hat{v}_{\mathcal{S}} \geq \max_{x \in \mathcal{X}'} \hat{g}_{\mathcal{S}}(x)$
- for all $x \in \mathcal{X}$, $\hat{g}_{\mathcal{N}}(x)$ is an unbiased estimator of $g(x)$: i.e., $\mathbb{E}\left[\hat{g}_{\mathcal{N}}(x)\right] = g(x)$

It follows that the estimator $\hat{v}_{\mathcal{S}}$ is a statistical upper bound on v^*:

$$\mathbb{E}\left[\hat{v}_{\mathcal{S}}\right] \geq \mathbb{E}\left[\max_{x \in \mathcal{X}^*} \hat{g}_{\mathcal{S}}(x)\right] \geq \max_{x \in \mathcal{X}^*} \mathbb{E}\left[\hat{g}_{\mathcal{S}}(x)\right] = \max_{x \in \mathcal{X}^*} g(x) = v^* \tag{6}$$

where $\mathcal{X}^* \subseteq \mathcal{X}$ denotes the set of optimal solutions to Equation 4. On the other hand, for any feasible solution $\tilde{x} \in \mathcal{X}$, $g(\tilde{x})$ is a lower bound on v^*:

$$v^* = \max_{x \in \mathcal{X}} g(x) \geq g(\tilde{x}) \tag{7}$$

The difference $\mathbb{E}\left[\hat{v}_{\mathcal{S}}\right] - \mathbb{E}\left[\hat{g}_{\mathcal{E}}(\tilde{x})\right]$ is an estimator of the optimality gap.

Using the theory of large deviations, [8] establish the following result: as $S \to \infty$, the probability that a solution to Equation 4 is an optimal solution to Equation 1 converges to 1 exponentially fast. In their implementation of SAA, the basic procedure is repeated P times, for increasing values of S (and E), until the estimate of the optimality gap is sufficiently small. However, as $S \to \infty$, the time and space complexity required to solve Equation 4 may increase superlinearly, making it impossible—for all practical purposes—to allow S to suitably increase, unless the desired optimality gap is sufficiently *large*. Rather than fix the number of policies P, and vary only S and E, we apply the SAA method by

searching for optimal settings of E, S, and P, within the time and space constraints of the problem.

2.1 Overview

In solving stochastic optimization problems, not only can increasing the number of scenarios S increase the quality of the solution, but increasing the number of policies P can also increase the quality of the solution (indeed, the probability that the $(P+1)$st policy outperforms the first P policies is $1/(P+1)$). Moreover, in Algorithm 1, the time complexity is linear in P and the space complexity is constant in P, whereas increasing S could increase time and space complexity in unpredictable ways because of the intricacies of solving Equation 4. In this paper, we study two stochastic optimization problems in the TAC domain, analyzing the tradeoff between solution quality and complexity, as a function of E, S, and P.

3 Sample Problem Domains

3.1 Stochastic Bidding Problem

We solve a stochastic bidding problem inspired by the classic Trading Agent Competition game. In this game, each agent's objective is to maximize the profits earned by delivering combinations of 28 types of goods, on offer in 28 auctions, to eight clients. The stochastic formulation of the bidding problem can be described informally as follows: given a stochastic model of auction clearing prices, and given an agent's clients' preferences, find an optimal set of bids (and asks). It is natural to formulate this problem as a two-stage stochastic program: in the first stage an agent makes its bidding decisions; in the second stage, when the agent is informed of its winnings, it allocates those winnings to its clients. This second stage problem, TAC Classic allocation—find the utility-maximizing set of packages that an agent can assemble from its winnings, given its clients preferences—is a winner determination problem (NP-hard). See Appendix B.

A TAC Classic Stochastic Pricing Model. The bidding problem takes as input a stochastic model of clearing prices. Our stochastic pricing model is inspired by the *expected* competitive equilibrium pricing model developed for Michigan's TAC Classic Agent, `Walverine` [6]. To compute the *expected* competitive equilibrium price of a good, the tatonnement process is run once making use of the expected market wide demand, which is calculated based on the expected values of the opposing agents' clients' preferences. In a *stochastic* modeler, rather than run the tatonnement process only once using expected market wide demand, the process is run multiple times, each time drawing a random selection of the opposing agents' clients' preferences from the probability distributions given in the game's specification. We ran this procedure 2000 times to generate 2000 scenarios, which comprise our model of stochasticity in our experiments.

3.2 Stochastic Scheduling Problem

We solve a stochastic scheduling problem inspired by the Trading Agent Competition in Supply Chain Management. The problem is to schedule production of computers at a factory with limited capacity when demand for different types of computers is stochastic. There are 16 computer types that can be produced. Production of one unit of a type requires a certain amount of production capacity. The computers are produced to meet customer demand that comes in the form of possible orders. A possible order is specified by a probability, price, computer type, and quantity. The probability says how likely a possible order is to become a real order. Real orders can be satisfied by delivering the quantity of the type of computer requested in the order. Revenue is earned by satisfying real orders. The objective is to distribute the available capacity among computer types in a way that maximizes expected revenue.

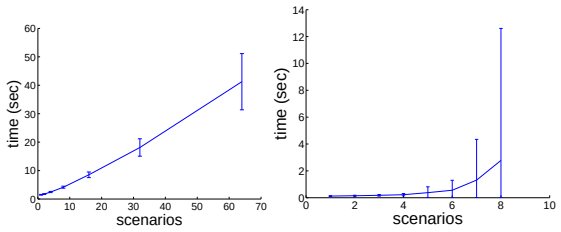
We express the problem as a two-stage stochastic program. In the first stage, the agent makes its production decisions. In the second stage, the agent is told which of the possible orders become real orders, and chooses the profit maximizing subset of orders to deliver using the computers produced in the first stage. The second stage problem is a 0-1 knapsack problem (NP-hard). See appendix C).

A problem instance is a set of 40 possible orders. The possible orders are generated based on uniform distributions of probabilities, computer types, and quantities. A possible order's price is inversely related to the probability of it becoming a real order. To make the problem interesting, we generate a range of quantities such that the expected demand is twice the production capacity.

4 Experimental Setup

The goal of our experiments was to optimize the parameters E, S, and P in Algorithm 1 in our two sample problems, given time (and space) constraints. At a high level, we ran multiple trials (that is, we solved multiple instances of each problem), with multiple settings of the parameters, and we averaged our results across trials to find the best parameter settings within reasonable time and space constraints. More specifically, our tests were conducted as follows: for each setting of the parameters (E, S, P), and for each trial $t = 1, \ldots, T$, we (i) generated a problem instance; (ii) ran Algorithm 1 to find the best policy; (iii) evaluated that policy using $E' > E$ scenarios.[2] For the bidding problem we set $T = 50$ and $E' = 250$, and we let $E \in \{1, 2, 4, 8, 16, 32, 64, 128\}$

[2] Anecdotally, we report that we found that more evaluations were necessary to accurately estimate the value of a policy, but that fewer evaluations were sufficient to rank policies.



Bidding Problem: $P = 1, E = 32$ Scheduling Problem: $P = 1, E = 100$

Figure 1: Time as a function of the number of scenarios.

and $S, P \in \{1, 2, 4, 8, 16, 32, 64\}$. For the scheduling problem, we set $T = 100$ and $E' = 5000$, and we let $E \in \{10, 100, 500, 1000\}$, $S \in \{1, \ldots, 8\}$, and $P \in \{1, \ldots, 375\}$. All experiments were run on an AMD Athlon 64 3000+ with 1GB of RAM. We used CPLEX 9.0 to solve the integer programs, solving to within .01% of optimality with the default settings.

5 Experimental Results

Our first set of experiments was intended to approximate the time complexity of the bidding problem and the scheduling problem. The graphs in Figure 1 depict time (in seconds) as a function of S, for fixed values of E and P. Perhaps surprisingly, in the bidding problem, this relationship is approximately linear. In the scheduling problem, however, time increases rapidly as the number of scenarios increases. The error bars in these graphs plot one standard deviation from the mean. The variance in these experiments is enormous, because we average results across multiple trials: i.e., problem instances of varying degrees of difficulty.

Figure 2 depicts the agent's reward in the bidding problem and its revenue in the scheduling problem, as a function of S and P. Here, we observe that increasing the number of scenarios from 1 to 64 in the bidding problem leads to an increase in reward from roughly 3300 to roughly 4200, on average. In the scheduling problem, increasing the number of policies, say from 1 to 20, leads to a substantial increase in revenue, but at 32 policies, revenue seems to stabilize. Of course, increasing the number of scenarios also leads to an increase in revenue, but in practice this may not be feasible. Indeed, revenue is still increasing between 7 and 8 scenarios, but we cannot solve this problem reliably with 9 scenarios.

Based on these observations, we postulate the following: In problems, like the bidding problem, where time complexity is an approximately linear function of S, a near optimal setting of the parameters can be obtained by setting S to the maximum value possible within the time and space constraints of the application, and then perhaps increasing P if time and space permit. On the other hand, in problems like the scheduling problem, where time complexity is a superlinear function of S, it is necessary to search the space of parameter settings, within the time and space constraints of the application, to find the best quality solution. Indeed, the next experiments provides evidence to support these hypotheses.

5.1 Optimizing E, S, and P

We tested only 243 of the 392 possible settings of the bidding parameters, and 282 of the 12000 possible settings of the scheduling parameters because of artificially-imposed time constraints. In the bidding problem, we ran all combinations of the parameter settings for which we predicted the running time would be less than 3 minutes; (in TAC Classic games, hotel bidding proceeds in 1 minute rounds). in the scheduling problem, we ran all combinations of the parameter settings for which we predicted the running time would be less than 25 seconds (in TAC SCM games, each day lasts 15 seconds) We made these predictions as follows. For all values of S, we recorded the average time to solve Equation 4; call this $\alpha(S)$. For all values of E, we recorded the average time to solve Equation 5; call this $\beta(E)$. For each combination of E, S, and P, we predicted the running time would be $P(\alpha(S) + \beta(E))$. Note that because of these time constraints, we never exhausted the space limitations of our machines.

The results of our search for optimal parameter settings are depicted in Table 1. Each table was generated as follows. First, we sorted our data according to time. Then, we traversed this sorted list. Whenever the value of the solution improved, we output new parameter settings and the corresponding values. The first column in each table lists time in seconds. For the bidding problem, the second column lists rewards; for scheduling, the second and third columns list the (estimated) lower and upper bounds on revenue. Note that the revenue earned in the scheduling problem is precisely the estimated lower bound. The last three columns list the settings of E, S, and P that generated these values.

In the bidding problem, the best performing parameter settings are those with 64 scenarios and only 2 policies, increasing the number of evaluations as time permits. It appears that the increase in solution quality obtained by increasing the number of scenarios exceeded any increase that could have been obtained by increasing the number of policies. Given that time increases linearly with the number of scenarios in the bidding problem, but that solution quality converges to the optimal exponentially fast, this outcome is not surprising.

In the scheduling problem, the best performing parameter settings are those with 4–6 scenarios and multiple policies. When the number of scenarios was 7 or 8, the policy generation process was much slower than it was when there were fewer scenarios. Indeed, only one "algorithm" with 7 scenarios appears in the table, and no

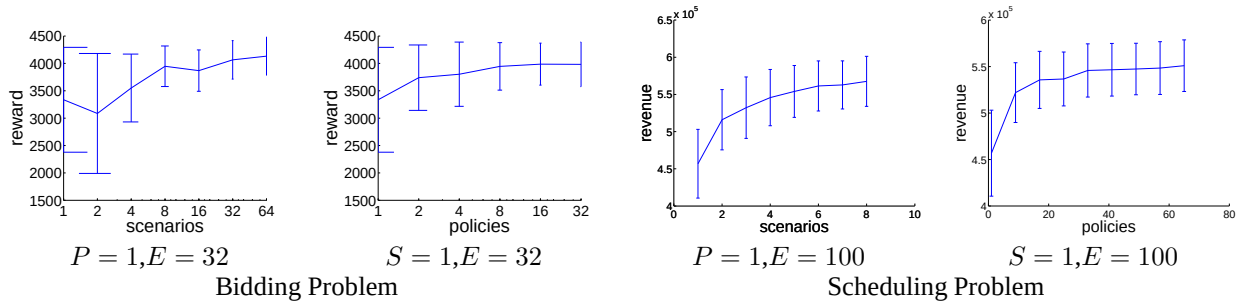


Figure 2: Reward and revenue as a function of the number of scenarios and policies.

"algorithms" with 8 scenarios appear at all. We conclude that in the scheduling problem, choosing the best among multiple policies yields better solutions than increasing the number of scenarios to a point at which only a few policies can be evaluated.

6 Related Work

The sample average approximation method has been applied to a variety of stochastic optimization problems of varying degrees of difficulty. It appears to us, however, that none were quite so hard as the problems attacked here. For example, [8] experiment with the stochastic knapsack problem, with only 20 first stage binary decision variables, and [1] experiment with a stochastic optimization problem with 2 continuous first stage variables and 4 integer second stage variables. In our formulation of the bidding problem, there are 70400 first stage variables and 201216 second stage variables; in our formulation of the scheduling problem, there are 16 first stage integer variables and 320 second stage binary variables. Moreover, nested inside each of our problems are NP-hard second stage decision problems.

To our knowledge, Algorithm 1 is the best-known method for solving stochastic optimization problems. Thus, rather than compare sample average approximation with other techniques, we conducted an in-depth study of the SAA method itself. We mention in passing that we did measure the performance of the expected value method in the scheduling problem, where all random variables are replaced with their means. Much like the results reported in [3], this method performs far worse than the SAA method, even with only one scenario and one policy. The revenue earned was only 311,064 (as compared to 468,310).

7 Conclusion

In this paper, we conducted an empirical study of the SAA method for solving stochastic optimization problems. We discovered that runtime can increase exponentially with the number of scenarios, so that even if solution quality increases exponentially with the number of scenarios, it may be advantageous to conduct a policy search, because runtime increases only linearly with the number of policies. In particular, we applied what we call the ESP methodology, which is a means of searching for optimal settings of the parameters of the SAA algorithm, within the time and space constraints of an application. Based on our observations, we formulated a quick test to determine whether maximizing S and then P, within the constraints of an application is sufficient—namely, do time and space usage grow only linearly with S? Often, however, we expect time and space usage to grow exponentially with S, in which case it is necessary to search the space of parameter settings to optimize the tradeoffs between increasing S or P.

The stochastic bidding and scheduling problems studied in this paper were inspired by the Trading Agent Competition. In both TAC Classic and TAC SCM, our agent's architecture [2; 7] is comprised of two main modules: a "modeler" and a "decider." Our modelers build stochastic models of their environments, which necessitates that our deciders solve stochastic optimization problems. In other words, decision-making under uncertainty, particularly the two problems defined and analyzed in this paper, are fundamental to our agents' designs. Generalizing from our experience in TAC Classic and TAC SCM, we expect that related stochastic optimization problems are fundamental to the design of trading agents for domains outside the scope of the Trading Agent Competition.

A Stochastic Programming Formulations

A.1 TAC Classic Bidding Problem

Index Sets

$a \in A$ indexes the set of auctions. $c \in C$ indexes the set of clients. $p \in P$ indexes the set of bid prices. $q \in Q_a$ indexes the set of goods in auction a. $s \in S$ indexes the set of scenarios. $t \in T$ indexes the set of packages.

Constants

$\mathcal{G}_{at}$ is an integer constant indicating how many goods from auction a are contained in a package t. $\mathcal{B}_{aqs}$ is an integer constant indicating the closing buy price of the

Time	Reward	E	S	P
1.47	3318	64	1	1
1.48	3456	128	1	1
1.48	3502	2	1	1
1.49	3548	16	1	1
2.45	3550	32	4	1
2.45	3577	2	4	1
3.38	3695	2	1	2
3.89	3705	4	1	2
4.12	3912	128	8	1
4.16	3947	32	8	1
8.43	3967	2	16	1
10.55	4014	8	8	2
16.75	4043	32	8	2
17.95	4045	64	32	1
18.09	4064	1	32	1
18.12	4065	32	32	1
33.50	4077	32	8	4
38.52	4099	16	32	2
41.26	4132	32	64	1
82.20	4134	1	64	2
84.81	4136	32	32	4
85.99	4141	16	64	2
88.81	4142	32	64	2
115.27	4146	128	64	2

Bidding Problem

Time	Lower	Upper	E	S	P
0.03	468310	754507	10	1	1
0.04	517559	688963	10	2	1
0.07	535059	657833	10	3	1
0.11	548218	647722	10	4	1
0.29	550930	639010	10	5	1
0.38	554046	637546	100	5	1
0.40	559796	630666	10	6	1
0.56	561418	628053	100	6	1
1.31	562798	624235	100	7	1
1.36	567807	661136	100	3	8
1.58	575676	647877	100	4	7
2.84	577965	646174	100	4	13
3.06	579369	638006	100	5	9
4.13	581433	636296	100	5	13
5.47	582306	629457	100	6	9
5.65	582504	635982	100	5	17
7.30	583621	637376	100	5	21
8.50	583998	630956	100	6	13
9.44	584043	646170	100	4	43
10.00	584287	636188	100	5	29
10.92	585094	645841	100	4	49
12.63	585543	636626	100	5	37

Scheduling Problem

Table 1: Optimal settings of the parameters as a function of time, with corresponding values.

qth good of auction a in scenario s. $\mathcal{Z}_{aqs}$ is an integer constant indicating the closing sell price of the qth good of auction a in scenario s. $\mathcal{U}_{ct}$ is an integer constant indicating the utility gained for client c having package t.

Decision Variables

$B = \{\beta_{apq}\}$ is a set of boolean variables indicating whether to bid price p for the qth good in auction a. $Z = \{\zeta_{apq}\}$ is a set of boolean variables indicating whether to ask price p for the qth good in auction a. $\Gamma = \{\gamma_{cst}\}$ is a set of boolean variables indicating whether client c gets package t in scenario s.

Objective Function

$$\max_{B,Z,\Gamma} \sum_{S} \Pr(s) \left(\overbrace{\left(\sum_{C,T} \mathcal{U}_{ct}\gamma_{cst} \right)}^{utility} - \overbrace{\left(\sum_{A,Q_a,p>\mathcal{B}_{aqs}} \mathcal{B}_{aqs}\beta_{aqp} \right)}^{cost} + \overbrace{\left(\sum_{A,Q_a,p<\mathcal{Z}_{aqs}} \mathcal{Z}_{aqs}\zeta_{aqp} \right)}^{revenue} \right) \quad (8)$$

The objective function (Equation 8) maximizes utility minus cost plus revenue.

Constraints

$$\sum_{T} \gamma_{cst} \leq 1 \quad \forall c \in C, s \in S \quad (9)$$

$$\sum_{P} B_{apq} \leq 1 \quad \forall a \in A, q \in Q_a \quad (10)$$

$$\sum_{P} Z_{apq} \leq 1 \quad \forall a \in A, q \in Q_a \quad (11)$$

$$\sum_{C,T} \gamma_{cst}\mathcal{G}_{at} \leq \left(\sum_{Q_a,p>\mathcal{B}_{aqs}} \beta_{apq} \right) - \left(\sum_{Q_a,p<\mathcal{Z}_{aqs}} \zeta_{aqs} \right) \quad \forall a \in A, s \in S \quad (12)$$

Equation 9 limits each client to one package in each scenario. Equation 10 prevents the agent from placing more than one bid for the same (auction, quantity) pair. Equation 11 prevents the agent from placing more than one ask for the same (auction, quantity) pair. Equation 12 prevents the agent from allocating goods that it does not own (number allocated $\leq$ number bought $-$ number sold).

A.2 TAC SCM Production Scheduling Problem

Index Sets

$i \in I$ indexes the set of RFQs. $j \in J$ indexes the set of SKUs. $n \in N$ indexes the set of scenarios.

Constants

C is the production capacity. SKU s_i, quantity q_i, price p_i, and penalty ρ_i. c_j is the number of cycles required to produce SKU j. ω_{in} is set to 1 if RFQ i becomes an order in scenario n.

Decision Variables
v_j is an integer variable indicating the amount of SKU j to produce. z_{in} is a boolean variable that is set to 1 if we fill order i in scenario n.

Objective Function

$$\max_z \sum_n \sum_i (p_i + \rho_i) z_{in} \tag{13}$$

The objective function (Equation 13) maximizes the revenue of allocating assembled computers to orders across scenarios. Note that the true value is obtained by subtracting the quantity $\sum_i \rho_i$ from the given value; but changing the objective function by a constant does not affect the solution.

Constraints
Stage 1:

$$\sum_j c_j v_j \leq C \tag{14}$$
$$v_j \in \mathbb{Z}_+, \quad \forall j$$

Stage 2:

$$z_{in} \leq \omega_{in} \qquad \forall i, n \tag{15}$$
$$\sum_{\{i \mid s_i = j\}} q_i z_{in} \leq v_j, \qquad \forall j, n \tag{16}$$
$$z_{in} \in \{0, 1\}, \qquad \forall i, n$$

Constraint 14 makes sure that we do not produce beyond our production capacity. Constraint 15 does not let us satisfy an RFQ for which we did not place a winning bid. Constraint 16 makes sure that we do not allocate to orders more computers than we assembled in the first stage.

B TAC Classic Allocation is NP-Hard

Generally speaking, *allocation* is equivalent to the winner determination problem in combinatorial auctions, an NP-hard problem [9]. The role of the agent is analogous to that of an auctioneer. In WD, an auctioneer is given a set of combinatorial bids in the form of package–price pairs, and seeks an allocation of goods to bids so as to maximize his profits, subject to the constraint that he cannot allocate more goods than he owns. Analogously, in *allocation*, the agent is given a set of "bids" in the form of package–utility pairs, and seeks an allocation of goods to bids so as to maximize the sum of its clients' utilities, subject to the constraint that it cannot allocate more goods than it owns.

In TAC Classic, agents are subject to the further constraint that only one package can be allocated to each client. TAC Classic allocation and allocation are polynomial-time reducible to one another. An arbitrary instance of TAC Classic allocation reduces to an instance of allocation by adding to every bid from client c a dummy good c of which the agent owns exactly one, so that every feasible allocation assigns at most one package per client. An arbitrary instance of allocation reduces to TAC Classic allocation by associating a unique client with each bid.

C Delivery Scheduling is NP-Hard

The NP-hardness of the Delivery Scheduling Problem (DSP) [2] can be proven by showing a polynomial-time reduction to DSP from the 0-1 Knapsack Maximization Problem (0-1KMP), a problem known to be NP-hard. In 0-1KMP, a set of items of varying weights and values is given, along with a knapsack of a fixed capacity (i.e., a weight limit). An optimal solution is a subset of the items that maximizes the total value, with a total weight that does not exceed the knapsack's capacity.

To perform the reduction take an arbitrary instance of 0-1KMP and create an instance of DSP in which only one SKU is present, say, SKU 2. Let each item in the 0-1KMP instance correspond to one order for SKU 2 such that the order's quantity is equal to the item's weight and the order's price is equal to the item's value. Let the given inventory of SKU 2 be equal to the given knapsack's capacity. Assume every order is due tomorrow, there are no late penalties, and there is nothing scheduled for production. The optimal solution to this instance of DSP is the most profitable subset of orders for which the total quantity does not exceed the inventory of SKU 2. This subset of orders corresponds to a subset of items that solves 0-1KMP.

References

[1] S. Ahmed and A. Shapiro. The sample average approximation method for stochastic programs with integer recourse. *Submitted for publication*, 2002.

[2] M. Benisch, A. Greenwald, I. Grypari, R. Lederman, V. Naroditskiy, and M. Tschantz. Botticelli: A supply chain management agent. In *Third International Conference on Autonomous Agents and Multiagent Systems*, volume 3, pages 1174–1181, July 2004.

[3] M. Benisch, A. Greenwald, V. Naroditskiy, and M. Tschantz. A stochastic programming approach to TAC SCM. In *Fifth ACM Conference on Electronic Commerce*, pages 152–160, May 2004.

[4] John Birge and Francois Louveaux. *Introduction to Stochastic Programming*. Springer, New York, 1997.

[5] H. Chang, R. Givan, and E. Chong. On-line Scheduling Via Sampling. In *Artificial Intelligence Planning and Scheduling (AIPS)*, pages 62–71, Breckenridge, Colorado, 2000.

[6] S.F. Cheng, E. Leung, K.M. Lochner, K.O'Malley, D.M. Reeves, L.J. Schvartzman, and M.P. Wellman. Walverine: A Walrasian trading agent. *Decision Support Systems*, page To Appear, 2004.

[7] A. Greenwald and J. Boyan. Bidding algorithms for simultaneous auctions: A case study. In *Proceedings of Third ACM Conference on Electronic Commerce*, 115-124 2001.

[8] A.J. Kleywegt, A. Shapiro, and T. Homem de Mello. The sample average approximation method for stochastic discrete optimization. *SIAM Journal of Optimization*, 12:479–502, 2001.

[9] M.H. Rothkopf, A. Pekeč, and R.M. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44(8), 1998.