The LP/POMDP Marriage: Optimization with Imperfect Information

Kirk A. Yost,[1] **Alan R. Washburn**[2]

[1] *U.S. Air Force*

[2] *Operations Research Department, Naval Postgraduate School, Monterey, California 93943-5000*

Received May 1999; revised June 2000; accepted 6 June 2000

Abstract: A new technique for solving large-scale allocation problems with partially observable states and constrained action and observation resources is introduced. The technique uses a master linear program (LP) to determine allocations among a set of control policies, and uses partially observable Markov decision processes (POMDPs) to determine improving policies using dual prices from the master LP. An application is made to a military problem where aircraft attack targets in a sequence of stages, with information acquired in one stage being used to plan attacks in the next. © 2000 John Wiley & Sons, Inc. Naval Research Logistics 47: 607–619, 2000

Keywords: linear program; Markov decision process; information

1. INTRODUCTION

We introduce a technique for solving optimization problems where constrained resources must be sequentially allocated to control a large number of objects, each of which has a finite number of states. The resources have random effects on the objects, and the outcomes are not known with certainty; that is, the object states are only partially observable. Our work is motivated by a military problem where weapons and sensors must be gradually allocated to targets, an example of which is given in Section 5. Another application might be in medicine, where treatments with random effects are made on the basis of error-prone tests by equipment whose availability is sometimes severely constrained.

While the literature has tended to treat such situations using decision-theoretic approaches (e.g., Marshall and Oliver [13]), these methods do not explicitly handle constrained resources, nor do they allow such resources to be shared among different objects. A problem of this class might be cast as a stochastic program with recourse (e.g., Klein Haneveld [9]) where one engages a three-step process: (i) making an initial resource allocation; (ii) observing the outcomes once the realization of some random variables generates a scenario; and (iii) reacting with a second, follow-on decision involving resource adjustments or reallocation. However, the structure of our problem makes this approach difficult for two reasons. First, the outcomes in step (ii) cannot be determined with certainty in our problem, so the recourse mechanism in step (iii) is not viable.

Correspondence to: A.R. Washburn

Second, the random variables of step (ii) depend on the allocations of step (i) in our problem, whereas they are assumed to be independent of those allocations in a recourse formulation. These difficulties appear to be insurmountable, so we have not pursued this line of thinking further.

Previous work in this area includes Jonsbraten [10], who studied a well-drilling application with partial observability using a decision tree with constraints. Jonsbraten, Wets, and Woodruff [11] developed a stochastic programming model where the random elements do depend on the actions taken; they note that the literature in this area is very sparse, and cite only one paper on the Markovian case (Pflug [16]).

Castanon [4] considers the allocation of one aircraft sensor type to find targets of multiple types. He combines POMDPs with Lagrange multipliers to meet the single sensor constraint, noting that finding a way to generalize the technique for larger problems is a subject for further research.

The LP/POMDP technique introduced by Yost [19] can be thought of as a logical descendent of Castanon's work where the Lagrange multipliers are the dual variables of a linear program (LP), rather than exogenously determined. The LP/POMDP technique employs a master linear program (LP) to assign costs to resources through its dual variables, and a set of POMDPs using these resource costs to find improving columns for the master LP. The resulting algorithm is a column-generation method that can be used to solve certain large allocation problems that would otherwise be intractable.

We begin by describing the case where there is only one object, even though the technique will perform at its best on problems where there are many.

2. CORE PROBLEM FOR A SINGLE OBJECT

Let I be a set of m constrained resources, and let R and $Y \equiv (Y_1, Y_2, \ldots, Y_m)$ be $m+1$ random variables, with R being a reward and Y_i being the amount of resources of type i consumed. The joint distribution of R and Y depends on which policy is chosen. We make no assumptions about R and Y other than that each has a known expected value for every policy. The object is to choose a policy within some finite feasible set S, possibly at random, to maximize $E(R)$ subject to $E(Y) \leq b$, where b is an appropriately dimensioned vector of resources and $E(\quad)$ is the expected value operator. Denote $E(R)$ and $E(Y_i)$ when using policy s as ER_s and EY_{si}, respectively. With this notation, we can express our problem as a linear program LP(S) where the variable x_s represents the probability of choosing policy s:

$$\begin{aligned}\text{LP(S)}: \quad & \text{maximize} \quad \sum_{s \in S} ER_s\, x_s \\ & \text{subject to} \quad \sum_{s \in S} EY_{si}\, x_s \leq b_i \qquad \forall i \in I, \\ & \sum_{s \in S} x_s = 1, \quad \text{and} \\ & x_s \geq 0 \qquad \forall s \in S.\end{aligned}$$

The sums in the objective function and resource constraints are $E(R)$ and $E(Y_i)$, respectively, by the conditional expectation theorem. We assume that $b \geq 0$ and that there is a null policy in S that consumes no resources, so LP(S) has a feasible solution. The value of LP(S) will in general be greater than the value of the restricted problem in which the variables are required to be 0 or 1; that is, it is significant that the decision maker is permitted to choose a strategy at random as long as the *expected* consumption of resources meets the constraints. Given that EY_{si} already represents an expected value, solving LP(S) as a 0/1 integer program would not in general have the effect of guaranteeing that resources never be exceeded, and that is not our intention at this point anyway. We will comment further on this in Section 7.

The set S is assumed to be so large that enumeration of all possible policies is out of the question. In such cases, the best approach is usually to find an ϵ-optimal algorithm. This requires upper and lower bounds on the maximized objective function value $v(S)$. A lower bound is readily available by solving LP(T), where T is any subset of S. To obtain an upper bound, consider the following Lagrangian relaxation LPU$(S;\lambda)$, where $\lambda = (\lambda_1, \dots, \lambda_m)$:

$$\begin{aligned} \text{LPU}(S;\lambda): \quad & \text{maximize} \quad \sum_{s\in S} ER_s\, x_s + \sum_{i\in I} \lambda_i \left(b_i - \sum_{s\in S} EY_{si}\, x_s \right) \\ & \text{subject to} \quad \sum_{s\in S} x_s = 1, \text{ and} \\ & x_s \geq 0 \qquad \forall s \in S. \end{aligned}$$

Let $u(S;\lambda)$ be the optimal value of LPU$(S;\lambda)$. As long as $\lambda \geq 0$, $u(S;\lambda) \geq v(S)$, with equality in the case where λ is equal to the optimal dual variables of the resource constraints of LP(S) (e.g., Parker and Rardin [15]). Furthermore, since LPU$(S;\lambda)$ contains one simple constraint, the optimal value is given by

$$u(S;\lambda) = \sum_{i\in I} \lambda_i b_i + \max_{s\in S} \left\{ ER_s - \sum_{i\in I} \lambda_i\, EY_{si} \right\}. \tag{1}$$

For any given set of resource prices λ, finding the upper bound $u(S;\lambda)$ is a matter of solving the maximization part of (1), which may be possible (see Section 3) in spite of the large size of S. If some policy that solves (1) is already in T, then LP(S) and LP(T) have the same solution; otherwise, some solution of (1) not already in T can be added to T and the process repeated. More formally, we can write a dynamic column generation algorithm for solving LP(S) (Gilmore and Gomory [7, 8]):

```
 Input:  An initial subset $T^1$ of $S$ and a nonnegative tolerance $\epsilon$.
Output:  An $\epsilon$-optimal solution of LP$(S)$.
Step 1:  Set $k = 1$.
Step 2:  Solve LP$(T^k)$ for $v(T^k)$ and a set of dual prices $\lambda^k$.
Step 3:  Solve (1) for $u(S;\lambda^k)$ and an optimizing policy $s$.
Step 4:  If $u(S;\lambda^k) - v(T^k) \leq \epsilon$, stop.
Step 5:  Let $T^{k+1} = T^k \cup \{s\}$.
Step 6:  Add 1 to $k$ and go to Step 2.
```

If $s \in T^k$, then $u(S;\lambda^k) = u(T^k;\lambda^k) = v(T^k)$ and the algorithm stops in Step 4. The union operation in Step 5 is therefore always nontrivial, so the algorithm will stop after at most $|S|$ steps even if $\epsilon = 0$. However, there is little comfort in this finite convergence property on account of the assumed large size of S. The main utility of the algorithm is in computing nearly optimal solutions after a number of steps that is much smaller than $|S|$. Regardless of the stopping criterion, $v(T^k)$ will always be a lower bound on $v(S)$ because T^k is a subset of S.

3. APPLYING PARTIALLY OBSERVABLE MARKOV DECISION PROCESSES (POMDPS)

The algorithm described in Section 2 makes no reference to time or information. It is a generic column-generation scheme that might be useful in any circumstance where LP(S) is hard to solve while (1) is not. However, the characteristics of the kind of problems described in the Introduction—sequential decisions and partial observability—make the set of strategies available in a Markov Decision Process an attractive choice for S. Since the state is only partially observable, the POMDP generalization is especially attractive.

A POMDP involves a sequence of decisions $U_0, \ldots, U_{N-1}$ and a sequence of observations $Z_0, \ldots, Z_{N-1}$, with a decision-making policy being *admissible* if U_k depends only on the observable history $I_k \equiv (Z_0, Z_1, \ldots, Z_k, U_0, U_1, \ldots, U_{k-1})$, for $k = 0, \ldots, N-1$ (e.g., Bertsekas [1]). The observation Z_k depends stochastically on the true state X_k of the process at time k, but the true state X_k is known only to the extent that it can be deduced from I_k. The set of admissible policies S is typically enormous, but POMDPs are solvable as a practical matter on account of the Markovian nature of state evolution and the way observations are generated. The crucial result is that the state probability distribution given I_k is a sufficient statistic for the decision U_k [1], which has the effect of converting the POMDP into an ordinary Markov Decision Process over a much larger but observable (via Bayes Theorem) state space.

The specification of a POMDP, as originally formalized by Smallwood and Sondik [18], requires a scalar reward function $G(k, U, X)$ that is interpreted to be the net reward at time k for making decision U when the true state of the process is X. In our case $G(k, U, X)$ will be the difference between a reward $r(k, U, X)$ and the cost of the resources consumed; specifically $G(k, U, X) \equiv r(k, U, X) - \lambda y(k, U, X)$, where $y(k, U, X)$ is an m-vector representing expected resource consumption and λ is an m-vector of prices for those resources. The expected total net profit for a given policy s is then the accumulation of all net profits:

$$p(s; \lambda) = E\left[r(X_N) + \sum_{k=0}^{N-1} G(k, U_k, X_k)\right], \tag{2}$$

where $r(X_N)$ represents a terminal reward that depends only on the final state. The random variables U_k and X_k have a joint distribution that depends on s. The optimal policy is the policy in S that maximizes the expected total net profit, which maximum we denote $p(S; \lambda)$. In spite of the large size of S, efficient techniques are available for calculating $p(S; \lambda)$.

The accumulated net profit can also be written as the difference $R - \lambda Y$, where R is the accumulated reward [including $r(X_N)$], and Y is the vector of accumulated resource consumption. Recalling the definitions of ER_s and EY_{si} from Section 1, the POMDP solution of (2) is equivalent to

$$p(S; \lambda) = \max_{s \in S} \left\{ ER_s - \sum_{i=1}^{m} \lambda_i \, EY_{si} \right\}. \tag{3}$$

This provides the link between the POMDP and the master LP. Solving the POMDP is equivalent to solving LPU($S; \lambda$), and the solution provides an upper bound on $v(S)$ through the relation

$$u(S;\lambda) = \sum_{i=1}^{m} \lambda_i b_i + p(S;\lambda). \tag{4}$$

Furthermore, the maximizing policy also defines a new column to be included in the master LP.

Let w^k be the dual value of the equality constraint in the solution of LP(T^k). The dual of LP(T^k) includes the constraints $w^k \geq ER_s - \sum_{i=1}^{m} EY_{si}\, \lambda_i^k \forall s$. From (3), w^k must therefore be $p(T^k;\lambda^k)$, and consequently Step 3 of the algorithm could be written as

3. If $p(S;\lambda^k) - w^k \leq \epsilon$, stop.

In other words, if the POMDP cannot find a new column with a reduced cost greater than ϵ, the algorithm terminates with an ϵ-optimal solution.

The LP/POMDP algorithm as so far specified can be regarded as a method for including resources in a POMDP that are constrained, rather than priced. This is a potentially important extension from the viewpoint of POMDP applications. The POMDP literature, with the notable exception of [4], assumes a known cost structure. A common situation in reality (as will be demonstrated in our example problem) is that there is no marginal cost structure for actions or observations; there are merely availability constraints.

Lovejoy [12] and Cassandra [2] provide good surveys of the POMDP literature. The POMDP problem has been shown to be PSPACE-complete by Papadimitirou and Tsitsiklis [14], and even the best of the current algorithms has difficulty when there are many states. By the basic theory of linear programming, there exists an optimal solution for LP(S) that involves at most $m+1$ basic variables. We assume m is much smaller than $|S|$, so it should come as no surprise that most of the computation time involved in solving LP(S) is spent in the column generation phase (POMDP calculations), rather than in the master LP. The LP/POMDP algorithm must solve a sequence of POMDPs, so the size of the state space will in practice have to be kept small.

4. SCALING UP TO MULTIPLE OBJECTS

Consider next a problem where there is a set J of classes of identical objects, with N_j being the number of objects in class j. Each class j has a separate set S_j of feasible policies, with each policy affecting only a single object in j. Let $S = \cup_{j\in J} S_j$. Also let the decision variable x_{sj} be the expected number of objects in class j to which policy s is applied. Note that the decision variables are not necessarily integer-valued and that they do not completely determine a method for choosing a collection of policies, since many joint distributions may have the same marginal expected values. The nonoptimized degrees of freedom are useful, as will be explained in Section 7, but only the averages x_{sj} are involved in the optimization. Finally, assume that both rewards and resource consumption are additive over classes. Then, since sums and expected values commute, the expanded master LP for maximizing the total expected reward is (with dual variables in parentheses):

$$\begin{aligned} \text{LP2}(S):\quad & \text{maximize} \quad \sum_{j\in J} \sum_{s\in S_j} ER_{sj} x_{sj} \\ & \text{subject to} \quad \sum_{j\in J} \sum_{s\in S_j} EY_{sji} x_{sj} \leq b_i \qquad \forall i \in I, \qquad (\lambda_i) \\ & \sum_{s\in S_j} x_{sj} = N_j \qquad \forall j \in J, \quad \text{and} \qquad (w_j) \end{aligned}$$

$$x_{sj} \geq 0 \qquad \forall j \in J, \quad s \in S_j$$

ER_{sj} is the expected reward when policy s is applied to an object of type j, so by the conditional expectation theorem the objective function can be interpreted as "average total reward." Similarly, since EY_{sji} is the average consumption of resource type i when policy s is applied to an object of type j, the expression bounded by b_i is still the average amount of resource type i used in applying policies to all objects.

The upper bound LP is now:

$$\text{LPU2}(S;\lambda): \quad \text{maximize} \quad \sum_{j\in J}\sum_{s\in S_j} ER_{sj}\,x_{sj} + \sum_{i\in I}\lambda_i\left(b_i - \sum_{j\in J}\sum_{s\in S_j} EY_{sji}\,x_{sj}\right)$$

$$\text{subject to} \quad \sum_{s\in S_j} x_{sj} = N_j \qquad \forall j \in J, \quad \text{and}$$

$$x_{sj} \geq 0 \qquad \forall j \in J, \quad s \in S_j$$

Note that the upper bound LP decomposes into $|J|$ separate optimizations. The algorithm of Section 1 still applies, except that a separate POMDP must be solved for each object class. Let

$$p_j(S_j;\lambda) = \max_{s\in S_j}\left\{ER_{sj} - \sum_{i=1}^{m}\lambda_i\,EY_{sji}\right\}. \tag{5}$$

The overall upper bound is given by

$$u(S;\lambda) = \sum_{i=1}^{m}\lambda_i b_i + \sum_{j\in J} N_j p_j(S_j;\lambda). \tag{6}$$

The multiple-object decomposition algorithm, with natural generalizations of the notation used in Section 1 (notably T^{kj} is the subset of S_j used in step k, T^k is the set of all such subsets, and w^{kj} is the jth component of w^k), is as follows:

```
**Input:** Initial nonempty subsets $T^{1j}$ of $S_j, j \in J$, and a nonnegative tolerance $\epsilon$.
**Output:** An $\epsilon$-optimal solution of LP2($S$).
**Step 1:** Set $k = 1$.
**Step 2:** Solve LP2($T^k$) for $v(T^k)$ and a set of dual prices $\lambda^k, w^k$.
**Step 3:** For all $j \in J$, solve POMDP (4) for $p_j(S_j;\lambda^k)$ and an optimizing policy $s^j$.
**Step 4:** If $u(S;\lambda^k) - v(T^k) \leq \epsilon$, stop.
**Step 5:** For all $j \in J$, if $p_j(S^j;\lambda^k) > w^{kj}$, let $T^{j,k+1} = T^{j,k} \cup \{s^j\}$; else let $T^{j,k+1} = T^{j,k}$.
**Step 6:** Add 1 to $k$ and go to Step 2.
```

The multiple-object algorithm stops in Step 4 if none of the object classes generates a new policy in Step 3, so it is finite. A total of $|J|$ POMDPs must now be solved in Step 3, so the computational burden increases when multiple object classes are considered. However, increasing N_j does not complicate the structure of the master LP, and does not affect the POMDP computations at all, so the algorithm will be at its best on problems where there are few classes but many objects per class.

As an aside, we note that the POMDPs could be solved in parallel, since the resource prices decouple the computations.

5. THE BDA PROBLEM

The task of assigning weapons and sensors to targets is a crucial one in the military, and is becoming more so. Modern armaments are lethal and can be employed from very long distances, but they are also expensive and in short supply. The long standoff distances require examination by sensors—Bomb Damage Assessment or BDA—to determine the outcome of the attack. The tradeoffs between sensors and weapons have been difficult to analyze and are of great current interest (see, e.g., Scott [17]). The LP/POMDP methodology can be useful for making these tradeoffs because it permits the incorporation of constraints on sensors as well as other more traditional combat resources.

The objects to be controlled are the targets. They all have two states (live or dead) and are divided into $|J|$ classes (bridge, tank column, command bunker, and so on). Each target in class j has a value C_j, and the terminal reward $r(X_N)$ is C_j if the terminal state is dead, otherwise 0. There are three types of resource. Two of these are aircraft and sensors, availabilities of which are constrained in each stage. The third resource is various kinds of weapons that can be dropped or launched from the aircraft, with total weapon consumption over all stages being constrained. The overall goal is to maximize the expected total value of targets destroyed.

Each action is either a *strike*, a *look*, or a *pause*. A strike is any permissible combination of aircraft, weapon, and target. Let A be the set of available strikes, with p_a denoting the known probability that strike a kills the intended target. Each strike consumes one aircraft sortie and at least one weapon of some kind. Similarly let L be the set of looks, each of which is the assignment of a sensor that will assess its target to be either "live" or "dead." Each look l is assumed to have known error probabilities $\alpha_l \equiv \Pr$ (assess target as live | target is actually dead) and $\beta_l \equiv \Pr$ (assess target as dead | target is actually live). The pause action consumes no resource and has no effect on the target.

Each target is assumed to be initially live, and the sole purpose of a look at stage k is to provide information about the target's state X_k at that time. We assume here that information gained from an observation can be used in the next stage, but note that the POMDP formulation can accommodate sensors with larger response delays. This is an important generality, since timeliness of information is one of its important attributes.

The BDA problem would be considerably simplified if the error probabilities α_l and β_l were taken to be 0, but modeling verity demands that these parameters be permitted to be positive. Figure 1 shows two views of an aircraft shelter struck by bombs in the Gulf War. The shelter appears to be heavily damaged, but in fact is still functional. On the other hand, penetrating bombs sometimes produce only a small hole on the outside of a shelter while completely destroying the interior and its contents. One must allow for the possibility of both type 1 and type 2 errors.

To show the hopelessness of trying to enumerate all possible policies in a problem of this sort, suppose there is only one target, suppose that $|A| = |L| = 1$, and let N_k be the number of possible policies in a k-stage problem. Then $N_1 = 3$, since one can either *strike*, *look*, or *pause* with one stage left, and N_k satisfies the recursion $N_{k+1} = 2N_k + N_k^2$ for $k > 0$. The coefficient of 2 is because the *strike* and *pause* options can each be followed by N_k k-stage policies, and the power

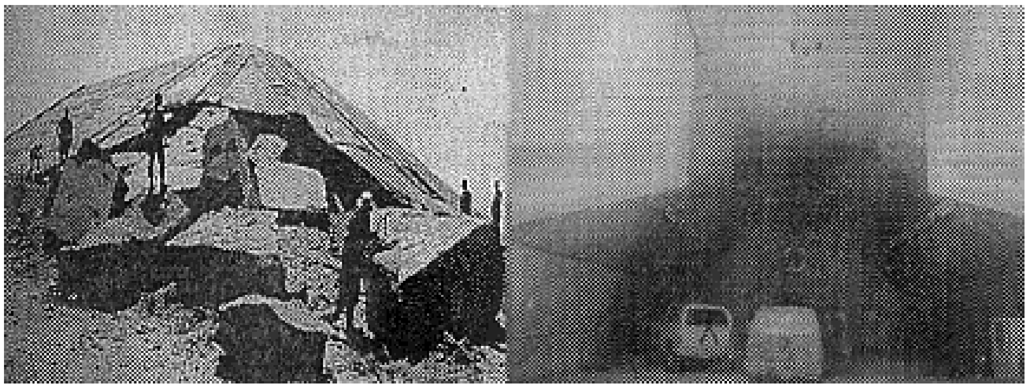

Figure 1. Exterior (left) and interior (right) photographs of an aircraft shelter hit by two bombs in the Gulf War (Cohen [6]). Apparently heavy damage is actually not.

of 2 is because the *look* option must be prepared for two possible assessments. One of the 15 possible 2-stage policies is "look, then strike if the look assesses live, or pause if the look assesses dead." The sequence N_k is 3, 15, 255, 65535, $4.3 * 10^9, \ldots$. Clearly policy enumeration will not be the method of choice in any realistically scaled problem.

Our example is on the scale of the attack-planning problem for a DESERT STORM-sized scenario. It has 9 stages (3 attack waves a day for 3 days, the length of a typical planning cycle), 9 aircraft types, 42 weapon types, 65 target types, and 10 sensor types. The master LP has 81 constraints for aircraft (one for each type and each stage), 42 weapon constraints, 65 target constraints, and 90 sensor constraints (one for each type and each stage), so $m = 278$. This problem contains the characteristics we have been discussing, as $|S|$ is intractably large while m is relatively small.

Since each target has only two states, the probability p of being alive is a sufficient statistic, and will itself be referred to as the target's state hereafter. Let $F_n^j(p, \lambda)$ be the expected net reward of the optimal policy over the last n stages for a target of type j in state p, the difference between the value of the targets killed and the resources required to kill them. Thus $p_j(S_j; \lambda)$ from Eq. (5) is $F_9^j(1, \lambda)$. The associated POMDP dynamic programming recursion $DP^j(\lambda)$ is as follows:

$$\mathrm{DP}^j(\lambda) : F_0^j(p, \lambda) = C_j(1 - p),$$

$$\text{for } 1 \le n \le 9, \qquad F_n^j(p, \lambda) = \max \begin{cases} F_{n-1}^j(p, \lambda) & \text{(pause)} \\ \max_{a \in A} F_{n-1}^j([1 - p_a]p, \lambda) - \lambda_a & \text{(strike)} \\ \max_{l \in L} E[F_{n-1}^j(Z, \lambda)] - \lambda_l & \text{(look)} \end{cases} \qquad (7)$$

In this recursion, λ_a and λ_l are the costs of actions a and l. The effect of an attack is to reduce p by the factor $(1 - p_a)$, the probability that the target survives the attack. A look produces a random state Z when there are $n - 1$ stages remaining, with two possibilities for Z because each look can result in either a "live" or "dead" assessment. The expectation is computed using Bayes'

Table 1. Solution statistics for the BDA problem on a 333 MHz Pentium II PC.

Section of algorithm	Exact POMDP solutions	Approx POMDP solutions
Master LPs (s)	31	28
POMDP subproblems (s)	1976	145
Total solution time (s)	2028	188
Initial columns	2214	2214
Columns generated	3373	2802
Total iterations	76	79

Theorem:

$$E[F_{n-1}(Z),\lambda] = [(1-\beta_l)p + \alpha_l(1-p)]F_{n-1}\left(\frac{(1-\beta_l)p}{(1-\beta_l)p + \alpha_l(1-p)}, \lambda\right) + [\beta_l p + (1-\alpha_l)(1-p)]F_{n-1}\left(\frac{\beta_l p}{\beta_l p + (1-\alpha_l)(1-p)}, \lambda\right). \quad (8)$$

6. IMPLEMENTATION AND COMPUTATIONAL RESULTS

Recent POMDP literature (e.g., Cassandra [3]) notes that while getting exact solutions to a POMDP can be very difficult, approximate solutions with bounds are much easier to compute. We employ Cheng's [5] "linear support algorithm," which works well for two-state POMDPs and can be adjusted to produce a solution with any desired level of accuracy. This is a crucial point, since it is wasteful to solve DP$^j(\lambda)$ to optimality in the early stages of the decomposition when λ is changing rapidly. A better strategy is to approximate the solution of DP$^j(\lambda)$ while still providing improving columns to the master LP. It is not necessary that DP$^j(\lambda)$ be solved exactly on every occasion, as long as an upper bound of controllable accuracy is available when computation stops.

Each solution of the collection of POMDPs (one for each target class) produces an upper bound on $v(S)$. Since each LP solution produces a lower bound on $v(S)$, this establishes a "gap" G that is the difference of the two bounds, and a relative gap RG that is just G divided by the upper bound. In our implementation, we gradually increase the accuracy of the POMDP solutions as the decomposition progresses, using the relative gap RG as a guide to how accurately the POMDPs should be solved. Table 1 shows the solution times for the BDA problem using both exact and approximate POMDP solutions. The code is written in Microsoft Visual Basic 5.0, and the master LPs were solved using the CPLEX 5.0 Callable Library. Both cases start with the same initial set of policies, and both terminate when $RG < .001$. The strategy of adjusting the POMDP accuracy cuts the solution time by over an order of magnitude.

The LP/POMDP algorithm generates only a few thousand columns in addition to the initial columns, a tiny fraction of those available in S. The decomposition has slow tail convergence, as is characteristic of column-generation schemes. It reaches a 0.05 relative gap in 38 s and a 0.01 gap in 80 s. Nearly all of this time is spent solving POMDPs, rather than LPs. Reducing the relative gap from 0.001 to 0 increases the solution time from 188 to 652 s, thereby increasing the lower bound by only 0.006%.

One feature of the optimal policies is worthy of comment. An optimal policy will sometimes pause for one or two stages before first striking a target. The reason for this is that the price of early strikes invariably turns out to be higher than the price of late ones, in spite of the fact that strikes are equally constrained in each stage. The reason for this decreasing strike price is that valuable targets may require multiple rounds of shooting and looking, which requires getting started immediately. Therefore, it makes sense to delay strike activity for less valuable targets that do not require that kind of treatment. This subtle but realistic tendency happens naturally in the LP/POMDP formulation, and is impossible to generate in an ordinary POMDP where resource prices are the same in each stage.

7. RIGID POLICY APPROXIMATION

The inequality constraints in LP2(S) state that resources must not be exhausted *on the average*, which we will refer to as the "soft" version of the constraints. The "rigid" version would require that resources *never* be exhausted. The quantity $y(k, U, X)$ was earlier defined as the mean resource consumption at time k if decision U is taken when the object under consideration is in state X. The resource quantities consumed may by implication be random, so the rigid problem is not well defined by the inputs discussed so far. The notion of "policy" needs to be enlarged in the rigid version to permit decisions made about one target to depend on observations made about another, and additional assumptions need to be made about joint distributions of random variables.

Whether the rigid or soft version is superior as a model depends on circumstances. For example, there has been a long debate within the Air Force about whether aircraft attrition is better modeled as a penalty in the objective function or as an explicit constraint. Putting attrition in the objective function raises the question of how to penalize it, while including a rigid constraint of (say) one lost aircraft would force the assignment of at most one sortie per stage in order to prevent even the *possibility* of multiple losses. A compromise position might be that attrition should be modeled as a soft constraint rather than a rigid one. In other words, only the *average* attrition might be constrained. There is a similar issue with respect to the availability of aircraft sorties within a given period, since aircraft usage always has a certain amount of flexibility in the short term. The soft/rigid modeling question can be argued either way.

However, there can be no debate about the relative tractability of the soft and rigid versions. We speculate that rigid versions of problems on the scale of our BDA problem will never be solvable, since rigid constraints force the consideration of a single large POMDP involving all objects, rather than a separate small POMDP for each one.

In spite of its theoretical intractability, any rigid problem can at least be approximated through the soft version. The optimal value of the soft formulation is an upper bound on the optimal value of the rigid formulation, since an upper bound on any random variable is also an upper bound on its mean.

There are a variety of ways to use soft policies as a template for rigid ones. In problems where stages are linked only by cumulative constraints of the type that link weapons consumption in the BDA problem, its makes sense to utilize only the first stage of a soft solution's policies, then evaluate the resulting resource consumption and target states based on the observations made so far, and then consider another soft problem with one less stage. The rest of this section gives the details of an example of this type of incremental rigid policy for the BDA problem.

The BDA problem has some characteristics that make it an apt subject for an incremental rigid policy. The current consumption of resources by each action is deterministic, for one thing, and in addition the initial resource constraints are all integral. This means that the POMDP will generate no policies that gamble with feasibility. The optimized soft decision variables x_{sj} may still be nonintegral, however, so they cannot be applied directly to the rigid problem. We resolve this difficulty by basing the actions in the current stage on a solution of LP2(S) where only policies that result in the consumption of resources in the current stage are required to be integral. Specifically, we solve LP2(S) to a specified gap, then, using the generated columns S' only, we solve a mixed-integer program MP2(S') that requires integral assignments for policies using resources in the current stage (policies that begin with any action other than "pause"). The associated actions are guaranteed to be feasible in the current stage, even in the rigid problem. After simulating the results of the allocations computed for the current stage and updating the state of each object, the entire procedure is repeated with one less stage, and so on.

As the number of remaining stages is gradually reduced in the incremental procedure, a given class of targets may fragment into individuals in many states on account of the randomness inherent in strikes and observations, as well as treatment by different policies. The number of constraints in LP2(S) must therefore grow to include one for every (class, state) pair. However, the POMDP solutions do not grow in difficulty because the POMDP solution for each target class already covers all states. Our experience is that solution times for LP2(S) are still smaller than POMDP times, even when LP2(S) grows in size and complexity due to the integrality restrictions and multiple allocation constraints.

The following procedure describes a Monte Carlo simulation of the incremental strategy:

Input: Data required for a 9-stage BDA problem.
Output: Terminal states for one replication of the incremental rigid strategy.
Step 1: Set $n = 9$. Initialize all resources and states for all targets.
Step 2: Solve LP2(S) for the current n-stage problem. Let S' be the set of strategies generated.
Step 3: Solve MP2(S') to get integral policy assignments for the first of n stages.

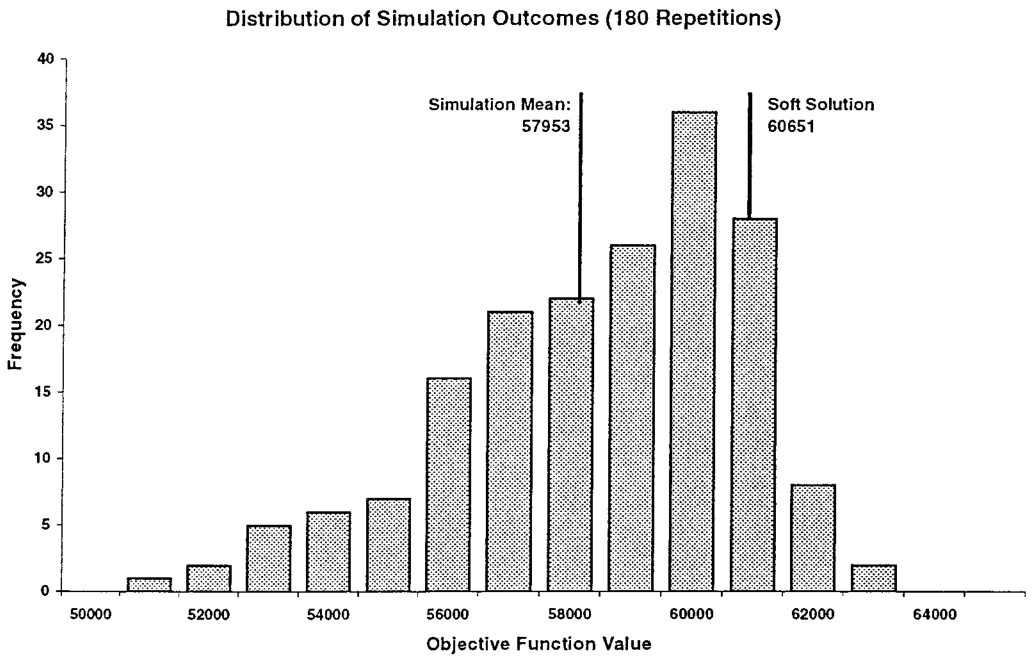

Figure 2. Rigid simulation outcomes compared to the soft solution in the BDA Problem.

Step 4: Simulate outcomes for the current stage and update object states and resources remaining.
Step 5: Let $n = n - 1$.
Step 6: Output states and quit if $n = 0$, or go to Step 4 if $n = 1$, or else go to Step 2.

Figure 2 shows the distribution of 180 such repetitions for the BDA problem. As expected, the soft solution is larger than the sample mean. However, the sample mean is within 5% of the soft solution, and would presumably be even closer if more sophisticated rigid strategies were employed. The BDA problem is on a large enough scale that the best rigid and soft solutions do not differ substantially from each other.

8. SUMMARY

The LP/POMDP marriage is a technique for solving large-scale allocation problems with partially observable states and constrained action and observation resources. The technique uses the strengths of LP in determining implicit resource prices, and the strengths of POMDPs in determining optimal policies extended in time. The technique can handle arbitrarily large numbers of objects as long as the number of possible states of each object remains small. It determines an optimal "soft" solution where resource constraints are satisfied on the average. The BDA example demonstrates the possibility of using the soft solution as an incremental guide to a rigid one in problems where a rigid solution is desired.

REFERENCES

[1] D.P. Bertsekas, Dynamic programming and stochastic control, Academic, New York, 1976, pp. 111–128.

[2] A.R. Cassandra, Optimal policies for partially observable Markov Decision Processes, Technical Report CS-94-14, Brown University, August 1994.

[3] A.R. Cassandra, Exact and approximate algorithms for partially observable Markov Decision Processes, Ph.D. thesis, Brown University, May 1998, 313 pp.

[4] D.A. Castanon, Approximate dynamic programming for sensor management, Proc 36th IEEE Conf Decision Control, San Diego, December 1997, pp. 1208–1213.

[5] H.-T. Cheng, Algorithms for partially observable Markov Decision Processes, Ph.D. thesis, University of British Columbia, August 1988, pp. 52–61.

[6] E. Cohen (Editor), Gulf War airpower survey, Volume II, Operations and effectiveness, Part II, Department of the Air Force, Washington, DC, 1993, pp. 39–43.

[7] P.C. Gilmore and R.E. Gomory, A linear programming approach to the cutting stock problem, Oper Res 9 (1961), 849–859.

[8] P.C. Gilmore and R.E. Gomory, A linear programming approach to the cutting stock problem—Part II, Oper Res 11 (1963), 863–888.

[9] W.K. Klein Haneveld, Stochastic linear programming models, Duality in stochastic linear and dynamic programming, Springer-Verlag, Berlin, 1986, pp. 22–47.

[10] T.W. Jonsbraten, Optimal selection and sequencing of oil wells under reservoir uncertainty, Technical Report, Department of Business Administration, Stavanger College, Stavanger, Norway, July 1997.

[11] T.W. Jonsbraten, R.J.-B. Wets, and D.L. Woodruff, A class of stochastic programs with decision dependent random elements, Technical Report, University of California Davis, August 1997.

[12] W.S. Lovejoy, A survey of algorithmic methods for partially observed Markov decision processes, Ann Oper Res 28(1) (1991), 47–65.

[13] K.T. Marshall and R.M. Oliver, Decision making and forecasting, McGraw-Hill, New York, 1995.

[14] C.H. Papadimitriou and J.N. Tsitsiklis, The complexity of Markov decision processes, Math Oper Res 12 (1987), 441–450.

[15] R.G. Parker and R.R. Rardin, Discrete optimization, Academic, San Diego, 1988, pp. 205–230.

[16] G. Pflug, On-line optimization of simulated Markov processes, Math Oper Res 15 (1990), 381–395.

[17] W.B. Scott, Computer/IW efforts could shortchange aircraft programs, Aviation Week and Space Technology (19 January 1998), 59.

[18] R. Smallwood and E.J. Sondik, The optimal control of partially observable Markov decision processes over a finite horizon, Oper Res 21 (1973), 1071–1088.

[19] K.A. Yost, Solution of large-scale allocation problems with partially observable outcomes, Ph.D. thesis, Naval Postgraduate School, September 1998.