

# Solving Large-Scale Allocation Problems with Partially Observable Outcomes

Kirk A. Yost and Alan R. Washburn<sup>1</sup>  
Naval Postgraduate School

## Abstract

We introduce a new technique for solving large-scale allocation problems with partially observable states and constrained action and observation resources. Our decomposition 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.

## 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 observations about the object states are “noisy.” Models with noisy observations are said to have *partial observability*, and this characteristic distinguishes this problem from typical stochastic programs.

Allocation problems with partial observability are common. In medicine, treatments with random effects are made on the basis of error-prone tests and examinations. In the criminal justice system, sentencing decisions are made under similar circumstances. Our work is motivated by a military problem where the objects are targets that are either “live” or “dead” and the resources are weapons and sensors.

While the literature has tended to treat this situation 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 differing types of objects. In the math programming world, a problem of this class would normally be cast as a stochastic program with recourse (e.g., Haneveld [9]). However, the structure of the problem makes this approach very difficult. First, the outcomes are not known with certainty, so the typical recourse mechanism of acting, observing the random outcomes, and correcting at some cost does not really apply. Also, the stochastic program with recourse does not generally allow the probability of realizing a scenario to be influenced by actions taken. In this model, the states of the objects *depend* on the actions taken, so modeling this is crucial. Finally, stochastic programs do not normally consider the notion of dedicated and constrained observation resources.

---

<sup>1</sup> Professor Alan R. Washburn, Code OR/Ws, Operations Research Dept, Naval Postgraduate School, Monterey, CA 93943-5000.

There is some work being done in this area. Jonsbraten [10] studies a well-drilling application with partial observability, and he approaches the problem as a decision tree with constraints. Jonsbraten, Wets, and Woodruff [11] develop a stochastic programming model where the random elements do depend on the actions taken. They note the literature in this area is very sparse, and were only able to find one paper on the Markovian case (Pflug [16]). We have been unable to find any papers in the math programming literature addressing partial observability and constrained observation resources.

Castenon [4] addresses a problem of this class in a paper analyzing the allocation of a single aircraft sensor to find targets of multiple types. He suggests using partially observable Markov decision processes (POMDPs) along with a Lagrange multiplier technique to handle a single constraint on observations. He notes that finding a way to generalize the technique for larger problems is a subject for further research.

We offer a method to solve such a generalized problem. First, we employ a master linear program (LP) to implicitly assign costs to resources; and second, we solve 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 allocation problems that are otherwise intractable. We begin by describing the case where there is only one object.

## 2. Core Problem for a Single Object

Let  $I$  be a set of  $m$  constrained resources, and let  $R$  and  $Y \equiv (Y_1, Y_2, \dots, Y_m)$  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  $E()$  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 maximization problem as a linear program  $LP(S)$  with variables  $x_s$ :

$$\begin{aligned}
 LP(S): \quad & \max \sum_{s \in S} ER_s x_s \\
 \text{st} \quad & \sum_{s \in S} EY_{si} x_s \leq b_i \quad \forall i \in I \\
 & \sum_{s \in S} x_s = 1 \\
 & x_s \geq 0 \quad \forall s \in S
 \end{aligned}$$

Variable  $x_s$  is the probability of choosing policy  $s$ , and the sums in the objective function and resource constraints are correct 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. Note that  $LP(S)$  is a “soft” problem in the sense we only constrain the *expected* resource consumption. We will comment further on this in Section 6.

We are interested in problems where  $S$  is so large that enumeration of all possible policies is out of the question. In such cases, the best approach is usually to find an  $\epsilon$ -optimal algorithm. This in turn 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  $\lambda = (\lambda_1, \dots, \lambda_m)$ :

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

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  $\lambda$  is equal to the 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

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

For any particular set of resource prices  $\lambda$ , finding the upper bound is a matter of solving the maximization part of  $u(S; \lambda)$ , which we assume to be possible in spite of the large size of  $S$ . Solving the maximization part of  $u(S; \lambda)$  means finding a policy in  $S$  that improves the current LP solution, which is the same as finding an improving column for  $LP(S)$ . With this in mind, we can write a solution algorithm for  $LP(S)$  by using dynamic column generation (Gilmore and Gomory [7], [8]):

1. Choose an initial subset  $T^1$  of  $S$ . Set  $k = 1$ .
2. Solve  $LP(T^k)$  for  $v(T^k)$  and the dual prices  $\lambda^k$ .
3. If  $u(S; \lambda^k) - v(T^k) \leq \epsilon$  stop; otherwise, let  $T^{k+1} = T^k \cup \{s\}$ , add 1 to  $k$  and go to 2.

If  $s \in T^k$ , then  $u(S; \lambda^k) = u(T^k; \lambda^k) = v(T^k)$ , so the union operation in step 3 is always nontrivial. It follows that the algorithm will stop after at most  $|S|$  steps, even if  $\epsilon=0$ . However, we take little comfort in this due to the assumed large size of  $S$ . Our aim is to find an  $\epsilon$ -optimal solution after a number of steps that is much smaller than the theoretical limit.

### 3. Applying Partially Observable Markov Decision Processes (POMDPs)

The algorithm described in section 2 could be useful in any circumstance where LP(S) is hard to solve while (1) is not, and is a generic column-generation scheme. However, the characteristics of this problem – sequential decisions and partial observability – make the POMDP an attractive choice for the subproblem.

A POMDP involves a sequence of decisions  $u_0, \dots, u_{N-1}$  and a sequence of observations  $z_0, \dots, z_{N-1}$ , with a decision-making policy being *admissible* if  $u_k$  depends only on the observable history  $I_k \equiv (z_0, z_1, \dots, z_k, u_0, u_1, \dots, u_{k-1})$ , for  $k = 0, \dots, N-1$  (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 to the decision-maker 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], normally includes a scalar reward function  $g(k, u, x)$  that is accumulated over time to form the objective. We retain that function, but in addition require that the expected resource consumption  $r(k, u, x)$ , a vector of dimension  $m$ , be specified for each action  $u$  that might be taken at time  $k$ . The cost of the resources consumed at prices  $\lambda$  is then the inner product  $\lambda r(k, u, x)$ . The net profit function  $G(k, u, x)$  is  $g(k, u, x) - \lambda r(k, u, x)$ , and the expected total net profit for a given policy  $s$  is the accumulation of all net profits:

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

where  $g(X_N)$  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  $p(S; \lambda)$ .

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

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

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

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

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 maximized net profit  $p(T^k; \lambda^k)$  is necessarily equal to  $w^k$ , so step 3 of the algorithm could be written as

3. If  $p(S; \lambda^k) - w^k \leq \epsilon$  stop; otherwise, let  $T^{k+1} = T^k \cup \{s\}$ , add 1 to  $k$  and go to 2.

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

As applied to POMDPs, the algorithm of section 2 is essentially a method for including resources that are constrained, rather than being modeled through explicit prices. From the point of view of the POMDP user, this methodology is an important extension. The entire 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.

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 POMDP problem has been shown to be PSPACE-complete by Papadimitriou and Tsitsiklis [14], and even the best of the current algorithms have difficulty solving a single POMDP with a long time horizon or many states. Since our algorithm solves a sequence of POMDPs, we must be concerned about the computational feasibility of such an approach. Nonetheless, we have found that the decomposition procedure works well for problems with small state spaces.

#### 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  $S = \bigcup_{j \in J} S_j$ . Also let  $x_{sj}$  be the average number of objects in class  $j$  to which policy  $s$

is applied, and assume that both rewards and resource consumption are additive over classes. Then, since sums and expected values commute, the expanded master LP (with dual variables in parentheses) is:

$$\begin{aligned}
\text{LP2(S): } & \max \sum_{j \in J} \sum_{s \in S_j} ER_{sj} x_{sj} \\
& \text{st } \sum_{j \in J} \sum_{s \in S_j} EY_{sji} x_{sj} \leq b_i \quad \forall i \in I \quad (\lambda_i) \\
& \sum_{s \in S_j} x_{sj} = N_j \quad \forall j \in J \quad (w_j) \\
& x_{sj} \geq 0 \quad \forall j \in J, s \in S_j
\end{aligned}$$

$ER_{sj}$  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::

$$\begin{aligned}
\text{LPU2(S; } \lambda): & \max \sum_{j \in J} \sum_{s \in S_j} ER_{sj} x_{sj} + \sum_{i \in I} \lambda_i \left( b_i - \sum_{j \in J} \sum_{s \in S_j} EY_{sji} x_{sj} \right) \\
& \text{st } \sum_{j \in J} \sum_{s \in S_j} x_{sj} = N_j \quad \forall j \in J \\
& x_{sj} \geq 0 \quad \forall j \in J, s \in S_j
\end{aligned}$$

Note the upper bound LP decomposes into  $|J|$  separate optimizations. The algorithm of section 2 still applies, except we have to solve a separate POMDP for each object class. Let

$$p_j(S_j; \lambda) = \max_{s \in S_j} \left\{ ER_{sj} - \sum_{i=1}^m \lambda_i EY_{sji} \right\} \quad (4)$$

The overall upper bound is given by:

$$u(S; \lambda) = \sum_{i=1}^m \lambda_i b_i + \sum_{j \in J} N_j p_j(S_j; \lambda) \quad (5)$$

The multiple-object decomposition algorithm, with natural generalizations of the notation used in section 2 (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  $j$ th component of  $w^k$ ), is as follows:

1. For all  $j$ , choose an initial subset  $T^{1j}$  of  $S^j$ . Set  $k = 1$ .

2. Solve LP2( $T^k$ ) for  $v(T^k)$  and the dual prices  $\lambda^k, w^k$ .
3. For all  $j$ , solve (4) using  $\lambda^k$  for  $p^j(S^j; \lambda^k)$  and optimizing policy  $s^j$ .
4. For all  $j$ , if  $p^j(S^j; \lambda^k) > w^{kj}$  let  $T^{j,k+1} = T^{j,k} \cup \{s^j\}$ , else let  $T^{j,k+1} = T^{j,k}$ .
5. If  $u(S, \lambda^k) - v(T^k) \leq \epsilon$ , stop. Otherwise, add 1 to  $k$  and go to 2.

The multiple-object algorithm stops if none of the object classes generates a new policy in step 4, so it stops after at most  $|S|$  steps. We must now solve  $|J|$  POMDPs in each step, so the computational burden increases when multiple object classes are considered. However, increasing  $N_j$  has no computational effect on any POMDP nor on the master LP, so the algorithm works best on problems where there are few classes but many objects per class. This comment is partially responsible for the words “large scale” in the title of this article.

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

## 5. Example Problem and Computational Experience

Our application is a crucial one in the military, that of assigning weapons and sensors to targets. Modern armaments can be fired from very long distances and have high probabilities of kill, but these weapons are both expensive and in short supply. Also, the long standoff ranges require looks from sensors 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, for example Scott [17].

The objects to be controlled are the targets. They have two states (live or dead), 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  $g(X_N)$  is  $C_j$  if the terminal state is dead, otherwise 0. Also, there are several types of resources. There are aircraft and sensors, whose availabilities are measured in terms of sorties per stage. There are also weapons that can be dropped from the aircraft, with total weapon consumption over all stages being constrained. The overall goal of the optimization 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  $A = \{a\}$  denote 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  $O = \{o\}$  be the set of looks, each of which is the assignment of a sensor to a target. Each look  $o$  is assumed to have known error probabilities  $\alpha_o \equiv \Pr(\text{assess target as live} \mid \text{target is actually dead})$  and  $\beta_o \equiv \Pr(\text{assess target as dead} \mid \text{target is actually live})$ , and each sensor is capable of

looking at each target. 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.

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), 9 aircraft types, 42 weapon types, and 65 target types. There are 5203 feasible aircraft-weapon-target combinations ("strikes"). There are also 10 sensor types, each of which can look at any target and deliver bomb-damage assessment (BDA). 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 in each stage), so  $m=278$ . This problem contains the characteristics we have been discussing, as  $|S|$  is intractably large while  $m$  is relatively small.

We solve the POMDP for each target, where the POMDP sufficient statistic is a probability distribution over the two target states. The probability  $p$  that the target is live implicitly determines the distribution, so we use it for the POMDP state. Let  $F_n^j(p)$  be the expected cost of the optimal policy over the last  $n$  stages for an object of type  $j$  in state  $p$ . The associated dynamic programming recursion  $DP^j(\lambda)$  is as follows:

$$DP^j(\lambda): \quad F_0^j(p) = C_j p$$

$$\text{for } n > 0, F_n^j(p) = \max \begin{cases} F_{n-1}^j(p) & \text{(pause)} \\ \max_{a \in A} F_{n-1}^j([1-p_a]p) - \lambda_a & \text{(attack)} \\ \max_{o \in O} E[F_{n-1}^j(Z)] - \lambda_o & \text{(look)} \end{cases}$$

In this recursion,  $\lambda_a$  and  $\lambda_o$  are the per-use costs of actions  $a$  and  $o$ . 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 POMDP 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' Theorem:

$$E[F_{n-1}(Z)] = [(1-\beta_o)p + \alpha_o(1-p)]F_{n-1}\left(\frac{(1-\beta_o)p}{(1-\beta_o)p + \alpha_o(1-p)}\right) +$$

$$[\beta_o p + (1-\alpha_o)(1-p)]F_{n-1}\left(\frac{\beta_o p}{\beta_o p + (1-\alpha_o)(1-p)}\right) \quad (7)$$

The optimal solution to  $DP^j(\lambda)$  is piecewise-linear and convex[18], and current solution algorithms exploit this structure. Space prevents us from discussing POMDP solution methods in detail; Lovejoy [12] and Cassandra [3] provide good surveys.

Recent POMDP literature (e.g., Cassandra [2]) 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 pointless to solve  $DP^j(\lambda)$  to optimality in the early stages of the decomposition when  $\lambda$  is changing rapidly. A better strategy is to approximate the solution of  $DP^j(\lambda)$  while still providing improving columns to the master LP. However, it is necessary to find bounds on optimal solutions provided by the POMDP to determine a global upper bound. This is another advantage of the linear support algorithm, as it provides these bounds.

In our implementation, we gradually increase the accuracy of the POMDP solutions as the decomposition progresses. Table 1 shows the solution times for the test problem using exact versus approximate POMDP solutions for the subproblems. We compute the decomposition gap as the (upper bound – lower bound)/upper bound, and stop when this gap  $< 0.001$ . Both cases start with the same initial set of policies. The strategy of adjusting the POMDP accuracy cuts the subproblem solution time by over an order of magnitude.

The computation times reported in Table 1 were obtained using a 333MhZ Pentium II PC running Microsoft Windows NT 4.0. The code itself is written in Microsoft Visual Basic 5.0; the master LPs were solved using the CPLEX 5.0 Callable Library. We used a simple heuristic to generate an initial set of policies, and then followed the algorithm as developed in Sections 2 and 3.

The algorithm generated only a few thousand columns in addition to the initial columns, a tiny fraction of the available policies. The decomposition has slow convergence, with the majority of the solution time spent achieving very small improvements. The approximate POMDP method, for example, reaches a 0.05 gap in 38 seconds and a 0.01 gap in 80 seconds.

The algorithm can compute an exact optimal solution, but requires considerably more time. Reducing the gap from 0.001 to 0 increased the solution time from 188 to 652 seconds, and increased the number of iterations from 79 to 121. This increased expenditure only improved the lower bound by 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 the price of early strikes invariably turns out to be higher than the price of late ones, in spite of the fact that 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 realistic tendency happens naturally in our formulation, and is impossible to generate in an ordinary POMDP where resource prices are constant.

## 6. Bounds, Policy Application, and Reoptimization

The soft constraints in LP2 state that resources must not be exhausted, on the average. The “rigid” version of this problem would require that resources *never* be exhausted. The quantity  $r(k,u,x)$  was earlier defined as the mean resource consumption at stage  $k$  if action  $u$  is taken when the object under consideration is in state  $x$ . The resource quantities consumed may by implication be random, so additional structural assumptions are needed in the rigid version about stochastic independence and what happens when resources are exhausted.

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. A compromise position might be that it should be modeled as a constraint, but a soft constraint rather than a rigid one. There is a similar issue with respect to the availability of aircraft sorties within a given period, since there is always a certain amount of flexibility in the short term.

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 the BDA problem in section 5 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. The rigid version is therefore much more difficult than the soft one. In spite of this intractability, any rigid problem can at least be approximated through the soft version. Depending on the additional structural assumptions mentioned above, the soft version may be a relaxation of the rigid one. Since the mean of any random variable bounded above by  $b$  is also necessarily bounded above by  $b$ , the optimized soft objective function will be an upper bound on the rigid one.

There are a variety of ways to use soft policies as a “template” for rigid policies. However, given the staged structure of the POMDP, it makes sense to reoptimize the soft problem between stages, using current resource levels and target states to establish a new set of optimal policies. The rest of this section gives the details of an example of this type of rigid policy for the BDA problem of section 4.

The BDA problem has several characteristics that make it an apt subject for rigid policy determination. The current consumption of resources by each action is deterministic, for one thing, and in addition the resource constraints are all integral. This means that there are no actions that gamble with feasibility, a

simplifying feature that guarantees that the soft solution is an upper bound on the rigid one. The optimized soft decision variables  $x_{sj}$  may nonetheless be non-integral, 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 policies that result in the consumption of resources in the current stage are required to be integral. 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 next stage is considered, and so on. In our example problem, the actions taken in the current stage lead to deterministic consumptions; only the future consumptions are probabilistic, based on the outcomes of the look actions. Therefore, we only need to enforce integrality for policies that consume resources in the current stage. There are many ways this integrality condition could be enforced, and an optimal approach would involve embedding the decomposition in a branch-and-bound procedure. Our approach is simpler; we solve LP2(S) to a specified gap, then, using the generated columns only (the set  $S'$ ), we solve a mixed-integer program MP2( $S'$ ) that requires integral assignments for policies using resources in the current stage.

As time progresses, 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 allocation constraints in LP2(S) must grow to include one for every (class, state) pair, so LP2(S) becomes increasingly larger. However, the POMDP solutions do not grow in difficulty because the POMDP solution for each target class covers all states automatically. Our experience is that solutions 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 implements and tests the above scheme by Monte Carlo simulation, with the index  $k$  being a repetition counter:

1. Set  $k = 0$ .
2. If  $k =$  desired number of repetitions, stop; otherwise, set  $k = k + 1$ , set each object to its initial (known) state, and set  $n =$  number of stages in the time horizon.
3. Solve LP2(S) for the current states of all the objects, the current resources available, and  $n$  stages yielding the set  $S'$ .
4. Solve MP2( $S'$ ) to get integral policy assignments for the current stage.
5. Simulate outcomes for the current stage and update object states and resources used.
6. If  $n = 1$ , set  $n = 0$  and go to 5; if  $n = 0$ , go to 2; otherwise, set  $n = n - 1$  and go to 4.

Figure 1 shows the distribution of 180 such repetitions for the BDA problem. As expected, the soft solution provided by LP2(S) is larger than the sample mean. However, the sample mean is within 5% of this upper bound for the tested rigid strategy, and would presumably be even closer if more sophisticated reoptimization strategies were employed. Our heuristic explanation of this closeness is that the POMDP is aware of the marginal costs of the resources and plans accordingly, so as columns are generated, the costs of the resources directly affect the distributions of their use in the policies. We expect to find that the percentage difference between the soft and rigid problems becomes small as the scale of the problem increases, and take these results as evidence to that effect. This is another reason for our use of the term “large-scale” in the title.

We offer one final comment on this simulation. Much of the stochastic programming literature is concerned with finding bounds on objective function values, and we have exploited a great deal of this theory in our work. However, by making the initial decomposition fast, we feel that the optimize-simulate-optimize procedure shown above has great value in modeling problems of this type. Objective function bounds do not describe the distribution of outcomes, and in allocation problems, the distributions of allocations and resource usage are often more important than the distribution of objective function values. Many authors, such as Geoffrion and Powers [6] have commented on the fragility of optimization solutions and their tendency to produce solutions that use resources in an extreme fashion. A Monte Carlo simulation such as this can give a better idea of allocation variability in time-staged problems and perhaps even provide more robust solution strategies.

## **7. Summary**

We have outlined a new LP/POMDP technique for solving large-scale allocation problems with partially observable states and constrained action and observation resources. Our decomposition technique uses the strengths of linear programming in determining resource allocation and implicit resource prices, and the strengths of partially observable Markov decision processes in determining optimal policies extended in time.

We believe the LP/POMDP technique holds great promise for situations where information-gathering resources must be shared among many objects. While our research has been oriented towards military problems, it is not hard to find other applications with these characteristics. In particular, the medical world is replete with constrained observation resources (X-rays, MRI’s, consultations with doctors) that only partially reveal a patient’s true state. Similar situations exist in education, criminal justice, and environmental management. In the information age, it will become increasingly important to consider problems such as these where opportunities for observation and transformation are intermixed. The LP/POMDP technique can solve some of them.

## References

1. Bertsekas, Dimitri P., *Dynamic Programming and Stochastic Control*, Academic Press, New York, 1976, pp. 111-128.
2. Cassandra, Anthony R., *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*, Ph.D. thesis, Brown University, May 1998, p. 313.
3. Cassandra, Anthony R., *Optimal Policies for Partially Observable Markov Decision Processes*, Technical Report CS-94-14, Brown University, August 1994.
4. Castanon, David A., "Approximate Dynamic Programming for Sensor Management," in *Proceedings of the 36<sup>th</sup> IEEE Conference on Decision and Control*, Vol. 2, IEEE Control Systems Society, Danvers, MA, pp. 1202-1207.
5. Cheng, Hsien-Te, *Algorithms for Partially Observable Markov Decision Processes*, Ph.D. thesis, University of British Columbia, August 1988, pp. 52-61.
6. Geoffrion, Arthur M., and R. F. Powers, "Twenty Years of Strategic Distribution System Design: An Evolutionary Perspective," *Interfaces*, Vol. 25, No. 5 (1995), pp. 105-127.
7. Gilmore, P. C., and R. E. Gomory, "A Linear Programming Approach to the Cutting Stock Problem," *Operations Research*, Vol. 9 (1961), pp. 849-859.
8. Gilmore, P. C., and R. E. Gomory, "A Linear Programming Approach to the Cutting Stock Problem — Part II," *Operations Research*, Vol. 11 (1963), pp. 863-888.
9. Haneveld, W. K. K., "Chapter 3: Stochastic Linear Programming Models," in *Duality in Stochastic Linear and Dynamic Programming*, Springer-Verlag, Berlin, 1986, pp. 22-47.
10. Jonsbraten, Tore W., *Optimal Selection and Sequencing of Oil Wells under Reservoir Uncertainty*, technical report, Department of Business Administration, Stavanger College, Norway, July 1997.
11. Jonsbraten, Tore W., Roger J-B Wets, and David L. Woodruff, *A Class of Stochastic Programs with Decision Dependent Random Elements*, technical report, University of California Davis, August 1997.
12. Lovejoy, William S., "A Survey of Algorithmic Methods for Partially Observed Markov Decision Processes," *Annals of Operations Research*, Vol. 28, No. 1 (1991), pp. 47-65.
13. Marshall, Kneale T., and Robert M. Oliver, *Decision Making and Forecasting*, McGraw-Hill, New York, 1995.
14. Papadimitriou, Christos H., and John N. Tsitsiklis, "The Complexity of Markov Decision Processes," *Mathematics of Operations Research*, Vol. 12 (1987), pp. 441-450.
15. Parker, R. Gary, and R. R. Rardin, *Discrete Optimization*, Academic Press, San Diego, 1988, pp. 205-230.
16. Pflug, Georg, "On-line Optimization of Simulated Markov Processes," *Mathematics of Operations Research*, Vol. 15 (1990), pp. 381-395.
17. Scott, William B., "Computer/IW Efforts Could Shortchange Aircraft Programs," *Aviation Week and Space Technology*, January 19, 1998, p. 59.
18. Smallwood, Richard, and E. J. Sondik, "The Optimal Control of Partially Observable Markov Decision Processes over a Finite Horizon," *Operations Research*, Vol. 21 (1973), pp. 1071-1088.

<b>Section of Algorithm</b>	<b>Exact POMDP Solutions</b>	<b>Approx POMDP solutions</b>
Master LPs	31 sec	28 sec
POMDP subproblems	1976 sec	145 sec
<b>Total Solution Time</b>	2028 sec	188 sec
<b>Initial Columns</b>	2214	2214
<b>Total Columns Generated</b>	3373	2802
<b>Total Iterations</b>	76	79

Table 1. Solution Statistics for the Test Weapon-Sensor-Target Assignment Problem.

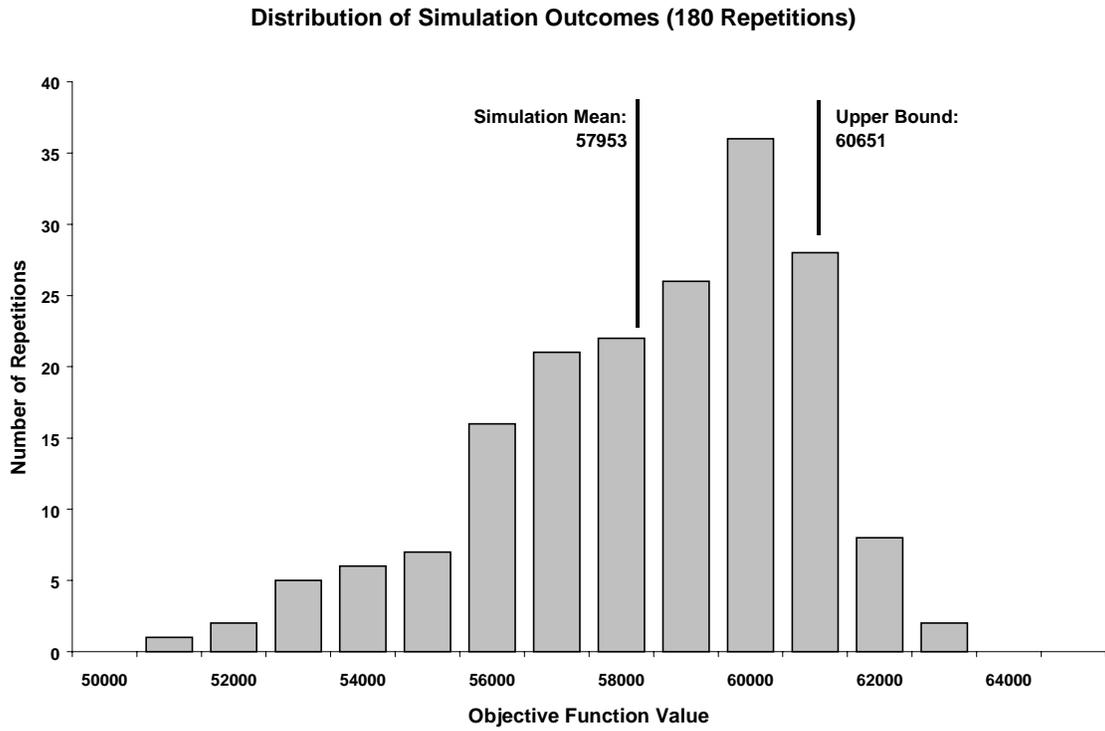


Figure 1. Comparison of Simulation Outcomes and Original Upper Bound for Test Weapon-Sensor-Target Assignment Problem. The mean objective function value through 180 repetitions of the simulation is within 5% of the value computed by the decomposition algorithm.