An Adaptive Dynamic Programming Algorithm for Dynamic Fleet Management, I: Single Period Travel Times

Gregory A. Godfrey • Warren B. Powell
Department of Operations Research and Financial Engineering, Princeton University, Princeton, New Jersey 08544

We consider a stochastic version of a dynamic resource allocation problem. In this setting, reusable resources must be assigned to tasks that arise randomly over time. We solve the problem using an adaptive dynamic programming algorithm that uses *nonlinear* functional approximations that give the value of resources in the future. Our functional approximations are piecewise linear and naturally provide integer solutions. We show that the approximations provide near-optimal solutions to deterministic problems and solutions that significantly outperform deterministic rolling-horizon methods on stochastic problems.

We consider the problem of assigning a set of homogeneous vehicles (or resources) to customer demands (or tasks) that arise randomly over time. Each vehicle is in a particular state (typically, a geographical location), and each task requires a resource in a particular state. The assignment of a resource to a task generates revenue and removes the task from the system. It also changes the state of the resource. A resource in one state can be transformed into a resource in selected other states at a cost. A task may be assigned to resources in a specified subset of states. Finally, we assume that all transitions require one time period (such as four hours or a day). Needless to say, we are generally interested in integer solutions to this problem.

Our problem arose in the context of managing fleets of vehicles that are allocated spatially over time. Vehicles might be trailers, boxcars, or containers, or they may be rental cars and trucks. The same problem structure also arises in machine scheduling with setups (moving a truck from one city to the next is identical to a machine with a setup cost and time), as well as other forms of resource transformation (such as personnel training). Even in transportation, the movement of a vehicle from one location to the next may involve more than a change in space: the truck may have less fuel; the driver may have reduced hours of service; and a chemical trailer may become dirty and, therefore, may need to be cleaned.

We refer to this problem as the stochastic dynamic resource allocation problem (SDRAP), a problem class with its roots in fleet management. It arises in a variety of settings, including: fleet management (how many empty box cars or trailers should be moved empty from one location to the next and which loads should be served first), car rental fleets (how many rental cars, and which types, should be assigned to a particular pool), dynamic crew scheduling (assigning drivers to move loads from origin to destination, taking into account all the characteristics of each type of driver), training of military personnel (how many recruits should receive a particular type of specialty training), and remnant inventory (given bins of pipes of different lengths, we wish to choose a pipe that is at least as long as the required demand, cut it down to the needed size, and then put the leftover piece in the appropriate bin).

0041-1655/02/3601/0021$05.00
1526-5447 electronic ISSN

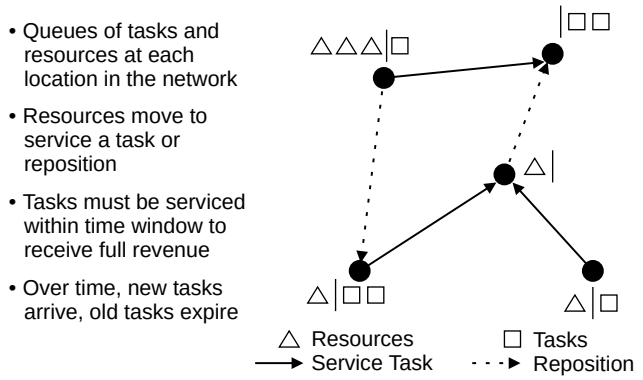


Figure 1 Physical Depiction of Dynamic Resource Allocation Problem

An illustration of the problem in a spatial context is given in Figure 1. Here, we have a set of points in space, and at each point are a queue of resources waiting to be assigned and a queue of tasks waiting to be completed. Each task will require moving a resource from one location to another. We have to decide which resources to assign to which tasks, and if we have excess resources, whether we want to transform (or in this setting, move) any resources from one location to another.

Three issues make solving this problem more interesting than simply matching resources to tasks at a common location. First, the planner may pay a cost to transform a resource from one state to another (for example, a change in location, cleaning a chemical trailer, or taking a driver off-duty). We call this *repositioning* the resource. Second, a planner may be penalized (receive a reduced revenue) for servicing a task too late. We define the task *time window* as the time during which a task may be serviced at full revenue (the reward function may be relatively complex). Third, over time, new resources and tasks become available via some random process. This mimics the real-world process of people calling for taxis or customers calling to have packages picked up. Although the planner cannot know the exact future, we assume that historical information is available about the demand process and can be used to estimate probability distributions.

This paper fills a gap in the research literature. Traditional dynamic programming methods focus on discrete state and action spaces (Puterman 1994). Forward dynamic programming using Monte Carlo methods (Bertsekas and Tsitsiklis 1996, Sutton and Barto 1998) is not as sensitive to large state spaces, but does suffer from large action spaces (they generally require the ability to search the entire action space to choose the best one). Forward dynamic programming methods also require the ability to estimate the value of the system being in a particular state. However, convergence proofs are only available under very strong assumptions (in particular, that each state will be visited infinitely often); in practice, we only visit the states generated by a particular policy, and if we produce a poor estimate of the value of a state, we may never again choose the actions that return us to this state.

A different line of investigation is to draw on the theory of stochastic linear programming (see the recent books by Infanger 1994, Kall and Wallace 1994, and Birge and Louveaux 1997). This literature can be divided between two-stage linear programs and their extension to multistage problems. Two-stage stochastic programs have been formulated as large-scale optimization problems subject to *nonanticipativity* constraints (Dantzig 1955, Rockafellar and Wets 1991), but these approaches typically limit the size of the problem we can consider. Techniques have been developed that approximate the second-stage recourse function (using either a static sample or with dynamic sampling). Approximations can be made through linearization methods (Ermoliev 1988, Ruszczynski 1980) or through the use of Benders cuts, which may be generated from a static sample (Van Slyke and Wets 1969) or dynamically through sampling-based methods (Higle and Sen 1991). Linearization methods usually benefit from nonlinear terms that stabilize the behavior. Examples include auxiliary function methods that combine stochastic gradient information with nonlinear stabilizing terms (Culioli and Cohen 1990, Cheung and Powell 2000) or proximal point terms (Ruszczynski 1987). These techniques are relevant to this research because it is always possible to take an approximation algorithm that is convergent for two-stage problems and apply it heuristically to a multistage problem.

Multistage problems are particularly problematic. Birge (1985) presents a nested Benders algorithm

for multistage stochastic programs, but the method is impractical for even medium-sized problems. Monte Carlo-based methods mitigate the difficulty of working with large sample spaces. Higle and Sen (1991) introduced the first sampling-based methods for two-stage stochastic programs. Pereira and Pinto (1991) suggest a sampling-based version of nested Benders decomposition (without a proof of convergence). Chen and Powell (1999) propose a convergent sampling-based version of nested Benders for multistage problems with independence across stages. Unfortunately, *all* of these methods depend on Benders cuts to approximate the impact of decisions on future time periods. This general approach suffers from extremely slow convergence for problems with even a modest number of dimensions. In addition, integrality constraints are problematic.

A number of approximations have been proposed in the context of fleet management, which represents a classic stochastic resource allocation problem. Jordan and Turnquist (1983) and Powell (1986) present methods for continuous (noninteger) flows. Powell (1987), Frantzeskakis and Powell (1990), Cheung and Powell (1996), and Powell and Carvalho (1998) present approximations that naturally produce integer solutions. Most of this work (Powell 1987, Frantzeskakis and Powell 1990, Cheung and Powell 1996) was not designed for problems with time windows (tasks that may be serviced over a range of time periods). The best results from this line of research are given in Cheung and Powell (1996), but these techniques are also relatively difficult to apply. Most recently, Carvalho and Powell (2000) propose a linear approximation with a multiplier adjustment procedure (LAMA) that works only on deterministic problems but gives promising results.

Our approach to solving this problem uses the CAVE (Concave Adaptive Value Estimation) algorithm of Godfrey and Powell (2001) to construct a concave, separable, piecewise-linear approximation of the value function that is more flexible and responsive than linear approximations. Previously, CAVE had been applied only to two-stage problems, but we apply it to multistage problems with single period travel times. The travel time assumption simplifies the mathematical presentation and avoids issues specific to multiperiod travel times that are beyond the scope of this paper. Readers are referred to Godfrey and Powell (2002) for a treatment of the problem with multiperiod travel times.

The contribution of this paper is a novel approximation strategy for solving multistage stochastic resource allocation problems. We avoid the curse of dimensionality of dynamic programming and the complexities associated with solving multistage stochastic linear programs using classical scenario methods. The method is computationally very fast and appears to produce near optimal solutions for deterministic problem instances. Most significantly, it outperforms rolling-horizon procedures, which represent the state of the art for engineering practice for this problem class.

In §1, we present the notation and formulate the problem as a dynamic program. Then, §2 describes our solution strategy, where we use a novel approach for approximating dynamic programs for multistage linear programs. Section 3 describes methods for producing nonlinear approximations of value functions using gradient information, including a summary of the CAVE algorithm used to construct piecewise linear concave approximations of the value function. In §4, we show how to apply the CAVE logic to solve multistage dynamic programs. Section 5 presents a series of deterministic and stochastic experiments that test our proposed algorithm over a wide range of data sets. Finally, in §6, we present our conclusions to the research.

1. Notation and Problem Formulation

In this section, we define the notation and problem dynamics for modeling the SDRAP over a fixed finite planning horizon. We have adopted the standard notation suggested by Powell et al. (2002), simplifying when possible and adding stochasticity where necessary. We conclude with a recursive dynamic programming formulation of the problem.

The Network

The physical and temporal elements of the underlying transportation network are defined as follows.

T = the number of planning periods in the planning horizon.

$\mathcal{T} = \{0, 1, \dots, T-1\}$, the times at which decisions are made.

$\mathcal{I}$ = the set of physical locations in the network, indexed by i and j.

We model the problem in discrete time, reflected in the set $\mathcal{T}$. In practical applications, we find that it is necessary to represent the exact time within a time period, but for the purposes of this paper, we represent all activities as beginning and ending at a particular instant in the set $\mathcal{T}$. A critical assumption in this paper is that the time required to move between any pair of locations is a single time period.

Resources and Tasks

At the beginning of the planning horizon, there is an initial set of tasks to be performed and an inventory of resources available to perform the tasks. All resources are the same, and any resource may serve any task as long as they are at the same location. Over time, additional resources may become available for the first time. For each $t \in \mathcal{T}$, $i \in \mathcal{I}$, we define

$\widehat{R}_{it}$ = the number of resources that first become available at location i at time t, with $\widehat{R}_t = (\widehat{R}_{it})_{i\in\mathcal{I}}$;

R_{it} = the total number of resources available at location i at time t before any new arrivals have been added in, with $R_t = (R_{it})_{i\in\mathcal{I}}$;

R_t^+ = the total number of resources that are available to be used in time period $t = R_t + \widehat{R}_t$;

To keep track of the available tasks, we define:

$\widehat{\mathcal{L}}_t$ = the set of tasks that first become available in time period t;

$\mathcal{L}_t$ = the tasks available at time t before the new arrivals in $\widehat{\mathcal{L}}_t$ are added to the system;

$\mathcal{L}_t^+$ = the set of tasks available to be serviced at time t, including new tasks that just arrived in this period $= \mathcal{L}_t \cup \widehat{\mathcal{L}}_t$;

$\mathcal{L}_{it}^+$ = the set of tasks that must be served by resources in state i ($\mathcal{L}_t^+ = \bigcup_{i\in\mathcal{I}} \mathcal{L}_{it}^+$);

b_ℓ = the vector of attributes of task $\ell \in \mathcal{L}_t^+$. We assume that the sequence $(b_\ell)_{\ell\in\mathcal{L}}$ is drawn from a known distribution;

$\mathcal{T}_l$ = the set of time periods over which task l may be served. If the task is not served within this interval (the time window), then it is assumed lost.

Our care in distinguishing between the sets $\mathcal{L}_t^+$ and $\mathcal{L}_t$ becomes important later when we define our state variable. Note that $\mathcal{L}_t$ is empty if all tasks must be served at a single point in time. This observation is pertinent when we compare experiments where all the tasks must be served at a point in time to those where there are time windows.

Let $W_t = (\widehat{R}_t, \widehat{\mathcal{L}}_t)$ represent the new information arriving in time period t. $(W_t)_{t=0}^T$ is our stochastic information process, with realization $W_t(\omega) = \omega_t = (\widehat{R}_t(\omega), \widehat{\mathcal{L}}_t(\omega))$. Let $\mathcal{H}_t$ be the σ-algebra generated by $(W_0, \dots, W_t)$, with $\mathcal{H} = \mathcal{H}_T$. Then our probability space is given by $(\Omega, \mathcal{H}, \mathcal{P})$, where $\mathcal{P}$ is a probability measure defined on $(\Omega, \mathcal{H})$.

Decision Variables

At each time period, a resource must service a task, reposition, or sit idle. To service a task, we assume that the resource and task are matched at the task origin and then move to the task destination. A resource repositions simply by moving from one location to another. The decision to sit idle is equivalent to a reposition from location i to location i that takes one period to complete (a reposition across time, but not space).

To formalize these decisions, we define for each $t \in \mathcal{T}$, $i, j \in \mathcal{I}$ and $l \in \mathcal{L}_t^+$,

$$x_{lt} = \begin{cases} 1 & \text{If a resource is assigned to a task } l \in \mathcal{L}_t^+ \text{ at time } t, \\ 0 & \text{otherwise,} \end{cases}$$

y_{ijt} = The number of resources repositioned from i to j beginning at time t.

We also define the vectors $x_t \equiv (x_{lt})_{l\in\mathcal{L}_t^+}$ and $y_t \equiv (y_{ijt})_{i,j\in\mathcal{I}}$.

State Variables

The relevant history of our system is captured by the state variable, giving the set of available resources and tasks:

$$S_t^+ = \left\{R_t^+, \mathcal{L}_t^+\right\}.$$

We have not found this version of the state variable to be the most useful. As an alternative, we define:

$$S_t = \{R_t, \mathcal{L}_t\}.$$

We refer to S_t as an *incomplete* state variable because it does not contain all the information required to make a decision. In the special case that all tasks must be served immediately or they are lost, the set $\mathcal{L}_t$ is empty (because it does not yet include the new arrivals $\widehat{\mathcal{L}}_t$). In general, $\mathcal{L}_t$ includes only the tasks left over from previous time periods, which in many problems is relatively small. Note that our decisions (x_t, y_t) are $\mathcal{H}_t$-measurable functions. Since S_t^+ includes W_t, (x_t, y_t) are random variables if we only know S_t.

We note that S_t is an $\mathcal{H}_{t-1}$-measurable function. In the future, we may write $S_{t+1}(\omega)$, and it needs to be understood that this is a function of $(\omega_0, \omega_1, \ldots, \omega_t)$ or, in our case, simply ω_t.

Objective Function

The cost parameters used to evaluate decisions are

c_{ij} = the cost to reposition a resource from location i to location j;

r_{lt} = the reward received for servicing task $l \in \mathcal{L}_t^+$ starting at time $t \in \mathcal{T}$.

The decision to sit idle is represented by y_{ii} at a cost c_{ii} (which is often zero). Let $g_t(x_t, y_t)$ be the one-period reward function:

$$\begin{aligned} g_t(x_t, y_t) &= \sum_{i\in\mathcal{I}} \sum_{l\in\mathcal{L}_{it}^+} r_{lt} x_{lt} - \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{I}} c_{ij} y_{ijt} \\ &= \text{the total profit gained from} \\ &\quad \text{the decisions made at time } t. \end{aligned} \tag{1}$$

We let the feasible region $\mathcal{X}_t(\omega)$ at time period t for a given outcome ω be given by:

$$\sum_{j\in\mathcal{I}} y_{ijt}(\omega) + \sum_{l\in\mathcal{L}_{it}^+(\omega)} x_{lt}(\omega) = R_{it} + \widehat{R}_{it}(\omega) \quad \forall\, i \in \mathcal{I}, \tag{2}$$

$$x_{lt}(\omega) \le 1 \quad \forall\, l \in \mathcal{L}_t^+(\omega), \tag{3}$$

$$x_{lt}(\omega) \ge 0 \quad \forall\, l \in \mathcal{L}_t^+(\omega), \tag{4}$$

$$y_{ijt}(\omega) \ge 0 \quad \forall\, i, j \in \mathcal{I}. \tag{5}$$

In addition, we have to express the dynamics of the system over time. For this purpose, we first need to define the set of tasks which are served:

$\mathcal{L}_t^e(x_t)$ = the tasks that *expire* in time period t, either because they are served or they hit the end of the time window.

The dynamics of the resources and tasks are then given by:

$$\mathcal{L}_{t+1} = \mathcal{L}_t^+ \setminus \mathcal{L}_t^e, \tag{6}$$

$$R_{j,\,t+1}(\omega) = \sum_{i\in\mathcal{I}} \left(y_{ijt}(\omega) + \sum_{l\in\mathcal{L}_{ijt}^+(\omega)} x_{lt}(\omega) \right) \quad \forall\, j \in \mathcal{I}. \tag{7}$$

The objective is to maximize expected profits over the T-stage planning horizon given an initial state S_0 . We may write the problem as:

$$\max_{x_0,\, y_0 \in \mathcal{X}_0} g_0(x_0, y_0) + E\left\{ \sum_{t\in\mathcal{T}\setminus 0} \max_{(x_t, y_t)\in\mathcal{X}_t} g_t(x_t, y_t) \right\}. \tag{8}$$

The functions g_t, decisions (x_t, y_t), and constraints $\mathcal{X}_t$ are all $\mathcal{H}_t$-measurable functions. The system dynamics are given by Equations (6)–(7). Of course, we now face the problem of solving this formulation.

2. Solution Strategy

In practical applications for this problem class, the most natural solution strategy is to use a rolling-horizon procedure, solving the problem at time t using what is known at time t and a forecast of future events over some horizon. Our goal is to produce a method that outperforms this basic approach. We begin by writing the optimality equations using Bellman's principle of optimality. For this purpose, we have to choose how to represent the state of our system. If we use S_t^+, then we have all the information we need to make decisions at time t; S_{t+1}^+ is random only because it includes the new information arriving in time period $t+1$. Using this formulation, we would obtain the recursion:

$$V_t^+(S_t^+) = \max_{(x_t, y_t)\in\mathcal{X}_t} g_t(x_t, y_t) + E\{V_{t+1}^+(S_{t+1}^+) | S_t^+\}. \tag{9}$$

In this formulation, the expectation is conditioned on the prior state S_t^+. We refer to $V_t^+(S_t^+)$ as the *value function* and assume a terminal reward of $V_T^+(\cdot) = 0$.

The problem with Equation (9) is that it is not computationally tractable. In this form, it suffers from the

three curses of dimensionality: the state space, the outcome space, and the action space. We solve these problems in the following way. First we reformulate the optimality equations in terms of the incomplete state variable:

$$V_t(S_t) = E\left\{\max_{(x_t, y_t)\in\mathcal{X}_t} g_t(x_t, y_t) + V_{t+1}(S_{t+1})\Big|S_t\right\}. \quad (10)$$

This allows us to drop the expectation and solve the problem for a single sample realization:

$$V_t(S_t, \omega) = \max_{(x_t(\omega), y_t(\omega))\in\mathcal{X}_t(\omega)} g_t(x_t(\omega), y_t(\omega)) + V_{t+1}(S_{t+1}(\omega)). \quad (11)$$

If we had retained the original formulation using the complete state variable, dropping the expectation would have meant that we were solving the problem for a sample realization ω_{t+1}, which would mean that we would be choosing x_t using ω_{t+1}, a violation of the measurability of our decision function. It is important to remember that S_{t+1} is $\mathcal{H}_t$-measurable.

Finally, we replace the value function $V_{t+1}(S_{t+1})$ with an approximation $\widehat{V}_{t+1}(R_{t+1})$ that depends only on the resource vector. Writing $\widehat{V}_{t+1}(R_{t+1})$ purely as a function of R_{t+1} (and excluding $\mathcal{L}_{t+1}$) is an approximation that will introduce errors, but it simplifies the algorithm considerably and, as we show below, produces very good results. This gives us:

$$\widetilde{V}_t(R_t, \omega) = \max_{(x_t(\omega), y_t(\omega))\in\mathcal{X}_t(\omega)} g_t(x_t(\omega), y_t(\omega)) + \widehat{V}_{t+1}(R_{t+1}(\omega)) \quad (12)$$

where $\widetilde{V}_t(R_t, \omega)$ serves as a placeholder. This problem is solved subject to:

$$\sum_{j\in\mathcal{I}} y_{ijt}(\omega) + \sum_{l\in\mathcal{L}_{it}^+(\omega)} x_{lt}(\omega) = R_{it} + \widehat{R}_{it}(\omega) \quad \forall i \in \mathcal{I}, \quad (13)$$

$$\sum_{i\in\mathcal{I}}\left(y_{ijt}(\omega) + \sum_{l\in\mathcal{L}_{ijt}^+(\omega)} x_{lt}(\omega)\right) = R_{j,t+1}(\omega) \quad \forall j \in \mathcal{I}, \quad (14)$$

$$x_{lt}(\omega) \le 1 \quad \forall l \in \mathcal{L}_t^+(\omega), \quad (15)$$

$$x_{lt}(\omega) \ge 0 \quad \forall l \in \mathcal{L}_t^+(\omega), \quad (16)$$

$$y_{ijt}(\omega) \ge 0 \quad \forall i, j \in \mathcal{I}. \quad (17)$$

The first constraint (13) enforces conservation of flow by specifying that each resource at location i at time t must either reposition (y_{ijt}) or service a task (x_{lt}) (since we are implicitly conditioning on S_t, we do not index R_{it} by ω). Constraint (14) counts how many resources move to location j at time $t+1$. Constraint (15) specifies that each task cannot be assigned more than once. Constraints (16) and (17) are the requisite nonnegativity constraints.

We now face the challenge of devising a suitable approximation, $\widehat{V}_t(R_t)$. The simplest approximation is a linear one, but these can be unstable. Instead, we prefer to use a nonlinear (piecewise linear) separable approximation that we express as:

$$\widehat{V}_t(R_t) = \sum_{i\in\mathcal{I}} \widehat{V}_{it}(R_{it}). \quad (18)$$

Our choice of this approximation is motivated by two issues. First, it is possible to show that the real value function is piecewise linear and concave with respect to R_t. This property follows from the well-known result that a nonlinear program with concave objective function is a concave function of the linear right-hand-side constraint vector (see Rockafellar 1972). Second, as we illustrate later, this form of approximation means that Equations (12)–(17) form a network problem. If we ensure that the breakpoints of the piecewise linear function $\widehat{V}_{it}(R_{it})$ occur at integer values of R_{it}, then the solution of our approximate subproblem is also guaranteed to be integer.

In the next section, we describe the algorithm that we use to estimate $\widehat{V}_{it}(R_{it})$ using dual information obtained from solving our sample subproblem.

3. Fitting Concave Functional Approximations

In the previous section, we formulated a dynamic program using a separable concave approximation of $V_t(R_t)$. In this section, we describe methods for estimating functions of this type. We assume that we can readily compute a valid stochastic subgradient $v(\omega) \in \partial V(R, \omega)$ or, in special cases, left and right derivatives that we denote:

$$v^+(\omega) = V(R+1, \omega) - V(R, \omega),$$

$$v^-(\omega) = V(R, \omega) - V(R-1, \omega).$$

Since $V(R,\omega)$ is a concave function, we obtain the property that $v^-(\omega) \geq v^+(\omega)$. At each iteration n of our algorithm, we sample the variable R^n, then choose a sample observation ω^n and calculate the slope $v^n(\omega^n)$ (or possibly the left and right gradients $v^{n-}(\omega^n)$ and $v^{n+}(\omega^n)$) of the function $V(R,\omega)$ for $R = R^n$. Given this information, we want to update our estimate of a concave function $\widehat{V}^n(R)$ to obtain a new concave estimate $\widehat{V}^{n+1}(R)$.

There is a family of algorithms for estimating piecewise linear concave functions from stochastic subgradients which maintain concavity at each iteration. In our numerical experiments we use the CAVE algorithm of Godfrey and Powell (2001), but the technical details of this algorithm, while useful in practical experiments, disguise the essential ideas. For this reason, we briefly describe simpler algorithms that accomplish the same function, and finally present the details of the CAVE algorithm itself.

The simplest version of the algorithm would be a piecewise linear version of the SHAPE algorithm. Assume that we start with a concave function such as any one of the following:

$$f(R) = \rho_0\left(1 - e^{-\rho_1 R}\right),$$
$$f(R) = \ln(R+1),$$
$$f(R) = -\rho_0(R-\rho_1)^2.$$

Now convert $f(R)$ into a piecewise linear function $\hat{f}(R)$, where the break points are defined at integer values of R, and let $\widehat{V}^0(R) = \hat{f}(R)$ be our initial approximation. Next let $\partial\widehat{V}^n(R^n)$ be any valid subgradient of our approximate function $\widehat{V}^n(R)$ evaluated at $R = R^n$. Our updating procedure at each iteration n would be:

$$\widehat{V}^{n+1}(R) = \widehat{V}^n(R) + \alpha^n(v^n(\omega^n) - \partial\widehat{V}^n(R^n))R, \qquad (19)$$

where α^n is the stepsize at iteration n. Equation (19) effectively adds a concave approximation to a linear correction term, which obviously maintains concavity. If we have access to left and right derivatives of $\widehat{V}^n$ (which we do for the problem addressed in this paper), then we can use a two-sided updating procedure:

$$\widehat{V}^{n+1}(R) = \begin{cases} \widehat{V}^n(R) + \alpha^n(v^{n+}(\omega^n) - \partial\widehat{V}^{n+}(R^n))R, & R \geq R^n, \\ \widehat{V}^n(R) + \alpha^n(v^{n-}(\omega^n) - \partial\widehat{V}^{n-}(R^n))R, & R \leq R^n. \end{cases} \qquad (20)$$

Equation (20) reflects a weighted average of two concave functions, which clearly maintains concavity.

An alternative algorithm works as follows. Let $v^n(R)$ for $R = 0, 1, \ldots$ be the estimates, at iteration n, of the slopes of the function $\widehat{V}^n(R)$. Assume that for each (integer) value of R, we are going to sample R^n and ω^n, calculate $v^n = v^{n+}(R^n, \omega^n)$, and update the right gradient (that is, the slope to the right of R^n) using:

$$\hat{v}^{n+1}(R^n) = (1-\alpha^n)\hat{v}^n(R^n) + \alpha^n v^n(\omega^n).$$

At this point, we have not updated the slopes for any other value of R. If the resulting function is concave, we stop. If not, then assume that concavity is violated because $\hat{v}^n(R^n) < \hat{v}^n(R^n+1)$. We are now going to smooth the estimate v^n over the function $\widehat{V}^n(R)$ for values of R larger than R^n, but only over a range large enough to maintain concavity (the converse is true if $\hat{v}^n(R^n) > \hat{v}^n(R^n-1)$). The difference between this version of the algorithm and the one- (or two-) sided SHAPE algorithm is that the SHAPE algorithm updated the entire function, whereas now we are only updating the function locally.

The updating interval is given by:

$$\mathcal{J}^{n+} = \left\{j : R^n < j \leq R^{\max}, \hat{v}^{n+1}(R^n) < \hat{v}^n_j\right\},$$
$$\mathcal{J}^{n-} = \left\{j : 0 \leq j < R^n, \hat{v}^n_j < \hat{v}^{n+1}(R^n)\right\}.$$

We then update the slopes within the updating sets $\mathcal{J}^{n+}$ and $\mathcal{J}^{n-}$:

$$\hat{v}^{n+1}_j = \hat{v}^{n+1}(R^n), \quad \text{for all } j \in \mathcal{J}^{n+} \cup \mathcal{J}^{n-},$$
$$\hat{v}^{n+1}_j = \hat{v}^n(j), \quad \text{for all } j \notin \mathcal{J}^{n+} \cup \mathcal{J}^{n-} \cup \{R^n\}.$$

This algorithm maintains concavity by simply lowering any slope to the right of R^n that violates concavity relative to the slope $\hat{v}^{n+1}(R^n)$, or raising any slope to the left of R^n that violates concavity on the left side.

The CAVE algorithm is closest in spirit to the second algorithm. It differs in two respects. First, rather than simply raising or lowering slopes that violate concavity, it smooths the new estimate v^n with the current estimates of slopes in the sets $\mathcal{J}^{n+}$ and $\mathcal{J}^{n-}$. Second, instead of maintaining an estimate of the slope for each possible value of R (we encounter scaling problems if $R^{\max}$ is large), it sequentially adds break points at intermediate points in the function, making it easier to use for problems where $R^{\max}$ might be small or quite large (in real problems, it is possible to have depots where $R^{\max}$ is quite large, in addition to customer locations where $R^{\max}$ is very small (and often zero)).

The detailed steps of the CAVE algorithm for updating the piecewise linear approximation appear in Figure 2. Three parameters control the amount of change in $\widehat{V}(R)$ at each iteration. We define the parameters $\varepsilon^-, \varepsilon^+$ such that $[R-\varepsilon^-, R+\varepsilon^+)$ is the smallest updating interval for the approximation. We also define α as the smoothing parameter used to update the segment slopes. At each iteration, we apply declining step-size rules to $\varepsilon^-, \varepsilon^+$, and α for stability (see §4 for details).

An illustration of the smoothing intervals for a particular state and set of gradient estimates appear in Figures 3a and 3b. In Figure 3a, the right gradient at $R = s$ produces a slope that is less than that in the interval from u^2 to u^3. Hence, the algorithm smooths the slope all the way up to u^3. Figure 3b illustrates the same situation for the left gradient.

STEP 1 Initialization:

- Let $\mathcal{N} = \{0\}$, where $\nu^0 = 0, u^0 = 0$.
- Initialize parameters $\varepsilon^-, \varepsilon^+, \alpha$.

STEP 2 Collect Gradient Information:

- Given a state $R \geq 0$, find the gradients v^- and v^+ for a given $\omega \in \Omega$.

STEP 3 Define Smoothing Interval:

- Let $n^- = \min\{n \in \mathcal{N} : \nu^n \leq (1-\alpha)\nu^{n+1} + \alpha v^-\}$ and $n^+ = \min\{n \in \mathcal{N} : (1-\alpha)\nu^{n-1} + \alpha v^+ > \nu^n\}$.
- Define the smoothing interval $\mathcal{Q} = \left[\min\{s-\varepsilon^-, u^{n^-}\}, \max\{s+\varepsilon^+, u^{n^+}\}\right)$.
- Create new breakpoints at R and the endpoints of $\mathcal{Q}$ as needed. Since a new breakpoint always divides an existing segment, the segment slopes on both sides of the new breakpoint are the same initially.

STEP 4 Perform Smoothing:

- For each segment in the interval $\mathcal{Q}$, update the slope according to $\nu^n_{new} = \alpha\pi + (1-\alpha)\nu^n_{old}$ where $\pi = v^-$ if $u^n < R$ and $\pi = v^+$ otherwise.
- Adjust $\varepsilon^-, \varepsilon^+, \alpha$ according to step size rules.
- Return to Step 2.

Figure 2 **The Concave Adaptive Value Estimation (CAVE) Algorithm**

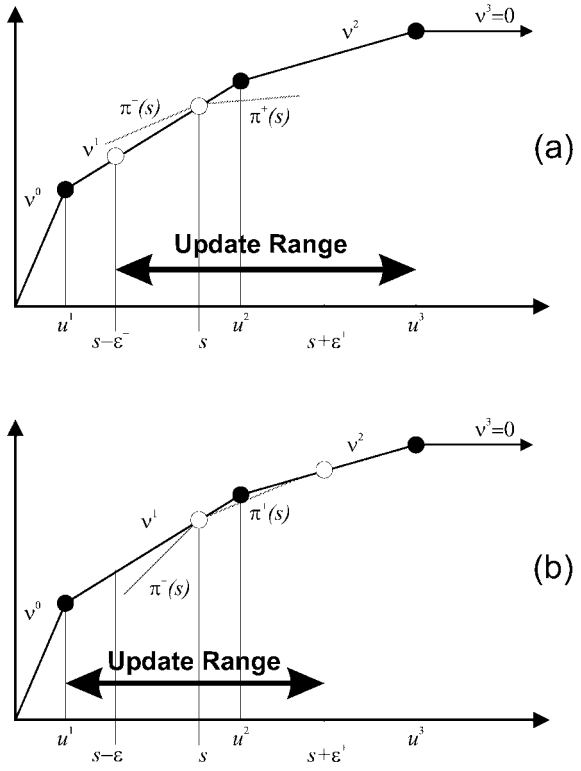


Figure 3 **Two Examples of Smoothing Intervals for the CAVE Update (from Godfrey and Powell 2001)**

A particularly important feature of the CAVE algorithm is that if we require that ε^+ and ε^- be strictly positive integers, then the breakpoints always occur at integer values of R. This logic will work if R ranges between 0 and 5, or 0 and 5,000. In the limit as ε decreases, we require that it never be smaller than one. Later, we show that this ensures that our approximation will always produce integer solutions.

Among the properties of the CAVE algorithm proven in Godfrey and Powell (2001) is that the concavity of the approximation is preserved under the updating rules. However, they applied the CAVE technique only to two-stage problems. In §4, we extend the methodology to use the CAVE algorithm to approximate the multistage dynamic program value function.

4. Value Function Approximation Method

In the previous sections, we formulated the SDRAP as a dynamic program and presented the CAVE algorithm for constructing piecewise linear approximations of one-dimensional concave functions. In this section, we bring the two ideas together to present an iterative algorithm for approximately solving the SDRAP.

The proposed algorithm has two parts, a forward simulation followed by an update. In the forward simulation, we generate a random future outcome $\omega \in \Omega$ and solve a sequence of network subproblems based on ω for $t = 0, 1, \ldots, T-1$ using the current approximation of the expected value function $V_t(R_t)$. We then use dual information derived from solving the network subproblems to update the value function approximations using the CAVE technique in §3.

The key to the forward simulation is constructing a concave separable piecewise linear approximation of the expected value function $V_t(R_t)$ in order to formulate each subproblem as a pure network. For each time period $t \in \mathcal{T}$, we approximate $V_t(R_t)$ by a sum of one-dimensional separable functions $\widehat{V}_t(R_t) = \sum_{i \in \mathcal{I}} \widehat{V}_{it}(R_{it})$. Note that this resource-centric approximation does not explicitly consider the task portion of the state. This approximation has no effect on problems with tight time windows, but does introduce errors as the time windows become wider.

Given the approximation $\widehat{V}_t(R_t)$, we generate a random outcome $\omega \in \Omega$ and solve a sequence of network subproblems from $t = 0$ through $t = T-1$:

$$\max_{x_t(\omega),\, y_t(\omega) \in \mathcal{X}_t(\omega)} g_t(x_t(\omega), y_t(\omega)) + \widehat{V}_{t+1}(R_{t+1}(\omega)). \qquad (21)$$

Prior work used a linear approximation for $\widehat{V}_t(R_t)$. Linear approximations are easy to calculate, and allow the problem to be decomposed into a sequence of easy subproblems (in this case, the subproblem is little more than a sort routine). However, linear approximations can be unstable. Powell and Carvalho (1998) apply an artificial control u_{ijt} to limit repositioning between locations i and j at time t. In contrast, we constrain the total flow into each destination using the concave piecewise linear approximation. These linear segments with decreasing marginal return control the flow more smoothly than the strict upper bound in previous work. However, our subproblem does not decompose by location, making it slightly harder to solve.

Figure 4 illustrates the time t subproblem as a two-stage, time-space network. In the first-stage decisions (time t, top half of Figure 4) each resource must: (1) service a task, (2) reposition, or (3) remain in inventory. Each node on the left-hand side of the network represents a location i with available resource supply R_{it}. From each origin node, we build three types of arcs: (1) service arcs to each task destination, (2) repositioning arcs to each destination within an allowable radius, and (3) inventory arcs to the same location one time period later. The task arcs have arc cost $+r_{lt}$ and upper bound $|\mathcal{L}^+_{ijt}(\omega)|$. The repositioning arcs have cost $-c_{ij}$ and upper bound $+\infty$. The first-stage network matches that of the myopic plan-

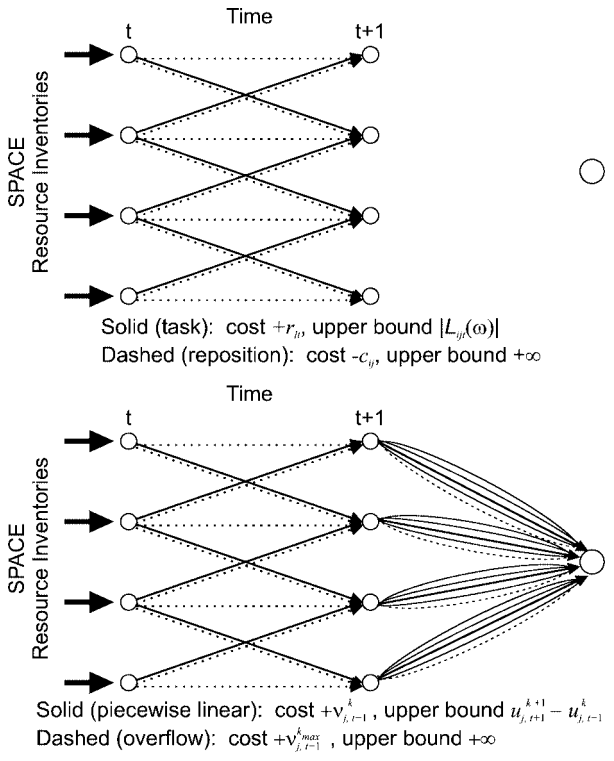


Figure 4 **Subproblem at Time t Expressed as a Network Problem in Two Stages, One for the Time t Decisions (Top) and One for the Time $t+1$ Decisions (Bottom)**

ner who will always choose to hold inventory (cost of 0) over repositioning (cost > 0).

The second-stage decisions (time $t+1$, bottom half of Figure 4) incorporate the CAVE approximation $\widehat{V}_{j,t+1}$. From each of the destination nodes $(j, t+1)$, we build one arc for each piecewise linear segment in $\widehat{V}_{j,t+1}$ to the supersink node. In general, the nth arc $(n = 0, 1, \ldots, n^{\max})$ has cost $+\nu^n_{j,t+1}$ and upper bound $u^{n+1}_{j,t+1} - u^n_{j,t+1}$ where $u^{n^{\max}+1}_{j,t+1} \equiv \infty$. The last arc always has a finite cost (usually zero) and infinite upper bound to absorb overflow and to ensure that the network always has a feasible solution.

Several observations follow. First, all arc capacities and node supplies/demands are naturally integer; as a result, the optimal solution to the network is also integer. Second, all resources flow to one destination node, then immediately to the supersink node. Third, repositioning flow is not constrained by the first-stage repositioning arcs, but instead by economic incentives in the second-stage arcs connecting the destination nodes to the supersink.

Finally, since $\nu^n_{j,t+1}$ decreases monotonically with n, all resources flow through the destination node in the order prescribed by $\widehat{V}_{j,t+1}$. That is, resources flow through the highest marginal link up to its upper bound, then through the next highest marginal link, and so on with the overflow link used only as a last resort. This is why it is critical that the CAVE algorithm preserve concavity. Otherwise, the value function approximation is not properly represented by the second-stage arcs.

This network structure leads to fast integer optimal solutions using a network solver. In addition, we also get dual values on all of the nodes. Of particular interest are the flow-augmenting and flow-decrementing dual values and paths introduced by Powell (1989) on the first-stage nodes. These values, call them v^+_{it} and v^-_{it}, respectively, give the marginal benefit (or loss) of one more (or one fewer) resource at each location. These dual values are used to update the value function approximation $\widehat{V}_{it}$ via the CAVE algorithm.

As a final step, we update the state from S^+_t to S^+_{t+1} by implementing the first-stage, time t network solution and adding new resources and tasks from $\widehat{R}_{t+1}$ and $\widehat{\mathscr{L}}_{t+1}$. We continue generating and solving this sequence of subproblems through $t = T-1$. Further iterations are run using new random samples as desired. In Figure 5, we summarize the steps of the algorithm.

STEP 1 Initialization:

- Set $\nu^0_{it} = 0$ and $u^0_{it} = 0$ as the piecewise-linear approximation $\hat{V}_{it}(R_{it})$ for all $i \in \mathcal{I}, t \in \mathcal{T}$.

STEP 2 Forward Simulation and CAVE Update:

- Generate a random sample $\omega \in \Omega$.
- For each time period $t = 0, 1, \ldots, T-1$,
  - Determine the new tasks $\hat{\mathcal{L}}_t(\omega)$ arriving to the system.
  - Solve the network subproblem $\max_{(\mathbf{x}_t, \mathbf{y}_t) \in \mathcal{X}_t} g_t(\mathbf{x}_t, \mathbf{y}_t) + \hat{V}_{t+1}(R_{t+1}(\omega))$.
  - Store dual values v^-_{it} and v^+_{it} associated with the first-stage nodes.
  - Update the value function approximation using $\hat{V}_{j,t} \leftarrow U^{CAVE}(\hat{V}_{jt}, v^+_{jt}, v^-_{jt}, R_{jt})$, using the CAVE updating rules in figure 2.
- Repeat Step 2 as desired.

Figure 5 The Multistage Stochastic Value Approximation Algorithm Using CAVE

5. Experimental Design and Results

The following single period travel time experiments address three key research questions. First, how does the solution quality using the CAVE logic compare against other algorithms for solving (or placing a bound on) the deterministic version of the problem over a wide variety of data sets? By deterministic, we mean a problem in which the future tasks are assumed to be known. Second, how does the CAVE algorithm compare in terms of CPU time, measured as a function of solution quality? The CPU time/solution quality trade-off is important to measure because the real-world version of this problem typically uses the generated solutions within a rolling-horizon environment. Third, for stochastic problems in which the future tasks are not known, how does the CAVE approximation, which can be constructed via resampling, compare to a deterministic approximation implemented on a rolling-horizon basis? It is this last experiment that is most interesting, since the CAVE algorithm readily handles uncertainty and returns integer solutions for this problem class.

We begin our experiments with a discussion of the generation of the data sets in §5.1. Next, §5.2 reports on runs using deterministic data, where we take advantage of the ability to obtain an optimal LP

relaxation to serve as a tight benchmark. Finally, §5.3 demonstrates the value of the method in the context of stochastic data by comparing the algorithm with a classical rolling-horizon approximation.

5.1. Experimental Design

We can divide our experiments along two major dimensions: deterministic vs. stochastic, and problems where tasks can be served within a time window ($|\mathcal{T}_l| > 1$) vesus those where the task must be served at a particular point in time ($|\mathcal{T}_l| = 1$). In both dimensions, we must compare the solution using our value function approximation to some benchmark. The deterministic case that has the tasks served at a point in time is a pure network, so we can easily obtain integer solutions. For the deterministic case with time windows we have to resort to an LP relaxation, which provides only an upper bound. Also, our value function approximation is purely a function of R_t, and not $\mathcal{L}_t$. When the time windows are tight, the set $\mathcal{L}_t$ is always empty (remember these are the tasks left over from the previous time period). Thus, the approximation introduced by ignoring $\mathcal{L}_t$ only arises when there are time windows.

When we conduct stochastic experiments we do not have access to optimal solutions, so we instead compare against rolling-horizon procedures, which represent the standard approach for these problems in practice. We then normalize the results for both our engineering approximation and the rolling-horizon procedure using a posterior bound obtained by solving each problem to optimality after all the information is known.

Two different sets of deterministic experiments are considered. In the first set, the number of resources and the cost structure varies. In the second set, the number of locations and planning-horizon length varies. In both cases, a commercial linear programming package generates reasonably tight upper bounds as a benchmark (if there are no time windows, the LP relaxation provides integer optimal solutions). For the stochastic experiments, the number of resources, the cost structure, and the planning-horizon length varies. All experiments are run on a 194-MHz Silicon Graphics Challenge R10000 workstation.

Table 1 contains the attributes for the first set of deterministic experiments. The 20 locations are scattered uniformly across space, with single period travel times between all locations. Using a Poisson process that is uniform over the 30-period planning horizon, we generated a demand sample containing 1,992 tasks.

Table 1 Parameters for the First Set of Deterministic Experiments

Problem Characteristic	Attribute Value(s)
Number of locations, $\|\mathcal{I}\|$	20 locations
Planning-horizon length, T	30 periods
Number of tasks over T	1,992 tasks
Time window length (fixed)	6 periods
Net task revenue per mile	\$1.00 per mile
Origin/destination weights	Negatively correlated
Repositioning cost per mile	\$0.80, \$1.40, \$2.00 per mile
Number of resources	50, 100, 200, 400 resources

Care was taken in the generation of the sets of tasks. If we had generated tasks randomly around the network, we would, on average, find that the total flow into a city was approximately the same as the total flow out. Such a problem would approximately decompose by location, making the problem too easy. To avoid this possibility, we randomly generated a weight $\beta_i \sim U[0, 1]$. We then made the demand from i to j proportional to $\beta_i(1-\beta_j)$. In this way, β_i is being used as a measure of the ability of location i to generate tasks, and $(1-\beta_i)$ is a measure of the ability of location i to attract tasks. This approach creates a negative correlation between outbound tasks and inbound tasks. This behavior will create significant surpluses and deficits of resources, requiring significant levels of repositioning from one location to the next.

Figure 6 illustrates the negative correlation at the 20 locations. Each location is represented by a dot that plots the number of tasks originating at that location against the number of tasks that terminate at that location. Problems with negatively correlated demand weights require more repositioning to obtain high-quality solutions (and are therefore harder to solve) than problems with independent or positively correlated weights (these problems tend to reduce into spatially separable problems that are much easier to approximate). The presence of significant levels of repositioning means that the problem is highly nonseparable in the inventories across locations.

Once a task is called into the system, it must be satisfied within six periods to receive the revenue. The

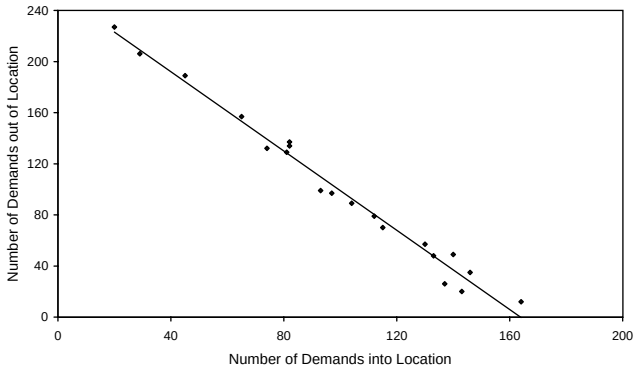


Figure 6 **Negative Correlation Between the Number of Tasks into and Out of Each Location**

net revenue per task and cost for repositioning is proportional to the distance between the two locations. For this set of experiments, the net revenue is fixed at \$1.00 per mile, and for separate runs, the repositioning cost per mile is fixed at \$0.80, \$1.40, and \$2.00. In the case of \$0.80, drivers realize a net profit by repositioning from location A to location B and then servicing a task from B back to A. The higher repositioning costs progressively restrict the incentive to reposition. However, we will show that these costs are not unduly restrictive. Even for the \$2.00 case, the negatively correlated tasks encourage several hundred repositions to occur.

The cost of repositioning is an important parameter in the evaluation of the CAVE approximation. If the repositioning cost is too high, then the problem decomposes by location, since resources in one location are not used to service tasks in another. We expect the CAVE algorithm to work better as the problem becomes more separable.

The number of resources is fixed at 50, 100, 200, or 400 for each simulation. All resources are considered to be available at the start of the simulation (time 0). The number of resources at each location j at the start of the simulation is proportional to the destination weight $1-\beta_j$. This initial allocation of resources reduces the start-up effect of the simulation.

5.2. Deterministic Experiments

Taking the combination of three repositioning costs and four resource sizes, we have 12 different data sets to evaluate the performance of the CAVE logic. We solve each problem using four algorithms: (1) the CAVE logic with a piecewise linear value function approximation (denoted CAVE), (2) the CAVE logic with a strictly linear value function approximation (denoted Linear), (3) the Linear Approximation and Multiplier Adjustment (LAMA) method of Carvalho and Powell (2000), and (4) the commercial linear programming package CPLEX. To present the deterministic formulation, we define:

$\mathscr{L}$ = the set of all tasks to be served over the horizon;

$\mathscr{L}_{ij}$ = the set of all tasks with origin i and destination j;

$\mathscr{L}_i = \bigcup_{j\in\mathscr{I}} \mathscr{L}_{ij}$.

The deterministic formulation of the problem is now given by:

$$\max_{x,y,R}\left\{\sum_{t\in\mathscr{T}} g_t(x_t, y_t)\right\} \tag{22}$$

subject to:

$$\sum_{j\in\mathscr{I}} y_{ijt} + \sum_{l\in\mathscr{L}_i} x_{lt} - R_{it} = 0 \quad \forall\, i\in\mathscr{I}, t\in\mathscr{T}, \tag{23}$$

$$\sum_{i\in\mathscr{I}}\left(y_{ijt} + \sum_{l\in\mathscr{L}_{ij}} x_{lt}\right) - R_{j,t+1} = 0 \quad \forall\, j\in\mathscr{I}, t\in\mathscr{T}, \tag{24}$$

$$\sum_{t\in\mathscr{T}_l} x_{lt} \le 1 \quad \forall\, l\in\mathscr{L}, \tag{25}$$

$$x_{lt} \ge 0 \quad \forall\, l\in\mathscr{L}, t\in\mathscr{T}, \tag{26}$$

$$y_{ijt} \ge 0 \quad \forall\, i,j\in\mathscr{I}, t\in\mathscr{T}. \tag{27}$$

This is basically the same formulation as the stochastic case, with obvious changes to reflect the lack of uncertainty. Perhaps the most important difference, however, is Equation (25) which is the only nonnetwork constraint. If there are no time windows (which is to say $|\mathscr{T}_l| = 1$), then this problem drops out. However, for the cases where we considered time windows, it means that the LP solver no longer returns integer solutions. We believe the LP relaxation is quite tight.

For the CAVE runs, we use a step size of $\alpha = 20/(40+n)$ for the nth iteration and minimal updating intervals of $\varepsilon^- = \varepsilon^+ = 2$ for the first 10 iterations and $\varepsilon^- = \varepsilon^+ = 1$ thereafter. Interestingly, the CAVE results are not much different using a constant step

size such as 0.2. For the linear runs, we use a step size of $\alpha = 2/(4+n)$ with the minimal updating interval $\varepsilon^- = \varepsilon^+ = 100$. Using this minimum updating interval and v_{it}^+ for both dual values ensures a linear approximation over a practical domain ($R_{it} \leq 100$).

We evaluate the algorithms based on two statistics: (1) the best objective function value after 1,000 iterations of CAVE and Linear (approximately 240 CPU seconds) and after 5,000 iterations of LAMA (approximately 1,200 CPU seconds), and (2) the CPU time required to reach different percentages of the CPLEX relaxed optimal solution. The extra LAMA iterations are needed due to its slow convergence as we show.

Table 2 lists the best objective function value (relative to CPLEX) obtained by the three competing algorithms as a function of the repositioning cost and the number of resources. The CAVE method is the most consistent of the algorithms, especially across different repositioning costs. However, as the number of resources decreases, so does the CAVE performance against the CPLEX bound. There are three primary factors that contribute to this degradation. The first is the nonintegrality of the CPLEX relaxed solution. For problems with 50 resources, about 50% of the nonzero variables in the CPLEX solution have integer values. For problems with 400 resources, about 90% of the nonzero values are integer. The second factor is the separable approximation (by location) of a nonseparable value function. The third factor is the exclusion of the task information in the value approximation state space. We investigate these factors further in the second set of experiments.

The Linear method performs well when resources are scarce because the pool of idle resources available for repositioning is small. However, unlike the CAVE method, the solution quality falls as the number of resources grows. The cause is a lack of restraint in repositioning decisions due to the linear approximation estimating the same marginal benefit to be gained regardless of how many resources reposition to that location. This also leads to wild fluctuations in solution quality from iteration to iteration that are not apparent in a table that lists only the best solution.

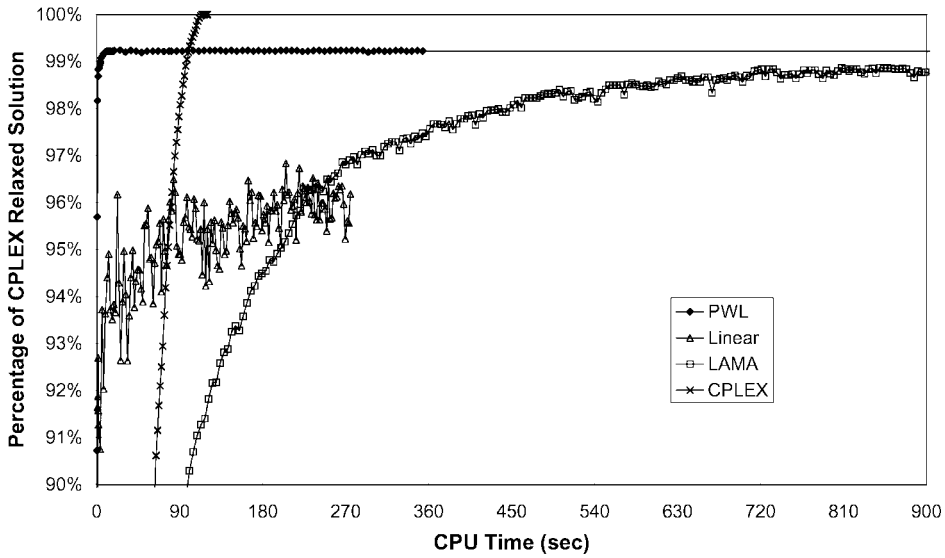
Finally, the LAMA method has the opposite problem from the Linear method. Due to the tight incremental control restricting repositioning, the LAMA method struggles when there are few resources, but does quite well when resources are plentiful. When the repositioning cost is low, the LAMA solution narrowly beats the CAVE solution for 100, 200, and 400 resources. However, as Figure 7 shows, this victory comes at a severe computational price.

Figure 7 illustrates the solution quality of each algorithm as a function of CPU time for the data set with 200 resources and \$1.40 per mile repositioning cost. Other data sets have the same general traits with differences in the final solution quality. For all of the data sets, the CPLEX solution required roughly 60 CPU seconds to reach 90% and another 30–40 CPU seconds to reach the optimal solution.

On the other hand, within five CPU seconds (20 iterations) the CAVE solution jumps to a 99% value, then stabilizes. This pattern of jumping to a near maximum within 5–10 CPU seconds and then stabilizing holds across all of the data sets. This fast convergence explains why it is not sensitive to reasonable step-size rules such as $20/(40+n)$ or 0.2, as long as the step size does not decrease too quickly. If the step size decreases too quickly, then the rate of convergence to high-quality solutions slows as well.

Table 2 Percentage of CPLEX Relaxed Optimal Value Obtained Using CAVE, Linear, and LAMA Algorithms for the First Set of Deterministic Experiments

No. of Resources	CAVE Method: Repositioning Cost			Linear Method: Repositioning Cost			LAMA Method: Repositioning Cost		
	\$0.80 (%)	\$1.40 (%)	\$2.00 (%)	\$0.80 (%)	\$1.40 (%)	\$2.00 (%)	\$0.80 (%)	\$1.40 (%)	\$2.00 (%)
50	97.6	97.4	97.4	97.6	97.2	97.1	89.7	90.6	91.4
100	98.6	98.7	98.7	97.2	98.0	98.2	98.8	98.6	98.1
200	99.2	99.2	99.1	96.2	96.9	98.1	99.5	99.1	98.8
400	99.4	99.6	99.4	86.6	95.6	93.0	99.6	99.1	99.1



Note. Not all iterations are plotted.

Figure 7 Objective Function Value Versus CPU Time for Deterministic Demand with 200 Resources and $1.40 Repositioning Cost per Mile

Table 3 CPU Time (in Seconds) Required to Reach 95%, 97%, 98%, and 99% of CPLEX Relaxed Optimal Value for the First Set of Deterministic Experiments

		Computation Time (CPU in seconds)					
		$0.80 Cost		$1.40 Cost		$2.00 Cost	
Resources	CPLEX Opt(%)	CAVE	LAMA	CAVE	LAMA	CAVE	LAMA
	95	1	471	1	267	1	154
100	97	1	703	1	474	1	435
	98	1	903	1	797	1	1233
	99	—	—	—	—	—	—
	Best	98.6%	98.8%	98.7%	98.6%	98.7%	98.1%
	95	1	316	1	197	1	106
200	97	1	417	1	287	1	235
	98	1	487	1	448	2	450
	99	3	741	4	2081	11	—
	Best	99.2%	99.5%	99.2%	99.1%	99.1%	98.8%
	95	1	13	1	18	1	13
400	97	1	29	1	59	1	19
	98	2	54	1	128	2	94
	99	3	115	4	1246	8	700
	Best	99.4%	99.6%	99.6%	99.1%	99.4%	99.1%

*— = threshold not reached.

Table 4 Attributes for the Second Set of Deterministic Experiments

Problem Characteristic	Attribute Value(s)
Number of locations, $\|\mathcal{I}\|$	20, 40, 80 locations
Planning-horizon length, T	15, 30, 60 periods
Number of tasks over T	Approximately 1,000, 2,000, 4,000
Time window length (fixed)	6 periods
Net task revenue per mile	\$1.00 per mile
Origin/destination weights	Negatively correlated
Cost per repositioning mile	\$1.40 per mile
Fleet size	200 resources

Table 3 compares the CAVE and LAMA solutions in terms of the CPU time required to reach a particular solution quality. The results for 50 resources are not included because the best LAMA solutions reach only around 90%. The general convergence pattern displayed in Figure 7 holds across all of the data sets. The main difference is the relatively faster convergence rate for LAMA with 400 resources, though it still lags behind CAVE.

The focus of the second set of deterministic experiments is the relationship between solution quality and CPU time as the physical size of the problem varies. The attributes for these experiments appear in Table 4. We vary two parameters, the number of locations and the length of the simulation. The number of locations is fixed at 20, 40, or 80 locations, with single period travel times between all locations. The geographic area of the network stays the same as before, but the density of locations in the space increases. The planning horizon length is fixed at 15, 30, or 60 periods. The fleet size is fixed at 200 resources with a repositioning cost of \$1.40 per mile. The number of tasks scales with the horizon length, leading to an approximate $2,000 : 30$ tasks-to-periods ratio as before. Since each of the nine data sets is generated independently, the exact task totals vary slightly from this ratio.

For each data set, we compare solutions generated by CAVE (1,000 iterations) and LAMA (5,000 iterations) against the CPLEX bound. One iteration of each algorithm takes approximately the same amount of CPU time. Table 5 lists the best objective function value (relative to CPLEX) obtained by CAVE and LAMA as a function of the number of locations and horizon length. The CAVE method does better for longer horizons and fewer locations, while LAMA has an advantage for shorter horizons and more locations.

Table 5 Percentage of CPLEX Relaxed Optimal Value Obtained Using CAVE and LAMA Algorithms for the Second Set of Deterministic Experiments

No. of Locations	CAVE Method: Horizon Length			LAMA Method: Horizon Length		
	15 (%)	30 (%)	60 (%)	15 (%)	30 (%)	60 (%)
20	99.0	99.2	99.5	99.1	99.1	98.7
40	98.2	98.4	98.9	98.8	98.9	98.8
80	97.5	97.0	97.6	98.8	98.0	96.8

The most noticeable effect was the drop in CAVE solution quality as the number of locations increases. This may be due in part to the increase in the integrality gap, but we suspect it is primarily due to the use of a separable approximation of a nonseparable value function.

If we collapse the time windows of the problem down to a single period, then this "no-time-window" problem can be solved to integer optimality as a pure network. Table 6 contains the no-time-window results and shows that the CAVE solutions are optimal or near optimal for this modified set of problems across all data parameters. Since there is exactly one period in which each task can be serviced, the task part of the system state is independent of the set of earlier decisions. In other words, for any set of decisions the task state stays exactly the same. In this case, adding the task information to the value approximation state is unnecessary since it does not vary.

The near-optimal results of Table 6 suggest that the previous optimality gap could be attributed to either an integrality gap or the lack of task information in the value function approximation. Further research is needed to answer this question.

Table 6 Percentage of Integer Optimal Value Obtained Using CAVE for the Second Set of Deterministic Experiments with Single Period Time Windows (Network Problems)

No. of Locations	Planning Horizon		
	15 (%)	30 (%)	60 (%)
20	100.00	100.00	100.00
40	100.00	99.99	100.00
80	99.99	100.00	99.99

Table 7 CPU Time (in Seconds) Required to Reach 95%, 97%, 98%, and 99% of CPLEX Relaxed Optimal Value for the Second Set of Deterministic Experiments

		Computation Time (CPU seconds)					
		15 Periods		30 Periods		60 Periods	
Locations	CPLEX Opt (%)	CAVE	LAMA	CAVE	LAMA	CAVE	LAMA
20	95	1	3	1	197	1	564
	97	1	5	1	287	2	839
	98	1	10	1	448	2	1050
	99	3	34	4	2,081	5	—
	Best	99.0%	99.1%	99.2%	99.1%	99.5%	98.7%
	CPLEX	14		119		1,059	
40	95	1	75	2	254	3	1,129
	97	2	110	4	318	5	1,509
	98	4	204	10	416	9	1,912
	99	—	—	—	—	—	—
	Best	98.2%	98.8%	98.4%	98.9%	98.9%	98.8%
	CPLEX	52		577		4,974	
80	95	4	230	11	878	14	3,652
	97	12	336	214	1,359	68	—
	98	—	503	—	3,540	—	—
	99	—	—	—	—	—	—
	Best	97.5%	98.8%	97.0%	98.0%	97.6%	96.8%
	CPLEX	120		1,785		11,587	

*—= threshold not reached.

Table 7 contains the CPU times for the full-time window CAVE and LAMA runs. The CPLEX CPU times are included for each location/horizon pair. Although the best LAMA solution dominates the best CAVE solution in half of the data sets, CAVE generates high-quality solutions an order of magnitude or two faster than LAMA. The CAVE solution time per iteration increases roughly linearly with the horizon length and slightly greater than linearly with the number of locations. The greater than linear increase with locations is due to the time needed to solve the intermediate network subproblems.

For the LAMA runs, the dramatic increase in time required to reach a specified solution quality as the horizon lengthens is surprising considering that the best LAMA solution over different horizons is quite similar, especially for 20 and 40 locations. The best guess is that as the horizon lengthens, the ability to estimate the marginal impact to the end of the horizon of an additional resource becomes more difficult and less accurate.

5.3. Stochastic Experiments

We now turn to the question of evaluating the performance of the CAVE approximation in the presence of uncertain future tasks. In these experiments, we do not consider LAMA because its control structure is not suited to stochastic problems. The data parameters in Table 8 are used to construct the stochastic experiment data sets and are similar to the first set of deterministic experiments. The main exception is the single period time window on the tasks that enables an optimal posterior bound to be calculated. In addition, as an objective comparison against CAVE we formulate a deterministic rolling-horizon algorithm that requires single period time windows.

All of our experiments use a 180-period simulation. We assume that the real-world future demand is represented by the outcome ω^0. In period t of the simulation, we will have observed only the first t periods of the outcome ω^0. We solve a T-period problem using resampling (stochastic training) or an expectation (deterministic solution) to produce decisions for times t through $t+T-1$ or 180, whichever is less.

Table 8 Parameters for Stochastic Experiments

Problem Characteristic	Attribute Value(s)
No. of locations, $\|\mathcal{I}\|$	20, 40, 80 locations
No. of resources	100, 200, 400 resources
Planning-horizon length, T	2, 3, 6, 9, 15, 30 periods
Simulation length	180 periods
No. of tasks over simulation	Approximately 12,000
Time window length (fixed)	1 period
Net task revenue per mile	\$1.00 per mile
Origin/destination weights	Negatively correlated
Repositioning cost per mile	\$1.40 per mile

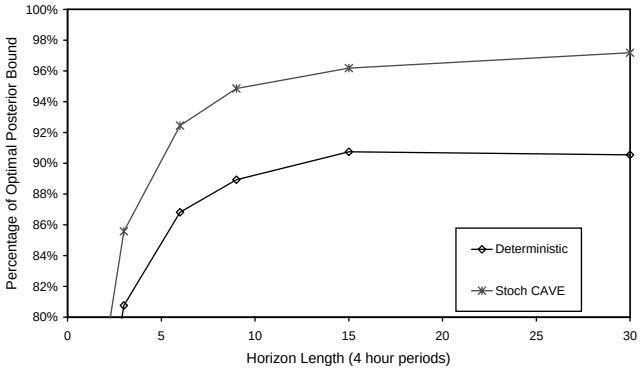


Figure 8 Rolling-Horizon Comparison of Deterministic Solution and Stochastic Training with 40 Locations and 200 Resources

We implement the first-period decisions, update the resources and tasks, and then start the process again at time $t+1$.

For the deterministic solution, we solve the T-period problem using a single expectation of the future. By *an* expectation, we mean a single demand sample that uses an integer value of the Poisson mean at each location and time period using randomized rounding. For example, if a pair of locations has a mean of 1.3 tasks during a given period, then a particular expectation outcome will have either one task (with probability 0.7) or two tasks (with probability 0.3). The purpose of the randomized rounding is to create an integer data set that is "closer" to the mean than a wider distribution Poisson sample.

Since the expectation has single period time windows, it can be solved as a deterministic network that yields an integer optimal solution under that future expectation. This approach is the best that one could ever do using a deterministic technique with an unknown future because the expectation comes from the same Poisson parameters that govern ω^0 and the solution technique provides integer optimal solutions. The remaining question is estimating the penalty for using the expectation (which is known) instead of ω^0 (which is not known).

For the stochastic training we solve the T-period problem using the multistage CAVE methodology with a different Poisson sample at each iteration. For repeatability, we pregenerated and stored 200 random outcomes from which we sampled at each iteration. For objectivity, ω^0 was not among the training samples. In the previous deterministic experiments, CAVE produced near-optimal solutions for pure network problems. However, we concluded that the near optimality was due in part to the task state being fixed. Under resampling, the task state changes every iteration, independent of the sequence of decisions and value approximation updates, making the value estimation more difficult.

For each combination of number of resources and locations, we perform stochastic and deterministic runs over the 180-period simulation. An optimal posterior bound is calculated by solving the entire simulation under the outcome ω^0 as a deterministic network. This posterior bound is used to scale the results of the simulations.

We undertook a series of experiments to determine an appropriate planning horizon. Figure 8 is typical of these experiments, showing the rolling-horizon results as a function of planning-horizon length for 40 locations and 200 resources. We concluded from these experiments that a 15-period planning horizon was "safe."

Table 9 summarizes the rolling-horizon results for the 15-period planning-horizon length. Using CAVE while resampling provides, on average, a 7% advantage (increasing dramatically as the number of locations increases) over solving a deterministic network problem using the expectation. In general, the CAVE solutions are halfway between the deterministic solution and the posterior bound. The CAVE advantage lies primarily with the superior repositioning decisions made under resampling. The purpose of repositioning is to move resources to locations with better

Table 9 Summary of Rolling-Horizon Results with a 15-Period Planning Horizon Using Deterministic and Stochastic Training

		Percentage of Posterior Bound	
No. of Locations	No. of Resources	Deterministic (Expectation) (%)	Stochastic Using CAVE (%)
20	100	92.2	96.3
20	200	96.3	97.8
20	400	96.6	98.1
40	100	81.0	90.5
40	200	90.7	96.2
40	400	92.6	96.8
80	100	66.3	82.1
80	200	81.4	93.3
80	400	84.8	94.5

future opportunities. However, if the future develops differently than expected, little benefit is gained from these movements. If resources are scarce, then the penalty is twofold: There is the wasted repositioning cost and the opportunity cost of the wrong decision.

The gap between resampling and using the expectation increases with the number of locations. Repositioning becomes more of a guessing game as the number of locations increases. Since resampling provides a richer view of the future than a single expectation, guessing under resampling is more likely to lead to better future opportunities. Also, the gap between resampling and using the expectation increases as the fleet size decreases. As before, it is due to the greater consequences of unnecessary repositioning when fewer resources are available.

6. Conclusions

We have made the following contributions to the solution of stochastic, dynamic resource allocation problems with single period travel times. First, we extended the CAVE methodology of Godfrey and Powell (2001) to solve multistage problems. Second, we showed that the CAVE logic provided consistently high-quality solutions within 0.5%–3% of a relaxed optimal bound on deterministic problems with a variety of resource sizes, repositioning costs, number of locations, and horizon lengths. In those cases where the solution quality was less than 99% of the relaxed bound, we argued that the reason was a combination of an integrality gap due to the relaxed bound (especially when resources are scarce), and the separable approximation of a nonseparable value function (especially as the number of locations increases), and the lack of task information in the approximation that we chose. Third, the CAVE method produces high-quality solutions (95%–99% of the relaxed bound) an order of magnitude or two faster than the competing LAMA method of Carvalho and Powell (2000) with similar final solution quality over a wide range of problems. Finally, we estimated the value of stochastic over deterministic modeling for stochastic problems with an uncertain future at 2%–16% where the realized benefit of CAVE increases as the number of resources decreases or as the number of locations increases.

Acknowledgments

This research was supported in part by Grant AFOSR-F49620-93-1-0098 from the Air Force Office of Scientific Research and by Grant DMI00-85368 from the National Science Foundation.

References

Bertsekas, D., J. Tsitsiklis. 1996. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA.

Birge, J. 1985. Decomposition and partitioning techniques for multistage stochastic linear programs. *Oper. Res.* **33**(5) 989–1007.

——, F. Louveaux. 1997. *Introduction to Stochastic Programming*. Springer-Verlag, New York.

Carvalho, T. A., W. B. Powell. 2000. A multiplier adjustment method for dynamic resource allocation problems. *Transportation Sci.* **34** 150–164.

Chen, Z-L., W. Powell. 1999. A convergent cutting-plane and partial-sampling algorithm for multistage linear programs with recourse. *J. Optim. Theory Appl.* **103**(3) 497–524.

Cheung, R. K-M., W. B. Powell. 1996. An algorithm for multistage dynamic networks with random arc capacities, with an application to dynamic fleet management. *Oper. Res.* **44**(6) 951–963.

——, ——. 2000. SHAPE: A stochastic hybrid approximation procedure for two-stage stochastic programs. *Oper. Res.* **48**(1) 73–79.

Culioli, J-C., G. Cohen. 1990. Decomposition/coordination algorithms in stochastic optimization. *SIAM J. Control Optim.* **28** 1372–1403.

Dantzig, G. 1955. Linear programming under uncertainty. *Management Sci.* **1** 197–206.

Ermoliev, Y. 1988. Stochastic quasigradient methods. Y. Ermoliev, R. Wets, eds. *Numerical Techniques for Stochastic Optimization*. Springer-Verlag, Berlin, Germany.

Frantzeskakis, L., W. B. Powell. 1990. A successive linear approximation procedure for stochastic dynamic vehicle allocation problems. *Transportation Sci.* **24**(1) 40–57.

Godfrey, G. A., W. B. Powell. 2001. An adaptive, distribution-free approximation for the newsvendor problem with censored demands, with applications to inventory and distribution problems. *Management Sci.* **48**(8) 1101–1112.

——, ——. 2002. An adaptive, dynamic programming algorithm for dynamic fleet management, II: Multiperiod travel times. *Transportation Sci.* **36**(1).

Higle, J., S. Sen. 1991. Stochastic decomposition: An algorithm for two stage linear programs with recourse. *Math. Oper. Res.* **16**(3) 650–669.

Infanger, G. 1994. *Planning Under Uncertainty: Solving Large-scale Stochastic Linear Programs*. The Scientific Press Series, Boyd & Fraser, New York.

Jordan, W., M. Turnquist. 1983. A stochastic dynamic network model for railroad car distribution. *Transportation Sci.* **17** 123–145.

Kall, P., S. Wallace. 1994. *Stochastic Programming*. John Wiley and Sons, New York.

Pereira, M., L. Pinto. 1991. Multistage stochastic optimization applied to energy planning. *Math. Programming* **52** 359–375.

Powell, W. B. 1986. A stochastic model of the dynamic vehicle allocation problem. *Transportation Sci.* **20** 117–129.

——. 1987. An operational planning model for the dynamic vehicle allocation problem with uncertain demands. *Transportation Res.* **21B** 217–232.

——. 1989. A review of sensitivity results for linear networks and a new approximation to reduce the effects of degeneracy. *Transportation Sci.* **23**(4) 231–243.

——, T. A. Carvalho. 1998. Dynamic control of logistics queueing network for large-scale fleet management. *Transportation Sci.* **32**(2) 90–109.

——, J. A. Shapiro, H. P. Simão. 2002. A representational paradigm for dynamic resource transformation problems. R. Fourer, C. Coullard, J. Owens, eds. *Ann. Oper. Res.* J. C. Baltzer AG. Forthcoming.

Puterman, M. L. 1994. *Markov Decision Processes*. John Wiley and Sons, New York.

Rockafellar, R. T. 1972. *Convex Analysis*, 2nd ed. Princeton University Press, Princeton, NJ.

——, R. Wets. 1991. Scenarios and policy aggregation in optimization under uncertainty. *Math. Oper. Res.* **16**(1) 119–147.

Ruszczynski, A. 1980. Feasible direction methods for stochastic programming problems. *Math. Programming* **19** 220–229.

——. 1987. A linearization method for nonsmooth stochastic programming problems. *Math. Oper. Res.* **12**(1) 32–49.

Sutton, R., A. Barto. 1998. *Reinforcement Learning*. The MIT Press, Cambridge, MA.

Van Slyke, R., R. Wets. 1969. L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM J. Appl. Math.* **17**(4) 638–663.

Received: December 2000; revision received: August 2001; accepted: September 2001.