TRANSPORTATION SCIENCE
Vol. 38, No. 4, November 2004, pp. 399–419
ISSN 0041-1655 | EISSN 1526-5447 | 04 | 3804 | 0399

DOI 10.1287/trsc.1030.0073

The Dynamic Assignment Problem

Michael Z. Spivey, Warren B. Powell
Department of Operations Research and Financial Engineering, Princeton University, Princeton, New Jersey 08544
{mzspivey@alumni.princeton.edu, powell@princeton.edu}

There has been considerable recent interest in the dynamic vehicle routing problem, but the complexities of this problem class have generally restricted research to myopic models. In this paper, we address the simpler dynamic assignment problem, where a resource (container, vehicle, or driver) can serve only one task at a time. We propose a very general class of dynamic assignment models, and propose an adaptive, nonmyopic algorithm that involves iteratively solving sequences of assignment problems no larger than what would be required of a myopic model. We consider problems where the attribute space of future resources and tasks is small enough to be enumerated, and propose a hierarchical aggregation strategy for problems where the attribute spaces are too large to be enumerated. Finally, we use the formulation to also test the value of advance information, which offers a more realistic estimate over studies that use purely myopic models.

Key words: dynamic vehicle routing; dynamic assignment; approximate dynamic programming
History: Received: June 2001; revisions received: June 2002, March 2003; accepted: May 2003.

The problem of dynamically assigning resources to tasks over time arises in a number of applications in transportation. In the field of freight transportation, truckload motor carriers, railroads, and shipping companies all have to manage fleets of containers (trucks, boxcars, and intermodal containers) that move one load at a time, with orders arriving continuously over time. In the passenger arena, taxi companies and companies that manage fleets of business jets have to assign vehicles (taxicabs or jets) to move customers from one location to the next. It is common to assume that the arrival of customer demands is random (e.g., known only through a probability distribution) over time, but it may also be the case that the vehicles become available in a random way. Finally, each assignment of a resource to a task generates a contribution to profits, which may also be random.

We refer to the problem of dynamically assigning resources to tasks as a *dynamic assignment problem*. In general, it may be possible to assign a resource to a sequence of two or more tasks at the same time, but we focus on problems where we assign a resource to one task at a time. We assume that resources and tasks are each characterized by a set of possibly unique attributes, where the contribution generated by an assignment will depend on the attributes of the resource and task. Resources do not have to be used and tasks do not all have to be covered, although there can be a cost for holding either one.

The dynamic assignment problem is a fundamental problem in routing and scheduling. It is a special case of the dynamic vehicle routing problem, without the complexities of in-vehicle consolidation. For this reason, it provides a natural framework for modeling the dynamic information processes and comparing myopic models with those that exploit distributional information about the future. It is common practice, for example, to model dynamic vehicle routing problems using myopic models, which ignore any forecasts of the future based on currently available data. These problems are themselves quite difficult because it is necessary to solve vehicle routing problems very quickly to respond to new information. (The static versions of these problems are already difficult.) In our problem, a static instance of an assignment problem is easy, allowing us to focus on the challenge of modeling the informational dynamics more carefully, and to study policies that consider the impact of decisions now on the future.

The dynamic assignment problem offers considerable richness relative to its static cousin. To gain an appreciation of the problem class, consider the following application drawn from the trucking industry in Powell (1996). Drivers call in over time asking to be assigned to loads. Customers call in over time asking for loads to be moved by drivers (one driver pulls one load). Both drivers and loads have characteristics that determine the contribution from assigning the driver to a load (for example, the current location of the driver and the point at which the load should be picked up). If we assign a driver to a load, the driver may refuse the assignment, which is only learned after the assignment is made (at which point a new assignment would have to be made). Finally, the contribution from assigning a driver to a load

may be estimated in advance. However, new information may arrive at the time that the assignment is made, and even more information may arrive after the assignment is completed (for example, we may only collect information on tolls after the driver completes the assignment).

This example illustrates three classes of information processes: (1) the arrival of drivers and loads to the system, (2) the information on whether a driver-to-load assignment is feasible, and (3) the contribution from the assignment. We have to make decisions now about assignments before we know about the availability of future resources and tasks, and possibly before we know about the contribution of an assignment or whether the assignment will be acceptable. Standard engineering practice has been to solve assignment problems myopically, using only the information available at the time the decision is made. However, not only is this process suboptimal, it is not even as good as that of an experienced dispatcher who will routinely hold a driver for a better load later in the day despite an acceptable assignment now.

The static assignment problem is one of the foundational problems in the field of operations research and has been studied and reported on extensively over the last 50 years (Dantzig 1963, Murty 1992). Many algorithms have been proposed for the assignment problem, including shortest augmenting path methods (see, for example, Balinski and Gomory 1964, Tomizawa 1972, Jonker and Volegnant 1987), variants of the primal simplex method (Barr et al. 1977, Hung 1983), relaxation methods (Bertsekas 1981, 1988), and signature methods (Goldfarb 1985; Balinski 1985, 1986). Variations of the static assignment problem, such as the bottleneck assignment problem (Gross 1959) and the stable assignment problem (Gale and Shapley 1962, Wilson 1977) have also received attention. Important properties of the static assignment problem have also been studied (see, for example, Shapley 1962).

Very little attention has been paid to explicitly extending the classical static assignment problem into a dynamic setting. For instance, the texts on networks by Murty (1992) and Ahuja et al. (1992) did not mention the assignment problem in a dynamic context. By contrast, there are many applications of dynamic assignment problems in industry, which are typically solved as sequences of static assignment problems. In the research literature, the closest problem class that has received a considerable amount of attention arises in machine scheduling. Pinedo (1995) provided a thorough summary of myopic policies for scheduling machines over time. There is a growing literature on the analysis of such algorithms, which are typically characterized as "online" algorithms (see Shmoys et al. 1995, Hall et al. 1997, Hoogeveen and Vestjens 2000), but all these algorithms are basically myopic models.

The literature on the dynamic vehicle routing problem can be divided into two broad problem classes: the so-called *full truckload problem* and the dynamic version of the general vehicle routing problem with multiple stops (*in-vehicle consolidation*). Psaraftis (1988) and Psaraftis (1995) discussed issues associated with the dynamic version of the general vehicle routing problem. Research on this problem class has primarily focused on simulations of algorithms solving myopic models (Cook and Russell 1978, Regan et al. 1998, Gendreau et al. 1999). Psaraftis (1980) was the first to attempt to explicitly solve a deterministic, time-dependent version of the vehicle routing problem using dynamic programming, but the formulation encountered the well-known problems with dimensionality. Swihart and Papastravrou (1999) and Secomandi (2001) both considered dynamic programming approximations for the single vehicle problem.

A separate line of research has focused on the simpler full truckload problem, where a vehicle serves one load at a time. Powell et al. (2000a) provided a myopic model and algorithm for the dynamic assignment problem, focusing on the problem of routing a driver through a sequence of more than one load. Powell (1996) provided a formulation of the dynamic assignment problem in the context of the load-matching problem for truckload trucking using a nonlinear approximation of the value of a resource in the future. A number of articles have been written on dynamic programming approximations for dynamic fleet management problems (see, for example, Godfrey and Powell 2002) but these problems do not share the discrete 0/1 behavior of the dynamic assignment problem.

General methods for this problem class can be divided between discrete dynamic programming and multistage linear programming. Traditional backward discrete dynamic programming approaches focus on calculating the value function explicitly. One can determine the optimal action at each stage by calculating the value function for all possible states at all possible times recursively (Puterman 1994). For our problem the number of states increases exponentially with the number of resources and tasks, making traditional applications of dynamic programming intractable. Forward dynamic programming methods (Bertsekas and Tsitsiklis 1996, Sutton and Barto 1998, Bertsekas et al. 1997) help mitigate the state-space problem by using simulation and Monte Carlo sampling, rather than by explicitly calculating the value function for all possible states in a backwards manner. However, these are general methods, and they do not take advantage of special problem structure such as the network structure of the dynamic assignment

problem. The action space of a dynamic assignment problem is also too large for forward dynamic programming methods to handle, and the challenge of estimating the value function for a large number of states remains.

Another algorithmic strategy is based on multistage linear programs. These techniques can be divided between scenario methods that explicitly enumerate the space of possible outcomes: those based on Bender's decomposition, and those that use other classes of approximations for the recourse function. Scenario methods (see Kall and Wallace 1994, Infanger 1994, Birge and Louveaux 1997) require enumerating a set of outcomes and explicitly solving a large-scale program. Aside from the challenge of enumerating the space of potential scenarios, this approach destroys the natural integrality of the dynamic assignment problem. Lageweg et al. (1988), Louveaux and van der Vlerk (1993), and Laporte and Louveaux (1993) addressed the problem of solving integer stochastic programs, but these would be difficult to implement in an online, dynamic fashion. Techniques that use Monte Carlo sampling (Higle and Sen 1991, Chen and Powell 1999) also destroy the problem's natural integrality. Methods based on Bender's decomposition (Van Slyke and Wets 1969, Birge 1985, Cheung and Powell 2000) seem more attractive, but they completely lose the inherent network structure of our problem class.

The original contributions of this paper are as follows. First, we provide a mathematical model of a general class of dynamic assignment problems, with an explicit representation of the exogenous information process. We introduce, apparently for the first time in the routing and scheduling literature, an explicit model of lagged information processes that captures the behavior of knowing about the arrival of a resource or task before the arrival actually occurs. Second, we introduce a family of adaptive learning algorithms that provide nonmyopic behavior, yet require only solving sequences of assignment problems no larger than would be required with a myopic algorithm. We provide variations for problems where the number of different types of resources and tasks is small and easy to enumerate, as well as a hierarchical aggregation strategy for handling large attribute spaces. We show that these algorithms can outperform myopic models. Third, we study experimentally the effect of advance information, and compare adaptive models to myopic models under varying degrees of advance information. These experiments show that the adaptive models will outperform myopic models with some advance information, but with sufficient advance information the myopic model actually outperforms an adaptive model.

In §1 we define our notation and formulate the problem. In §2 we establish some properties of the static and dynamic assignment problems that are used in developing our algorithm. We present our solution strategy in two stages. First, §3 presents the basic strategy of approximating the future with different types of linear approximations. In this formulation, we assume that there is a fixed set of resources and tasks (which is not too large), where we assume that we can easily enumerate all the resources and tasks that might arrive in the future. This model is equivalent to a problem where the set of attributes of resources and tasks is small enough that it can be enumerated. In the second stage, §4 describes a hierarchical aggregation strategy for handling problems with very large, or infinite, attribute spaces. Section 5 compares the adaptive approximations presented in this paper to a myopic approximation under varying levels of advance information. Finally, §6 draws some conclusions and suggests areas for further research.

The dynamic assignment problem offers tremendous richness, creating the challenge of balancing completeness with simplicity. The central ideas of the problem are covered in §1.1 (the basic model), and §3 (the algorithm and experimental results). We suggest that the reader might wish to initially read only these sections. Section 1.2, which provides generalizations of the basic model, and §2, which describes mathematical properties of assignment problems, are optional and can be read independently depending on the reader's background. Section 4 will be of interest to the algorithmically oriented reader looking to generalize the adaptive learning logic to more realistic situations. By contrast, §5 reports on experiments to quantify the value of advance information which requires only a reading of §1.

1. Problem Formulation

We model the dynamic assignment problem using the language of Markov decision processes. Because decisions are made at a particular time based on the current information available and with an eye toward the future, we feel the Markov decision process paradigm is the appropriate one. However, because the algorithms we propose are approximate, we do not have to assume that the exogenous information processes are actually Markovian.

We assume that we are modeling our problem in discrete time over the time instants $\mathcal{T} = \{0, 1, \ldots, T\}$ and that there are finite numbers of possible resources and tasks available. Define

$\mathcal{R}$ set of all resource indices that might possibly enter the system;

$\mathcal{L}$ set of all task indices that might possibly enter the system.

We assume that the indices in the sets $\mathcal{R}$ and $\mathcal{L}$ are distinct, meaning that $\mathcal{R} \cap \mathcal{L} = \varnothing$. This allows us to form a single set of indices $\mathcal{I} = \mathcal{R} \cup \mathcal{L}$, where the element $i \in \mathcal{I}$ uniquely identifies whether it is a resource or a task.

In sharp contrast to the static assignment problem, the dynamic assignment problem offers a rich set of variations based purely on the types of information that arrive and the possibility of advance information. In addition, there are two ways of representing new information: (1) the vector form and (2) the set form. We start with the vector form, which provides for a more classical representation of information as random variables.

We introduce the dynamic assignment problem as a sequence of models with increasingly general exogenous information processes. Section 1.1 describes the most basic model with random arrivals of resources and tasks, then §1.2 describes a series of variations that differ purely in the types of exogenous information arriving to the system.

1.1. A Basic Dynamic Assignment Problem

We divide our description of the basic dynamic assignment problem between descriptions of the exogenous information process (§1.1.1), the decision process (§1.1.2), system dynamics (§1.1.3), and the objective function (§1.1.4).

1.1.1. The Exogenous Information Process. Our basic assignment problem considers only the dynamic arrival of resources and tasks. The resource and task processes are independent of each other. Let

$$\widehat{R}_{tr} = \begin{cases} 1 & \text{if resource } r \in \mathcal{R} \text{ becomes known in period } t; \\ 0 & \text{otherwise;} \end{cases}$$

$$\widehat{R}_t = (\widehat{R}_{tr})_{r \in \mathcal{R}}$$

$$\widehat{L}_{tl} = \begin{cases} 1 & \text{if task } l \in \mathcal{L} \text{ becomes known in period } t; \\ 0 & \text{otherwise;} \end{cases}$$

$$\widehat{L}_t = (\widehat{L}_{tl})_{l \in \mathcal{L}}.$$

We let the assignment contributions be given by

$$\begin{aligned} c_{trl} &= \text{the contribution from assigning resource } r \\ &\qquad \text{to task } l \text{ at time } t; \\ &= f^c(r, l, t). \end{aligned}$$

$f^c(r, l, t)$ is a deterministic function of r, l, and t.

In this simple model, the exogenous information arriving in time period t is given by

$$W_t = (\widehat{R}_t, \widehat{L}_t). \tag{1}$$

Following standard convention, we let ω be a sample realization of $(W_t)_{t \in \mathcal{T}}$. We assume that Ω represents the set of elementary outcomes. If $\mathcal{F}$ is the σ-algebra on Ω and $\mathcal{P}$ is a probability measure on Ω, then $(\Omega, \mathcal{F}, \mathcal{P})$ is a probability space. We let $\mathcal{F}_t$ be the σ-algebra generated by $(W_0, W_1, \ldots, W_t)$, where the sequence of increasing sub-σ-algebras $\mathcal{F}_t$ forms a filtration. We assume that our information process satisfies $\sum_{t \in \mathcal{T}} \widehat{R}_{tr}(\omega) \le 1$ and $\sum_{t \in \mathcal{T}} \widehat{L}_{tr}(\omega) \le 1$ almost surely (a.s.), which is to say that every resource and task can become known at most once, and not every resource and task will become known in every outcome.

To describe the state of our system, we define

$$R_{tr} = \begin{cases} 1 & \text{if resource } r \in \mathcal{R} \text{ is known and available to be assigned in time period } t; \\ 0 & \text{otherwise;} \end{cases}$$

$$L_{tl} = \begin{cases} 1 & \text{if task } l \in \mathcal{L} \text{ is known and available to be assigned in time } t; \\ 0 & \text{otherwise.} \end{cases}$$

We let R_t, L_t be the corresponding vectors of these quantities. The state of our system is given by $S_t = (R_t, L_t)$.

The use of vector notation is not the only, nor necessarily the most natural, way to model the problem. The vectors $\widehat{R}_t$, R_t, $\widehat{L}_t$, and L_t imply enumerating the entire sets $\mathcal{R}$ and $\mathcal{L}$. A more natural representation is to use sets. Let

$$\widehat{\mathcal{R}}_t = \{r \mid \widehat{R}_{tr} = 1\}$$

$$\mathcal{R}_t = \{r \mid R_{tr} = 1\}.$$

$\widehat{\mathcal{L}}$ and $\mathcal{L}_t$ are defined analogously. The state of the system would then be given by $\mathcal{S}_t = (\mathcal{R}_t, \mathcal{L}_t)$.

1.1.2. The Decision Process. We represent the decisions we have made by:

$$x_{trl} = \begin{cases} 1 & \text{if resource } r \text{ is assigned to task } l \text{ at time } t; \\ 0 & \text{otherwise;} \end{cases}$$

$$x_t = (x_{trl})_{r \in \mathcal{R},\, l \in \mathcal{L}};$$

$$x^L_{tl} = \sum_{r \in \mathcal{R}} x_{trl} = \begin{cases} 1 & \text{if any resource is assigned to task } l \text{ at time } t; \\ 0 & \text{otherwise;} \end{cases}$$

$$x^R_{tr} = \sum_{l \in \mathcal{L}} x_{trl} = \begin{cases} 1 & \text{if resource } r \text{ is assigned to any task at time } t; \\ 0 & \text{otherwise.} \end{cases}$$

If we are using set notation, we would define the sets $\mathscr{R}_t^R$ and $\mathscr{R}_t^L$ as the sets of resources and tasks that are served at time t.

The function that returns a decision can be represented as

X_t^π a member of a family of functions $(X^\pi)_{\pi\in\Pi}$ that returns a decision vector x_t at time t.

We refer to a particular function X^π as a *policy*, and let the set Π represent our family of policies.

We can write our decision function as taking the general form

$$X_t^\pi = \arg\max_x C_t^\pi(x \mid S_t). \tag{2}$$

X_t^π is an $\mathscr{F}_t$-measurable function providing an assignment at time t from a given policy π. Our goal is to find a computationally tractable function X^π that provides optimal or near-optimal results. Equation (2) must be solved subject to

$$\sum_{l\in\mathscr{L}} x_{trl} \le R_{tr}$$

$$\sum_{r\in\mathscr{R}} x_{trl} \le L_{tl}.$$

We define the *feasible set* $\mathscr{X}_t(S_t)$ as this set of actions available at time t. The constraints have the structure of an assignment problem.

The specific class of assignment problem depends on the structure of C^π. The contribution function $C_t^\pi(x_t \mid S_t)$ effectively determines our policy. The quality of the overall solution depends on how C^π is formed. The simplest contribution function considers only what is known, while more sophisticated policies incorporate information that reflects distributional information about the future. Section 3 describes different strategies that balance current costs and rewards against the future.

1.1.3. System Dynamics. The dynamics of our system are given by

$$R_{t+1} = R_t - x_t^R + \widehat{R}_{t+1} \tag{3}$$

$$L_{t+1} = L_t - x_t^L + \widehat{L}_{t+1}. \tag{4}$$

We assume, of course, that our decision function returns decisions $x_t = X_t^\pi$ that satisfy the flow conservation equations

$$\sum_{l\in\mathscr{L}} x_{trl} \le R_{tr} \tag{5}$$

$$\sum_{r\in\mathscr{R}} x_{trl} \le L_{tl}. \tag{6}$$

It is clear from Equations (3) and (4) that the random variables R_t and L_t, and of course S_t, are defined for a given policy π. We could write our state variable as S_t^π to express this dependence, but we suppress the reference to the policy π for notational simplicity.

1.1.4. Objective Function. The cost of an assignment is given by

$$C_t(x_t) = \sum_{r\in\mathscr{R}_t}\sum_{l\in\mathscr{L}_t} c_{trl}x_{trl}. \tag{7}$$

For a state S_t and a policy π, define, for each t,

$$F_t^\pi(S_t) = E\left\{\sum_{t'=t}^{T} C_{t'}(X_{t'}^\pi)\,x \,\middle|\, S_t\right\}.$$

We note the difference between the cost function C^π used to choose x_t, and the cost function $C_t(x_t)$ used to evaluate our decision at time t. Our global optimization problem can now be formally stated as

$$F_t^*(S_t) = \sup_\pi F_t^\pi(S_t).$$

The solution to our dynamic assignment problem can be found by solving

$$F_0^*(S_0) = \sup_\pi F_0^\pi(S_0). \tag{8}$$

Section 3 poses the problem of finding the best policy by presenting several classes of cost approximations C^π. We next present a series of generalizations of our basic assignment problem.

1.2. Variants of the Exogenous Information Process

The basic dynamic assignment problem provides a foundation for several important variations which reflect more general exogenous information processes. Below, we present some of the most important generalizations.

1.2.1. Modeling Resource and Task Attributes. In our simplest model, the costs (c_{trl}) effectively become known as soon as the resources and tasks become known. It is often convenient in practice to assume that each resource and task is associated with a vector of attributes that we might define

a_r = vector of attributes associated with resource r, where we assume that $a_r \in \mathscr{A}$;

b_l = vector of attributes associated with task l, where $b_l \in \mathscr{B}$.

When these attribute vectors are defined explicitly, we obtain our assignment cost from the function

$C(t, a, b)$ = the cost of assigning a resource with attribute a to a task with attribute b at time t.

Using this function, the cost of an assignment would be computed using $c_{trl} = C(t, a_r, b_l)$, and the total cost of an assignment is still given by Equation (7).

We next need to model the information process that contains the attributes of resources and tasks. For this purpose, we define

$\widehat{A}_{tr}$ = the attributes of resource r entering the system at time t;

$\widehat{A}_t = (\widehat{A}_{tr})_{r\in\mathcal{R}}$;

$\widehat{B}_{tl}$ = the attributes of task l entering the system at time t;

$\widehat{B}_t = (\widehat{B}_{tl})_{l\in\mathcal{L}}$.

The exogenous information process would now be written

$$W_t = (\widehat{R}_t, \widehat{A}_t, \widehat{L}_t, \widehat{B}_t). \tag{9}$$

To describe the state of our system, we define

A_{tr} = the attributes of resource r at time t;

B_{tl} = the attributes of task l at time t.

Our system state vector is now

$$S_t = (R_t, A_t, L_t, B_t).$$

The dynamics of R_t and L_t are still given by Equations (3) and (4). The dynamics of A_t and B_t are given by

$$A_{t+1,r} = \begin{cases} \widehat{A}_{t+1,r} & \text{if } \widehat{R}_{t+1,r} = 1; \\ A_{r,t} & \text{otherwise;} \end{cases} \tag{10}$$

$$B_{t+1,l} = \begin{cases} \widehat{B}_{t+1,l} & \text{if } \widehat{L}_{t+1,l} = 1; \\ B_{l,t} & \text{otherwise.} \end{cases} \tag{11}$$

Equations (10) and (11) assume that information on the attributes of a resource arrives at the same time as information about the resource itself. This is a reasonable model for most academic studies but represents a simplification of what happens in practice. We could allow updates of the attributes of a resource or task at any time after it initially becomes "known." This model would require replacing $\widehat{R}_{t+1,r}$ and $\widehat{L}_{t+1,l}$ in Equations (10) and (11) with $R_{t+1,r}$ and $L_{t+1,l}$, respectively. If the attributes of a resource or task become known when the resource or task first becomes known, and are never updated again, then this model is effectively equivalent to the model given in §1.1.

We note in passing that our exogenous information process is, by definition, independent of the state of the system. In principle, this means that we may be receiving information on the status of a resource or task after it has already been assigned and removed from the system. While this seems awkward in principle, it does not represent a problem in practice, since new information about a resource or task after it has been assigned and removed would simply be ignored.

1.2.2. Lagged Information Processes. A common dimension of dynamic assignment problems is that we may know about a resource or task before we can act on it. Using our driver assignment problem, a customer may call in an order to be handled several days in the future. Alternatively, a driver may notify the company that he will be available for assignment later in the day. We refer to these as "lagged information processes."

The problem of time lags arises widely in the yield management area where customers make reservations before they show up for a flight, but the issue is largely overlooked in the routing and scheduling community. The issue has been addressed in the dynamic programming literature by Bander and White (1999), who posed the problem in the context of delayed state observations. (In our setting information is known in advance, whereas with delayed state observations the information about the state is known later.)

Let

τ_r = the time at which we know about resource r (similarly, this would be the time at which we know about the attribute vector a_r);

τ_r^a = the time at which the resource first becomes *actionable*.

Actionability refers to the time at which we can actually assign a resource to a task, and then remove it from the system. We would similarly define τ_l and τ_l^a as the knowable and actionable times of task l. We note that the actionable time can be simply an attribute of a resource or a task, but it is special in that it determines when we can act on a resource or task.

This notation allows us to define the arrival process of resources and tasks using

$$\widehat{R}_{t,rt'} = \begin{cases} 1 & \text{if resource } r\in\mathcal{R} \text{ becomes known in period } t, \text{ and is actionable at time } t'; \\ 0 & \text{otherwise;} \end{cases}$$

$\widehat{R}_{tt'} = (\widehat{R}_{t,rt'})_{r\in\mathcal{R}}$;

$\widehat{R}_t = (\widehat{R}_{tt'})_{t'\geq t}$;

$$\widehat{L}_{t,lt'} = \begin{cases} 1 & \text{if task } l\in\mathcal{L} \text{ becomes known in period } t, \text{ and is actionable at time } t'; \\ 0 & \text{otherwise;} \end{cases}$$

$\widehat{L}_{tt'} = (\widehat{L}_{t,lt'})_{l\in\mathcal{L}}$;

$\widehat{L}_t = (\widehat{L}_{tt'})_{t'\geq t}$.

In this case, $\widehat{R}_t$ and $\widehat{L}_t$ become families of vectors. With this interpretation, we can still let our exogenous information process be represented by $W_t = (\widehat{R}_t, \widehat{L}_t)$.

If we view the actionable time as an attribute, then we may use the representation given in §1.2.1.

Our resource and task state vectors become

$$R_{t,rt'} = \begin{cases} 1 & \text{if resource } r \in \mathcal{R} \text{ is known and first available to be assigned in period } t, \text{ and is actionable at time } t'; \\ 0 & \text{otherwise;} \end{cases}$$

$$L_{t,lt'} = \begin{cases} 1 & \text{if task } l \in \mathcal{L} \text{ is known and first available to be assigned in period } t, \text{ and is actionable at time } t'; \\ 0 & \text{otherwise.} \end{cases}$$

As we did earlier, we can define analogous sets $\mathcal{R}_{tt'}$ and $\mathcal{L}_{tt'}$. We note that $R_{t,rt'} = 1$ implies that $\tau_r^a = t'$.

We have to handle the case where a resource (or task) is knowable and actionable at time t, but we do not act on it, and it is held. In this event, we would have a resource that is known at time $t+1$ but actionable at time t. That is, it is possible for a resource to be actionable in the past. We can reasonably expect that this would never be true when a resource or task first becomes known (that is, it should never be the case that $\widehat{R}_{t+1,rt} = 1$). However, because resources and tasks can be held arbitrarily, we have to allow for the more general case when representing our state vector; in fact, there is no compelling reason to enforce $t' \geq t$ even in the exogenous information process.

We assume that we can only assign resources to tasks that are in the sets $(\mathcal{R}_{tt'})_{t' \leq t}$ and $(\mathcal{L}_{tt'})_{t' \leq t}$, respectively. However, we can *plan* assignments of resources or tasks, or both, that are known but not yet actionable. Let

$x_{t,rlt'}$ = the decision, made at time t (using what is known at time t), to assign resource r to task l at time t'. Both r and l must be knowable at time t, and actionable on or before time t'.

We refer to $x_{t,rlt}$ as an *action* while $x_{t,rlt'}$, $t' > t$, is a *plan*. Later in this paper we will provide conditions under which it would never be optimal to assign a resource to a task, both of which are actionable on or before time t', at a time later than t'. We impose the condition that $x_{t,rlt'} = 0$ for $t' < t$ because we cannot take actions in the past. We express the constraints on actionability using

$$\sum_{l \in \mathcal{L}} x_{t,rlt'} \leq \sum_{s' \leq t'} R_{rt,s'} \tag{12}$$

$$\sum_{r \in \mathcal{R}} x_{t,rlt'} \leq \sum_{s' \leq t'} L_{lt,s'}. \tag{13}$$

When $t = t'$, this constraint means that we can only act on resources that are actionable now or earlier. However, it also allows us to plan the assignment of the same resource at multiple times in the future. While this does not violate any physical constraints, allowing multiple assignments would probably yield poor results. For this reason, we add the constraints

$$\sum_{s' \geq t} \sum_{l \in \mathcal{L}} x_{t,rls'} \leq \sum_{s' \in \mathcal{T}} R_{rt,s'} \tag{14}$$

$$\sum_{s' \geq t} \sum_{r \in \mathcal{R}} x_{t,rls'} \leq \sum_{s' \in \mathcal{T}} L_{lt,s'}. \tag{15}$$

Under the assumption that planned assignments (where $x_{t,rt'} = 1$ for $t' > t$) may be replanned in the next period, the dynamics of the system with lagged information are given by

$$R_{t+1,rt'} = \begin{cases} R_{t,rt'} - x^R_{t,rt} + \widehat{R}_{t+1,rt'} & t' \leq t \\ R_{t,rt'} + \widehat{R}_{t+1,rt'} & t' > t \end{cases}$$

$$L_{t+1,lt'} = \begin{cases} L_{t,lt'} - x^L_{t,lt} + \widehat{L}_{t+1,lt'} & t' \leq t \\ L_{t,lt'} + \widehat{L}_{t+1,lt'} & t' > t, \end{cases}$$

where

$$x^L_{t,lt} = \sum_{r \in \mathcal{R}} x_{t,rlt}$$

$$x^R_{t,rt} = \sum_{l \in \mathcal{L}} x_{t,rlt}.$$

Finally, the one-period cost function given in Equation (7) must be replaced with

$$C_t(x_t) = \sum_{r \in \mathcal{R}_{tt}} \sum_{l \in \mathcal{L}_{tt}} c_{trl} x_{t,rlt}.$$

This function considers only decisions that are actionable at time t.

1.2.3. A Model with Cost Uncertainties. In some applications, the decision to assign a resource or task may have to be made before the cost of the assignment becomes known. For example, in our trucking problem we may not know about travel times (which may be affected by congestion, for instance) or tolls until after the assignment is complete. Also, the revenue received for serving the task may be something that depends on the total volume of the account, which may not be determined until the end of the accounting period.

There are several models we could reasonably use to handle cost uncertainties. A simple model assumes that there is a flow of information updates to the estimate of the cost of an assignment. Let

$\widehat{C}_{trl}$ = random variable giving the change in the cost of assigning r to l at time t.

As with the other random variables describing exogenous information processes, we assume that $\widehat{C}$ is

represented by a probability distribution that in practice would be computed using observations of costs after the fact. Interestingly, the simple exercise of computing a distribution to measure the difference between what we assumed before the fact and what actually occurred is often overlooked in practice.

The assignment cost, then, would be given by

$$c_{trl} = c_{t-1,\,rl} + \widehat{C}_{trl}. \tag{16}$$

Our information process would then be $W_t = (\widehat{R}_t, \widehat{L}_t, \widehat{C}_t)$, and our system state would be $S_t = (R_t, L_t, c_t)$.

It is easiest to represent the information process as the change in costs. In practice, a real information process on costs would appear as a stream of updates to the cost, if there is a change, or no update, if there is no change.

We note that this process evolves independently of whether the resource or task is actually in the system. This is quite realistic. We may have updates to our estimate of the cost before an assignment is made, and we may continue to have updates after the assignment is completed as we receive more information on the outcome of an assignment.

1.3. Solution Strategy

Given the large state space, we are not going to be able to solve this using either the classical backward dynamic techniques (Puterman 1994) or the approximate forward techniques, which depend on discrete representations of the value function (Bertsekas and Tsitsiklis 1996, Sutton and Barto 1998). However, we do feel that approximate dynamic programming can be effective, as long as we use the right approximation for the value function. We propose to use a class of policies of the form

$$X_t^{\pi}(S_t) = \arg\max_{x_t \in \mathscr{X}_t} c_t x_t + E\{\widehat{V}_{t+1}(S_{t+1}(x_t)) \mid S_t\},$$

where $\mathscr{X}_t$ describes our feasible region, and $\widehat{V}_{t+1}(S_{t+1}(x))$ is an appropriately chosen approximation of the value of being in state S_{t+1} at time $t+1$. Because $\widehat{V}_{t+1}(S_{t+1}(x))$ is, in effect, an approximation of assignment problems later in time, we undertake in §2 a study of the properties of assignment problems. Section 3 then outlines a class of solution strategies that require iteratively solving sequences of assignment problems.

2. Some Properties of Assignment Problems

Our algorithm strategy depends on making decisions at time t using an approximation of the problem at time $t+1$. At time t, we are often faced with estimating the value of a resource or a task in the future. Furthermore, when we assign a resource to a task, we need to estimate the impact of dropping the resource *and* the task from the future. In §3 we introduce several different types of approximations. In this section, we summarize several properties of assignment problems. In addition, they provide bounds on the marginal value of resources and tasks in the future.

Section 2.1 summarizes the properties of a single, static assignment problem. Section 2.2 gives a useful result on the behavior of assignment problems over time.

2.1. Properties of the Static Assignment Problem

We first establish some properties for the static assignment problem. By *static* we mean a problem in which there is only one assignment problem to be solved rather than a sequence of assignment problems over time. Assume a superset of resources $\mathscr{R}$ and a superset of tasks $\mathscr{L}$. Given an ordered pair of sets $\mathscr{S}' = (\mathscr{R}', \mathscr{L}')$ consisting of resources and tasks to be assigned and contributions c_{rl} for assigning each resource r to each task l, define

$$\begin{aligned} C(\mathscr{S}') = \max_x \quad & c \cdot x \\ \text{subject to:} \quad & \sum_{l \in \mathscr{L}} x_{rl} \le \begin{cases} 1 & \forall r \in \mathscr{R}', \\ 0 & \forall r \notin \mathscr{R}', \end{cases} \\ & \sum_{r \in \mathscr{R}} x_{rl} \le \begin{cases} 1 & \forall l \in \mathscr{L}', \\ 0 & \forall l \notin \mathscr{L}', \end{cases} \\ & x_{rl} \in \{0,1\} \quad \forall r \in \mathscr{R}, \forall l \in \mathscr{L}, \end{aligned}$$

where the assignment vector x is indexed by the elements of the cross product $\mathscr{R} \times \mathscr{L}$.

Although technically $\mathscr{S}'$ is not a set but rather an ordered pair of sets, we can extend some normal set operations to $\mathscr{S}' = (\mathscr{R}', \mathscr{L}')$, such as:

Subset: $\mathscr{S}' \subset \mathscr{S}''$ provided $\mathscr{R}' \subset \mathscr{R}''$ and $\mathscr{L}' \subset \mathscr{L}''$.

Union: If $\mathscr{R}'' \subset \mathscr{R}$, then $\mathscr{S}' \cup \mathscr{R}'' = (\mathscr{R}' \cup \mathscr{R}'', \mathscr{L}')$. (The same is true for $\mathscr{S}' \cup \mathscr{L}''$.)

Also, given a network $\mathscr{S}' = (\mathscr{R}', \mathscr{L}')$, define

$\mathscr{X}^*(\mathscr{S}')$ = the set of optimal assignments for $\mathscr{S}'$;

$x^*(\mathscr{S}')$ = an element of $\mathscr{X}^*(\mathscr{S}')$;

$c^*(r)$ = the contribution on the arc containing the flow out of resource r in the optimal assignment for $\mathscr{S}'$;

$c^*(l)$ = the contribution on the arc containing the flow into task l in the optimal assignment for $\mathscr{S}'$;

$l^*(r)$ = the task assigned to resource r under $x^*(\mathscr{S}')$. If r is not assigned under x^*, then $l^*(r)$ is the supersink;

$r^*(l)$ = the resource assigned to task l under $x^*(\mathcal{S}')$. If l is not assigned under x^*, then $r^*(l)$ is the supersource.

We are primarily interested in the behavior of assignment problems after we add or drop resources and tasks. For these purposes it is useful to define

$$C_r^+(\mathcal{S}') = C(\mathcal{S}' \cup \{r\}), \quad \text{for } r \notin \mathcal{R}';$$
$$C_r^-(\mathcal{S}') = C(\mathcal{S}' - \{r\}), \quad \text{for } r \in \mathcal{R}';$$
$$C_{rl}^+(\mathcal{S}') = C(\mathcal{S}' \cup \{r\} \cup \{l\}), \quad \text{for } r \notin \mathcal{R}', l \notin \mathcal{L}';$$
$$C_{r_1r_2}^+(\mathcal{S}') = C(\mathcal{S}' \cup \{r_1, r_2\}), \quad \text{for } r_1, r_2 \notin \mathcal{R}'.$$

$C_l^+(\mathcal{S}')$, $C_l^-(\mathcal{S}')$, $C_{rl}^-(\mathcal{S}')$ and $C_{r_1r_2}^-(\mathcal{S}')$ are defined similarly. Now let

$$v_r^+(\mathcal{S}') = C_r^+(\mathcal{S}') - C(\mathcal{S}'), \quad \text{provided } r \notin \mathcal{S}';$$
$$v_r^-(\mathcal{S}') = C(\mathcal{S}') - C_r^-(\mathcal{S}'), \quad \text{provided } r \in \mathcal{S}';$$
$$v_r(\mathcal{S}') = \begin{cases} v_r^+(\mathcal{S}') & \text{if } r \notin \mathcal{S}' \\ v_r^-(\mathcal{S}') & \text{if } r \in \mathcal{S}'. \end{cases}$$

We define $v_l^+(\mathcal{S}')$, $v_l^-(\mathcal{S}')$ and $v_l(\mathcal{S}')$ correspondingly.

We often have to consider the problem of adding (removing) both a resource and a task to (from) a network. For a network $\mathcal{S}'$, define

$$v_{rl}^+(\mathcal{S}') = C_{rl}^+(\mathcal{S}') - C(\mathcal{S}'), \quad \text{provided } r, l \notin \mathcal{S}';$$
$$v_{rl}^-(\mathcal{S}') = C(\mathcal{S}') - C_{rl}^-(\mathcal{S}'), \quad \text{provided } r, l \in \mathcal{S}';$$
$$v_{rl}(\mathcal{S}') = \begin{cases} v_{rl}^+(\mathcal{S}') & \text{if } r, l \notin \mathcal{S}' \\ v_{rl}^-(\mathcal{S}') & \text{if } r, l \in \mathcal{S}'. \end{cases}$$

When we solve a problem at time t, our decisions impact the assignment problem that will have to be solved at $t+1$. Our solution strategy depends on our ability to approximate the impact of present decisions on the future.

To begin, the assignment problem is, of course, a linear program, and therefore shares the basic properties of linear programs. For example, an assignment problem is piecewise linear concave with respect to the right-hand side constraints. For our work, we may be adding or dropping multiple resources and tasks, and we often have to evaluate the cost of adding or dropping a resource-task pair. A fundamental result relating the values of adding additional resources and tasks separately to the value of adding them together is due to Shapley (1962):

THEOREM 1. *Given a network* $\mathcal{S}'$,

1. $(C(\mathcal{S}' \cup \{r_1\}) - C(\mathcal{S}')) + (C(\mathcal{S}' \cup \{r_2\}) - C(\mathcal{S}')) \geq C(\mathcal{S}' \cup \{r_1, r_2\}) - C(\mathcal{S}')$ (*subadditivity*);
2. $(C(\mathcal{S}' \cup \{l_1\}) - C(\mathcal{S}')) + (C(\mathcal{S}' \cup \{l_2\}) - C(\mathcal{S}')) \geq C(\mathcal{S}' \cup \{l_1, l_2\}) - C(\mathcal{S}')$ (*subadditivity*);
3. $(C(\mathcal{S}' \cup \{r\}) - C(\mathcal{S}')) + (C(\mathcal{S}' \cup \{l\}) - C(\mathcal{S}')) \leq C(\mathcal{S}' \cup \{r, l\}) - C(\mathcal{S}')$ (*superadditivity*) (*Shapley* 1962).

This means that, in Shapley's terminology, two resources or two tasks are *substitutes*, while a resource and a task are *complements*.

As a corollary to Theorem 1, known as Shapley's theorem, relating the values of removing resources and tasks from a network separately to removing them together, we have

COROLLARY 1. *Given a network* $\mathcal{S}'$ *with resources* r, r_1, r_2, *and tasks* l, l_1, l_2,

1. $(C(\mathcal{S}') - C(\mathcal{S}' - \{r_1\})) + (C(\mathcal{S}') - C(\mathcal{S}' - \{r_2\})) \leq C(\mathcal{S}') - C(\mathcal{S}' - \{r_1, r_2\})$;
2. $(C(\mathcal{S}') - C(\mathcal{S}' - \{l_1\})) + (C(\mathcal{S}') - C(\mathcal{S}' - \{l_2\})) \leq C(\mathcal{S}') - C(\mathcal{S}' - \{l_1, l_2\})$;
3. $(C(\mathcal{S}') - C(\mathcal{S}' - \{r\})) + (C(\mathcal{S}') - C(\mathcal{S}' - \{l\})) \geq C(\mathcal{S}') - C(\mathcal{S}' - \{r, l\})$.

Given a network $\mathcal{S}'$, Shapley's theorem (Theorem 1) and Corollary 1 can be written

$$v_r^+(\mathcal{S}') + v_l^+(\mathcal{S}') \leq v_{rl}^+(\mathcal{S}')$$
$$v_r^-(\mathcal{S}') + v_l^-(\mathcal{S}') \geq v_{rl}^-(\mathcal{S}').$$

When we assign a resource to a task, we are dropping the resource-task pair from the assignment problem in the future. We can approximate this effect by simply adding the value of adding (dropping) the resource plus the value of adding (dropping) a task. From Shapley's result, this will underestimate the impact of adding (dropping) the resource and the task together. Because it is computationally expensive to estimate the value of adding (dropping) a resource-task pair, it can be useful to be able to approximate this. Below we provide a bound on the resource-task gradients.

THEOREM 2. *Let* $\mathcal{S}'$ *be a network. Then we have*

$$c_{rl} \leq v_{rl}^+(\mathcal{S}') \leq \max_{r', l'} \{c_{rl'} + c_{r'l} - c_{r'l'}\};$$
$$c_{rl} \leq v_{rl}^-(\mathcal{S}') \leq c_{rl^*(r)} + c_{r^*(l)l} - c_{r^*(l)l^*(r)}.$$

PROOF.

1. Let y be the flow-augmenting path consisting of the link (r, l). Then $C(y) = c_{rl}$. Thus $v_{rl}^+(\mathcal{S}') = C_{rl}^+(\mathcal{S}') - C(\mathcal{S}') = C(y_{r,l}^*) \geq c_{rl}$.
2. Because $v_{rl}^-(\mathcal{S}' \cup \{r\} \cup \{l\}) = v_{rl}^+(\mathcal{S}')$, we have $c_{rl} \leq v_{rl}^-(\mathcal{S}')$.
3. To prove $v_{rl}^-(\mathcal{S}') \leq c_{rl^*(r)} + c_{r^*(l)l} + c_{r^*(l)l^*(r)}$, we require two cases:

Case a. If r and l are assigned to each other in the optimal solution for $\mathcal{S}'$, the only flow-augmenting path from l to r that preserves feasibility in the original network is $y = (l, r)$. Then $l^*(r) = l$ and $r^*(l) = r$.

So we have $C(y^*_{l,r}) = C(y) = -c_{rl}$. Because $-v^-_{rl}(\mathcal{S}') = -C(y^*_{l,r})$, we have $v^-_{rl}(\mathcal{S}') = c_{rl} = c_{rl} + c_{rl} - c_{rl} = c_{rl^*(r)} + c_{r^*(l)l} - c_{r^*(l)l^*(r)}$.

Case b. If r and l are not assigned to each other in the optimal solution for $\mathcal{S}'$, all flow-augmenting paths from l to r that preserve feasibility in the original network must contain at least the mirror arcs $(l, r^*(l))$ and $(l^*(r), r)$. Let y be the flow-augmenting path from l to r consisting of $(l, r^*(l))$, $(r^*(l), l^*(r))$ and $(l^*(r), r)$. Then $C(y) = -c_{l,r^*(l)} + c_{r^*(l),l^*(r)} - c_{l^*(r),l}$. Thus $C(y^*_{l,r}) \geq -c_{l,r^*(l)} + c_{r^*(l),l^*(r)} - c_{l^*(r),l}$. Because $-v^-_{rl}(\mathcal{S}') = -C(y^*_{l,r})$, we have $v^-_{rl}(\mathcal{S}') \leq c_{l,r^*(l)} + c_{l^*(r),l} - c_{r^*(l),l^*(r)}$.

4. Because $v^+_{rl}(\mathcal{S}') = v^-_{rl}(\mathcal{S}' \cup \{r\} \cup \{l\})$, we have $v^+_{rl}(\mathcal{S}') \leq c_{l,r^*(l)} + c_{l^*(r),l} - c_{r^*(l),l^*(r)}$ for the network $\mathcal{S}' \cup \{r\} \cup \{l\}$. Clearly, then, $v^+_{rl}(\mathcal{S}') \leq \max_{r',l'}\{c_{rl'} + c_{r'l} + c_{r'l'}\}$ for the network $\mathcal{S}'$. □

2.2. Some Properties of Deterministic Dynamic Assignment Problems

For a deterministic dynamic assignment problem the resource and task arrival processes as well as the contribution processes are known at all times. Thus there is no expectation in the definition of F^π_t; we have $F^\pi_t(\mathcal{S}_t) = \sum_{t'=t}^{T+1} c_{t'} \cdot X^\pi_{t'}$.

Recall the definition of a policy π:

π = A set of decisions $\mathcal{X}_t(\mathcal{S})$ that specify the action to take if the system is in state S.

For a deterministic problem, a policy π is equivalent to a set of decisions $(x_t)_{t=0}^T$ that are specified in advance and that are independent of the state of the system at each point in time.

There are two interesting properties of deterministic dynamic assignment problems. The first, which is a known result (Glasserman and Yao 1994, pp. 234–238), establishes a condition under which a myopic solution can be optimal. Let $\mathcal{A}'$ be the set of paired elements (r, l) in $\mathcal{S}'$ and let σ be an ordered sequence of such pairs. Assume that elements r and l occur before either r' or l'. Then σ is said to be a *Monge sequence* provided $c_{rl} + c_{r'l'} - c_{rl'} - c_{r'l} \geq 0$. If a Monge sequence can be constructed of all the pairs in $\mathcal{A}'$ then the static assignment problem can be solved optimally using a simple greedy solution.

The next result, which is original, establishes a condition under which a resource and task that are assigned to each other in the optimal solution should be assigned as soon as both are actionable. This result is stated as follows:

THEOREM 3. *Consider a deterministic dynamic assignment problem in which c_{trl} is a strictly decreasing function of t. Let τ^a_{rl} be the first-time resource r and task l, which are both actionable. If $x^*_{trl} = 1$ for some t in an optimal solution x^*, then $x^*_{rl\tau^a_{rl}} = 1$.*

PROOF. Suppose not. Then there exists an optimal solution x^* such that for some r' and l', $x^*_{r'l'\tau^a_{r'l'}} = 0$ but $x^*_{r'l't'} = 1$ for some $t' > \tau^a_{r'l'}$. Let π be a policy such that $X^\pi = x^*$ for all r, l, t except that $X^\pi_{r'l'\tau^a_{r'l'}} = 1$ and $X^\pi_{r'l't'} = 0$. Then $F^\pi(\mathcal{S}_0) = F^*(\mathcal{S}_0) - c_{r'l't'} + c_{r'l'\tau^a_{r'l'}}$. However because c_{trl} is a strictly decreasing function of t, we have $c_{r'l'\tau^a_{r'l'}} > c_{r'l't'}$. Thus $F^\pi(\mathcal{S}_0) > F^*(\mathcal{S}_0)$, which contradicts the assumption that x^* is an optimal solution. □

3. Solution Strategy for Problems with Small Attribute Spaces

Although in prior sections we have considered general dynamic assignment problems, in the rest of the paper we concentrate, for practical purposes, on simpler problems: those in which $\tau_r = \tau^a_r$ and $\tau_l = \tau^a_l$, i.e., those in which the time that the existence of a resource or task becomes known is the same as the time at which the resource or task becomes actionable for assignment.

We define the value function as follows:

$$V_t(\mathcal{S}_t) = \max_{x_t \in \mathcal{X}_t}\{c_t \cdot x_t + E[V_{t+1}(\mathcal{S}_{t+1}) \mid \mathcal{S}_t]\}; \quad t = 0, \ldots, T,$$
$$= 0; \quad t = T+1. \tag{17}$$

The traditional dynamic programming approach is to calculate the value function explicitly for each state. By the principle of optimality we have $V_t(\mathcal{S}_t) = F^*_t(\mathcal{S}_t)$ (Puterman 1994). In particular, $V_0(\mathcal{S}_0) = F^*_0(\mathcal{S}_0)$, and thus we could solve the original problem by solving the value function recursions. Unfortunately, the number of possible states $\mathcal{S}_t$ is on the order of the number of possible combinations of available resources and tasks, which is $2^{|\mathcal{R}|+|\mathcal{L}|}$. Because solving the value function recursions involves calculating $V_t(\mathcal{S}_t)$ for each state $\mathcal{S}_t$, calculating the value function explicitly is feasible only for the smallest sets $\mathcal{R}$ and $\mathcal{L}$. Instead, we use an approximation $\widehat{V}$ of the value function at $t+1$ when solving the value function recursion at time t. Our approximation at $t+1$ can be made into an explicit function of the time t decision variables, and then the value function at t can be solved by embedding it into a network structure.

More explicitly, we replace the expression $V_{t+1}(\mathcal{S}_{t+1})$ with an approximation of the form

$$\widehat{V}_{t+1}(S_{t+1}) = \widehat{V}^R \cdot R_{t+1} + \widehat{V}^L \cdot L_{t+1}, \tag{18}$$

where $\widehat{V}^R$ and $\widehat{V}^L$ are, respectively, the vectors consisting of the resource and task value approximations.

Substituting this expression in the objective function of (17) yields

$$\begin{aligned}
&c_t \cdot x_t + E\big[\widehat{V}^R \cdot R_{t+1} + \widehat{V}^L \cdot L_{t+1} \mid \mathcal{S}_t\big] \\
&\quad = c_t \cdot x_t + E\big[\widehat{V}^R \cdot (R_t - x_t^R + \widehat{R}_{t+1}) \\
&\qquad\qquad + \widehat{V}^L \cdot (L_t - x_t^L + \widehat{L}_{t+1}) \mid \mathcal{S}_t\big] \\
&\quad = c_t \cdot x_t + E\big[\widehat{V}^R \cdot R_t \mid \mathcal{S}_t\big] - E\big[\widehat{V}^R \cdot x_t^R \mid \mathcal{S}_t\big] \\
&\qquad + E\big[\widehat{V}^R \cdot \widehat{R}_{t+1} \mid \mathcal{S}_t\big] + E\big[\widehat{V}^L \cdot L_t \mid \mathcal{S}_t\big] \\
&\qquad - E\big[\widehat{V}^L \cdot x_t^L \mid \mathcal{S}_t\big] + E\big[\widehat{V}^L \cdot \widehat{L}_{t+1} \mid \mathcal{S}_t\big] \\
&\quad = c_t \cdot x_t + \widehat{V}^R \cdot R_t - \widehat{V}^R \cdot x_t^R + \widehat{V}^R \cdot E[\widehat{R}_{t+1}] \\
&\qquad + \widehat{V}^L \cdot L_t - \widehat{V}^L \cdot x_t^L + \widehat{V}^L \cdot E[\widehat{L}_{t+1}], \qquad (19)
\end{aligned}$$

where Equation (19) arises because R_t, x_t^R, L_t, and x_t^L are deterministic given S_t, and $\widehat{R}_{t+1}$ and $\widehat{L}_{t+1}$ are independent of S_t.

Because R_t, $\widehat{R}_{t+1}$, L_t, and $\widehat{R}_{t+1}$ do not contain an x_t term, they do not affect the choice of x_t in the maximization. Thus for practical purposes the terms $\widehat{V}^R \cdot R_t$, $\widehat{V}^R \cdot E[\widehat{R}_{t+1}]$, $\widehat{V}^L \cdot L_t$ and $\widehat{V}^L \cdot E[\widehat{L}_{t+1}]$ can be dropped from the objective function. This gives us the following approximation of (17) for $t \le T$:

$$\widetilde{V}_t(\mathcal{S}_t) = \max_{x_t \in \mathcal{X}_t} \{c_t \cdot x_t - \widehat{V}^R \cdot x_t^R - \widehat{V}^L \cdot x_t^L\}. \qquad (20)$$

Note also that the expectation in (17) has disappeared. This simplification is important because the expectation is itself computationally intractable.

We consider three main classes of approximations. The simplest is the greedy or myopic approximation; in this case $\widehat{V} = 0$. We use this approximation only as a means of comparison for our other two classes of approximations. Another is to let the value of a state be the sum of the values of the individual resources and tasks in the state: $\widehat{V}(\mathcal{S}_t) = \sum_{r \in \mathcal{R}} v_{tr} \cdot R_{tr} + \sum_{l \in \mathcal{L}} v_{tl} \cdot L_{tl}$. This approximation is separable. We also consider a nonseparable approximation based on the values of resource-task pairs: $\widehat{V}(\mathcal{S}_{t+1}) = -\sum_{r \in \mathcal{R}} \sum_{l \in \mathcal{L}} v_{t+1,rl} \cdot x_{trl}$. All of our approximations are calculated adaptively; that is, the values of the resources, tasks, and resource-task pairs that we use are calculated over a number of iterations.

3.1. Separable Approximations

First, we consider an approximation of $V_{t+1}(\mathcal{S}_{t+1})$ based on values of individual resources and tasks. We define the value of a resource to be the impact of adding or removing the resource from the system, i.e., the resource's *gradient*.

$$\frac{\partial V_t^*(\mathcal{S}_t)}{\partial R_{tr}} = \begin{cases} V_t(\mathcal{S}_t) - V_t(\mathcal{S}_t - \{r\}) & \text{if } R_{tr} = 1, \\ V_t(\mathcal{S}_t \cup \{r\}) - V_t(\mathcal{S}_t) & \text{if } R_{tr} = 0. \end{cases}$$

Thus $\partial V_t^*(\mathcal{S}_t)/\partial R_{tr}$ is either the left or the right derivative, depending on the situation. The gradient of a task is defined similarly.

We see that $\partial V_t^*(\mathcal{S}_t)/\partial R_{tr}$ is equivalent to the natural extension of the definition of $v_r(\mathcal{S}')$ in §2.1 to $v_{tr}(\mathcal{S}_t)$:

$$v_{tr}(\mathcal{S}_t) = \begin{cases} V_t(\mathcal{S}_t) - V_t(\mathcal{S}_t - \{r\}) & \text{if } r \in \mathcal{S}_t, \\ V_t(\mathcal{S}_t \cup \{r\}) - V_t(\mathcal{S}_t) & \text{if } r \notin \mathcal{S}_t. \end{cases}$$

Thus $\partial V_t^*(\mathcal{S}_t)/\partial R_{tr} = v_{tr}(\mathcal{S}_t)$.

We also define the following.

$\hat{v}_{tr}^k$ = the estimate of the value of resource r at time t obtained directly in iteration k. We have that $\hat{v}_{tr}^k = v_{tr}(\mathcal{S}_t)$ if $\mathcal{S}_t$ is the system state at iteration k, time t.

$\bar{v}_{tr}^k$ = the smoothed estimate of the value of resource r at time t after iteration k. In particular, for smoothing function α^k, $\bar{v}_{tr}^k = \alpha^k \hat{v}_{tr}^k + (1 - \alpha^k)\bar{v}_{tr}^{k-1}$.

The quantities $\hat{v}_{tl}^k$ and $\bar{v}_{tl}^k$ are defined similarly. We believe the smoothing is necessary because the resource value estimates for a particular iteration are dependent on the realization of the stochastic resource and task processes in that iteration. To get a more accurate estimate of the true value of the resource, the estimates from each iteration must be combined in some manner. (For a more careful discussion of how the resource value estimates themselves are calculated in a particular iteration, please see §3.3.)

Using the resource and task gradients we can approximate $V_{t+1}(\mathcal{S}_{t+1})$ during iteration k in any one of three ways:

$$\widehat{V}_{t+1}^{r,k}(\mathcal{S}_{t+1}) = \sum_{r \in \mathcal{R}} \bar{v}_{t+1,r}^{k-1} \cdot R_{t+1,r}, \qquad (21)$$

$$\widehat{V}_{t+1}^{l,k}(\mathcal{S}_{t+1}) = \sum_{l \in \mathcal{L}} \bar{v}_{t+1,l}^{k-1} \cdot L_{t+1,l}, \qquad (22)$$

$$\widehat{V}_{t+1}^{r,l,k}(\mathcal{S}_{t+1}) = \sum_{r \in \mathcal{R}} \bar{v}_{t+1,r}^{k-1} \cdot R_{t+1,r} + \sum_{l \in \mathcal{L}} \bar{v}_{t+1,l}^{k-1} \cdot L_{t+1,l}. \qquad (23)$$

These approximations are both linear and separable.

Substituting the resource gradients approximation (21) (the others would be similar) for $V_{t+1}(\mathcal{S}_{t+1})$ in the value function recursion (17) yields, as in (20):

$$\widetilde{V}_t^{r,k}(\mathcal{S}_t) = \max_{x_t \in X_t} \left\{ \sum_{r \in \mathcal{R}} \sum_{l \in \mathcal{L}} (c_{trl} - \bar{v}_{t+1,r}^{k-1}) \cdot x_{trl} \right\}. \qquad (24)$$

We can see from the formulation (24) that if resource r is assigned to some task, then the quantity $\bar{v}_{t+1,r}^{k-1}$ is subtracted from the original objective function. We can thus view the resource gradients as contributions for *not* assigning the corresponding resources. This leads to a network formulation of

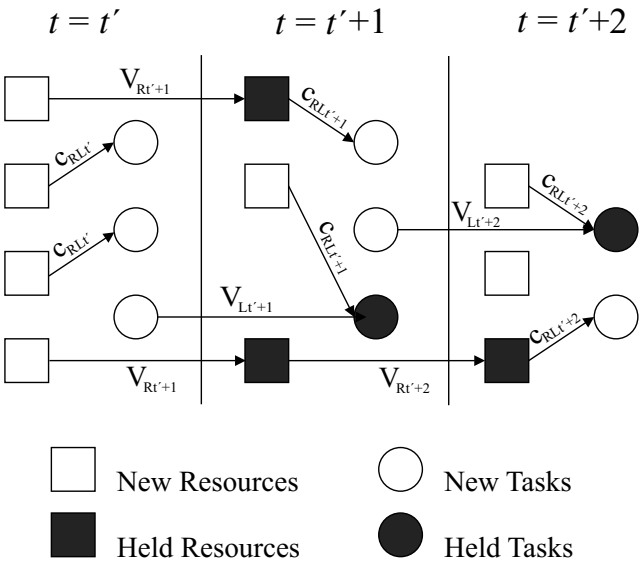

Figure 1 A Dynamic Assignment Problem with Resource and Task Gradients

the problem consisting of the usual arcs connecting resources and tasks but also including *no-assignment* arcs from each resource to a supersink with the appropriate resource gradient as the contribution of the no-assignment arc. (When utilizing the task gradients instead of or in addition to the resource gradients we include them in the network formulation as the contributions on no-assignment arcs from the supersource to the tasks.) Figure 1 illustrates a dynamic assignment problem over three periods, and the flow augmenting paths represented by the resource and task gradients.

3.2. A Nonseparable Approximation

The basic decision unit in a dynamic assignment problem is a single decision variable at time t, x_{trl}. It seems reasonable, then, to consider an approximation of $V_{t+1}(\mathcal{S}_{t+1})$ based on these decision variables.

In a network formulation each decision variable x_{trl} is associated with a particular arc in the network; thus we refer to a marginal value with respect to x_{trl} as an *arc gradient*.

We define the following.

$$\frac{\partial V^*_{t+1}(\mathcal{S}_{t+1})}{\partial x_{trl}} = \begin{cases} V_{t+1}(\mathcal{S}_{t+1} - \{r,l\}) - V_{t+1}(\mathcal{S}_{t+1}) \\ \quad \text{if } x_{trl} = 0,\ r,l \in \mathcal{S}_t, \\ V_{t+1}(\mathcal{S}_{t+1} - \{r\}) - V_{t+1}(\mathcal{S}_{t+1} \cup \{l\}) \\ \quad \text{if } x_{trl} = 0,\ r \in \mathcal{S}_t,\ l \notin \mathcal{S}_t, \\ V_{t+1}(\mathcal{S}_{t+1} - \{l\}) - V_{t+1}(\mathcal{S}_{t+1} \cup \{r\}) \\ \quad \text{if } x_{trl} = 0,\ r \notin \mathcal{S}_t,\ l \in \mathcal{S}_t, \\ V_{t+1}(\mathcal{S}_{t+1}) - V_{t+1}(\mathcal{S}_{t+1} \cup \{r,l\}) \\ \quad \text{if } x_{trl} = 0,\ r,l \notin \mathcal{S}_t, \\ V_{t+1}(\mathcal{S}_{t+1}) - V_{t+1}(\mathcal{S}_{t+1} \cup \{r,l\}) \\ \quad \text{if } x_{trl} = 1. \end{cases}$$

The four cases under $x_{trl} = 0$ are necessary to cover the various instances pertaining to the availability of r and l under $\mathcal{S}_t$. In the latter three of these four cases in which $x_{trl} = 0$, calculating the marginal value with respect to x_{trl} actually violates feasibility because x_{trl} cannot be set equal to 1 if either of r and l is not available at time t. Hence these three definitions are needed.

For the two feasible cases, the definition of $\partial V^*_{t+1}(\mathcal{S}_{t+1})/\partial x_{trl}$ is similar to the extension of the definition of $v_{rl}(\mathcal{S}')$ in §2.1 to $v_{t+1,\,rl}(\mathcal{S}_{t+1})$. In addition to the cases $r, l \in \mathcal{S}_{t+1}$ and $r, l \notin \mathcal{S}_{t+1}$ covered in §3, we also wish to examine the cases $r \in \mathcal{S}_{t+1}$, $l \notin \mathcal{S}_{t+1}$, and $r \notin \mathcal{S}_{t+1}$, $l \in \mathcal{S}_{t+1}$. This gives us the following definition of $v_{t+1,rl}(\mathcal{S}_{t+1})$:

$$v_{t+1,\,rl}(\mathcal{S}_{t+1}) = \begin{cases} V_{t+1}(\mathcal{S}_{t+1}) - V_{t+1}(\mathcal{S}_{t+1} - \{r,l\}) \\ \quad \text{if } r,l \in \mathcal{S}_{t+1}, \\ V_{t+1}(\mathcal{S}_{t+1} \cup \{l\}) - V_{t+1}(\mathcal{S}_{t+1} - \{r\}) \\ \quad \text{if } r \in \mathcal{S}_{t+1},\ l \notin \mathcal{S}_{t+1}, \\ V_{t+1}(\mathcal{S}_{t+1} \cup \{r\}) - V_{t+1}(\mathcal{S}_{t+1} - \{l\}) \\ \quad \text{if } r \notin \mathcal{S}_{t+1},\ l \in \mathcal{S}_{t+1}, \\ V_t(\mathcal{S}_{t+1} \cup \{r,l\}) - V_{t+1}(\mathcal{S}_{t+1}) \\ \quad \text{if } r,l \notin \mathcal{S}_{t+1}. \end{cases}$$

The conditions on these four cases are equivalent, respectively, to (1) $x_{trl} = 0$ and $r, l \in \mathcal{S}_t$, (2) $x_{trl} = 0$ and $r \in \mathcal{S}_t$, $l \notin \mathcal{S}_t$, (3) $x_{trl} = 0$ and $r \notin \mathcal{S}_t$, $l \in \mathcal{S}_t$, and (4) either $x_{trl} = 0$ and $r, l \notin \mathcal{S}_t$ or $x_{trl} = 1$. Thus we have the relationship $\partial V^*_{t+1}(\mathcal{S}_{t+1})/\partial x_{trl} = -v_{t+1,\,rl}(\mathcal{S}_{t+1})$. This means we can also think of the arc gradients as gradients with respect to resource-task pairs.

We define the following.

$\hat{v}^k_{trl}$ = the estimate of the value of resource-task pair (r,l) at time t obtained directly in iteration k. We have that $\hat{v}^k_{trl} = v_{trl}(\mathcal{S}_t)$ if $\mathcal{S}_t$ is the system state at iteration k, time t.

$\bar{v}^k_{trl}$ = the smoothed estimate of the value of resource-task pair (r,l) at time t after iteration k. In particular, for smoothing function α^k, $\bar{v}^k_{trl} = \alpha^k \hat{v}^k_{trl} + (1-\alpha^k)\bar{v}^{k-1}_{trl}$.

Using the arc gradients we can then approximate $V_{t+1}(\mathcal{S}_{t+1})$ by

$$\begin{aligned} \widehat{V}^{rl}_{t+1}(\mathcal{S}_{t+1}) &= \sum_{r\in\mathcal{R}} \sum_{l\in\mathcal{L}} \frac{\partial V^*_{t+1}(\mathcal{S}_{t+1})}{\partial x_{trl}} \cdot x_{trl} \\ &= \sum_{r\in\mathcal{R}} \sum_{l\in\mathcal{L}} -v_{t+1,\,rl}(\mathcal{S}_{t+1}) \cdot x_{trl}. \end{aligned}$$

At iteration k, time t, we have

$$\widehat{V}^{rl,\,k}_{t+1}(\mathcal{S}_{t+1}) = \sum_{r\in\mathcal{R}} \sum_{l\in\mathcal{L}} -\bar{v}^{k-1}_{t+1,\,rl} \cdot x_{trl}.$$

This gives us the following approximation

$$\widetilde{V}_t^{rl,k}(\mathcal{S}_t) = \max_{x_t \in X_t} \left\{ \sum_{r \in \mathcal{R}} \sum_{l \in \mathcal{L}} \left(c_{trl} - \bar{v}_{t+1,rl}^{k-1}\right) \cdot x_{trl} \right\}. \quad (25)$$

This leads to a network formulation of the problem consisting of the usual resources, tasks, and arcs but with the modification that, for each arc, the contribution is now the original contribution minus the corresponding arc gradient. Because the decision variables are not independent of each other, this approximation is, of course, nonseparable.

3.3. An Adaptive Dynamic Programming Algorithm

We now present our basic algorithm. In each iteration k and at each time t we solve a network assignment problem consisting of the currently known resources, tasks, and contributions. In addition, we incorporate our resource, task, resource and task, or arc gradients from iteration $k-1$ into the network. (For instance, resource gradients would be included as contributions on the arcs from the respective resources to the supersink. Arc gradients would be subtracted from the contributions on the corresponding arcs.) After solving this problem using a network solver we remove the assigned resources and tasks from the system and roll forward in time.

The gradients for iteration k are not calculated until after the forward pass has been completed. Thus the history of the entire stochastic process for iteration k is known. We calculate the gradients backward by constructing, for each time t, a network consisting of the currently known information as well as all of the information that becomes available at times $t+1, \ldots, T$. We believe that calculating the gradients in this fashion captures more accurately the actual marginal impact of the resources, tasks, or resource-task pairs on the system. This is because including this later information, rather than just the information available at time t, incorporates into the gradients some of the downstream impact of removing the resources or tasks, or both, from the system. These gradients are then smoothed in some fashion with the gradients from iteration $k-1$ for use in iteration $k+1$. Calculating the gradients in this fashion does not violate knowledge measurability constraints because none of the gradients in iteration k are used until iteration $k+1$. By the start of iteration $k+1$, all of the information from iteration k is known, and thus we can use all of this information in calculating the gradients.

We present our algorithm using the resource gradients approximation.

The algorithm can be easily modified to include the task gradients or to use the arc gradients approximations instead, as we detail below. The algorithm with resource gradients is as follows.

Step 0. Determine a maximum number of iterations K. Set $\hat{v}_{tr}^0 = 0$ and $\bar{v}_{tr}^0 = 0$ for all r and t. Set $k = 1$, $t = 0$.

Step 1. For the current k and t, solve the assignment problem

$$\begin{aligned} \widetilde{V}_t^{r,k}(\mathcal{S}_t) = \max_{x_t} \quad & \left\{ \sum_{r \in \mathcal{R}} \sum_{l \in \mathcal{L}} \left(c_{trl} - \bar{v}_{t+1,r}^{k-1}\right) \cdot x_{trl} \right\} \\ \text{subject to} \quad & \sum_{l \in \mathcal{L}} x_{trl} \le \begin{cases} 1 & \forall\, r \in \mathcal{R}_t, \\ 0 & \forall\, r \notin \mathcal{R}_t, \end{cases} \\ & \sum_{r \in \mathcal{R}} x_{trl} \le \begin{cases} 1 & \forall\, l \in \mathcal{L}_t, \\ 0 & \forall\, l \notin \mathcal{L}_t, \end{cases} \\ & x_{trl} \in \{0,1\} \quad \forall\, r \in \mathcal{R},\ \forall\, l \in \mathcal{L}. \end{aligned} \quad (26)$$

Step 2: *Transition.* Once the $\arg\max x_t$ in Step 1 is determined, let $\mathcal{R}_{t+1} = \mathcal{R}_t \cup \widehat{\mathcal{R}}_{t+1} - \mathcal{X}_t^R$ and $\mathcal{L}_{t+1} = \mathcal{L}_t \cup \widehat{\mathcal{L}}_{t+1} - \mathcal{X}_t^L$.

Step 3. If $t < T$ then $t = t+1$ and go to Step 2.

Step 4: *Backwards Calculation of Resource Gradients.* Let N_t be the network consisting of all resources and tasks available at iteration k and times $t' \ge t$. Let $c_{rl} = c_{r,l,\tau_{rl}^a}$. Then, for the current k and t, and for each r and l that become available by time t (even if one or both were assigned before time t), calculate $\hat{v}_{tr}^k$ according to one of the following cases.

Case 1. If r is available at time t, then $\hat{v}_{tr}^k = C(N_t) - C_r^-(N_t)$.

Case 2. If r is not available at time t, then $\hat{v}_{tr}^k = C_r^+(N_t) - C(N_t)$.

Step 5: *Smoothing.* For each r, set $\bar{v}_{tr}^k = \alpha^k \hat{v}_{tr}^k + (1-\alpha^k)\bar{v}_{tr}^{k-1}$ (for some smoothing function α^k).

Step 6. If $t > 0$, then $t = t-1$ and go to Step 4.

Step 7. If $k < K$, then $k = k+1$ and go to Step 1; otherwise stop.

Modifications of the algorithm include using the resource and task gradients approximation rather than just the resource gradients approximation in Step 1. In this case we would denote the value function by $\widetilde{V}_t^{r,l,k}(\mathcal{S}_t)$ rather than by $\widetilde{V}_t^{r,k}(\mathcal{S}_t)$. This would also involve calculating task and resource gradients in Step 4.

We could also calculate arc gradients instead of resource and task gradients. This would involve the following logic in Step 4.

Case 1. If r and l are available at time t, then $\hat{v}_{trl}^k = C(N_t) - C_{rl}^-(N_t)$.

Case 2. If r and l are not available at time t, then $\hat{v}_{trl}^k = C_{rl}^+(N_t) - C(N_t)$.

Case 3. If either of r and l is available at time t and the other is not, then $\hat{v}_{trl}^k = c_{r,l,\tau_{rl}^a}$. (See Theorem 2.)

These arc gradients would then be incorporated into solving the value function (a $\widetilde{V}_t^{rl}$ in this case) in Step 1.

Remark on Case 3. If r is available and l is not, for instance, then the logical extension from Cases 1 and 2 to Case 3 is (and in fact our definition of $v_{trl}(\mathcal{S}_t)$ implies) something like $C_l^+(N_t) - C_r^-(N_t)$. However, this calculation includes in neither term the value of the basis network N_t. From Theorem 2 the lower bound for $\hat{v}_{trl}^k$ in Cases 1 and 2 is c_{r,l,τ_{rl}^a}. While this is not guaranteed to be a lower bound for $\hat{v}_{trl}^k$ in Case 3, it seems a reasonable approximation, since in practice the value of $\hat{v}_{trl}^k$ in Cases 1 and 2 is often found to be close to its lower bound. The approximation $\hat{v}_{trl}^k = c_{r,l,\tau_{rl}^a}$ has the additional advantages of being extremely easy to calculate and works well in our experiments. Of course, the more obvious $C_l^+(N_t) - C_r^-(N_t)$ calculation as well as other types of logic are also possible.

The gradients $\hat{v}_{tr}^k$, $\hat{v}_{tl}^k$, and $\hat{v}_{trl}^k$ can be calculated either with flow-augmenting paths (as discussed in §2.1) or numerically, by calculating (in the case of $\hat{v}_{tr}^k$) $C(N_t)$ and $C_r^-(N_t)$ (or $C_r^+(N_t)$ and $C(N_t)$) and taking the difference. The numerical calculations using a network simplex code can be quite fast, provided one uses the solution of $C(N_t)$ as a warm start, rather than cold starting, in calculating $C_r^-(N_t)$ and $C_r^+(N_t)$.

3.4. Experimental Results

We are interested in the relative performance of the different classes of functional approximations, and their comparison against posterior optimal solutions. We created datasets with the intent of approximating a single region for a major truckload motor carrier. (Most companies use approximately 100 regions to represent the United States.) To capture the range from sparse to dense problems, we created 20 test problems by randomly selecting different numbers of points on a grid to represent the resources and tasks. The datasets range in size from 2 resources and 2 tasks (Dataset 5) to 55 resources and 55 tasks (Dataset 100). The number associated with a particular dataset is roughly the total number of resources and tasks in the set. The initial contribution for assigning a resource to a task was made to be a function of the distance between the resource point and the task point. The contribution for assigning a particular resource to a particular task decreases over time from its initial value.

In the experiments described below, we would perform 100 training iterations for estimating the value of resources in the future, followed by 100 testing iterations to evaluate solution quality. The code was not tuned for speed, but since we were only solving sequences of single-period assignment problems, none of the run times were especially large. The most difficult algorithm used the arc gradients to estimate the value of resources in the future. This algorithm required approximately 0.3 minutes per iteration on the largest datasets; the run times are computed assuming a 2 GHz Intel processor. In an operational setting, the training iterations would be done in background, and real-time operations would require solving only a single assignment problem, because the remainder of the forward pass is used only to approximate the value of resources in the future.

We first performed runs on deterministic datasets where the posterior optimal solution represents a tight bound. Table 1 summarizes the results, comparing a myopic model against models using resource gradients alone, resource and task gradients, and the arc gradients. The results show that all three classes of nonmyopic strategies outperform the myopic model. Furthermore, the arc gradients version performs the best, with the resource and task gradients second best, as we would expect. It is significant that the arc gradient algorithm produces near-optimal (and often optimal) solutions, demonstrating that it is doing a better job of approximating the value function. However, the computational burden is quite high because it requires a calculation for every arc, and not just for every node.

The next set of runs was performed under uncertainty. We held the set of resource and tasks fixed, but introduced randomness in the cost on an assignment arc. We introduced a random cost that reflected

Table 1 Resources and Tasks Arrive Over Time: Deterministic Runs

Dataset size	Myopic	Resource gradients	Resource and task gradients	Arc gradients
5	100	100	100	100
10	93.4	100	100	100
15	90.8	90.8	100	100
20	89.3	89.3	100	100
25	87.4	98.0	98.6	100
30	90.6	97.9	98.4	100
35	93.5	98.4	99.6	99.8
40	84.8	96.5	98.5	98.7
45	96.9	96.9	100	100
50	84.6	98.5	99.3	99.8
55	83.5	83.5	98.6	99.9
60	84.9	84.9	88.2	100
65	82.7	96.4	99.5	100
70	86.1	94.8	98.7	99.9
75	84.8	84.8	84.8	98.1
80	81.6	80.1	99.6	99.9
85	92.5	91.7	98.6	100
90	88.5	99.0	99.9	99.9
95	83.5	96.8	99.5	100
100	89.0	89.0	89.0	99.9
Mean	88.4	93.4	97.5	99.8
Median	88.0	96.5	99.4	100

whether the user felt the assignment was "acceptable"; see Powell et al. (2000b) for a discussion of user-acceptance issues. We made the assumption that any resource-task assignment would be acceptable with probability 50%; this is equivalent to assuming that the cost on an assignment arc is zero or "big M" with equal probability. For the arc (r, l) this cost was not known until τ^a_{rl}, the earliest time at which resource r and task l were both available.

We tested our algorithm using the resource, resource and task, and arc gradients variations. For our training phase on each dataset we averaged the gradients from 100 iterations. In each iteration we used the posterior optimal, as described in the previous section, as the solution on which to base the gradient calculations. We then ran another 100 iterations to test these gradients.

Table 2 presents the stochastic runs. On average, each of the three gradients variations of our algorithm outperforms the myopic solution. As in the deterministic case, the variation that only uses resource gradients performs the worst of the three. However unlike the deterministic case, the resource and task gradients version slightly outperforms the arc gradients version. A possible explanation for this is that there are so many more arc gradients than resource and task gradients to calculate that it requires a much larger training phase to achieve a similar degree of accuracy. The results suggest that the arc gradients version of the algorithm may not be useful in the context of stochastic applications.

Table 2 Resources and Tasks Arrive Over Time: Stochastic Runs

Dataset size	Myopic	Resource gradients	Resource and task gradients	Arc gradients
5	97.1	93.6	93.6	97.6
10	90.2	93.9	93.7	95.4
15	90.4	86.6	90.8	92.6
20	87.9	84.5	91.8	88.4
25	87.6	94.2	95.1	95.9
30	86.2	90.4	91.5	92.7
35	92.4	93.8	93.4	96.1
40	84.0	89.1	91.2	90.5
45	92.0	90.0	96.5	97.1
50	85.8	93.4	93.8	93.6
55	83.1	80.7	92.4	88.2
60	81.5	84.5	88.4	83.5
65	82.4	92.0	93.4	91.1
70	85.4	91.8	93.0	89.5
75	83.0	86.4	90.0	91.5
80	80.1	79.5	93.9	89.7
85	88.4	86.3	93.9	88.9
90	87.1	95.8	96.3	93.8
95	83.2	91.7	94.2	91.6
100	84.3	85.5	89.7	91.7
Mean	86.6	89.2	92.8	92.0
Median	86.0	90.2	93.4	91.7

4. Hierarchical Aggregation for Problems with Large Attribute Spaces

Up to now we have looked at the value of a particular resource or task in the future. Implicitly, this approach allows us to model the attributes of a resource or task at a high level of detail. Such an approach requires either that we be able to enumerate all the resources and tasks (and their attributes) that might arise in the future or that we be able to enumerate all the possible attribute vectors of resources and tasks. In practice, this assumption will generally not hold. For real problems, attribute vectors can be quite detailed, making it impossible to enumerate all possible outcomes, and still produce a computationally tractable algorithm. Even if we could enumerate them, we would encounter a problem of statistical reliability.

We would like to be able to make a decision to assign a resource with attribute vector a_r and take into consideration the value of a resource with this attribute vector in the future. Thus, rather than estimating the value of resource r at time $t + 1$, we would like to estimate the value of a resource with attribute a_r in the future, which we might denote by $\hat{v}_{t+1,a}$. Because we are creating these estimates through Monte Carlo sampling, we have to face the problem that we may need to use an estimate of $\hat{v}_{t+1,a}$ based on very few observations; often, for example, we have never observed a particular attribute vector.

To illustrate, assume that the only attribute of a driver is his location, expressed as a set of continuous coordinates (x, y). Assume we want to consider assigning a driver to a load with a destination at the point (x', y'). To properly evaluate the value of assigning a driver to this load, we would need to know the value of a driver at location (x', y'). Now we face a statistical estimation problem: How do we estimate the value of a driver with these coordinates? Defined over continuous space, the likelihood of sampling another driver with the same coordinates is negligible, and we would need many observations to obtain a statistically reliable estimate. The natural strategy is to divide the region into a set of zones. This approach, however, introduces the classic tradeoff between statistical error (larger zones provide larger samples) and structural error (smaller zones are better).

A common strategy in dynamic programming is to choose a level of aggregation that seems to strike a reasonable balance between statistical error and structural error. A single level of aggregation, however, ignores the fact that some regions of our network have a higher density of activity and will produce

larger samples. More problematical is that algorithms in the early stages have to deal with value functions that are estimated with a small number of iterations, and small samples, which means that we may have more observations of drivers in some areas than others. The decision of choosing the right level of aggregation can be a function of how many iterations the algorithm has progressed. We propose and test a hierarchical aggregation procedure that simultaneously estimates the value of a driver at different levels of aggregation. We have not seen this technique in the general dynamic programming literature, and it is certainly new to the routing and scheduling literature.

Aggregation remains a widely used technique in the operations research literature to handle complex problems; see, in particular, the survey paper by Rogers et al. (1991). Most of the aggregation algorithms in the dynamic programming literature also involve aggregating the original problem, solving the aggregated problem, and disaggregating to find a solution. The aggregation is done in order to reduce the size of the state space. Some algorithms of this type include those in Hinderer (1978), Mendelssohn (1982), Bean et al. (1987) (which is for deterministic dynamic programs only), and Bertsekas and Castanon (1989). Morin (1978) is a general survey paper of the older literature. Puterman (1994) presented a general technique for approximating countable-state Markov decision processes using a finite number of states, complete with error bounds, but this technique essentially consists of truncating the number of states. Whitt (1978, 1979) performed some error analysis.

Bertsekas and Tsitsiklis (1996) provided a good general discussion of partitioning techniques, including using grids, exploiting special features, breaking the value function approximation into a global value and the sum of local values, solving small subproblems exactly and only approximating the large ones, and soft partitioning, which smooths the values of the partition at the edges. Our approach represents a contribution to this class of strategies.

4.1. An Algorithm Based on Hierarchical Aggregation

We perform aggregation of resources and tasks through the use of a collection of aggregation functions

$$G^n\colon \mathscr{A} \to \mathscr{A}^{(n)},$$

where G^n represents the nth level of aggregation of the attribute space $\mathscr{A}$. It is not necessarily the case that $\mathscr{A}^{(n)} \subseteq \mathscr{A}^{(m)}$ for $n \le m$.

We also define

$a_r^{(n)} = G^n(a_r)$, the nth level aggregation attribute vector associated with resource r;

$a_l^{(n)} = G^n(a_l)$, the nth level aggregation attribute vector associated with task l;

$R_t^{(n)}$ = vector of aggregated resources at time t, where $R_{ta}^{(n)} = \sum_{r\in\mathscr{R}_t} 1_{\{G^n(a_r)=a\}}$;

$L_t^{(n)}$ = vector of aggregated tasks at time t, where $L_{ta}^{(n)} = \sum_{l\in\mathscr{L}_t} 1_{\{G^n(a_l)=a\}}$.

Our algorithmic strategy is the same as before, using linear approximations of the future to make better decisions now. As we did in §3, we can use the value functions based on resources, resources and tasks, and resource-task combinations. Because these all have the same basic structure, we illustrate the use of hierarchical aggregation using only the task gradients. In this case, our approximation would look like

$$\widehat{V}_{t+1}^{(n)r}(\mathscr{S}_{t+1}) = \sum_{a\in\mathscr{A}^{(n)}} \hat{v}_{t+1,a}^{(n)} \cdot R_{t+1,a} \tag{27}$$

$$= \sum_{a\in\mathscr{A}^{(n)}} \hat{v}_{t+1,a}^{(n)} \sum_{r\in\mathscr{R}_t} \left(1 - \sum_{l\in\mathscr{L}_t^+} x_{trl}\right) 1_{\{G^n(a_r)=a\}} \tag{28}$$

$$= \sum_{a\in\mathscr{A}^{(n)}} \hat{v}_{t+1,a}^{(n)} \sum_{r\in\mathscr{R}_t} \left(1 - \sum_{l\in\mathscr{L}_t^+} x_{trl}\right) 1_{\{G^n(a_r)=a\}}. \tag{29}$$

We are only interested in the portion of $\widehat{V}$ that depends on x, so we drop the constant term and retain only the portion that includes x, giving us

$$\widehat{V}_{t+1}^{(n)r}(\mathscr{S}_{t+1}) = -\sum_{r\in\mathscr{R}_t} \sum_{l\in\mathscr{L}_t^+} x_{trl} \left(\sum_{a\in\mathscr{A}^{(n)}} \hat{v}_{t+1,a}^{(n)} 1_{\{G^n(a_r)=a\}} \right). \tag{30}$$

Let $\hat{a}_r$ satisfy $1_{\{G^n(a_r)=\hat{a}_r\}} = 1$, which means that a_r maps to $\hat{a}_r$ under aggregation G^n; we only use $\hat{a}_r$ when the level of aggregation is clear. This allows us to write (30) as

$$\widehat{V}_{t+1}^{(n)r}(\mathscr{S}_{t+1}) = -\sum_{r\in\mathscr{R}_t} \sum_{l\in\mathscr{L}_t^+} x_{trl} \hat{v}_{t+1,\hat{a}_r}^{(n)}. \tag{31}$$

Combining Equations (17) and (31) gives

$$\widetilde{V}_t(\mathscr{S}_t) = \max_{x_t\in\mathscr{X}_t} \sum_{r\in\mathscr{R}_t} \sum_{l\in\mathscr{L}_t^+} \left(c_{trl} - \hat{v}_{t+1,\hat{a}_r}^{(n)}\right). \tag{32}$$

We now turn to the problem of actually calculating $\hat{v}_{t,\hat{a}_r}^{(n)}$. Our approximation methodology involves solving the function $\widetilde{V}_t(\mathscr{S}_t)$ in Equation (17). We compute the value of a resource r in the set $\mathscr{S}_t$ using

$$\tilde{v}_{tr}(\mathscr{S}_t) = \begin{cases} \widetilde{V}_t(\mathscr{S}_t) - \widetilde{V}_t(\mathscr{S}_t - \{r\}) & \text{if } r \in \mathscr{S}_t, \\ \widetilde{V}_t(\mathscr{S}_t \cup \{r\}) - \widetilde{V}_t(\mathscr{S}_t) & \text{if } r \notin \mathscr{S}_t, \end{cases}$$

where the calculation of these gradients can be simplified using flow augmenting path algorithms (see Powell 1989). We now need to produce aggregated estimates of these values. Aggregated versions have to be defined relative to the attribute spaces $\mathcal{A}^R$, $\mathcal{A}^L$. Aggregation can be formed in different ways (Ling and Butler 1999). Assume we have a collection of resources $\mathcal{R}_{\hat{a}}$ where if $r, r' \in \mathcal{R}_{\hat{a}}^{(n)}$, then $\hat{a}_r = \hat{a}_{r'}$. Then let

$f^a(\mathcal{V})$ = the aggregating function that takes a family of values $\mathcal{V}$ and combines them to produce the values of the aggregate vectors.

This function could produce the mean, a weighted mean, the median, or some other combination of the disaggregate values. Let

$$\mathcal{V}_{ta}^{(n)r} = \{v_{tr} \mid r \in \mathcal{S}_t,\ a_{tr}^{(n)} = a\}$$

= the set of values (in this case, for resources) that aggregate up to the same attribute vector $a \in \mathcal{A}^{(n)}$.

We use $\mathcal{V}_{ta}^{(n)}$ when we want to refer to a generic set of values to be aggregated. We then define

$$\tilde{v}_{ta}^{(n)}(\mathcal{S}_t) = f^{\text{avg}}(\mathcal{V}^{(n)}).$$

As an example, f^{avg} as the mean function gives

$$\tilde{v}_{ta}^{(n)}(\mathcal{S}_t) = \frac{\sum_{\{r \in \mathcal{V}_{ta}^{(n)}\}} \tilde{v}_{tr}}{|\mathcal{V}_{ta}^{(n)}|}.$$

In an iterative setting, we would let $\tilde{v}_{tr}^k$ represent an estimate of the value of resource r at time t, and let $\tilde{v}_{ta}^{(n),k}$ be an aggregated estimate at iteration k.

Finally, having obtained an aggregated estimate of the value of a resource (task or resource-task), we perform the usual smoothing to obtain

$$\hat{v}_{ta}^{(n),k+1} = (1-\alpha^k)\hat{v}^{(n),k} + \alpha^k \tilde{v}_{ta}^{(n),k},$$

where $0 < \alpha^k < 1$ is a stepsize.

4.2. Adaptive Hierarchical Aggregation

In this section we consider using different levels of aggregation in an adaptive setting, and we present a modification of our algorithm from §3. The basic idea is to estimate the value of a resource v_{ta} with attribute a at different levels of aggregation. Clearly, as the level of aggregation increases, we gain statistical accuracy but increase structural errors. For each level n of aggregation and time t, define

$s^2(v_{t,a_r}^{(n)})$ = the estimated variance of the value of resource r with attribute vector a at the nth level of aggregation at time t. (We use the estimated variance s^2 rather than actual variance σ^2 because we do not know σ^2.)

Then, when we need the value of resource r at time t, we set

$$\hat{v}_{tr} = v_{t,a_r}^{(m)},$$

where

$$m = \arg\min_n \{s^2(v_{t,a_r}^{(n)})\}.$$

We incorporate this into our resource gradients algorithm by adaptively choosing the level of aggregation at each iteration. Our algorithm, then, will estimate the value of a resource in the future by choosing the level of aggregation which produces the lowest error.

4.3. Experimental Testing of Hierarchical Aggregation

We now present our experimental results. We examine the effects of different levels of aggregation as well as the performance of our resource, task, resource and task, and arc gradients algorithms. Our major question, however, is how well the hybrid algorithm from §4.2 works.

Our datasets are created by associating each resource and each task with a random point on a $1{,}000 \times 1{,}000$ square. Thus the attribute vector of each resource and task consists of a unique label and a location on the square. The contribution for assigning a resource to a task is an inverse function of the distance between the resource and the task. Aggregation is achieved by imposing different-sized grids on the square, with the value of a resource or task set to be the value of the grid cell in which the resource or task lies. Our results are presented as a percentage of the posterior optimal solution.

We now present our results on the effects of different grid sizes. In conducting these experiments we used a battery of 20 deterministic datasets of various sizes. We tested our resource and task gradients algorithm on these datasets using several grid sizes. Figure 2 shows the performance of several grid sizes on the stochastic version of dataset 60 for iterations 1 through 1,000. The results that are plotted are exponentially smoothed with a smoothing factor of 0.05.

As we would expect, the performance at the most aggregate level performs the best over the earliest iterations. As the number of iterations increases, successively lower levels of aggregation perform better. If we were to use a single level of aggregation, we would be likely to obtain poor results if we can only run a small number of iterations, and yet we would never achieve the best results if we were to run a large number of iterations.

We now present results from using our hybrid algorithm discussed in §4.2. For these experiments our two levels of aggregation were chosen to be the 1×1 grid and the 5×5 grid. We ran 1,000 iterations of the resource and task gradients algorithm on a series

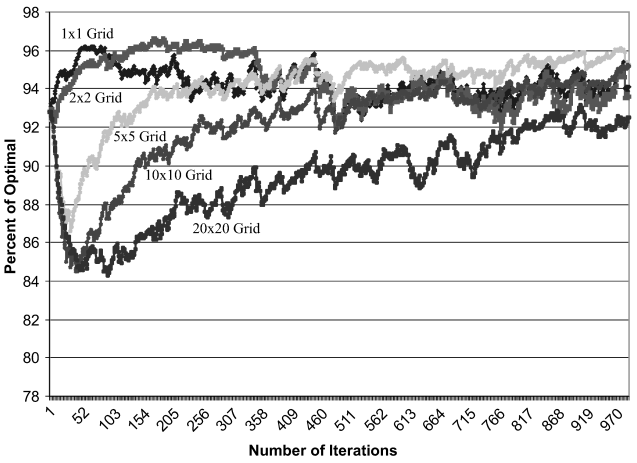

Figure 2 Comparison of Different Grid Sizes on Dataset 60

of stochastic datasets of increasing sizes for the pure 1×1 grid algorithm, the pure 5×5 grid algorithm, and the hybrid algorithm. The choice between levels of aggregation for a particular resource r at time t was made based on the smaller of the two values of $s^2(v_{tr})$ (the estimate of the sample variance of the value of resource r at time t) for each level of aggregation. What is important about these runs is not the final solution, but rather the rate of convergence.

Figure 3 illustrates the performance of each of the three algorithms (aggregation fixed on a 1×1 grid size, aggregation fixed on a 5×5 grid size, and the hybrid aggregation algorithm). Finer levels of aggregation will, as a rule, always work best after sufficiently many iterations, but practical problems require good convergence after just a few iterations. The 1×1 fixed aggregation works the best initially, but significantly underperforms the 1×1 grid after about 500 iterations. The hybrid outperforms both algorithms over the entire range.

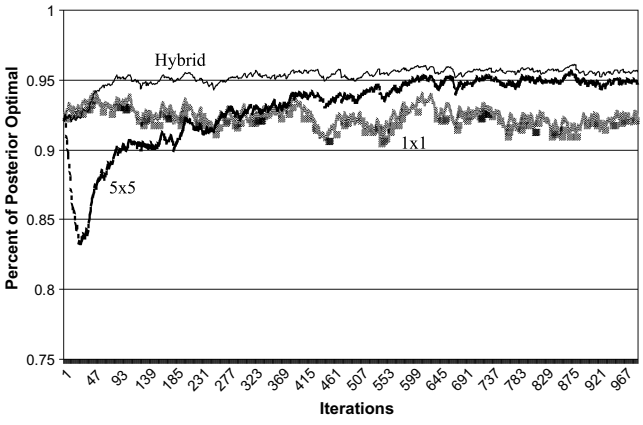

Figure 3 Comparison of 1×1, 5×5, and Hybrid Algorithms on Dataset 100

5. The Value of Advance Information

An important issue in the study of real-time models is the value of advance information. It is possible to undertake comparisons of different degrees of advance information using a myopic model, but this ignores the ability people have to anticipate the future in the form of distributional information. For example, a dispatcher might think, "I am going to hold this truck in this area because I have a lot of customers nearby and I will probably get a phone call," which we would represent using a probability distribution describing the number of phone calls we expect to receive. For this reason, it is useful to use our dynamic policy to estimate the value of advance information. We also undertake comparisons against a myopic model, where the expectation is that a myopic model should work fine if there is enough advance information.

Our study focuses on the difference between when a resource or task becomes known τ and when it becomes actionable τ^a. We refer to the probability distribution describing the difference $\tau^a - \tau$ as the *booking profile*. Although it is most common to think of random information arising about tasks, there are problems where the same issue arises with the vehicles (for example, this may be a major problem in the movement of empty rail cars). Up to now our experiments assumed that $\tau = \tau^a$. Thus when resources and tasks become known they are immediately actionable. However, this is often not the case in the real world; frequently information about resources and tasks becoming available in the future is known in advance.

We conducted a study where the knowable times, τ_r and τ_l, are generated as they were before, while the actionable times (for a particular task l) were computed using

$$\tau_l^a = \tau_l + \beta_l,$$

where β_l is a random variable representing the booking profile. We assumed, as would typically be the case, that β_l becomes known at time τ^a. We further assumed that β_l was uniformly distributed between 0 and $\beta^{\max}$; we used the same distribution for both resources and tasks. When the assignment model is run at time t, all resources and tasks with $\tau \le t$ and $\tau^a \le t + \tau^{ph}$ are considered, where τ^{ph} is the planning horizon. Any decisions that are actionable at time t are implemented, while any decisions that are actionable in the future are reoptimized in the next period.

The value of knowing the future is, of course, a function of the cost of making the wrong decision. For our problem, this is controlled by the transportation cost. If the transportation cost were zero, then there would be no value of advance information. In our test problems each resource and each task has a

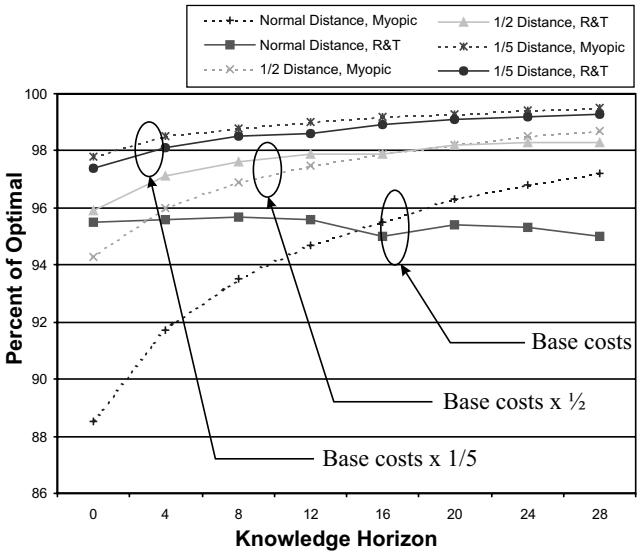

Figure 4 **The Difference Between the Adaptive Dynamic Profits Using Resource and Task Gradients (R&T) and the Myopic Algorithm, as a Function of the Planning Horizon, for Different Transportation Costs**

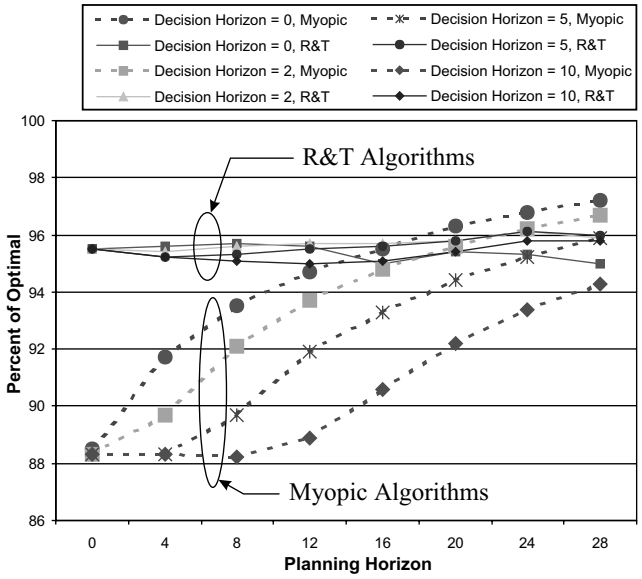

Figure 5 **The Effect of the Decision Horizon as a Function of the Planning Horizon, for the Myopic Algorithm and the Adaptive Dynamic Programming Algorithm with Resource and Task Gradients (R&T)**

location on a $1{,}000 \times 1{,}000$ grid, and the value of serving a task is fixed at \$2,000, while the cost of serving a task is equal to the distance from the resource to the task times a transportation cost of \$1 per mile. We then conducted experiments where the transportation cost was one half and one fifth of the base costs. The results are shown in Figure 4. The results show that as the transportation cost declines, the difference between the two algorithms over different planning horizons diminishes quickly. The obvious conclusion is that if it does not really matter what you will be doing in the future, then you do not have to anticipate the future.

In Figure 5 we examine the effects of decision horizons in the presence of advance information. A decision horizon is the period into the future in which decisions, once made, are locked into place. Up to now, our decision horizon has always been a single period. Locking in decisions now that cannot be implemented until some time in the future (perhaps for reasons of driver notification) would always be expected to perform worse than policies that do not lock in decisions. Implementing a decision horizon of length τ^{dh}, in these experiments, means that any assignment made at time t that is actionable between t and time $t + \tau^{dh}$ is "locked in" and acted upon at the time it does become actionable. For these runs, we used $\beta^{\max} = 30$.

The results are shown in Figure 5. They indicate, as we would expect, that the myopic model performs worse as the decision horizon is lengthened, over the entire range of planning horizons. The dynamic programming approximation, however, is relatively independent of the decision horizon, with results that are consistently better than all the myopic models as long as the planning horizon is not too long.

6. Conclusions

This paper suggests a strategy for solving dynamic assignment problems that is computationally tractable, requiring no specialized algorithms. The experimental work is promising, but more research is required to test the effectiveness of this approach on different problem classes. For example, the section on hierarchical aggregation did not explicitly test the method on multiattribute resources and tasks. Also, it is likely that other statistical methods such as nonparametric regression would produce even better results.

Our presentation has focused on the dynamic assignment problem where resources and tasks may be held if they are not acted on, but vanish from the system if a resource is coupled to a task. Often, the task vanishes from the system but the resource reappears. Because our approach captures the value of a resource if it is held, the same approach can be used to capture the value of a resource in the future.

We have considered only linear value function approximations. For larger problems, it is likely that the value of a vehicle in a particular region, for example, would depend on the number of other vehicles in the region at the same time. For this, it would be possible to use nonlinear functional approximations. Possible options include the use of polynomial

approximations, such as those investigated in Tsitsiklis and Van Roy (1997), or the separable, piecewise linear approximations used in Godfrey and Powell (2002).

Acknowledgments

This research was supported in part by Grant AFOSR-F49620-93-1-0098 from the Air Force Office of Scientific Research. The authors would also like to acknowledge the many helpful comments of the reviewers and the assistance of the editors.

References

Ahuja, R., T. Magnanti, J. Orlin. 1992. *Network Flows: Theory, Algorithms and Applications*. Prentice Hall, Upper Saddle River, NJ.

Balinski, M. 1985. Signature methods for the assignment problem. *Oper. Res.* **33** 527–537.

Balinski, M. 1986. A competitive (dual) simplex method for the assignment problem. *Math. Programming* **34**(2) 125–141.

Balinski, M., R. Gomory. 1964. A primal method for the assignment and transportation problems. *Management Sci.* **10** 578–593.

Bander, J., C. C. White. 1999. Markov decision processes with noise-corrupted and delayed state observations. *J. Oper. Res. Soc.* **50**(6) 660–668.

Barr, R., F. Glover, D. Klingman. 1977. The alternating path basis algorithm for assignment problems. *Math. Programming* **13** 1–13.

Bean, J., J. Birge, R. Smith. 1987. Aggregation in dynamic programming. *Oper. Res.* **35** 215–220.

Bertsekas, D. 1981. A new algorithm for the assignment problem. *Math. Programming* **21** 152–171.

Bertsekas, D. 1988. The auction algorithm: A distributed relaxation method for the assignment problem. *Ann. Oper. Res.* **14** 105–123.

Bertsekas, D., D. Castanon. 1989. Adaptive aggregation methods for infinite horizon dynamic programming. *IEEE Trans. Automatic Control* **34**(6) 589–598.

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

Bertsekas, D., J. Tsitsiklis, C. Wu. 1997. Rollout algorithms for combinatorial optimization. *J. Heuristics* **3**(3) 245–262.

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

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

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

Cheung, R. K.-M., W. B. Powell. 2000. SHAPE: A stochastic hybrid approximation procedure for two-stage stochastic programs. *Oper. Res.* **48**(1) 73–79.

Cook, T., R. Russell. 1978. A simulation and statistical analysis of stochastic vehicle routing with timing constraints. *Decision Sci.* **9** 673–687.

Dantzig, G. 1963. *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ.

Gale, D., L. Shapley. 1962. College admissions and the stability of marriage. *Amer. Math. Monthly* **69** 9–15.

Gendreau, M., F. Guertin, J. Potvin, E. Taillard. 1999. Parallel tabu search for real-time vehicle routing and dispatching. *Transportation Sci.* **33** 381–390.

Glasserman, P., D. Yao. 1994. *Monotone Structure in Discrete-Event Systems*. John Wiley and Sons, New York, 234–238.

Godfrey, G., W. B. Powell. 2002. An adaptive, dynamic programming algorithm for stochastic resource allocation problems, I: Single period travel times. *Transportation Sci.* **36**(1) 21–39.

Goldfarb, D. 1985. Efficient dual simplex methods for the assignment problem. *Math. Programming* **33** 187–203.

Gross, O. 1959. The bottleneck assignment problem. Technical Report p-1630, The RAND Corporation.

Hall, L., A. Schulz, D. Shmoys, L. Wein. 1997. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Math. Oper. Res.* **22** 513–544.

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

Hinderer, K. 1978. On approximate solutions of finite-stage dynamic programs. M. Puterman, ed. *Dynamic Programming and Its Applications.* Academic Press, New York.

Hoogeveen, J., A. Vestjens. 2000. A best possible deterministic on-line algorithm for minimizing delivery time on a single machine. *SIAM J. Discrete Math.* **13** 56–63.

Hung, M. 1983. A polynomial simplex method for the assignment problem. *Oper. Res.* **31** 595–600.

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

Jonker, R., A. Volegnant. 1987. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing* **38** 325–340.

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

Lageweg, B., J. Lenstra, A. R. Kan, L. Stougie. 1988. Stochastic integer programming by dynamic programming. Y. Ermoliev, R. Wets, eds. *Numerical Techniques for Stochastic Optimization*. Springer-Verlag, Berlin, Germany, 403–412.

Laporte, G., F. Louveaux. 1993. The integer l-shaped method for stochastic integer programs with complete recourse. *Oper. Res. Lett.* **13**(3) 133–142.

Ling, B., R. Butler. 1999. Comparing effects of aggregation methods on statistical and spatial properties of simulated spatial data. *Photogrammatic Engrg. Remote Sensing* **65**(1) 73–84.

Louveaux, F., M. van der Vlerk. 1993. Stochastic programming with simple integer recourse. *Math. Programming* **61** 301–325.

Mendelssohn, R. 1982. An iterative aggregation procedure for Markov decision processes. *Oper. Res.* **30**(1) 62–73.

Morin, T. L. 1978. Computational advances in dynamic programming. M. Puterman, ed. *Dynamic Programming and Its Applications.* Academic Press, New York.

Murty, K. 1992. *Network Programming.* Prentice Hall, Englewood Cliffs, NJ.

Pinedo, M. 1995. *Scheduling: Theory, Algorithms, and Systems.* Prentice Hall, Englewood Cliffs, NJ.

Powell, W. B. 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.

Powell, W. B. 1996. A stochastic formulation of the dynamic assignment problem, with an application to truckload motor carriers. *Transportation Sci.* **30**(3) 195–219.

Powell, W., W. Snow, R. Cheung. 2000a. Adaptive labeling algorithms for the dynamic assignment problem. *Transportation Sci.* **34** 67–85.

Powell, W., M. T. Towns, A. Marar. 2000b. On the value of globally optimal solutions for dynamic routing and scheduling problems. *Transportation Sci.* **34**(1) 50–66.

Psaraftis, H. 1980. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Sci.* **14** 130–154.

Psaraftis, H. 1988. Dynamic vehicle routing problems. B. Golden, A. Assad, eds. *Vehicle Routing: Methods and Studies.* North Holland, Amsterdam, The Netherlands, 223–248.

Psaraftis, H. 1995. Dynamic vehicle routing: Status and prospects. *Ann. Oper. Res.* **61** 143–164.

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

Regan, A., H. S. Mahmassani, P. Jaillet. 1998. Evaluation of dynamic fleet management systems—Simulation framework. *Transportation Res. Record* **1648** 176–184.

Rogers, D., R. Plante, R. Wong, J. Evans. 1991. Aggregation and disaggregation techniques and methodology in optimization. *Oper. Res.* **39**(4) 553–582.

Secomandi, N. 2001. A rollout policy for the vehicle routing problem with stochastic demands. *Oper. Res.* **49**(5) 796–802.

Shapley, L. S. 1962. Complements and substitutes in the optimal assignment problem. *Naval Res. Logist. Quart.* **9** 45–48.

Shmoys, D. B., J. Wein, D. P. Williamson. 1995. Scheduling parallel machines online. *SIAM J. Comput.* **24**(6) 1313–1331.

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

Swihart, M., J. D. Papastravrou. 1999. A stochastic and dynamic model for the single-vehicle pickup and delivery problem. *Eur. J. Oper. Res.* **114**(3) 447–464.

Tomizawa, N. 1972. On some techniques useful for solution of transportation network problems. *Networks* **1** 179–194.

Tsitsiklis, J., B. Van Roy. 1997. An analysis of temporal-difference learning with function approximation. *IEEE Trans. Automatic Control* **42** 674–690.

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.

Whitt, W. 1978. Approximations of dynamic programs I. *Math. Oper. Res.* **3** 231–243.

Whitt, W. 1979. Approximations of dynamic programs II. *Math. Oper. Res.* **4** 179–185.

Wilson, L. 1977. Assignment using choice lists. *Oper. Res. Quart.* **28**(3) 569–578.