An Approximate Dynamic Programming Algorithm for Large-Scale Fleet Management: A Case Application

Hugo P. Simao* Jeff Day** Abraham P. George*
Ted Gifford** John Nienow** Warren B. Powell*

*Department of Operations Research and Financial Engineering,
Princeton University
**Schneider National

October 29, 2009

Abstract

We address the problem of developing a model to simulate at a high level of detail the movements of over 6,000 drivers for Schneider National, the largest truckload motor carrier in the United States. The goal of the model is not to obtain a better solution but rather to closely match a number of operational statistics. In addition to the need to capture a wide range of operational issues, the model had to match the performance of a highly skilled group of dispatchers, while also returning the marginal value of drivers domiciled at different locations. These requirements dictated that it was not enough to optimize at each point in time (something that could be easily handled by a simulation model) but also over time. The project required bringing together years of research in approximate dynamic programming, merging math programming with machine learning, to solve dynamic programs with extremely high-dimensional state variables. The result was a model that closely calibrated against real-world operations, and produced accurate estimates of the marginal value of 300 different types of drivers.

In 2003, Schneider National, the largest truckload motor carrier in the United States, contracted with CASTLE Laboratory at Princeton University to develop a model that would simulate its long-haul truckload operations to perform analyses to answer questions ranging from the size and mix of its driver pool to questions about valuing contracts and getting drivers home. The requirements for the simulator seemed quite simple: it had to capture the dynamics of the real problem, producing behaviors that closely matched corporate performance along several dimensions, and it had to provide estimates of the marginal value of different types of drivers. If the model accurately matched historical performance, the company would be able to use the system to test changes in the mix of drivers, the mix of freight and other operating policies.

The major challenge we faced was that these requirements required that we do much more than develop a classical simulator. It was not enough to optimize decisions (in the form of matching drivers to loads) at a point in time. It had to optimize decisions over time so that decisions took into account downstream impacts. Formulating the problem as a deterministic, time-space network problem was both computationally intractable (the problem is huge) and too limiting (we needed to model different forms of uncertainty, as well as a high degree of realism that was beyond the capabilities of classical math programs). Classical techniques from Markov decision processes applied to this setting are limited to problems with only a small number of identical trucks moving between a few locations (see Powell (1988) or Kleywegt et al. (2004)). Our problem involved modeling thousands of drivers at a high level of detail.

We solved the problem using approximate dynamic programming, but even classical ADP techniques (Bertsekas & Tsitsiklis (1996), Sutton & Barto (1998)) would not handle the requirements of this project. Three years of development produced a model that closely matches a range of historical metrics. Achieving this goal required drawing on the research of three Ph.D. dissertations (Spivey (2001), Marar (2002), and George (2005)), and depended on the extensive participation of the sponsor to produce a model that accurately simulated operations. The model is able to handle a host of engineering details to allow the sponsor to run a broad range of simulations. To establish credibility, the model had to match historical performance of a dozen major operating statistics. Two of particular importance to our presentation

included our ability to match the average length of haul for different types of drivers, and to get drivers home with the same frequency as the company. A central hypothesis of the research, which is supported by the evidence we present in this paper, was that the behavior of a group of dispatchers could be described by an optimization model using a suitably designed objective function.

The contributions of this paper include: 1) We show, for the first time in a production setting for a truckload motor carrier, that approximate dynamic programming can provide high quality solutions while capturing operational issues at a high level of detail, including all business rules such as hours of service, returning drivers home and operational restrictions on the use of specific driver types. This appears to be the first optimization model of any form that captures the complex dynamics of a truckload motor carrier where decisions produce behavior that optimizes over time. 2) We demonstrate that the framework of approximate dynamic programming, with methods adapted to this problem class, produces a model that accurately captures the performance of a well-run company based on comparisons with historical metrics. This appears to be the first demonstrated calibration of an optimization model for truckload trucking for planning purposes. 3) We show that the value function approximations used in the dynamic programming formulation produce accurate estimates of the marginal value of particular driver types (for example, the value of adding additional team drivers domiciled in a particular region) over the entire simulation when compared against brute-force derivatives computed using the model (adding additional drivers and running the simulation again). These marginal values would not be available from a traditional simulator (which does not use the framework of dynamic programming to capture the value of a driver over the entire simulation), and mimic dual variables from a linear program (which is not able to handle the complex dynamics of this system).

The presentation begins in section 1 with a general description of the problem. Section 2 then provides a formal model of the problem. Section 3 describes the algorithmic strategies that are used, focusing primarily on the use of approximate dynamic programming to solve the problem of optimizing over time. Section 4 describes the results of calibration experiments that show that the model closely matches historical performance, which required using recent

research describing how to make cost-based models match rule-based patterns. Then, section 5 shows that the model can be used to estimate the value of particular types of drivers, which is then used to change the mix of drivers. The value of a particular type of driver, which requires estimating a derivative of the simulation, can only be achieved using the approximate dynamic programming strategies that were used to optimize over time. Section 6 concludes the paper.

1 Problem description and literature review

On the surface, truckload trucking can appear to be a relatively simple operational problem. At any point in time, there will be a set of drivers available to be dispatched, and a set of loads that need to be moved (typically from one city to another). The loads in this industry are typically quite long, generally requiring anywhere from one to four days to complete. As a result, at a point in time we will assign a driver to at most one load. This can easily be modeled as an assignment problem, where the cost of assigning a driver to a load includes both the cost of moving empty to pick up the load, plus the net revenue from moving the load.

In real applications, the problem is much richer. While dispatchers do their best to minimize the empty miles and move the most profitable loads, real decisions have to balance profits now and in the future, as well as accomplish objectives such as getting drivers home in a reasonable amount of time. An important issue in this project was matching historical behavior in terms of the average length of loads handled by different types of drivers. We modeled three "capacity types" (using the terminology of the carrier): teams (two drivers in the same tractor so they could trade off while the other rested), solos (a single driver who had to rest according to a schedule determined by Federal law), and IC's (independent contractors who owned the tractor they drove). Drivers in each of these three fleets had different expectations regarding the length of the loads to which they were assigned. Teams were generally given the longest loads so that their total revenue per week would reasonably compensate two people. Solos exhibited the shortest average length of haul. Getting the model to match historical performance for length of haul for each of the three driver classes required special algorithmic

measures.

The standard approach for modeling such large-scale problems (we worked with over 6,000 drivers) at a high level of detail would be to simply simulate decisions over time. In this setting, this would involve solving a series of network problems to assign drivers to loads at a point in time. While such an approach would handle a high level of detail, the decisions would not be able to reflect the future impact of decisions made now. For example, this logic would not take into account that sending drivers whose home is in Dallas on loads to Chicago is a good way of getting them home. It is also unable to realize that a long (and high revenue) load from Maryland to Idaho is not as good as a shorter load from Maryland to Cleveland (which offers more opportunities for drivers once they unload).

In addition to producing an accurate simulation of the company, we also wanted to produce estimates of the marginal value of different types of drivers distinguished by their home domicile and capacity type. For example, we would like to know the marginal value of adding 10 teams with a home domicile in central Illinois. It is not practical to run a simulation, add 10 drivers of a particular type (there were 300 types) and simulate again. If this were repeated 10 times (to reduce statistical error), we would have to run 3,000 simulations.

There is a fairly extensive literature on models and algorithms for the full-truckload problem, and in particular dynamic versions of this problem. Much of this work has solved sequences of deterministic problems which reflect only what is known at a point in time (for reviews, see Psaraftis (1995), Powell et al. (1995), Gendreau & Potvin (1998) and Larsen et al. (2002)). This work has often focused on the algorithmic challenge of solving problems in real-time (e.g. Gendreau et al. (1999), Taylor et al. (1999)). A number of papers simulated dynamic operations to study questions such as the value of real time information or other dynamic operating policies (Tjokroamidjojo (2001), Regan et al. (1998), Chiu & Mahmassani (2002), Yang et al. (2004)). Ichoua et al. (2006) also proposes a policy for dynamically routing vehicles with the intent of optimizing over time. Their research focuses on myopic policies which adjust behavior now based on probabilistic estimates of future demands. Secomandi (2000) and Secomandi (2001) provides a more formal treatment of policies for solving stochastic vehicle routing problems. This line of research, however, is limited to single-vehicle routing

problems.

The general problem of routing drivers so they return home on time has received very little attention. Caliskan & Hall (2003) propose a deterministic model for routing drivers in trucking, but this model does not capture either the complexity of drivers or the challenge of getting drivers home in the presence of the type of uncertainty that characterizes truckload trucking. There is a rich literature on planning pilot schedules capturing all the attributes of a pilot and a full set of work rules (see Desrosiers et al. (1995) and Desaulniers et al. (1998)). However, these problems are deterministic, and benefit from the highly scheduled nature of airline operations. Also, these problems are much smaller than the problem we address here.

A separate line of research has focused on developing models which produce solutions which optimize over an entire planning horizon. A summary of different modeling and algorithmic strategies for dynamic fleet management problems is given in Powell (1988) and Powell et al. (1995). Early work in this area focused on managing large fleets of relatively similar vehicles, such as would arise in the optimization of empty freight cars for railroads, or aggregate models of fleets for truckload motor carriers. Such problems could be formulated as space-time models (where each node represented a point in space and time) and solved as a network problem if there was a single vehicle type (see, for example, White (1972)) or multicommodity flow problem if there were multiple vehicle types and substitution (Tapiero & Soliman (1972), Crainic et al. (1984)). These models do not allow us to model drivers with any of the richness needed for our project.

The research closest to this project is given in Spivey & Powell (2004), which provides a formal model of the stochastic dynamic driver management problem. We build on this model, but introduce a number of new strategies to overcome challenges that arose when we made the transition from a laboratory experiment to a production application.

2 Problem Formulation

We model the problem using the language of dynamic resource management (see Powell et al. (2001)) where drivers are "resources" and loads are "tasks." The state of a single resource is

defined by an attribute vector, a, composed of multiple attributes, which may be numerical or categorical. For our model, we used:

$$a = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \\ a_9 \\ a_{10} \end{pmatrix} = \begin{pmatrix} \text{Location} \\ \text{Domicile} \\ \text{Capacity type} \\ \text{Scheduled time-at-home} \\ \text{Days away from home} \\ \text{Available time} \\ \text{Geographical constraints} \\ \text{DOT road hours} \\ \text{DOT duty hours} \\ \text{Eight day duty hours} \end{pmatrix}$$

$\mathcal{A}$ = Set of all possible driver attribute vectors a.

A brief discussion of the driver attributes (and the load attributes below) provides an appreciation of some of the complexities in an industrial-strength system. Driver locations were captured at a level that produced 400 locations around the country. Driver domiciles were also captured at a level that divided the country into 100 regions. As discussed earlier, there were three capacity types: team, solo and IC (independent contractor). These three attributes (location, domicile and capacity type) were particularly important and will play a major role throughout the analysis. Field a_4 is the time that we would like to get the driver back home by (e.g. next Saturday), but the cost of not doing this is also influenced by the number of days the driver has been away from home (a_5). Our ability to get drivers home on time was one of the major metrics that we had to calibrate to.

The remaining attributes were needed to produce an accurate simulation. For example, a_6 (available time) captured the fact that a driver might be headed to Chicago (a_1 = Chicago) but would not arrive until 3:17pm tomorrow (all activities were modeled in continuous time). a_7 captured constraints such as the fact that Canadian drivers in the U.S. had to return to Canada, or other drivers had to stay within 500 miles of their home. a_8 and a_9 (DOT road hours and DOT duty hours) captured how many hours a driver had been behind the wheel (road hours) or on-duty (duty hours) on a given day. a_{10} is actually an eight element vector, capturing the number of hours a driver had worked on each of the last eight days.

Similarly, we let b be the vector of attributes of a load, including elements such as origin, destination, appointment time and type, priority, revenue and delivery window. Some windows are tight, but many are fairly loose, providing some flexibility in when a load is served. We let $\mathcal{B}$ be the space of all load types.

We can think of a_t, the attribute vector of a driver at time t, as the state of the driver. We model the state of all the drivers using the resource state vector, which is defined using

$$\begin{aligned} R_{ta} &= \text{The number of resources with attribute vector } a \text{ at time } t. \\ R_t &= \text{The resource state vector at time } t. \\ &= (R_{ta})_{a\in\mathcal{A}}\,. \end{aligned}$$

We then let D_{tb} be the number of loads with attribute b, and let $D_t = (D_{tb})_{b\in\mathcal{B}}$. Our system state vector is then given by

$$S_t = (R_t, D_t).$$

We measure the state S_t just before we make a decision. These *decision epochs* are modeled in discrete time $t = 0, 1, 2, \ldots, T$, but the physical process occurs in continuous time. For example, the available time of a driver, a_6, and the "ready time" (time at which it is available for pickup) of a load, b_6, are both continuous.

There are two types of exogenous information processes: updates to the attributes of a driver, and new customer demands. We let

$$\begin{aligned} \hat{R}_{ta} &= \text{The change in the number of drivers with attribute } a \text{ due to information arriving between time } t-1 \text{ and } t. \\ \hat{D}_{tb} &= \text{The number of new loads that first became known to the system with attribute } b \text{ between time } t-1 \text{ and } t. \end{aligned}$$

For example, $\hat{D}_{tb} = +1$ if we have a new customer order with attribute vector b. If a driver attribute randomly changed from a to a' (arising, for example, from a delay), we would have $\hat{R}_{ta} = -1$ and $\hat{R}_{ta'} = +1$. We let $W_t = (\hat{R}_t, \hat{D}_t)$ be our generic variable for new information. We view information as arriving continuously in time, where the interval between time instant

$t-1$ and t is labeled as time interval t. Thus, W_1 is the information that arrives between now ($t = 0$) and the first decision epoch ($t = 1$).

The major decision classes for this problem include whether a truck is to be used to move a load to a particular destination or whether it needs to move empty to another location in the anticipation of future loads with better rewards. When a decision is applied to a resource, it produces a contribution. A loaded move would generate revenue, whereas an empty move would incur some cost. Decisions are described using

$$\begin{aligned}
d &= \text{An elementary decision,} \\
\mathcal{D}^L &= \text{The set of all decisions to cover a type of load, where an element } d \in \mathcal{D}^L \\
&\quad \text{represents a decision to cover a load of type } b_d \in \mathcal{B}, \\
d^\phi &= \text{The decision to hold a driver,} \\
\mathcal{D} &= \mathcal{D}^L \cup d^\phi, \\
x_{tad} &= \text{The number of times decision } d \text{ is applied to resource with attribute vector } a \\
&\quad \text{at time } t, \\
x_t &= (x_{tad})_{a\in\mathcal{A},d\in\mathcal{D}}.
\end{aligned}$$

The decision variables x_{tad} have to satisfy the following constraints:

$$\sum_{d\in\mathcal{D}} x_{tad} = R_{ta} \quad \forall a \in \mathcal{A}, \tag{1}$$

$$\sum_{a\in\mathcal{A}} x_{tad} \leq D_{tb_d} \quad \forall d \in \mathcal{D}^L, \tag{2}$$

$$x_{tad} \geq 0 \quad a \in \mathcal{A}, d \in \mathcal{D}. \tag{3}$$

Equation (1) captures flow conservation for drivers (we cannot assign more than we have of a particular type), and equation (2) is flow conservation on loads (we cannot assign more drivers to loads of type b_d than there are loads of this type). We let $\mathcal{X}_t$ be the set of all x_t that satisfy equations (1) - (3). The feasible region $\mathcal{X}_t$ depends on S_t. Rather than write $\mathcal{X}(S_t)$, we let the subscript t in $\mathcal{X}_t$ indicate the dependence on the information available at time t. Finally, we assume that decisions are determined by a decision function denoted

$$\begin{aligned}
X^\pi(S_t) &= \text{A function that determines } x_t \in \mathcal{X}_t \text{ given } S_t, \text{ where } \pi \in \Pi, \\
\Pi &= \text{A set of decision functions (or policies).}
\end{aligned}$$

We next need to model the dynamics of the system. Both R_t and D_t evolve over time, but for the moment we focus purely on the evolution of R_t. If we act on a driver with attribute a using decision d, we represent the change in the attribute vector using

$$a' = a^M(a, d).$$

We model the transition function deterministically, which means that a' is the attribute vector that we think results from a decision, but before any new information has arrived. So, if we decide to move a truck from Dallas to Chicago leaving at time 12.2, with an expected travel time of 17.5, then immediately after the assignment, this would be a truck with the attribute that we expect it to be in Chicago at time 29.7 (later information may change this). For algebraic purposes, define

$$\delta_{a'}(a, d) = \begin{cases} 1, & \text{if } a^M(a, d) = a', \\ 0, & \text{otherwise.} \end{cases}$$

We now define the *post-decision* resource vector, which is the resource vector after we make a decision, but before any new information arrives. This can be written

$$R^x_{ta'} = \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} \delta_{a'}(a, d) x_{tad}. \tag{4}$$

Finally, our next pre-decision resource vector would be given by

$$R_{t+1,a} = R^x_{ta} + \hat{R}_{t+1,a}. \tag{5}$$

It is more conventional in stochastic dynamic systems to write the transition from R_t to R_{t+1}. Explicitly capturing the post-decision resource vector provides significant computational advantages, as we illustrate later.

The transition function for the demands is symmetrical. In addition to the state variable D_t, we would define the post-decision demand vector D^x_t, along with an indicator function similar to δ to describe how decisions change the attributes of a load. In the simplest model, a demand is either moved (in which case it leaves the system) or it waits until the next time

period. In our project, it was possible to have a driver move to pick up a load, move the load to an intermediate location and then drop it off so that a different driver can finish the move (this is known as a relay). While such strategies are used for only a small percentage of the total demand, trucking companies will use such a strategy to help get drivers home. A driver may pick up a load that takes the driver too far from his home. Instead, he may move the load part-way so that a different driver can pick up the load and complete the trip.

We define the objective function using

c_{tad} = The contribution generated by applying decision d to resource with attribute vector a at time t.

The contributions were divided between "hard dollar" and "soft dollar" contributions. Hard dollar contributions include the revenue generated from moving a load minus the cost of actually moving the truck (first moving empty to pick up the load, followed by the actual cost of moving the load). The soft dollar costs capture bonuses for getting the driver home, penalties for early or late pickup of a load, and penalties for getting a driver home before or after the time that he was scheduled to be at home.

If we assume that the contributions are linear, the contribution function for period t would be given by

$$C_t(S_t, x_t) = \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} c_{tad} x_{tad}. \tag{6}$$

The optimal policy maximizes the expected sum of contributions, discounted by a factor γ, over all the time periods

$$F_0^*(S_0) = \max_{\pi \in \Pi} \mathbb{E} \left\{ \sum_{t=0}^{T} \gamma^t C\left(S_t, X_t^\pi(S_t)\right) | S_0 \right\}. \tag{7}$$

One policy for solving this problem is the myopic policy, given by

$$X_t^M(S_t) = \arg\max_{x_t \in \mathcal{X}_t} \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} c_{tad} x_{tad}$$

which involves assigning known drivers to known loads at each point in time. This is a straightforward assignment problem, involving the assigning of drivers (that is, attributes a where $R_{ta} > 0$) to loads (attributes b where $D_{tb} > 0$). One of the biggest challenges we faced was the sheer size of this problem, involving over 2,000 available drivers and loads at each time period. Using careful engineering, we limited the number of links per driver (or load) to approximately 10, which still required generating about 20,000 links for each time period (the costing of each link required considerable calculations to enforce driver work rules and to handle the service constraints on each load). Given a solution $x_t = X_t^M(S_t)$, we would then use our transition functions to compute (R_t^x, D_t^x), and then find (R_{t+1}, D_{t+1}) by sampling $\hat{R}_{t+1}(\omega)$ and $\hat{D}_{t+1}(\omega)$.

A central hypothesis of this research is that an algorithm that does a better job of solving equation (7) will do a better job of matching the historical performance of the company. Although we use approximations, our strategy works from a formal statement of the objective function (something that is typically missing from most simulation papers) rather than heuristically policies. As we show, a byproduct of this strategy is that we also obtain estimates of the derivative of $F_0^*(S_0)$ with respect to R_{0a} (for a at some level of aggregation) that would tell us the value of hiring additional drivers in a particular domicile.

In the next section, we describe the strategies we tested for solving (7).

3 Algorithmic strategies

In dynamic programming, instead of solving equation (7) in its entirety, we divide the problem into time stages. At each time period, depending on our current state, we can search over the set of available actions to identify a subset that is optimal. The value associated with each state can be computed using Bellman's optimality equations which are typically written

$$V_t(S_t) = \max_{x_t \in \mathcal{X}_t} \left(C_t(S_t, x_t) + \gamma \sum_{s' \in \mathcal{S}} p(s'|S_t, x_t) V_{t+1}(s') \right) \tag{8}$$

where $p(s'|S_t, x_t)$ is the one-step transition matrix giving the probability that $S_{t+1} = s'$, and $\mathcal{S}$ is the state space. Solving equation (8) encounters three curses of dimensionality: the state

vector S_t (which has dimensionality $|\mathcal{A}| + |\mathcal{B}|$, which can be extremely large), the outcome space (the expectation is over a vector of random variables measuring $|\mathcal{A}| + |\mathcal{B}|$), and the action space (the vector x_t is dimensioned $|\mathcal{A}| \times |\mathcal{D}|$).

Section 3.1 provides a sketch of a basic approximate dynamic programming algorithm for approximating the solution of (7). Section 3.2 describes how we update the value function. Section 3.3 shows how we solve the statistical problem of estimating the value of drivers with hundreds of thousands of attribute vectors. Section 3.4 briefly describes research on stepsizes that was motivated by this project. In section 3.5, we describe how we implemented a backward pass to accelerate the rate of convergence. Finally, 3.6 reports on a series of comparisons of different algorithmic choices we had to make.

3.1 An approximate dynamic programming algorithm

Approximate dynamic programming has been emerging as a powerful technique for solving dynamic programs that would otherwise be computationally intractable. Our approach requires merging math programming with the techniques of machine learning used within approximate dynamic programming. Our algorithmic strategy differs markedly from what is presented in classic texts on approximate dynamic programming, particularly in our use of the post-decision state variable. A comprehensive treatment of our algorithmic strategy is contained in Powell (2007).

We solve (7) by breaking the dynamic programming recursions into two steps:

$$V_{t-1}^x(S_{t-1}^x) = \mathbb{E}\left\{V_t(S_t) \mid S_{t-1}^x\right\}, \tag{9}$$

$$V_t(S_t) = \max_{x_t \in \mathcal{X}_t} \left(C(S_t, x_t) + \gamma V_t^x(S_t^x)\right), \tag{10}$$

where $S_t = S^{M,W}(S_{t-1}^x, W_t)$ and $S_t^x = S^{M,x}(S_t, x_t)$. The basic algorithm strategy works as follows. At iteration n, assume we are following sample path ω^n and that we find ourselves at the post-decision state $S_{t-1}^{x,n}$ after making the decision x_{t-1}^n. Now compute the next pre-decision state S_t^n using

$$S_t^n = S^{M,W}(S_{t-1}^{x,n}, W_t(\omega^n)).$$

From state S_t^n, we compute our feasible region $\mathcal{X}_t^n$ (which depends on information such as $\hat{R}_t^n$ and $\hat{D}_t^n$). Next solve the optimization problem

$$\hat{v}_t^n = \max_{x_t \in \mathcal{X}_t^n} \big(C_t(S_t^n, x_t) + \gamma \bar{V}_t^{n-1}(S^{M,x}(S_t^n, x_t))\big), \tag{11}$$

and assume that x_t^n is the value of x_t that solves (18). We then compute the post-decision state $S_t^{x,n} = S^{M,x}(S_t^n, x_t)$ to continue the process.

We next wish to use the solution of (18) to update our value function approximation. With traditional approximate dynamic programming (Bertsekas & Tsitsiklis (1996), Sutton & Barto (1998)) we would use $\hat{v}^n$ to update a value function approximation around S_t^n. Using the post-decision state variable, we use $\hat{v}_t^n$ update $\bar{V}_{t-1}^{n-1}(S_{t-1}^{x,n})$ around $S_{t-1}^{x,n}$. The updating strategy depends on the specific structure of $\bar{V}_{t-1}^{n-1}(S_{t-1}^x)$.

To design our value function approximation, we took advantage of two properties of our problem. First, most loads are served at a given point in time. If we were to define a post-decision demand vector D_t^x (comparable to the post-decision resource vector R_t^x) which gives the number of loads left over after assignment decisions have been made, we would find that most of the elements of D_t^x were zero. Second, given the complexity of the attribute vector, R_{ta} was typically 0 or 1. For this reason, we used a value function approximation that was linear in R_{ta}, given by

$$\begin{aligned} \bar{V}_t^{n-1}(S_t^x) &= \bar{V}_t^{n-1}(R_t^x) \\ &= \sum_{a' \in \mathcal{A}} \bar{v}_{ta'} R_{ta'}^x. \end{aligned} \tag{12}$$

We have worked extensively with nonlinear (piecewise linear) approximations of the value function to capture nonlinear behavior such as "the fifth truck in a region is not as useful as the first" (see Topaloglu & Powell (2006), for example), but in this project, the focus was less on determining how many drivers to move, and more on what type of driver to use.

It is easy to rewrite (12) using

$$\bar{V}_t(R_t^x) = \sum_{a' \in \mathcal{A}} \bar{v}_{ta'} \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} \delta_{a'}(a, d) x_{tad}, \tag{13}$$

where equation (13) is obtained by using the state transition equation (4). This enables us to write the problem of finding the optimal decision function using

$$\begin{aligned} X_t^\pi(S_t) &= \arg\max_{x_t \in \mathcal{X}_t} \left(\sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} c_{tad} x_{tad} + \gamma \sum_{a' \in \mathcal{A}} \bar{v}_{ta'} \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} \delta_{a'}(a,d) x_{tad} \right) \\ &= \arg\max_{x_t \in \mathcal{X}_t(\omega)} \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} \left(c_{tad} + \gamma \sum_{a' \in \mathcal{A}} \bar{v}_{ta'} \delta_{a'}(a,d) \right) x_{tad}. \end{aligned} \tag{14}$$

Recognizing that $\sum_{a' \in \mathcal{A}} \delta_{a'}(a,d) = \delta_{a^M(a_t,d_t)}(a,d) = 1$, we can write (14) as

$$X_t^\pi(S_t) = \arg\max_{x_t \in \mathcal{X}_t(\omega)} \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} \left(c_{tad} + \gamma \bar{v}^{n-1}_{t,a^M(a,d)} \right) x_{tad}. \tag{15}$$

Clearly, (15) is no more difficult than solving the original myopic problem, with the only difference being that we have to solve it iteratively in order to estimate the value function approximation. Fortunately, it is neither necessary nor desirable to re-estimate the value functions each time we undertake a policy study.

We face two challenges at this stage. First, we have to find a way to update the values of $\bar{v}^{n-1}_{t-1,a}$ using information derived from solving the decision problems. Section 3.2 describes a Monte-Carlo based approach, but this introduces a statistical problem. As illustrated in figure 1, to decide which load a driver should move, we have to know the value of the driver at the end of each load. This means it is not enough to know the value of drivers with attributes that actually occur (that is, $R_{ta} > 0$), but we also have to know the value of attributes that we *might* visit.

3.2 Value function updates

Once we have settled on a value function approximation, we face the challenge of estimating it. The general idea in approximate dynamic programming is that we iteratively simulate the system forward in time. At iteration n, we follow a sample path ω^n which determines $\hat{R}^n_t = \hat{R}_t(\omega^n)$ and $\hat{D}^n_t = \hat{D}_t(\omega^n)$. The decision function in equation (15) is computed using value functions $\bar{v}^{n-1}_{ta'}$ computed using information from iteration $n-1$. We then use information

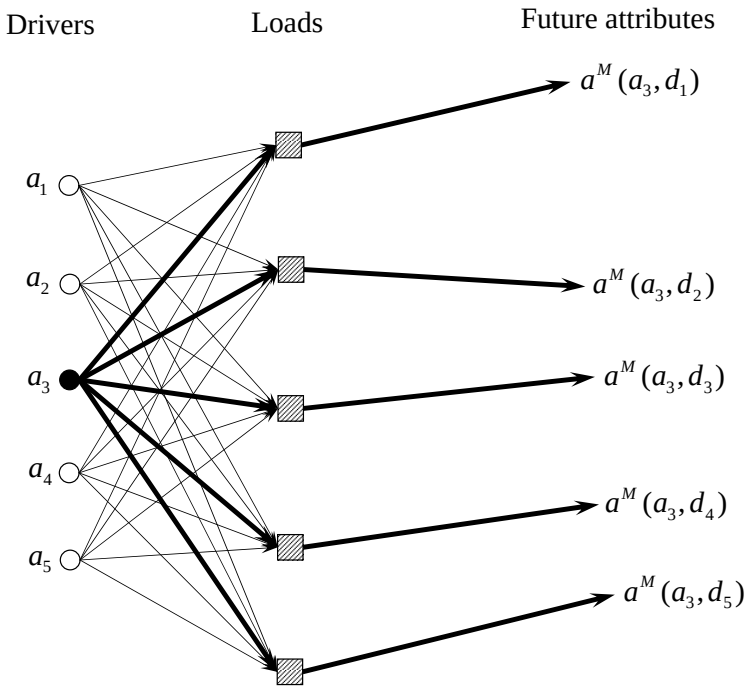


Figure 1: Driver assignment problem, illustrating the different future driver attributes that have to be evaluated.

from iteration n to update $\bar{v}_{t-1,a}^{n-1}$ giving us $\bar{v}_{t-1,a}^{n}$. This section describes how this updating is accomplished.

Assume that our previous decision (at time $t-1$) left us in the post-decision state $S_{t-1}^{x,n}$. Following the sample path ω^n then puts us in state $S_t^n = S^{M,W}(S_{t-1}^{x,n}, W_t(\omega^n))$ which determines our feasible region $\mathcal{X}_t^n$. We then make decisions at time t by solving

$$F_t(S_t^n) \;=\; \max_{x_t \in \mathcal{X}_t^n} \big(C_t(S_t^n, x_t) + \gamma \bar{V}_t^{n-1}(S_t^x)\big), \tag{16}$$

where $S_t^x = S^{M,x}(S_t^n, x_t)$. We let x_t^n be the value of x_t that solves (16). Note that $R_{t-1}^{x,n}$ affects (16) through the flow conservation constraint (1). Keep in mind that $R_{ta}^n = R_{t-1,a}^{x,n} + \hat{R}_{ta}(\omega^n)$. $\hat{R}_{ta}$ may be the random arrival of a new driver, but for our work, it primarily captures random changes in the status of a driver (e.g. travel delays or equipment failures). If these transitions change the status of a driver from a to a', then we would have $\hat{R}_{ta} = -1$ and $\hat{R}_{ta'} = +1$. If there are no random changes of this sort (which means that $\hat{R}_{ta} = 0$), then it is easy to see

that

$$\hat{v}^n_{t-1,a} = \frac{\partial F(S_t)}{\partial R^x_{t-1,a}} = \frac{\partial F(S_t)}{\partial R_{t,a}} = \hat{\nu}^n_{ta}, \tag{17}$$

where $\hat{\nu}^n_{ta}$ is the dual variable for the flow conservation constraint (1). If we do allow random changes (say, from a to a'), we would use $\hat{\nu}^n_{ta'}$ to update $\hat{v}^n_{t-1,a}$.

We want to use information from (16) to update the value functions used at time $t-1$, given by $\bar{v}^{n-1}_{t-1,a}$. Keeping in mind that these are estimates of slopes, what we need is the derivative of $F_t(S^n_t)$ with respect to each $R^{x,n}_{t-1,a}$ where $a = a^x_{t-1}$, which we compute using

$$\begin{aligned}\hat{v}^n_{t-1,a} &= \frac{\partial F(S_t)}{\partial R^x_{t-1,a}} \\ &= \sum_{a' \in \mathcal{A}} \frac{\partial F(S_t)}{\partial R_{ta'}} \frac{\partial R_{ta'}}{\partial R^x_{t-1,a}}\bigg|_{\omega=\omega^n}.\end{aligned} \tag{18}$$

$\frac{\partial F(S_t)}{\partial R_{ta'}}$ is just the dual of the optimization problem (15) associated with the flow conservation constraint (1), which we denote by $\nu^n_{ta'}$. For the second part of the derivative, we have

$$\frac{\partial R_{ta'}}{\partial R^x_{t-1,a}} = \begin{cases} 1 & \text{if } a' = a^{M,W}(a^x_{t-1}, W_t(\omega^n)) \\ 0 & \text{otherwise.} \end{cases}$$

This simply means that if we had a truck with attribute a^x_{t-1}, which then evolves (due to exogenous information) into a truck with attribute $a' = a_t = a^{M,W}(a^x_{t-1}, W_t(\omega^n))$, then

$$\hat{v}^n_{t-1,a} = \nu^n_{ta'}.$$

We do not have to execute the summation in equation (18). We just need to keep track of the transition from a^x_{t-1} to a_t. We note, however, that we are unable to compute $\nu^n_{ta'}$ for each attribute $a' \in \mathcal{A}$ (the attribute space is too large). Instead, for each a_{t-1} where $R^{x,n}_{t-1,a_{t-1}} > 0$, we found $a' = a_t = a^{M,W}(a_{t-1}, W_t(\omega^n))$ and computed $\nu_{ta'}$. We then found $\hat{v}^n_{t-1,a_{t-1}}$ from (18).

Once we have computed $\hat{v}^n_{t-1,a_{t-1}}$, we update the value function approximation using

$$\bar{v}^n_{t-1,a} = (1-\alpha_{n-1})\bar{v}^{n-1}_{t-1,a} + \alpha_{n-1}\hat{v}^n_{t-1,a} \tag{19}$$

Step 0: Initialization:

Step 0a. Initialize $\bar{V}_t^0, \quad t \in \mathcal{T}$.

Step 0b. Initialize the state S_0^1.

Step 0c. Set $n = 1$.

Step 1. Choose a sample path ω^n.

Step 2: Do for $t = 0, 1, \ldots, T$:

Step 2a: Solve the optimization problem:

$$\max_{x_t \in \mathcal{X}_t^n} \left(C_t(S_t^n, x_t) + \gamma \bar{V}_t^{n-1}(S^{M,x}(S_t^n, x_t))\right) \tag{20}$$

Let x_t^n be the value of x_t that solves (20), and let ν_{ta_t} be the dual corresponding to the resource conservation constraint for each R_{ta_t} where $R_{ta_t} > 0$.

Step 2b: Update the value function using

$$\bar{v}_{t-1,a}^n = (1 - \alpha_{n-1})\bar{v}_{t-1,a}^{n-1} + \alpha_{n-1}\hat{\nu}_{ta}^n$$

Do this for each attribute a for which we have computed $\hat{\nu}_{ta}^n$.

Step 2c: Update the state:

$$\begin{aligned} S_t^{x,n} &= S^{M,x}(S_t^n, x_t^n) \\ S_t^n &= S^{M,W}(S_{t-1}^{x,n}, W_t(\omega^n)) \end{aligned}$$

Step 3. Increment n. If $n \leq N$ then set $S_0^{x,n} = S_T^{x,n-1}$ and go to step 1.

Step 4: Return the value functions, $\{\bar{v}_{ta}^n, t = 1, \ldots, T, a \in \mathcal{A}\}$.

Figure 2: An approximate dynamic programming algorithm to solve the driver assignment problem.

where α_{n-1} is a stepsize between 0 and 1 (discussed in greater detail in section 3.4).

We outline the steps of a typical approximate dynamic programming algorithm for solving the fleet management problem in figure 2. This algorithm employs a single pass to simulate a sample trajectory using the current estimates of the value functions. We start from an initial state $S_0^1 = (R_0, D_0)$ of drivers and loads with a value function approximation $\bar{V}_t^0(S_t^x)$. From this we determine an assignment of drivers to loads, x_0^1. We then find the post-decision state $S_0^{x,1}$ and then simulate our way to the next state $S_1 = S^{M,W}(S_0^{x,1}, W_1(\omega^1))$. This simulation includes both new customer orders, as well as random changes to the status of the drivers. All of the complexity of the physics of the problem is captured in the transition functions, which impose virtually no limits on our ability to handle the realism of the problem.

At an early stage of the project, the company expressed concern that the results might be overfitted to a particular set of drivers (input to the model as R_0^x) and loads. We took two

steps in response to this concern. First, we randomized the loads, choosing a subset from a larger set of loads at each iteration. Second, we took the final resource state vector (R_T^x), and used this as the new initial resource state vector (see step 3).

A major technical challenge in the algorithm is computing the value function approximation $\bar{V}_t = (\bar{v}_{ta})_{a \in \mathcal{A}}$. Even if the attribute vector a has only a few dimensions, the attribute space is too large to update using (19). Furthermore, we only obtain updates $\hat{v}_{ta}^n$ for a subset of attributes a at each iteration. In principle, we could have solved our decision problem for a resource vector R_{ta} using all the attributes in $\mathcal{A}$. This is completely impractical. For our simulation, we only generated nodes for attributes a where $R_{ta}^n > 0$ (as a rule, we generated a unique node for each driver), which means we obtain $\hat{v}_{ta}^n$ only for a subset of attributes, but we need an estimate $\bar{v}_{ta}$ not just for where we have drivers (that is, $R_{ta} > 0$) but where we *might* want to send drivers. We address this problem in the next section.

3.3 Approximating the value function

The full attribute vector a needed to completely capture the important characteristics of the driver produces an attribute space that is far too large to enumerate. Fortunately, it is not necessary to use all these attributes for the purpose of approximating the value function. In addition to time (we have a finite horizon model, so all value functions are indexed by time), three attributes were considered essential: the location of the driver, the home domicile of the driver, and his capacity type (team, solo and independent contractor). The company divided the country into 100 regions for the purpose of representing location and domicile (this is only for the value function). Combined with three capacity types and 20 time periods, this produced a total of 600,000 attributes for which we would need an estimated value. While dramatically smaller than the original attribute space, this is still extremely large. Most of these attributes will never be visited, and many will be visited only a few times. As a result, we have serious statistical issues in our ability to estimate $\bar{v}_{ta}$.

The standard approach to overcoming large state spaces is to use aggregation. We can use aggregation to create a hierarchy of state spaces, $\left\{\mathcal{A}^{(g)}, g = 0, 1, 2, \ldots, |\mathcal{G}|\right\}$, with successively fewer elements. We illustrate four levels of aggregations in table 1. At level 0, we have 20 time

g	Time	Location	Domicile	Capacity type	$\lvert\mathcal{A}\rvert$
0	*	Region	Region	*	600,000
1	*	Region	-	*	6,000
2	*	Region	-	-	2,000
3	*	Area	-	-	200

Table 1: Levels of aggregation used to approximate value functions. A '$*$' corresponding to a particular attribute indicates that the attribute is included in the attribute vector, and a '$-$' indicates that it is aggregated out.

periods, 100 regions for location and domicile, and three capacity types, producing 600,000 attributes. At aggregation level 1, we ignored the driver domicile; at aggregation level 2, we ignored the capacity type; at aggregation level 3, we represented location as one of 10 areas, which had the effect of insuring that we always had some type of estimate for any attribute.

Choosing the right level of aggregation to approximate the value functions involves a tradeoff between statistical and structural errors. If $\left\{\bar{v}_{ta}^{(g)}, g \in \mathcal{G}\right\}$ denotes estimates of a value v_{ta} at different levels of aggregation, we can compute an improved estimate as a weighted combination of estimates of the values at different levels of aggregation using

$$\bar{v}_{ta} = \sum_{g \in \mathcal{G}} w_{ta}^{(g)} \cdot \bar{v}_{ta}^{(g)}, \tag{21}$$

where $\{w_{ta}^{(g)}\}_{g \in \mathcal{G}}$ is a set of appropriately chosen weights. George et al. (2005) shows that good results can be achieved using a simple formula, called WIMSE, that weights the estimates at different levels of aggregation by the inverse of the estimates of their mean squared deviations (obtained as the sum of the variances and the biases) from the true value. These weights are easily computed from a series of simple calculations. We briefly summarize the equations

without derivation. We first compute

$$\begin{aligned}
\bar{\beta}_{ta}^{(g,n)} &= \text{Estimate of bias due to smoothing a transient data series,} \\
&= (1-\eta_{n-1})\bar{\beta}_{ta}^{(g,n-1)} + \eta_{n-1}(\hat{v}^n - \bar{v}_{ta}^{(g,n-1)}) \qquad (22) \\
\bar{\mu}_{ta}^{(g,n)} &= \text{Estimate of bias due to aggregation error,} \\
&= \bar{v}_{ta}^{(g,n)} - \bar{v}_{ta}^{(0,n)}. \\
\bar{\bar{\beta}}_{ta}^{(g,n)} &= \text{Estimate of total squared variation,} \\
&= (1-\eta_{n-1})\bar{\bar{\beta}}_{ta}^{(g,n-1)} + \eta_{n-1}(\hat{v}_{ta}^n - \bar{v}_{ta}^{(g,n-1)})^2.
\end{aligned}$$

We are using two stepsize formulas here. $\alpha_{ta}^{(g,n-1)}$ is the stepsize used in equation (19) to update $\bar{v}_{ta}^{n-1}$. This is discussed more carefully in section 3.4. η_n is typically a deterministic stepsize which might be a constant such as 0.1, although we used McClain's stepsize rule

$$\eta_n = \frac{\eta_{n-1}}{1+\eta_{n-1}-\bar{\eta}} \qquad (23)$$

where $\bar{\eta} = 0.10$ has been found to be very robust (George et al. (2005)).

We estimate the variance of the observations at a particular level of aggregation using

$$(s_{ta}^2)^{(g,n)} = \frac{\bar{\bar{\beta}}_{ta}^{(g,n)} - (\bar{\beta}_{ta}^{(g,n)})^2}{1+\lambda_{ta}^{(g,n)}} \qquad (24)$$

where $\lambda_{ta}^{(g,n)}$ is computed using

$$\lambda_{ta}^{(g,n)} = \begin{cases} (\alpha_{ta}^{(g,n-1)})^2 & n = 1 \\ (1-\alpha_{ta}^{(g,n-1)})^2\lambda_{ta}^{(g,n-1)} + (\alpha_{ta}^{(g,n-1)})^2 & n > 1. \end{cases}$$

This allows us to compute an estimate of the variance of $\bar{v}_{ta}^{(g,n)}$ using

$$\begin{aligned}
(\bar{\sigma}_{ta}^2)^{(g,n)} &= Var[\bar{v}_{ta}^{(g,n)}] \\
&= \lambda_{ta}^{(g,n)}(s_a^2)^{(g,n)}. \qquad (25)
\end{aligned}$$

The weight to be used at each level of aggregation is given by

$$w_{ta}^{(g,n)} \propto \left((\bar{\sigma}_{ta}^2)^{(g,n)} + \left(\bar{\mu}_{ta}^{(g,n)}\right)^2\right)^{-1} \qquad (26)$$

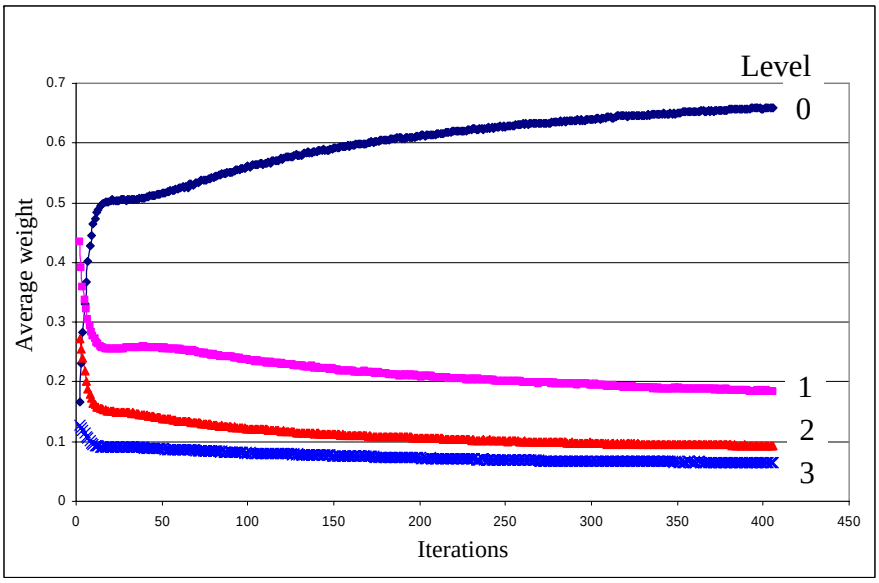


Figure 3: Average weight put on each level of aggregation by iteration.

where the weights are normalized so they sum to one. This formula is easy to compute even for very large scale applications such as this. All the statistics have to be computed for each attribute a, for all levels of aggregation, that is actually visited. From this, we can compute an estimate of the value of any attribute regardless of whether we visited it or not. Figure 3 shows the average weight put on each level of aggregation from one run of the model. As is apparent from the figure, higher weights are put on the more aggregate estimates, with the weight shifting to the more disaggregate estimates as the algorithm progresses. It is very important that the weights be adjusted as the algorithm progresses; using the final set of weights at the beginning produces very poor results.

3.4 Stepsizes

Stepsizes are often treated as the soft-science of approximate dynamic programming, with people using simple formulas such as a constant (.1 or .05 is typical) or a declining stepsize rule such as $a/(a+n)$ for some a. A popular rule is McClain's formula, given by (23), which provides $1/n$ behavior initially, quickly converging to the constant $\bar{\alpha}$ (we used 0.10, which is typical).

We genuinely struggled with stepsizes for this problem. If the stepsize was too small, the rate of convergence was much too slow. If the stepsize was too large, the performance was unstable, and the variance of the estimates $\bar{v}_{ta}$ were too large (later, we show that we use $\bar{v}_{ta}$ in our policy studies).

As a byproduct of this research, we developed a new stepsize formula that significantly improved the performance of the algorithm (faster initial convergence, with better stability in the limit). The stepsize rule is developed in George & Powell (2006) where it was named the optimal stepsize algorithm (OSA), and is given by

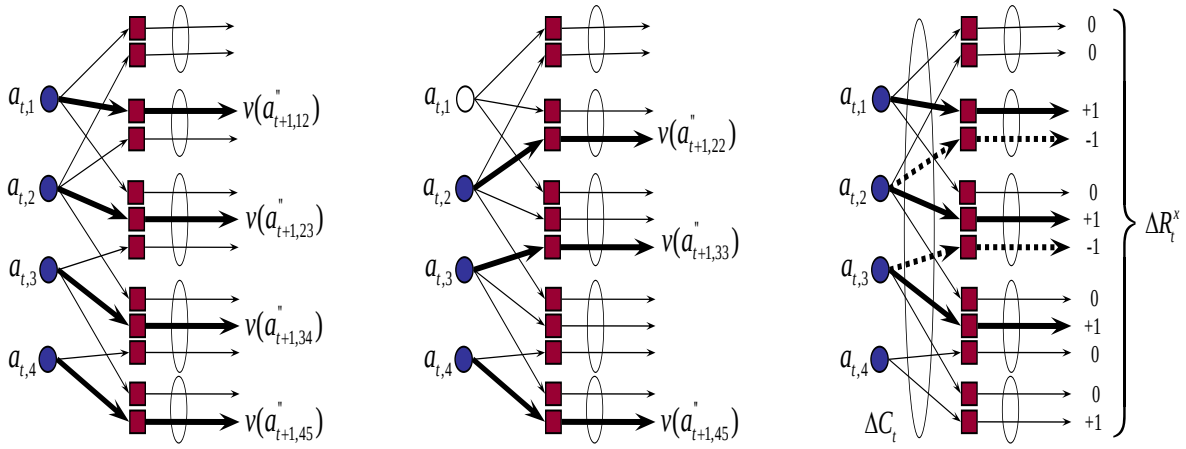
$$\alpha_n = 1 - \frac{(\bar{\sigma}^2)^n}{(1+\bar{\lambda}^{n-1})(\bar{\sigma}^2)^n + \left(\bar{\beta}^n\right)^2} \tag{27}$$

where $(\bar{\sigma}^2)^n$ is computed using equation (25), and $\bar{\beta}^n$ is given by equation (22) (we have dropped the indexing by aggregation level g and attribute a for simplicity). The stepsize rule balances the estimate of the noise $(\bar{\sigma}^2)^n$ and the estimate of the bias $\bar{\beta}^n$ which is attributable to the transient nature of the data. If the data is found to be relatively stationary (low bias) then we want a smaller stepsize; as the estimate of the noise variance decreases, we want a larger stepsize.

3.5 ADP using a double-pass algorithm

The steps in figure 2 describe the simplest implementation of an approximate dynamic programming algorithm which steps forward in time, updating value functions as we proceed. This is also known as a TD(0) algorithm (Bertsekas & Tsitsiklis (1996)). While easy to implement, this algorithm can suffer from slow convergence since $\hat{v}_{ta}^n$ depends on $\bar{v}_{ta}^{n-1}$ which is typically initialized to zero and slowly rises, producing a downward bias in all the value function estimates. This does not necessarily produce poor decisions, but it does mean that $\bar{v}_{ta}^n$ underestimates the value of a driver with attribute a at time t.

A strategy for overcoming this slow convergence, which proved to be particularly valuable for this project, involves using a two-pass procedure (also known as TD(1)). In this procedure, we simulate decisions forward in time without updating the value functions. The derivative



4a: Initial solution. 4b: Without driver a_1. 4c: Difference.

Figure 4: Illustration of numerical derivative. (a) is the base solution with four drivers, (b) is the solution with driver a_1 dropped out, and (c) is the difference in assignment costs and post-decision resource vector.

$\hat{v}^n_{ta}$ is then computed in a backward pass. In the forward pass implementation, $\hat{v}^n_{ta}$ depends on $\bar{v}^{n-1}_{ta}$. With the backward pass, $\hat{v}^n_{ta}$ depends on $\hat{v}^n_{t+1,a}$.

In classical discrete dynamic programs, implementing a backward pass (or backward traversal, as it is often referred to), is fairly straightforward (see Bertsekas & Tsitsiklis (1996), Sutton & Barto (1998)). If we are in state S^n_t, we choose an action x^n_t according to some policy, compute a contribution $C(S^n_t, x^n_t)$, then observe information $W_{t+1}(\omega^n)$, which leads us to state S^n_{t+1}. After following a path $S^n_t, x^n_t, S^n_{t+1}, x^n_{t+1}, \ldots$, we can compute $\hat{v}^n_t = C(S^n_t, x^n_t) + \hat{v}^n_{t+1}$ recursively by stepping backward through time.

This logic is completely intractable for the problem class that we are considering. Instead, we perform a numerical derivative for each driver, which means that after solving the original assignment problem (at time t), we loop over all the drivers (that is, all the attributes a where $R_{ta} > 0$), set $R_{ta} = 0$ and reoptimize. The process is illustrated in figure 4, where 4a shows the initial assignment of four drivers. Since the downstream value from assigning a driver to a load in the future depends on the driver-load combination, we have duplicated each load for each driver, using an ellipse to indicate which driver-load combinations represent the same load. If the driver with attribute a_{t1} is assigned to the second load, then this creates a driver in the future with attribute $a''_{t+1,12}$ with value $v(a''_{t+1,12})$.

In figure 4b, we show the solution without driver a_1. Since driver a_2 shifts up to cover load 2, we no longer have a driver in the future with attribute $a''_{t+1,12}$, but instead we have a driver with attribute $a''_{t+1,22}$. Figure 4c shows the difference, where we are interested in the change in the immediate contribution, and the change in the future availability of drivers. To represent these quantities, let $X_t(R_t)$ be the initial driver-load assignments, and let $X_t(R_t - e_a)$ be the perturbed solution, where e_a is a vector of 0's with a 1 in the element corresponding to R_{ta}. Now let

$$\Delta C_t(a) = C(S_t, X_t(R_t)) - C(S_t, X_t(R_t - e_a))$$

be the change in costs due to changes in flows over the driver to load assignment arcs as a result of the perturbation. Next let $R^x_t(R_t)$ be the post-decision resource vector given R_t, and let

$$\Delta R^x_t(a) = R^x_t(R_t) - R^x_t(R_t - e_a)$$

be the change in the post-decision state vector due to the perturbation. Figure 4c indicates the change in flows that drive $\Delta C_t(a)$, and the vector $\Delta R^x_t(a)$, where $\Delta R^x_{ta'}(a) = 1$ if including a driver with attribute a produces an additional driver of type a', or $\Delta R^x_{ta'}(a) = -1$ if the change takes away a driver of type a'.

In the double-pass algorithm, we compute $\Delta C_t(a)$ (which is a scalar) and $\Delta R^x_t(a)$ (which is a vector of +1's and -1's) for each attribute a (which we have chosen to represent). After we have completed the forward pass, we obtain $\hat{v}^n_{ta}$ in a backward pass using

$$\hat{v}^n_{ta} = \Delta C_t(a) + \sum_{a' \in \mathcal{A}} \Delta R^x_{ta'}(a) \hat{v}^n_{t+1,a'}.$$

where we have made a slight notational simplification by assuming that $a^x_t = a_{t+1}$ (that is, there is no noise in the attribute transition function), which means that $R^x_t = R_{t+1}$.

3.6 Comparisons

This section has produced a number of algorithmic choices: Should we use the forward pass (figure 2) or backward pass (section 3.5)? Should we compute $\hat{v}^n$ using numerical derivatives or dual variables? And should we perform smoothing using the OSA stepsize (equation (27)) or a deterministic formula such as McClain (equation (23)).

Figure 5 compares all of these strategies. We can draw several conclusions from this figure. First, it is apparent that the value functions computed from the backward pass show much faster convergence in the early iterations than those computed from using a forward pass. This is a well-known property of dynamic programming when we start with initial value function approximations equal to zero. However, the difference between these approaches disappears after 50 iterations. We also have to consider that the backward pass is much harder to implement. The real value of the backward pass is that we appear to be obtaining good value functions after as few as 25 iterations (a result supported by other experiments reported below). For very large scale applications such as this (where each iteration requires almost 10 minutes of CPU time on a 3Ghz Pentium processor), reducing the number of iterations needed from 50 to 25 is a significant benefit.

The figure also compares value functions computed using the OSA stepsize versus the McClain stepsize. The OSA stepsize produces faster convergence (this is particularly noticeable when using a forward pass), as well as more stable estimates (this is primarily apparent when using gradients computed using a backward pass).

Finally, we also see that there is a significant difference between value functions computed using dual variables versus numerical derivatives. It is easy to verify that the numerical derivative is greater than or equal to the dual variable, but it is not at all obvious that the difference would be as large as that shown in the figure. Of course, this comes at a significant price computationally. Run times using numerical derivatives are 30 to 40 percent greater than if we used dual variables. We have found, however, that while numerical derivatives produce much more accurate value functions (important in our study), they do not produce better dispatching decisions. If the interest is in a realistic simulation of the fleet (and not the

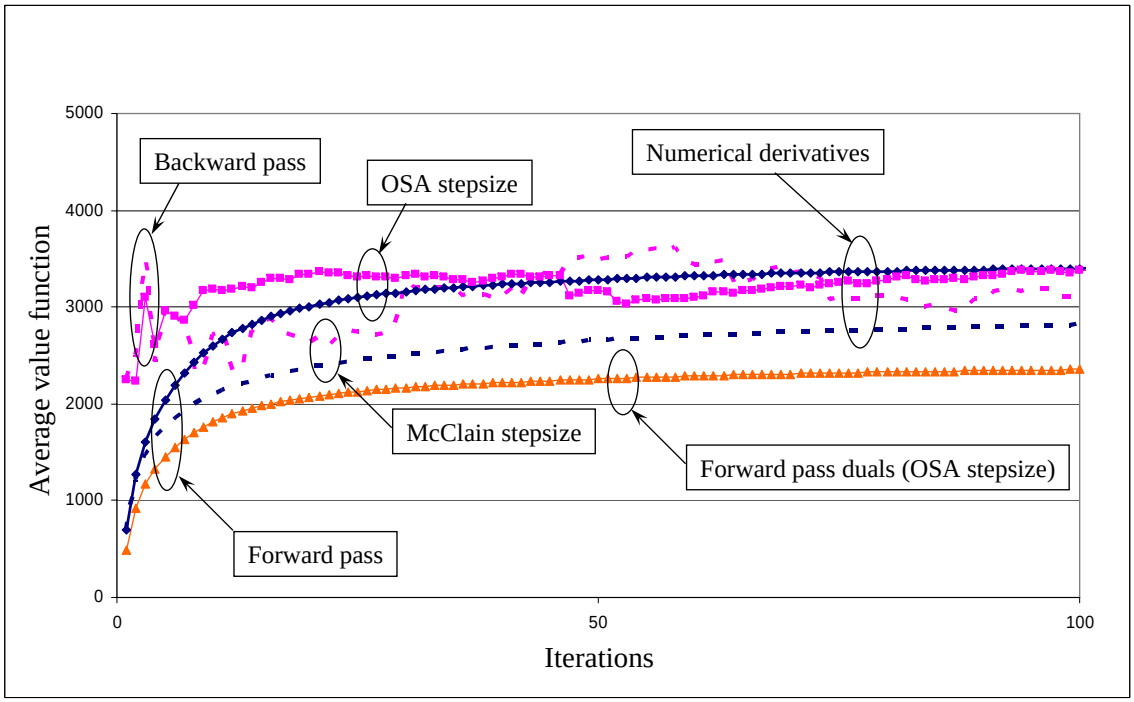


Figure 5: Average value function when we use forward and backward passes, numerical derivatives and dual variables, and the OSA stepsize or the McClain stepsize.

value functions themselves), then we have found that dual variables work fine. In this paper, we wish to use the value functions to estimate the value of different types of drivers,

4 Model Calibration

Before the model could be used for policy analyses, the company insisted that it closely replicate a number of operating statistics including the average length of haul (the length of a load to which a driver is assigned), the average revenue per truck per day, equipment utilization (miles per day), the percent of drivers that were sent home on a weekend. These statistics had to fall between historical minimums and maximums for each of the three capacity types. Model calibration means matching the performance of the collective decisions made by their dispatchers (see figure 6). Perhaps one of the surprising (and significant) outcomes of the research is that a properly calibrated optimization model was required to closely match the performance of an experienced group of dispatchers.

Figure 6: The Schneider dispatch center in Green Bay, WI.

Average length of haul is particularly important since drivers are only paid while they are driving, and longer loads mean less idle time. For this application, it was important to match the average length of haul for each of the three types of drivers (known as "capacity types"). Of the three capacity types, teams (drivers that work in pairs) prefer the longest loads because they pay the most. The company was not willing to consider the results of a simulation that produced an average length of haul that was significantly different (for each capacity type) from their historical performance. This could have an impact on driver turnover, which was not captured in the objective function.

When we look at the historical pattern of loads for a particular driver class, we obtain a distribution such as that shown in table 2. Thus, while this driver class may have an 800 mile average length of haul, this average will include a number of loads that are significantly longer or shorter. Using penalties to discourage assignments to loads that differ from the average would seriously distort the model.

Section 4.1 describes an algorithmic strategy to produce assignments that match historical patterns of behavior. Section 4.2 then describes how well the model matched the historical metrics, where we depend on both the contribution of the value functions as well as the

LOH (miles)			Relative frequency (%)
0	-	390	8.3
390	-	689	33.9
690	-	1089	36.6
1090	-	1589	15.6
1590	-		5.4

Table 2: Illustrative length-of-haul distribution for a single driver type.

pattern-matching logic described in the next section. Section 4.3 compares the contribution of value functions against the pattern-matching logic.

4.1 Pattern-matching

The problem of matching the historical averages for length-of-haul (LOH) by capacity type can be viewed as an example where the results of a model need to match exogenous patterns of behavior. Our presentation follows the work in Marar et al. (2006) and Marar & Powell (2004). In this work, we assume that we are given a *pattern vector* ρ where

$$\begin{aligned} \rho^e &= (\rho^e_{\bar{a}\bar{d}})_{\bar{a}\in\mathcal{A},\bar{d}\in\mathcal{D}}, \\ \rho^e_{\bar{a}\bar{d}} &= \text{The exogenous pattern, representing the percent of time that resources with attribute } \bar{a} \text{ are acted on by decisions of type } \bar{d} \text{ based on historical data.} \end{aligned}$$

We refer to ρ^e as an exogenous pattern since it describes desired behaviors, rather than a specific cost for a decision. In most applications, the indices $\bar{a}$ and $\bar{d}$ for $\rho^e_{\bar{a}\bar{d}}$ are aggregations of the original attribute a and decision d. For the purpose of matching the length of haul, $\bar{a}$ consists only of the capacity type, and $\bar{d}$ represents a decision to assign a driver of type $\bar{a}$ to a load whose length is within some range.

We next have to determine the degree to which the model is matching the exogenous pattern. Let $\rho_{\bar{a}\bar{d}}(x)$ be the average pattern flow from a solution $X(R)$ of the model corresponding to the attribute-decision pair $(\bar{a},\bar{d})$. Also let $R_{\bar{a}}$ be the total number of resources with attribute $\bar{a}$ over the entire horizon. The goal is for the model pattern flow $\rho_{\bar{a}\bar{d}}(x)$ to match closely the desired exogenous pattern flows, $\rho^e_{\bar{a}\bar{d}}$.

The deviation from the desired frequency is captured in the objective function using a penalty function. The actual term included in the objective function for each pattern is denoted as $H(\rho(x), \rho^e)$ where we used the square of the difference of the two, given by

$$H(\rho(x), \rho^e) = \sum_{\bar{a}} \sum_{\bar{d}} R_{\bar{a}} (\rho_{\bar{a}\bar{d}}(x) - \rho^e_{\bar{a}\bar{d}})^2. \tag{28}$$

The aim is to penalize the square of the deviations of the observed frequencies from the desired frequencies. In practice, a quadratic approximation of H is used. The pattern matching term H is multiplied by a weighting parameter θ and subtracted from the standard net revenue function. The objective function that incorporates the patterns is written as follows

$$x_t(\theta) = \arg\max_{x_t \in \mathcal{X}_t} \left[\sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} c_{tad} x_{tad} - \theta H(\rho(x), \rho^e) \right]. \tag{29}$$

θ permits control over how much emphasis is put on the patterns relative to the remainder of the objective function. Setting θ to zero turns the patterns off.

We use an algorithm proposed by Marar & Powell (2004) (and modified by Powell et al. (2004)) that incorporates this feature.

4.2 Comparison to history

We are finally ready to compare the model to historical measures. We have four types of statistics, which are measured for each of the three capacity types, giving us 12 statistics altogether. The company derived what it considered to be acceptable ranges for each statistic. Figures 7a-d gives the length-of-haul, revenue per driver, utilization (miles per driver per day) and the percent of drivers that are sent home on a weekend. The last statistic reflected the ability of the model to get drivers home on weekends, which was viewed as being important to the drivers.

All the results of the model closely matched historical averages. The units of the vertical axis have been eliminated due to the confidentiality of the data, but the graphs accurately show the relative error (the bottom of the vertical axis is zero in all the plots). The bands

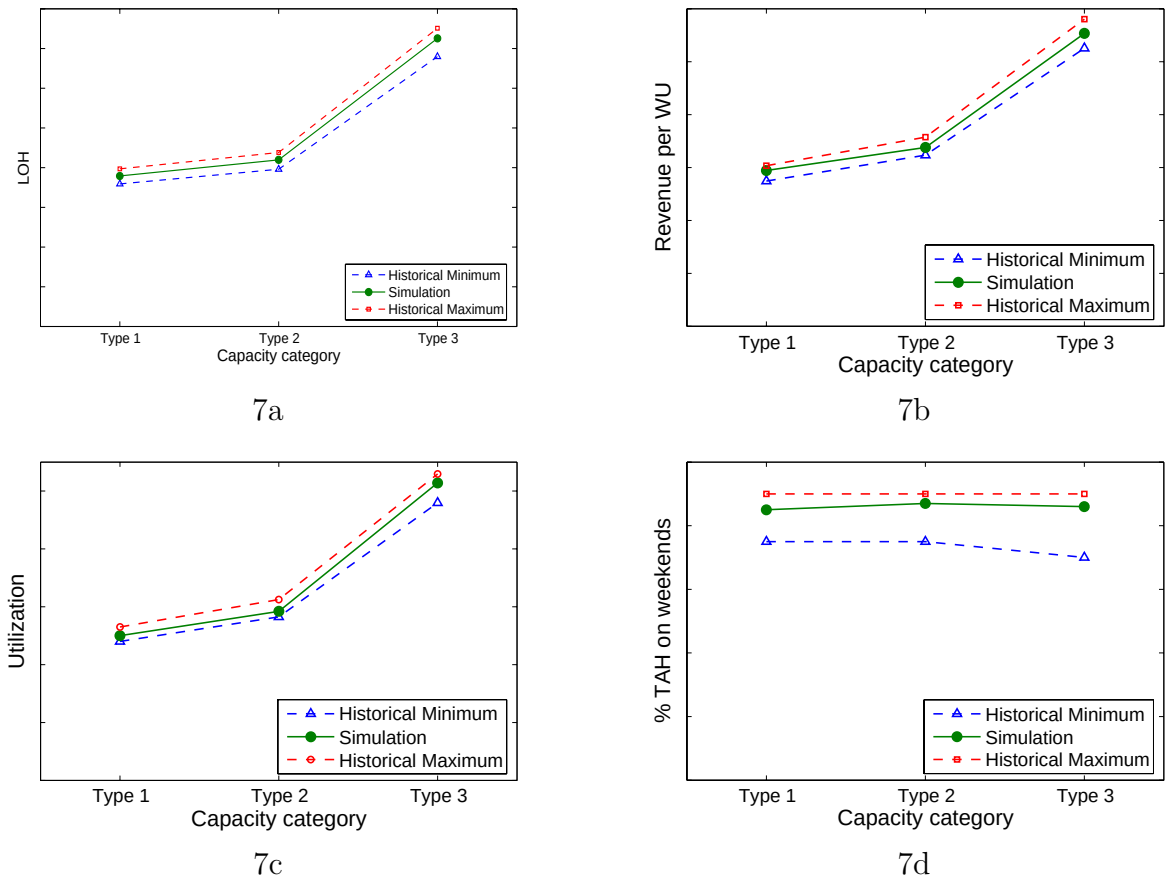


Figure 7: Simulation results compared against historical performance for (a) length of haul, (b) revenue per working unit, (c) utilization and (d) percent time at home on weekends.

were developed by company management before the model was run. It is easy to see that three of the four sets of max/min bands are quite tight. We also note that while we used specific pattern logic to match the length of haul statistics, the other statistics were a natural output of the model, calibrated through the use of cost-based rules.

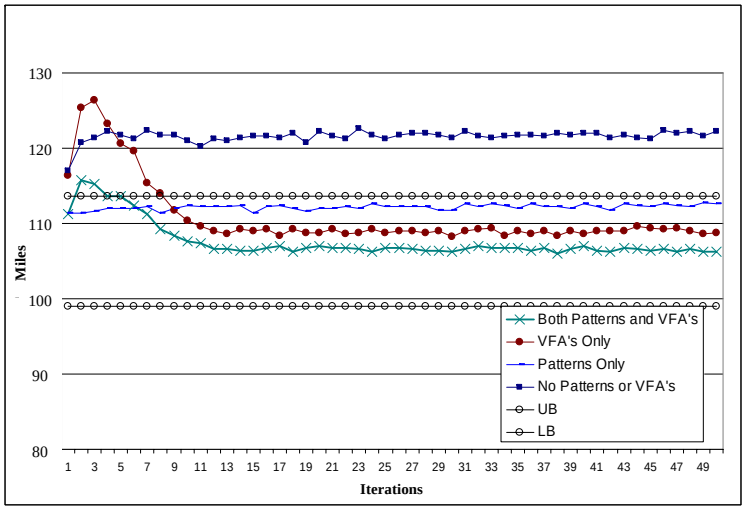
At this point, company management felt comfortable concluding that the model was well-calibrated and could be used for policy studies. Although the model has many applications, in section 5 we focus specifically on the ability of the model to evaluate the value of drivers by capacity type and domicile. This study requires that the value functions do more than simply produce good driver assignment decisions; the value functions themselves had to accurately estimate the value of each driver type.

4.3 VFA's versus patterns

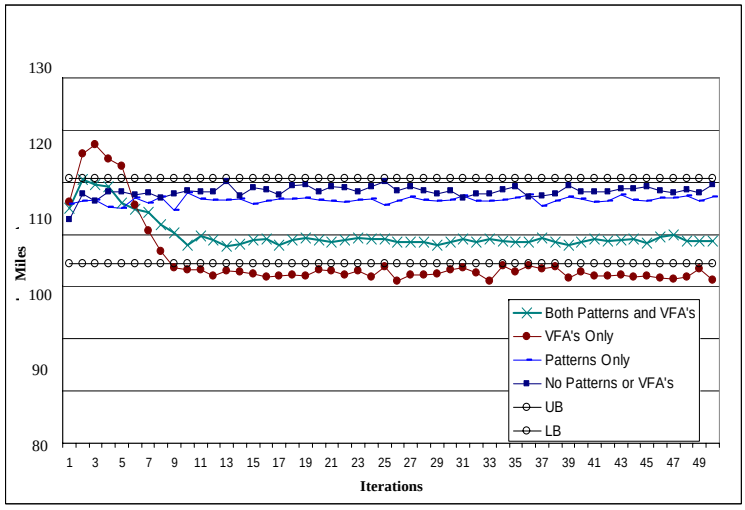
We have introduced two major algorithmic strategies for improving the performance of the model: value function approximations which produce the behavior of optimizing over time, and pattern matching, which is specifically designed to help the model match the length of haul for each driver class. These strategies introduce two questions: How do value function approximations and patterns each contribute to the ability of the model to match historical performance? And how do they individually affect the quality of the solution as measured by the objective function?

Figure 8 shows the average length of haul as a function of the number of iterations a) with patterns and VFA's, b) with patterns and without VFA's, c) without patterns and with VFA's, and d) without patterns or VFA's. We show the results for two different driver classes because the behavior is somewhat different. In both figures, we show upper and lower bounds specified by management as the limit of what they consider acceptable (the middle of this range is considered the best). Both figures show that we obtain the worst results when we do not use VFA's or patterns, and we obtain the best results with patterns and VFA's. Of interest is the individual contribution of VFA's versus patterns. In figure 8a, the use of VFA's alone improves the ability of the model to match history, while in figure 8b, VFA's actually make the match worse. But even in figure 8b, VFA's and patterns together outperform either alone.

We next examine the effect of VFA's and patterns on the objective function. We define the objective function as the total contribution earned by following the policy determined by using patterns and value functions. The contributions include the revenues from covering loads, minus the cost of moving the truck and any penalties for activities such as arriving late to a service appointment or allowing a driver to be away from home for too long (the "soft costs"). Figure 9 shows the objective function for the same four combinations (with and without patterns, with and without value functions). The figure shows that the results using the value functions significantly outperform the results without the value functions. Including the patterns with the value function does not seem to change the objective function (although it obviously improves our ability to match history performance measures). Interestingly,



8a - Driver type 1



8b - Driver type 2

Figure 8: Length of haul for two driver classes a) with patterns and VFA's, b) with patterns and without VFA's, c) without patterns and with VFA's, and d) without patterns or VFA's. Upper and lower bounds (UB and LB) represent the acceptable range set by management.

using patterns without the value functions produces a noticeable improvement over the results without the patterns (or value functions), suggesting that the patterns do, in fact, contribute to the overall objective function. However, the point of the patterns is to achieve goals that are not captured by the objective function, so this benefit appears to be incidental.

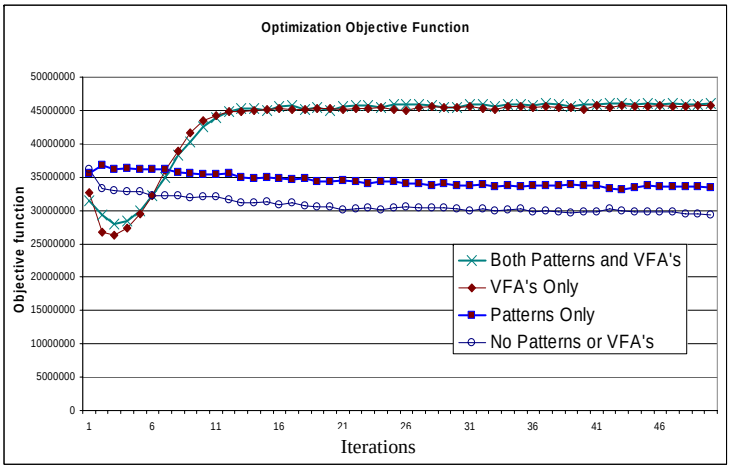


Figure 9: Objective function a) without patterns and VFA's, b) with patterns and without VFA's, c) without patterns and with VFA's, and d) with patterns and VFA's.

5 Fleet mix studies

All truckload motor carriers are continuously hiring drivers just to maintain their fleet size. It is not unusual for companies to experience over 100 percent turnover (that is, if the fleet has 1000 drivers, they have to hire 1000 drivers a year to maintain the fleet). Since companies are constantly advertising and processing applications, it is necessary to decide each week how many jobs to offer drivers based on their home domicile, and which of the three capacity types they would belong to. We studied the ability of the model to help guide the driver hiring process.

We divided the study into two parts. In section 5.1, we assessed our ability to estimate the marginal value of a driver type (defined by the driver domicile and capacity type) using the value function approximations. Then, section 5.2 reports on the results of simulations where we used the value functions to change the mix of drivers (while holding the fleet size constant).

5.1 Driver valuations

For our project, the 100 domicile regions and three capacity types produced 300 driver types. If this were a traditional simulator, we could estimate the value of each of these driver types by starting from a single base run, then incrementing the number of drivers of a particular type and then running the simulation again. There is a fair amount of noise inherent in the results of a single simulation run, so it might be reasonable to replicate this 10 times and take an average. With 300 driver types, this implies 3,000 runs of the simulation model.

We can avoid this by simply using the value functions. If we run N iterations of our ADP algorithm, we might expect that the final value functions for time $t = 0$, given by $\bar{v}_0^N = (\bar{v}_{0a}^N)_{a \in \mathcal{A}}$, could be used to estimate the value of each driver type. The value functions are indexed by three attributes (location, domicile, and capacity type) while we only need values indexed by domicile and capacity type. The value indexed by domicile and capacity type is nothing more than an aggregation on location, and was estimated using the same methods we used to estimate the value functions at different levels of aggregation. For the remainder of this section, we use the attribute a to represent driver domicile and capacity type only. In addition, we will let $\bar{v}_{0a} = \bar{v}_{0a}^N$ be our final estimate of the marginal value of a driver (at time 0) with attribute a.

While it is certainly reasonable to expect that $\bar{v}_{0a}$ to be the marginal value of a driver of type a, we needed to verify that this was, in fact, an accurate estimate. We ran experiments adding 10, 20, 30, 40 and 50 drivers for four different driver classes. These experiments convinced us that the model produced relatively linear behavior when we add up to 20 drivers.

We then estimated the value of adding 20 different types of drivers (different domiciles and capacity types) by adding 20 drivers and averaging the marginal value over 10 repetitions of the experiment. In each case, we computed a 95 percent confidence interval for the slope (based on the estimated mean and standard deviation of both the base case and the results of the 10 iterations with 20 additional drivers). Figure 10 shows the confidence intervals for the slope estimated from adding 20 additional drivers and the point estimate from the value function for the 20 different driver types. For 18 driver types, the value function estimate fell

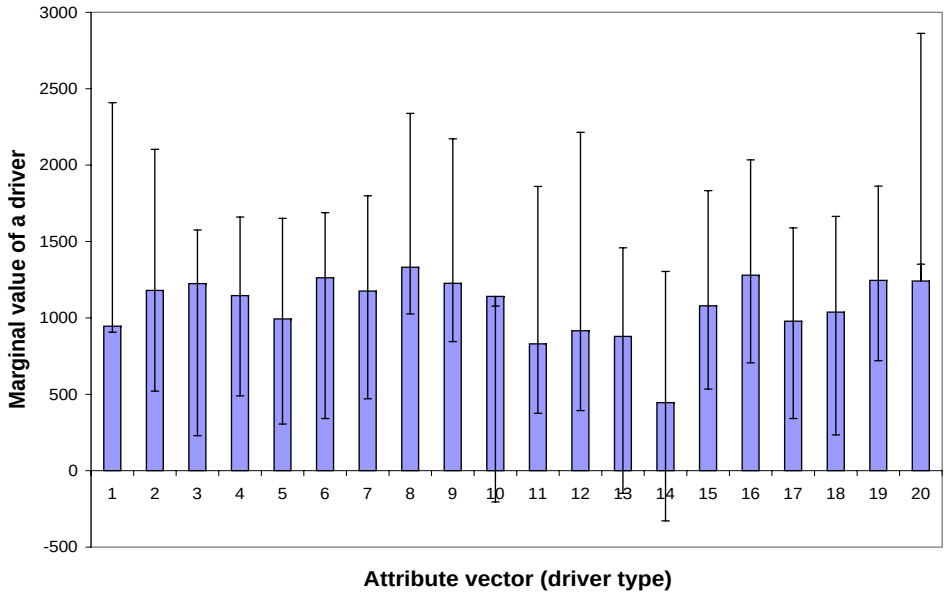


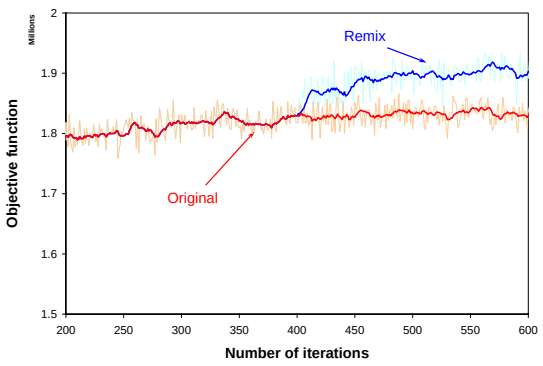
Figure 10: Predicted values compared against observed values from actual scenarios. The columns represent the approximations of the marginal values for different driver types. The error bars denote a 95% confidence interval around the mean marginal value, computed from observed scenarios.

within the confidence interval (with a 95 percent confidence interval, we would expect 19 of the driver types to fall within the confidence interval).

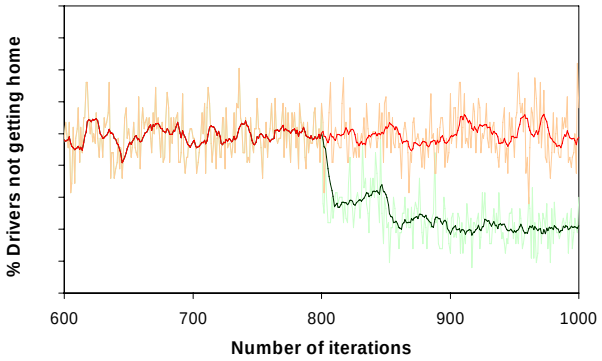
5.2 Driver Remix Experiments

In this section, we attempt to optimize the number of drivers belonging to each class so that there is an increase in the objective function. The method that we adopt for this purpose is to redistribute the drivers between the various driver types, such that there are more drivers of types with higher marginal values as compared to the ones with lower values.

To find the number of drivers to be added or removed from each class, we apply a stochastic gradient algorithm where we use a correction term to smooth the original number of drivers of each class. The correction term is a function of the difference in the marginal value from



11a - Objective function



11b - Percent not getting home

Figure 11: Result of driver remix experiments. a) Shows the change in the objective function, b) shows change in the percent of drivers not getting home.

the mean marginal value of all the driver classes. We define the following:

$$\begin{aligned} \bar{v}_a^n &= \text{The marginal value of a driver with attribute } a \text{ at iteration } n. \\ R_a^n &= \text{The number of drivers with attribute } a \text{ at iteration } n. \\ \bar{v}_* &= \bar{v}_a^n \text{ averaged over all attribute vectors } a. \end{aligned}$$

The algorithm for computing the new number of drivers of class a consists of the following step:

$$R_a^{n+1} = \max\{0, R_a^n + \beta\,(\bar{v}_a^n - \bar{v}_*^n)\} \tag{30}$$

where β is a scaling factor which we set, after some experimentation, to 0.10. After the update, we then rescaled R^{n+1} so that $\sum_a R_a^{n+1} = \sum_a R_a^n$.

In figure 11, we show the effect of shifting to a new mix of drivers. Figure 11a) shows the improvement in the objective function when we used value functions to adjust the mix of drivers. We did not adjust the driver mix until iteration 400 so that the value functions had a chance to stabilize. Figure 11b) shows the percent of drivers that did not get home within the simulation. This figure shows a significant improvement in our ability to get drivers home when we shift the fleet based on the value function approximations.

6 Conclusions

This paper has demonstrated that approximate dynamic programming allows us to produce an accurate simulation of a large-scale fleet that a) allowed us to capture real-world operations at a very high level of detail, b) produced operating statistics that closely matched historical performance, and c) provided accurate estimates of the marginal value of 300 different driver types from a single simulation. The technology of approximate dynamic programming allows us to capture all the relevant features of drivers and loads to produce a very realistic simulation, including decisions that balance immediate contributions against down-stream impacts. The logic is able to handle different types of uncertainty including random customer demands and travel times. Value function approximations produced not only more realistic behaviors (measured in terms of our ability to match historical performance) but also provided the marginal value of different types of drivers from a single run of the model.

This project motivated other important results. Although the value functions were approximated in terms of only four driver attributes (location, driver type, domicile and time), this still produced 600,000 parameters to be estimated, creating a significant statistical problem. A new approach for estimating parameters using a weighted average of estimates at different levels of aggregation was developed specifically for this project. This method was shown to produce better, more stable estimates. This project also motivated the development of a new stepsize formula that eliminated the need to constantly tune parameters in deterministic formulas. Finally, we used novel pattern-matching logic to produce behaviors (the average length of a load for different driver types) that matched historical performance.

The simulator has been adopted at Schneider National as a planning tool that is, as of this writing, used continually to perform studies of policies that affect the performance of the network. A partial list of benefits from studies that have been undertaken using the simulator are

- Getting drivers home - A major component for retaining drivers in a long-haul carrier is the ability to return them home in a predictable way. Schneider had developed a plan to make stronger commitments to drivers, but the simulator showed that the plan

would have cost the company $30M per year. Using the model, an alternative strategy was developed which provided 93 percent of the proposed self-scheduling flexibility while costing only $6M per year.

- Quantifying the cost of hours-of-service rules – Using the model, Schneider has been able to quantify the cost of changes in the hours-of-service rules set by the Department of Transportation. With this information, we are able to effectively negotiate adjustments in customer billing rates and freight tendering/handling procedures, leading to margin improvements of 2-3 percent.

- Setting appointments - The model has been used to evaluate the value of new policies for setting appointments. Preliminary results suggest margin impacts from improved utilization are in a range of 4 to 10 percent, and the number of late deliveries was reduced by half.

- Cross-border driver management - With recent changes in security and border policies, it is necessary to maintain a pool of drivers who are trained with these policies. Using the the model, Schneider was able to reduce the number of drivers engaged in border-crossing by 91 percent, and restrict relays to three designated points. This has resulted in an initial avoidance of $3.8M in training/identification/certification costs and ongoing annual cost avoidance of $2.3M.

- Hiring drivers - The home location of long-haul truck drivers has a significant impact on network operating efficiency. Schneider is continually hiring drivers, and can control the number of drivers hired in each region. Using the model, Schneider has been able to quantify the marginal contribution of changes in regional driver populations, leading to an estimated annual profit improvement of $5M.

The next step with the model is to focus on the loads. The model currently does not model the difference between tendered loads (loads offered to the company which may be refused), committed loads (tendered loads which the carrier has made a commitment to move) and contracted loads (loads which are offered to the carrier under a standing contract). Our goal is to use the model to identify good policies for making commitments to loads as they are

tendered, taking into consideration the state of the system. Once this policy is in place, the next goal would be to determine how to evaluate customer contracts as a foundation for determining contractual commitments. We are not aware of any existing technology that can evaluate loads in the presence of driver management issues.

This project has offered an important insight into the process of implementing optimization models for operational problems. The research community has traditionally focused on developing optimization models which produce the best possible solutions, presumably better than what can be achieved by a company. Our experience, with this and other similar projects, is that the first and most important goal is to produce a model that calibrates against history. Of particular importance was the ability of the model to handle a high level of detail, allowing the model to accurately represent hours-of-service rules, detailed service commitments, and complex rules governing driver relays and foreign drivers. Only after the model proved to be realistic did the carrier begin to believe the results. But perhaps the most remarkable conclusion was that an optimization model that used optimal solutions at a point in time, and near-optimal solutions over time, accurately reproduced (at an aggregate level) the performance of a well-run company.

References

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

Caliskan, C. & Hall, R. W. (2003), 'A dynamic empty equipment and crew allocation model for long-haul networks', *Transportation Research A* **5**, 405–418.

Chiu, Y. & Mahmassani, H. S. (2002), 'Hybrid real-time dynamic traffic assignment approach for robust network performance', *Transportation Research Record* **1783**, 89–97.

Crainic, T., Ferland, J. & Rousseau, J.-M. (1984), 'A tactical planning model for rail freight transportation', *Transportation Science* **18**(2), 165–184.

Desaulniers, G., Desrosiers, J., Gamache, M. & Soumis, F. (1998), Crew Scheduling in Air Transportation, *in* T.G. Crainic & G. Laporte, eds, 'Fleet Management and Logistics', Kluwer Academic Publishers, Norwell, MA, pp. 169–185.

Desrosiers, J., Solomon, M. & Soumis, F. (1995), Time constrained routing and scheduling, *in* C. Monma, T. Magnanti & M. Ball, eds, '*Handbook in Operations Research and Management Science*, Volume on *Networks*', North Holland, Amsterdam, pp. 35–139.

Gendreau, M. & Potvin, J. Y. (1998), Dynamic vehicle routing and dispatching, *in* T. Crainic & G. Laporte, eds, 'Fleet management and logistics', Kluwer Academic Publishers, pp. 115–126.

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

George, A. (2005), Optimal Learning Strategies for Multi-Attribute Resource Allocation Problems, PhD thesis, Princeton University.

George, A. & Powell, W. B. (2006), 'Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming', *Machine Learning* **65**(1), 167–198.

George, A., Powell, W. B. & Kulkarni, S. (2005), Value function approximation using hierarchical aggregation for multi-attribute resource management, Technical report, Princeton University, Department of Operations Research and Financial Engineering.

Ichoua, S., Gendreau, M. & Potvin, J.-Y. (2006), 'Exploiting knowledge about future demands for real-time vehicle dispatching', *Transportation Science* **40**(2), 211–225.

Kleywegt, A., Nori, V. S. & Savelsbergh, M. W. P. (2004), 'Dynamic programming approximations for a stochastic inventory routing problem', *Transporation Science* **38**, 42–70.

Larsen, A., Madsen, O. B. G. & Solomon, M. M. (2002), 'Partially dynamic vehicle routing - models and algorithms', *Journal of the Operational Research Society* **53**, 637–646.

Marar, A. (2002), Information Representation in Large-Scale Resource Allocation Problems: Theory, Algorithms and Applications, PhD thesis, Princeton University.

Marar, A. & Powell, W. B. (2004), Using static flow patterns in time-staged resource allocation problems, Technical report, Princeton University, Department of Operations Research and Financial Engineering.

Marar, A., Powell, W. B. & Kulkarni, S. (2006), 'Capturing expert knowledge in resource allocation problems through low-dimensional patterns', *IIE Transactions* **38**(2), 159–172.

Powell, W. B. (1988), A comparative review of alternative algorithms for the dynamic vehicle allocation problem, *in* B. Golden & A. Assad, eds, 'Vehicle Routing: Methods and Studies', North Holland, Amsterdam, pp. 249–292.

Powell, W. B. (2007), *Approximate Dynamic Programming: Solving the curses of dimensionality*, John Wiley and Sons, New York.

Powell, W. B., Jaillet, P. & Odoni, A. (1995), Stochastic and dynamic networks and routing, *in* C. Monma, T. Magnanti & M. Ball, eds, '*Handbook in Operations Research and Management Science*, Volume on *Networks*', North Holland, Amsterdam, pp. 141–295.

Powell, W. B., Shapiro, J. A. & Simão, H. P. (2001), A representational paradigm for dynamic resource transformation problems, *in* R. F. C. Coullard & J. H. Owens, eds, 'Annals of Operations Research', J.C. Baltzer AG, pp. 231–279.

Powell, W. B., Wu, T. T. & Whisman, A. (2004), 'Using low dimensional patterns in optimizing simulators: An illustration for the airlift mobility problem', *Mathematical and Computer Modeling* **29**, 657–2004.

Psaraftis, H. (1995), 'Dynamic vehicle routing: Status and prospects', *Annals of Operations Research* **61**, 143–164.

Regan, A., Mahmassani, H. S. & Jaillet, P. (1998), 'Evaluation of dynamic fleet management systems - simulation framework', *Transportation Research Record* **1648**, 176–184.

Secomandi, N. (2000), 'Comparing neuro-dynamic programming algorithms for the vehicle routing problem with stochastic demands', *Computers and Operations Research* **27**(11), 1201–1225.

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

Spivey, M. & Powell, W. B. (2004), 'The dynamic assignment problem', *Transportation Science* **38**(4), 399–419.

Spivey, M. J. (2001), The dynamic assignment problem, PhD thesis, Princeton University.

Sutton, R. & Barto, A. (1998), *Reinforcement Learning*, The MIT Press, Cambridge, Massachusetts.

Tapiero, C. & Soliman, M. (1972), 'Multicommodities transportation schedules over time', *Networks* **2**, 311–327.

Taylor, G., Meinert, T. S., Killian, R. C. & Whicker, G. L. (1999), 'Development and analysis of alternative dispatching methods in truckload trucking', *Transportation Research E* **35**(3), 191–205.

Tjokroamidjojo, E. Kutanoglu, G. T. (2001), Quantifying the value of advance load information in truckload trucking, Technical report, University of Arkansas.

Topaloglu, H. & Powell, W. B. (2006), 'Dynamic programming approximations for stochastic, time-staged integer multicommodity flow problems', *Informs Journal on Computing* **18**(1), 31–42.

White, W. (1972), 'Dynamic transshipment networks: An algorithm and its application to the distribution of empty containers', *Networks* **2**(3), 211–236.

Yang, J., Jaillet, P. & Mahmassani, H. (2004), 'Real-time multi-vehicle truckload pick-up and delivery problems', *Transportation Science* **38**, 135–148.