Dynamic Management of Heterogenous Resources

Warren B. Powell
Joel A. Shapiro
Hugo P. Simão

Department of Civil Engineering and Operations Research,
Princeton University, Princeton, NJ 08544
Statistics and Operations Research
Technical Report SOR-98-06

December, 1998

Abstract

In this paper we develop a new method for solving Dynamic Resource Allocation Problems (DRAP) that occur in the operation of freight transportation systems. Such problems involve the allocation of resources to perform tasks over a discrete-time, dynamic network. Our particular interest is in ultra-large scale problems, that involve managing thousands of resources (such as drivers or vehicles). We focus on problems with dynamic attributes, where the characteristics of the resource may change as it handles each task. We provide a general and very flexible formulation, and provide a solution based on the principle of dynamic programming. A newly proposed linearization approximation is shown to provide high quality solutions with reasonable execution times. The technique is illustrated in the context of the driver management problem for a large motor carrier.

We consider the problem of optimizing, in a dynamic setting, the management of thousands of discrete resources characterized by a number of attributes. Our application is motivated by a driver scheduling problem arising in freight transportation, where we were faced with the challenge of routing and scheduling over 5,000 drivers to serve 30,000 loads over a four day horizon. Similar problems arise in other resource management problems where there is a relatively high level of uncertainty. Examples include both crew scheduling and asset management (locomotives and aircraft) for railroads and the Air Mobility Command. Complex equipment such as locomotives and aircraft generally need to be described by a rich set of attributes.

Our problem is different in a significant way from two closely related problem classes. The first is crew scheduling, which is generally done in the context of a deterministic environment (all information is known) and where the attributes of a resource (crew) can be very complex, consisting possibly of dozens of dimensions. Our problem class is characterized by a relatively high level of uncertainty; partly as a result of this uncertainty, it makes sense to use a moderate level of aggregation and characterize resources with fewer attributes. The second problem class is the dynamic vehicle allocation problem, where uncertainty in customer demands is often modeled explicitly, but where the vehicles are either homogeneous (the so-called "single commodity" problem) or with a single attribute giving the type of vehicle (where the number of vehicle types might range between one and three dozen).

The focus of the project was to develop a system that would run in real time, but which would provide tactical information to a group of centralized planners. As a result, we were not interested in assigning a particular driver to a particular load "here and now" but rather to provide forecasts of available drivers and loads waiting to be moved over a four day horizon. For this reason, we did not need to model each driver individually, but we did need to capture the attributes of a driver with sufficient accuracy to model basic flows. An optimization model was needed since we needed to project decisions (assigning drivers to loads, moving empty, holding freight) in the future.

We approach the problem as a dynamic resource allocation problem (DRAP) which involves the management of two types of multiattribute resources (drivers and loads) over time. A driver may move a load, or move alone (an empty move). Loads, of course, cannot move without a driver, but may sit idle if no driver is assigned to move a load. Loads incur a service penalty for each shipment that arrives at its destination after its delivery date.

Dynamic resource allocation problems of the type we consider arise in a number of settings in

transportation. Examples include:

- Management of fleets of boxcars for railroads, including empty repositioning, load prioritization and fleet sizing.
- Operational planning of truckload fleets.
- Empty repositioning of containers moving over truck and rail networks.
- Management of rail locomotives.
- Capacity forecasting and pricing for one–way rental fleets.
- Management of drivers and crews in networks that arise in motor carriers and railroads.
- Management of chemical containers.

The unique challenge that we faced involved both the large size of our problem, measured in terms of the number of drivers and loads to be moved, and the relatively complex set of attributes that describe a driver. Since we are solving the problem at a somewhat aggregate level, we do not face the complexity addressed in formulations that manage individual drivers or crews (as is done in crew scheduling for airlines [8] or trucking [20]). However, our problem is *much* larger than is usually considered in these problem settings (measured in terms of the number of drivers and loads being managed).

Problems with lots of resources, such as fleet management problems, are typically modeled as network flow problems. These models are relatively insensitive to the number of vehicles being managed, but are very sensitive the number of different states that a resource can occupy. For example, a fairly complex instance of a fleet management problem might involve the flows of, say, 20 types of box cars between 100 locations, producing a state space of 2000 possible car-type/location combinations. By contrast, our problem produces a potential state space of over one million possible resource types. Thus, our problem shares the large state space of a crew scheduling problem and the large number of resources of a fleet management problem.

Prior research on dynamic resource allocation problems can be divided into two classes. The first is crew scheduling (CSP), which involves the routing and scheduling of airline crews, typically with complex attributes, over a set of flight legs. (Crew scheduling problems are deterministic, but

the techniques can be applied to deterministic approximations of dynamic problems on a rolling basis.) While a variety of heuristics have been proposed, this class of problems can be solved effectively using column generation techniques. An excellent survey of this approach is presented by Desrosiers *et al.* [8]. In this context, the CSP is formulated as a set partitioning problem, with rows corresponding to flight legs, and columns corresponding to feasible pairings. In order to keep the number of columns manageable, the set partitioning problem is solved using Dantzig-Wolfe decomposition. Columns may be generated using heuristics (see Crainic and Rousseau [7]) for maximum flexibility in handling workrules and non-additive path costs. They can, however, be generated optimally by solving a classical shortest path problem in an expanded network (Desrosiers *et al.* [9], Lavoie *et al.* [13]).

A second line of research has been in the dynamic fleet management literature, which easily handles fleets of thousands of vehicles, typically under the banner of the dynamic vehicle allocation problem. A review of this field can be found in Powell, Odoni and Jaillet [16] and Powell [18]. This problem has been formulated deterministically by White [24] and stochastically by Jordan and Turnquist [12], Frantzeskakis and Powell [11], and Cheung and Powell [6]. This work typically involves a single type of flow and tight time windows (the work of Jordan and Turnquist [12] provides for backlogging).

Optimization models for multiple equipment types with substitution typically produce large (integer) multicommodity flow problems. This problem has usually been solved in practice using network decomposition techniques, or by solving the linear relaxation, and then using rounding heuristics to obtain an integer solution. We could not solve even an LP relaxation of our problem using a commercial solver. We also experimented with network decomposition techniques, again without success. Our biggest difficulty with this class of methods was the one-sided time windows on the loads. Each load may be served at any point after its initial point of availability, although there is an increasing penalty as a load is delayed as the service requirement of each shipment on the trailer is violated.

Powell and Carvalho [22, 21] present an optimal control approach for planning the flows of single and multicommodity networks. Their technique also accomodates time windows on tasks in a heuristic fashion. Comparisons to optimal (noninteger) solutions indicates that the results are typically within two to five percent of a linear programming relaxation when applied to fleet management problems. The technique is appealing because it not only scales easily to very large

problems, it also can be trivially extended to handle forecasting uncertainties. This prior work, however, took advantage of the relatively small state space that arises in equipment management problems. In these problems, it is common to have numerous (e.g. 10 or 20) vehicles in the same state at the same time. Driver management problems, with much larger state spaces, will exhibit a state space where most of the states have no drivers in them, and many states with only one or two drivers per state. It is also easy to enumerate all possible states in advance. These properties were used in the design of both the mathematical algorithm, and the software, in the work of Powell and Carvalho. In this paper, we were unable to use these properties and had to redesign portions of the algorithm. Most significantly, it was not apparent that the technique would work as well on our problem class as it did on problems reported in the earlier work.

The contribution of this paper is to document the successful application of a dynamic programming approximation to a very large scale driver management problem. To our knowledge, this is the first time a dynamic programming approximation has been used in the context of a large-scale driver management problem. The successful use of this technique in the fleet management arena, given the much smaller state space, was no guarantee of success in this problem class. Most important, we are unaware of any competing method that can be applied to a problem of this scale. Our numerical work shows solutions that are within a few percent of optimal bounds. Even more important, the method compares favorably against historical performance. Computationally, we get good solutions from a cold start within approximately 20 minutes, and can provide updates in response to new data within a minute.

One of the challenges that we had to address in this paper was the evaluation of the quality of our solution method, which depends on an approximation of the value function in a dynamic program. Prior research into this method, reported in [22, 21], addressed problems that were sufficiently small that continuous approximations could be solved to optimality using a commercial LP solver. By contrast, the problem we address in this paper is far too large to yield to such an approach, regardless of acceptable run times. For this reason, three measures of performance are provided. In the first, we test our adaptation of the method in [22] on problems that can be solved as a linear program (since the LP produces non-integer solutions, this is a bound). Next, we take the full problem and solve a static approximation to optimality. Both of these benchmarks appear to be very tight. Finally, we compare our results to actual historical performance achieved by the motor carrier, and show that our method produces lower overall empty miles. While none of

these three measures provides a perfect yardstick, combined, they present a fairly compelling case supporting the quality of our solution.

In the next section we establish the notation which is used to describe the problem. Our notation captures a highly complex problem in a compact form that easily lends itself to introducing a wide range of operational issues in a simple way. We then present, in Section 2, the DRAP in a simultaneous formulation suitable for solution by integer programming. Unfortunately, the size of this formulation, and the presence of wide time windows, make it unsolvable by standard means. In response, we propose in Section 3 a new method for solving the DRAP as a dynamic program (DP), called the Logistics Queueing Network (LQN) approach. We show that a dynamic programming formulation of the DRAP is both natural and helpful in studying the problem, although the resulting optimality recursion is computationally intractable. To work around this difficulty, an approximation to the DP value function is suggested that leads to the main discussion of this paper. We also deal with issues of state aggregation and its impacts on solution efficiency and accuracy. In Section 4, we pose several questions on the efficiency and accuracy of our approach and set out to answer them in a systematic fashion.

1 Problem Notation

In this section we introduce the notation which is used to describe the problem throughout the paper. Our notation is actually a powerful tool for representing dynamic resource allocation problems, allowing practitioners to capture complex real-world details in an easy and intuitive manner.

As a general rule, we use lowercase roman letters to represent decision variables and script roman letters to represent sets. The set of positive integers $\{1, 2, 3, \ldots\}$ is denoted by $\mathbb{N}$.

Because the model is dynamic, we need to make decisions over a number of time periods. Recall that we have defined the parameter $T \in \mathbb{N}$ to be the number of planning periods in our planning horizon. We allow decisions to be made at T points in our planning horizon, which we refer to as time indices $t = 0, 1, \ldots, T-1$. Time period $t \in \mathcal{T}$ represents the interval $[t, t+1)$ and is of uniform length. Time indices and time periods are represented by the letters s and t.

The network

We define the following set:

$$\mathcal{C} \;=\; \text{the set of all physical terminals in the transportation network.}$$

We use the letters i and j to index terminals in $\mathcal{C}$. The DRAP is solved on a discrete-time, dynamic network which is generated by replicating the physical network at each time index. Each terminal $i \in \mathcal{C}$ appears T times in the dynamic network, and is referred to as node (i, t) at time $t \in \mathcal{T}$. A terminal i is connected to a terminal j by the dynamic link (i, j, t) with travel time $\tau_{ij} \in \mathbb{N}$. We require $\tau_{ii} = 1$ for all $i \in \mathcal{C}$ and define $\tau_{\max} = \max_{i,j \in \mathcal{C}} \tau_{ij}$ as the maximum travel time in the transportation network. By assumption $\tau_{\max}$ is a strictly positive, finite integer.

Resources

To account for the resources moving through our dynamic network we define the following set for each $i \in \mathcal{C}$, $t \in \mathcal{T}$ and $s = 0, \ldots, \tau_{\max} - 1$:

$$\mathcal{R}_{it}(s) \;=\; \text{the set of all resources currently allocated to arrive to terminal } i \text{ at time } t + s.$$

We index resources by r. To simplify our notation we define for each $t \in \mathcal{T}$

$$\mathcal{R}_t = \bigcup_{s=0}^{\tau_{\max}-1} \bigcup_{j \in \mathcal{C}} \mathcal{R}_{jt}(s)$$

as the set of all resources in the network at time t. We assume that $\mathcal{R}_0$ is given as input data.

Because of the heterogeneity of resources, we must track each individual resource over time. To this end, we define the following resource state vector:

$$a \;=\; \{\ldots, a_{rt}, \ldots\},$$

where

$$a_{rt} \;=\; \text{attribute vector of resource } r \in \mathcal{R}_t \text{ at time } t \in \mathcal{T}.$$

$$a_t \;=\; (\ldots, a_{rt}, \ldots)$$

A resource's attribute vector completely describes its state at any point on our space-time grid. These attributes are used to determine the cost and feasibility of a potential resource-to-task

Example 1(Resource Attribute Vector): Elements 1 through 5 are as follows:

$\mathbf{a_{rt}(1)}$ = the domicile terminal or 'home base' of driver r,
$\mathbf{a_{rt}(2)}$ = the current/next terminal of driver r,
$\mathbf{a_{rt}(3)}$ = the ETA of driver r at his current/next terminal,
$\mathbf{a_{rt}(4)}$ = the cumulative road (i.e., driving) time of driver r,
$\mathbf{a_{rt}(5)}$ = the cumulative duty time of driver r.

assignment. We refer to the kth component of a resource's attribute vector as $a_{rt}(k)$. In Example 1 we specify typical elements which a resource attribute might contain in a motor carrier setting. We denote the set of all possible resource attribute vectors by $\mathcal{A}$.

Tasks

Similarly, to account for the tasks that need to be moved about our dynamic network we define the following set for each $i \in \mathcal{C}$ and $t \in \mathcal{T}$:

$$\mathcal{L}_{it} \;=\; \text{the set of all currently uncovered tasks with origin terminal } i \text{ at time } t.$$

We index tasks by l. The set of all uncovered tasks in the network at time $t \in \mathcal{T}$ is defined as

$$\mathcal{L}_t = \bigcup_{i \in \mathcal{C}} \mathcal{L}_{it}.$$

The initial set $\mathcal{L}_0$ must be given as input data.

The task state vector is:

$$b \;=\; \{\, \ldots, b_{lt}, \ldots \,\},$$

where

$$b_{lt} \;=\; \text{attribute vector of task } l \in \mathcal{L}_t \text{ at time } t \in \mathcal{T}.$$

As with resource attribute vectors, the kth component of a task's attribute vector is referred to as $b_{lt}(k)$. We present typical elements of a task attribute vector as they might arise in a motor carrier setting in Example 2. We denote the set of all possible task attribute vectors as $\mathcal{B}$.

Decision Variables

We use two types of decision variables: assignment variables, x, that are used to assign a resource

Example 2(Task Attribute Vector): We define components 1 through 5 as:

$\mathbf{b_{lt}(1)}$ = the terminal most recently visited by load l,
$\mathbf{b_{lt}(2)}$ = the destination terminal of load l,
$\mathbf{b_{lt}(3)}$ = the time of availability of load l at its origin terminal,
$\mathbf{b_{lt}(4)}$ = the due time of load l at its destination terminal,
$\mathbf{b_{lt}(5)}$ = the remaining travel time for load l.

to a task, and repositioning variables, y, that are used to move a resource from one location to the next. More precisely, for each $t \in \mathcal{T}$, $i \in \mathcal{C}$, $r \in \mathcal{R}_{it}(0)$ and $l \in \mathcal{L}_{it}$ we define

$$x_{rlt} = \begin{cases} 1 & \text{if resource } r \text{ is assigned to `cover' task } l \text{ beginning during time period } t, \\ 0 & \text{otherwise} \end{cases}$$

and for $j \in \mathcal{C}$:

$$y_{rjt} = \begin{cases} 1 & \text{if resource } r \text{ is allocated to move unassigned from terminal } i \text{ to} \\ & \text{terminal } j \text{ beginning during time period } t, \\ 0 & \text{otherwise.} \end{cases}$$

Notice that we have restricted resource assignments to tasks available at the resource's current terminal. However, we have intentionally used the vague term "cover" to describe the assignment of a resource to a task, so that we are not limited to any one mode of transportation or type of service. We account for the coverage of tasks by defining for each $t \in \mathcal{T}$, $i \in \mathcal{C}$ and $l \in \mathcal{L}_{it}$ the integer variable z_{lt} by

$$z_{lt} = \sum_{r \in \mathcal{R}_{it}(0)} x_{rlt}.$$

Clearly $z_{lt} = 1$ if task l is first covered during period t and $z_{lt} = 0$ otherwise. For notational clarity we define the vectors of decision variables as $x_t = \{ \ldots, x_{rlt}, \ldots \}$, $y_t = \{ \ldots, y_{rjt}, \ldots \}$ and $z_t = \{ \ldots, z_{lt}, \ldots \}$.

System dynamics

It is convenient to describe our collection of resources and tasks as an abstract 'system'. We denote the 'state' of the system at time $t \in \mathcal{T}$ immediately *prior* to the knowledge of the decisions in x_t and y_t by S_t. The vector S_t is a function of S_{t-1}, x_{t-1} and y_{t-1}. Stated more formally,

$$S_t = (a_{rt}, \forall r \in \mathcal{R}_t; b_{lt}, \forall l \in \mathcal{L}_t),$$

where $x_t = y_t = 0$ for $t < 0$. S_0 is required as input data.

To keep our system state variable current, we must update it after each time period has passed and the accompanying set of decisions have been implemented. We can describe a system state variable update in more detail using the algebra which we have developed. We define operators A and B to update the resource and task attribute vectors, respectively.

The A operator is used to update the resource attribute vectors. Given x_t and y_t at their optimal values, we obtain a_{t+1} via

$$a_{t+1} = A(x_t, y_t, a_t, b_t).$$

The operation of A on x_t, y_t, a_t and b_t to produce a_{t+1} is really a sequence of actions. In Example 3 we detail the workings of the A operator on the sample resource attribute vector given in Example 1.

Example 3(The A Operator): We fix $r \in \mathcal{R}_t$ and let $i = a_{rt}(2)$ for clarity. Element 1 of a_{rt} is fixed and thus $a_{rt+1}(1) = a_{rt}(1)$. We can update element 2 of a_{rt} by

$$a_{rt+1}(2) = \begin{cases} j & \text{if } x_{rlt} = 1 \text{ for } l \in \mathcal{L}_{it} \text{ with } b_{lt}(2) = j \text{ and } \tau_{ij} = 1, \\ k & \text{if } y_{rkt} = 1 \text{ for } k \in \mathcal{C} \text{ with } \tau_{ik} = 1, \end{cases}$$

if $r \in \mathcal{R}_{it}(0)$. If $r \in \mathcal{R}_{jt}(1)$ for some $j \in \mathcal{C}$ then $a_{rt+1}(2) = j$. Otherwise, $a_{rt+1}(2) = a_{rt}(2)$.
To update the resource's ETA, set

$$a_{rt+1}(3) = \begin{cases} a_{rt}(3) & \text{if } r \notin \mathcal{R}_{it}(0), \\ t + \sum_{l \in \mathcal{L}_{it}} x_{rlt} b_{lt}(5) + \sum_{j \in \mathcal{C}} y_{rjt} \tau_{ij} & \text{otherwise.} \end{cases}$$

We update the resource's road time in the following fashion:

$$a_{rt+1}(4) = \begin{cases} a_{rt}(4) & \text{if } r \in \mathcal{R}_{it}(0) \text{ and } a_{rt+1}(2) = a_{rt}(2), \\ a_{rt}(4) + 1 & \text{otherwise.} \end{cases}$$

Finally, the resource's duty time is updated by setting $a_{rt+1}(5) = a_{rt}(5) + 1$.

The B operator is used to update the task attribute vectors. In other words,

$$b_{t+1} = B(b_t, x_t).$$

B works in sequence on elements $b_{lt}(1)$ through $b_{lt}(5)$ from Example 2 as described in Example 4.

We generally update the sets $\mathcal{R}_t$ and $\mathcal{L}_t$ at the same time as the system state variable S_t is updated, using the R and L operators, respectively.

Example 4(The B Operator): We fix $l \in \mathcal{L}_t$ and let $i = b_{lt}(1)$ for clarity. A task's last terminal is updated as follows:

$$b_{lt+1}(1) = \begin{cases} b_{lt}(2) & \text{if } b_{lt}(5) = 1 \text{ and } \sum_{\tau=0}^{t} z_{l\tau} = 1, \\ b_{lt}(1) & \text{otherwise.} \end{cases}$$

We assume that elements $b_{lt}(2), \ldots, b_{lt}(4)$ are fixed. Finally, we observe that

$$b_{lt+1}(5) = \begin{cases} b_{lt}(5) & \text{if } l \in \mathcal{L}_{it} \ \sum_{\tau=0}^{t} z_{l\tau} = 0, \\ \max(b_{lt}(5)-1, 0) & \text{otherwise.} \end{cases}$$

The R operator updates the set $\mathcal{R}_{it}$ to produce $\mathcal{R}_{it+1}$ for all $i \in \mathcal{C}$. Hence,

$$\mathcal{R}_{it+1} = R(\mathcal{R}_{it}, a_{t+1}).$$

For our sample resource attribute vector (Example 1) it operates as in Example 5.

Example 5(The R Operator): For each $i \in \mathcal{C}$ we set

$$\mathcal{R}_{it+1}(s) = \mathcal{R}_{it}(s+1) \bigcup \{r \in \mathcal{R}_t(0) : a_{rt+1}(2) = i \text{ and } a_{rt+1}(3) = t + s + 1\}$$

if $s \in \{1, \ldots, \tau_{\max} - 1\}$.

The L operator updates the set $\mathcal{L}_{it}$ to produce $\mathcal{L}_{it+1}$ for all $i \in \mathcal{C}$. Thus,

$$\mathcal{L}_{it+1} = L(\mathcal{L}_{it}, z_t).$$

For our sample task attribute vector (Example 2), the operator L is defined in Example 6.

Example 6(The L Operator): We set

$$\mathcal{L}_{it+1} = \{l \in \mathcal{L}_{it} : z_{lt} \neq 1\}$$

for all $i \in \mathcal{C}$.

The operators A, B, R and L are designed to accommodate general rules governing the evolution of the state of the system. Invariably, they are application specific, but do not directly affect the optimization procedures. These operators are an attempt to merge the strengths of simulation and

optimization. Judicious use of these operators will allow a practitioner to model even the most complicated workrules at relatively little cost to the efficiency of our algorithm.

The objective function

We complete our notation system by defining the cost and reward parameters which are used to evaluate each resource assignment:

$$
\begin{aligned}
h(a,b) &= \{\, \ldots, h(a_{rt}, b_{lt}), \ldots \,\} \\
&= \text{The vector of costs of assigning a resource with attributes } a_{rt} \text{ to cover} \\
&\quad\ \text{A task with attributes } b_{lt} \text{ beginning during period } t \in \mathcal{T},
\end{aligned}
$$

$$
\begin{aligned}
r &= \{\, \ldots, r_{lt}, \ldots \,\} \\
&= \text{The vector of rewards received for beginning task } l \in \mathcal{L}_{it} \text{ during time} \\
&\quad\ \text{period } t \in \mathcal{T},\ i \in \mathcal{C}.
\end{aligned}
$$

Typically, the cost vector h will include spatial components which account for the cost of physical movement, as well as temporal components which reward or penalize a delivery depending on whether it arrives to its destination by the specified due date or not.

To account for the repositioning movements of resources, we define:

$$
c(a) = \{\, \ldots, c_j(a_{rt}), \ldots \,\}
$$

where for each $i, j \in \mathcal{C}$, $t \in \mathcal{T}$ and $r \in \mathcal{R}_{it}(0)$,

$$
\begin{aligned}
c_j(a_{rt}) &= \text{The cost of a resource with attributes } a_{rt} \text{ moving unassigned from} \\
&\quad\ \text{terminal } i \text{ to terminal } j \text{ beginning during time period } t.
\end{aligned}
$$

2 Optimization Formulation

In this section we present the optimization formulation of the DRAP. We begin by considering the objective function. For each $t \in \mathcal{T}$ we define

$$
f_t(x_t, y_t) = r_t \bullet z_t - h_t(a_t, b_t) \bullet x_t - c_t(a_t) \bullet y_t,
$$

$$= \text{the net `profit' from the subset of decisions made at time } t,$$

where h_t and c_t are the subvectors of h and c at time t. The objective of the problem is to maximize the total profits over our planning horizon. Adding the necessary constraints, the DRAP is:

$$\max_{x,y} \sum_{t=0}^{T} f_t(x_t, y_t) \tag{1}$$

subject to, for all $t \in \mathcal{T}$:

Physical constraints:

$$\sum_{j \in \mathcal{C}} y_{rjt} + \sum_{l \in \mathcal{L}_{it}} x_{rlt} = 1 \quad \forall i \in \mathcal{C}, \quad \forall r \in \mathcal{R}_{it}(0) \tag{2}$$

$$\sum_{r \in \mathcal{R}_{it}(0)} x_{rlt} = z_{lt} \quad \forall i \in \mathcal{C}, \quad \forall l \in \mathcal{L}_{it} \tag{3}$$

$$\sum_{\tau=0}^{t} z_{l\tau} = 1 \quad \forall i \in \mathcal{C}, \quad \forall l \in \mathcal{L}_{it} \tag{4}$$

System dynamics:

$$a_{t+1} - A(x_t, y_t, a_t, b_t) = 0 \tag{5}$$

$$b_{t+1} - B(b_t, x_t, z_0, \ldots, z_t) = 0 \tag{6}$$

$$\mathcal{R}_{it+1} - R(\mathcal{R}_{it}, a_{t+1}) = 0 \quad \forall i \in \mathcal{C} \tag{7}$$

$$\mathcal{L}_{it+1} - L(\mathcal{L}_{it}, z_t) = 0 \quad \forall i \in \mathcal{C} \tag{8}$$

Nonnegativity and Integrality:

$$x_{rlt} \in \{0,1\} \quad \forall i \in \mathcal{C}, \quad \forall l \in \mathcal{L}_{it}, \quad \forall r \in \mathcal{R}_{it}(0) \tag{9}$$

$$y_{rjt} \in \{0,1\} \quad \forall i,j \in \mathcal{C}, \quad \forall r \in \mathcal{R}_{it}(0) \tag{10}$$

$$z_{lt} \in \mathbb{N} \quad \forall i \in \mathcal{C}, \quad \forall l \in \mathcal{L}_{it} \tag{11}$$

We would be remiss not to point out the deficiencies of this formulation. The principal disadvantage of this formulation is its size. The large scale problems that we would like to model typically contain over 6,000 resources, 35,000 tasks, 550 terminals and 60 time periods. The global formulation of such a problem instance would require millions of integer assignment variables x and millions of constraints and would be impossible to solve as an LP, let alone as an integer program. Our methodology reduces the dimension of the problem by decomposition and produces integer solutions in a natural manner.

3 Solution Approach

In this section we describe a relatively new approach to solving the DRAP. The method, called the "Logistics Queueing Network" approach (LQN) was introduced by Powell *et al.* [15]. This approach involves transforming the single large math program given in the previous section into a dynamic program, and then replacing the value function with a linear approximation. The result is thousands of small, almost trivial, subproblems, rather than a single, massive optimization problem.

To begin our description of the LQN method we restate the global formulation in recursive form. We define the following functional for each time index $t \in \mathcal{T}$:

$$F_t(S_t) = \max_{x_t, y_t}(f_t(x_t, y_t) + F_{t+1}(S_{t+1}(x_t, y_t))) \tag{12}$$

subject to constraints (2) through (10), where $F_T(S_T) \equiv 0$. This problem, which determines F_t, is a subproblem of the DRAP at time t—we refer to it as DRAP$_t$. Note that we write $S_{t+1}(x_t, y_t)$ as a function of the period t decision variables x_t and y_t.

Our original optimization problem is to solve:

$$F_0(S_0) = \max_{x_0, y_0}(f_0(x_0, y_0) + F_1(S_1(x_0, y_0))). \tag{13}$$

Using the terminology suggested by Powell *et al.* [17], we refer to (1) et sequentia as the *simultaneous* formulation, while we refer to (13) as the *recursive* formulation.

It seems natural to suggest that we solve DRAP$_t$ using dynamic programming. Unfortunately, the optimality recursion defining the value function F_t is unsolvable using standard methods from dynamic programming. If we can approximate the value function accurately we may be able to solve the problem to near-optimality. This is the idea underlying the LQN approach, as well as an emerging body of literature oriented toward approximating value functions (see, for example, [5]). In contrast with many dynamic programs, we take advantage of the fact that our value function (12) is (piecewise-linear) concave. Note that we may model the decision variables (x_t, y_t) and the state variable S_t as continuous, knowing that any solution of a subproblem will be integer.

We propose to solve equation (12) by replacing F_{t+1} by a linear approximation, $\hat{F}_{t+1} =$

$\xi_{t+1} \bullet S_{t+1}$. This gives us:

$$\begin{aligned} \tilde{F}_t(S_t) &= \max_{x_t,y_t}(f_t(x_t,y_t) + \hat{F}_{t+1}(S_{t+1}(x_t,y_t))) \\ &= \max_{x_t,y_t}(f_t(x_t,y_t) + \xi_{t+1} \bullet S_{t+1}(x_t,y_t)) \end{aligned} \tag{14}$$

instead of the exact recursion defining F_t. Notice that while $\hat{F}_{t+1}$ is a linear approximation of F_{t+1}, $\tilde{F}_t$ is a *nonlinear* approximation of F_t because of the *max* operator, and the resource constraints (2) - (4). The calculation of the linear approximation is described in section 3.2.

3.1 Aggregation Strategy

Given that we choose to approximate the value function F_t, it is reasonable to consider approximating the system state vector S_t as well. Although $\mathcal{A}$ and $\mathcal{B}$ are generally very large, typically a much smaller subset of $\mathcal{A}$ or $\mathcal{B}$ accounts for most potential resource and task states in a real-world planning problem. To capitalize on this property, we select aggregate attribute spaces $\hat{\mathcal{A}} \subseteq \mathcal{A}$ and $\hat{\mathcal{B}} \subseteq \mathcal{B}$ as approximations to the original state spaces $\mathcal{A}$ and $\mathcal{B}$, respectively. We refer to the subsets of $\hat{\mathcal{A}}$ and $\hat{\mathcal{B}}$ represented at a given node (i,t) as $\hat{\mathcal{A}}_{it}$ and $\hat{\mathcal{B}}_{it}$, respectively. Since our initial data, $\mathcal{R}_0$ and $\mathcal{L}_0$, may include resources and tasks whose initial states are not in $\hat{\mathcal{A}}$ and $\hat{\mathcal{B}}$, we define aggregation operators $\mathcal{Q} : \mathcal{A} \mapsto \hat{\mathcal{A}}$ and $\mathcal{P} : \mathcal{B} \mapsto \hat{\mathcal{B}}$ which 'round' a resource or task state vector to its closest counterpart in $\hat{\mathcal{A}}$ or $\hat{\mathcal{B}}$, respectively.

Aggregation has long been used in building and solving optimization models. In one of the earliest such applications, Balas [2] shows that by aggregating supply nodes, large-scale transportation problems can be solved more quickly, especially when the supply and demand nodes have an uneven spatial distribution. Zipkin extends this work to minimum cost network flow problems with linear [26] and generalized convex [28] objective functions. A key issue in aggregation is the tradeoff between higher solution accuracy (suggesting a *fine* aggregation, with many attributes) and reduced computational effort (suggesting a *coarse* aggregation, with few attributes).

The extent to which an augmented attribute vector will produce higher quality solutions is not easy to predict. Zipkin attempts to quantify this tradeoff between realism and computational effort by establishing several bounds on the error introduced when a solution to the "original network problem" is obtained from the optimal solution of the "aggregate network problem". His results show that aggregation error increases as the similarity between aggregated entities drops (see also

Evans [10]).

The implication for our work is that the ‘rounding error’ introduced by the aggregation operators $\mathcal{P}$ and $\mathcal{Q}$ must not be too great or the solution that we obtain by solving the DRAP on the attribute spaces $\hat{\mathcal{A}}$ and $\hat{\mathcal{B}}$ will not be meaningful in the context of the original attribute spaces $\mathcal{A}$ and $\mathcal{B}$ (which represent the real operational setting of the carrier). We must be careful to cluster similar states when forming $\hat{\mathcal{A}}$ and $\hat{\mathcal{B}}$ from $\mathcal{A}$ and $\mathcal{B}$, respectively. Zipkin generalizes his network results to the case of aggregated variables in generic linear programs in [27]. Using the terminology that he presents therein, we form the aggregate attribute space $\hat{\mathcal{A}}$ from the disaggregate attribute space $\mathcal{A}$ by “partitioning” columns in the constraint matrix corresponding to $\mathcal{A}$, so that several disaggregate states (columns) in $\mathcal{A}$ become a single state (column) in $\hat{\mathcal{A}}$. Zipkin's results imply that any feasible solution to the aggregate problem (using $\hat{\mathcal{A}}$) will be feasible for the original problem (using $\mathcal{A}$) but generally suboptimal.

Wright [25] applies aggregation to stochastic linear programming. In such a stochastic setting, the flow of information across time is conveniently described as a sequence of information σ-fields, $\mathcal{F}_t$, or σ-algebras. This is a very natural framework within which to study information aggregation, since aggregation corresponds exactly to replacing each σ-algebra with a sub σ-algebra. For reasons of measurability, the information structure is assumed to be a *filtration* which is defined as a sequence of σ-algebras, $\{\mathcal{F}_t\}$ such that $\mathcal{F}_t \subseteq \mathcal{F}_{t+1}, \forall t \in \mathcal{T}$. Aggregation is performed by simply restricting decision variables to be *adapted* to a coarser filtration than that of the original problem. For more information on aggregation in dynamic programming, the reader is referred to [1, 3, 4]. A general survey of aggregation in optimization is given in Rogers *et al.* [23].

Most allocation decisions in large-scale systems do not rely upon knowing the exact state of every element in the system at each point in time. Rather, such decisions focus on the short and long-term ‘balance’ of resources and tasks of a small number of states. We can promote this kind of focus in our solution strategy by aggregating the information in S_t into a more approximate form, say as $\hat{S}_t$. More specifically, for each $t \in \mathcal{T}$ we let

$$\hat{S}_t = (w_a, \forall i \in \mathcal{C}, \forall a \in \hat{\mathcal{A}}_{it}; q_b, \forall i \in \mathcal{C}, \forall b \in \hat{\mathcal{B}}_{it}),$$

where

w_a = the number of resources with state $a \in \hat{\mathcal{A}}_{it}$,

and

$$q_b \quad = \quad \text{the number of tasks with state } b \in \hat{\mathcal{B}}_{it}.$$

We suggest that the information in $\hat{S}_t$ will be sufficient to make accurate allocation decisions on a system-wide scale. In effect, we are using different levels of detail to obtain results for different decision makers and different time spans. This is certainly the case in reality, where dispatchers work with a much higher level of detail than do operations planners or strategic analysts.

To account for this change in perspective, we define the following notation for our decision vectors:

x_{ab} = The total number of resources in state $a \in \hat{\mathcal{A}}_{it}$ assigned to tasks in state $b \in \hat{\mathcal{B}}_{it}$ at terminal $i \in \mathcal{C}$ at time $t \in \mathcal{T}$,

y_{aj} = The total number of resources in state $a \in \hat{\mathcal{A}}_{it}$ which begin to move unassigned from terminal $i \in \mathcal{C}$ to terminal $j \in \mathcal{C}$, at time $t \in \mathcal{T}$.

We also need to revise the notation for our cost and reward parameters. Let:

h_{ab} = The cost of assigning a resource in state $a \in \hat{\mathcal{A}}_{it}$ to a task in state $b \in \hat{\mathcal{B}}_{it}$ at terminal $i \in \mathcal{C}$ at time $t \in \mathcal{T}$,

c_{aj} = The cost of a resource in state $a \in \hat{\mathcal{A}}_{it}$ moving unassigned from terminal $i \in \mathcal{C}$ to terminal $j \in \mathcal{C}$, beginning at time $t \in \mathcal{T}$,

r_b = The reward received for covering a task in state $b \in \hat{\mathcal{B}}_{it}$ at terminal $i \in \mathcal{C}$ beginning at time $t \in \mathcal{T}$.

The equation for task coverage becomes

$$z_b = \sum_{a \in \hat{\mathcal{A}}_{it}} x_{ab},$$

for each $i \in \mathcal{C}$, $t \in \mathcal{T}$ and $b \in \hat{\mathcal{B}}_{it}$. Under this notation, we rewrite $f_t(x_t, y_t)$ as

$$f_t(x_t, y_t) = r_t \bullet z_t - h_t \bullet x_t - c_t \bullet y_t,$$

where for instance,

$$x_t = \{\ldots, x_{ab}, \ldots\},\ \forall a \in \hat{\mathcal{A}}_{it}, \forall b \in \hat{\mathcal{B}}_{it}, \forall i \in \mathcal{C}.$$

3.2 Dynamic Programming Formulation

Our solution approach begins with the recursion in (12), which is solved by replacing the value function F_{t+1} with a linear approximation, as described in equation (14). The challenge we face is that replacing F_{t+1} by a linear approximation may lead to instability in the solution process. Specifically, small changes in the linear approximation can produce large changes in the flows. Hence, we add the following upper bound variables for each $i, j \in \mathcal{C}$, time $t \in \mathcal{T}$ and resource state $a \in \hat{\mathcal{A}}_{it}$:

u_{aj} = maximum allowable flow of unassigned resources in state a from node (i, t) to terminal j.

Similarly, we must add upper bound variables to limit the flows of tasks by defining:

v_{bi} = maximum allowable outflow of tasks in state b from node (i, t),

for all $i \in \mathcal{C}$, times $t \in \mathcal{T}$ and task states $b \in \hat{\mathcal{B}}_{it}$.

Through the introduction of the upper bounds on flows, we transform a single-level optimization problem into a bi-level problem where at the top level, we control the upper bounds, and then allow the dynamic programming approximation to produce a "simulation" of flows at the lower level. *This is exactly how this problem is solved in practice.* Most decisions are made at the lower level using relatively simple rules, but these decisions are subject to higher level flow limits that are controlled by centralized planners. As a result, our method effectively formalizes what is used in practice.

We develop our linear approximations of the value function by using subgradients of the approximation $\tilde{F}$ in equation (14). We can characterize the subdifferential of $\hat{F}_t$ with respect to $\hat{S}_t$ by the sets

$\partial_{w_a} \tilde{F}_t$ = the set of all subgradients of $\tilde{F}_t$ with respect to w_a,

and

$$\partial_{q_b} \tilde{F}_t \quad = \quad \text{the set of all subgradients of } \tilde{F}_t \text{ with respect to } q_b.$$

Let

$$\nu_a \in \partial_{w_a} \tilde{F}_t$$

be a subgradient of $\tilde{F}_t$ with respect to the resource inventory w_a and

$$\mu_b \in \partial_{q_b} \tilde{F}_t$$

be a subgradient of $\tilde{F}_t$ with respect to the task inventory q_b. We can then replace F_t by the approximation:

$$\tilde{F}_t(\hat{S}_t, u_t, v_t, \nu_{t+1}, \mu_{t+1}) = \max_{x_t, y_t, \hat{S}_{t+1}} (f_t(x_t, y_t) + \nu_{t+1} \bullet w_{t+1} + \mu_{t+1} \bullet q_{t+1}). \tag{15}$$

In effect, we have broken ξ_{t+1} into two components in equation (15): ν_{t+1} that accounts for resource states and μ_{t+1} that accounts for task states.

In our work, we introduced the approximation:

$$\mu_t = 0 \qquad \forall t$$

This approximation helped to simplify the calculations, with little apparent loss in accuracy (we emphasize apparent, since we have not tested the method without this approximation, and the solution quality still appears to be quite high). It was essential to calculate ν as accurately as possible, since using $\nu = 0$ means that we would never move any drivers empty. By contrast, the only decision regarding a load involved sending with a driver, or doing nothing. Using $\mu_t = 0$ simply means that we are ignoring the impact of covering (or not covering) a load now on the system at the next time period.

Assuming that the optimal solution of (15) is $(x_t(u_t, v_t, \nu_{t+1}), y_t(u_t, v_t, \nu_{t+1}))$, we can write the global objective as

$$H(u, v, \nu) = \sum_{t=0}^{T} f_t(x_t(u_t, v_t, \nu_{t+1}), y_t(u_t, v_t, \nu_{t+1})). \tag{16}$$

We find the best values of the upper bound variables and the resource and task gradients by solving the following control-theoretic problem:

$$\begin{aligned} \max_{u,v,\nu} \quad & H(u,v,\nu) \\ \text{s.t.} \quad & u_t, v_t \geq 0 \quad \forall t \in \mathcal{T} \end{aligned} \tag{17}$$

Equation (17) communicates the property of our solution that we are really trying to find the right values for u, v, ν, rather than all the detailed decisions regarding specific assignments.

The LQN algorithm is summarized in Figure 1. In STEP 2, the forward pass, we successively

---

STEP 1 Initialization:

- Set $\nu = 0$. (All value function approximations are set to zero.)

STEP 2 Forward Pass: for $t = 0, \ldots, T$:

- Solve $\tilde{F}_t(\hat{S}_t, u_t, v_t, \nu_{t+1})$ to get the movements of drivers and loads, and update $\hat{S}_t$ to $\hat{S}_{t+1}$.

STEP 3 Global Update:

- Compute ν_a for all $i \in \mathcal{C}$, $a \in \hat{\mathcal{A}}_{it}$ and $t = T, T-1, \ldots, 0$.
- Update ν and adjust the upper bound variables u and v.

Figure 1: The LQN Algorithm

solve linear approximations of the value function, updating the approximate state vector after each solution is generated. STEP 2 tells us how to allocate our resources, given the available resources at time t, the value function approximation at future time periods, and the upper bounds.

In effect, we are simulating the solution of the problem. We can write the optimality recursion $\tilde{F}_t$ as:

$$\begin{aligned} \tilde{F}_t(\hat{S}_t, u_t, \nu_{t+1}) \quad = \quad & \max_{x_t, y_t, \hat{S}_{t+1}} \sum_{i \in \mathcal{C}} \sum_{b \in \hat{\mathcal{B}}_{it}} r_b z_b - \sum_{i \in \mathcal{C}} \sum_{a \in \hat{\mathcal{A}}_{it}} \sum_{b \in \hat{\mathcal{B}}_{it}} h_{ab} x_{ab} - \sum_{i \in \mathcal{C}} \sum_{a \in \hat{\mathcal{A}}_{it}} \sum_{j \in \mathcal{C}} c_{aj} y_{aj} + \\ & \sum_{i \in \mathcal{C}} \sum_{a' \in \hat{\mathcal{A}}_{it+1}} \nu_{a'} w_{a'}. \end{aligned}$$

Using the separability of the objective function (in addition to obvious technical conditions on the finiteness of the sums), we may pass the max operator inside the first sum over $\mathcal{C}$. $\tilde{F}_t$ then decomposes into a separate component, $\tilde{F}_{it}$, for each $i \in \mathcal{C}$:

$$\tilde{F}_t(\hat{S}_t, u_t, \nu_{t+1}) \quad = \quad \sum_{i \in \mathcal{C}} \Bigg(\max_{x_t, y_t, \hat{S}_{t+1}} \sum_{b \in \hat{\mathcal{B}}_{it}} r_b z_b - \sum_{a \in \hat{\mathcal{A}}_{it}} \sum_{b \in \hat{\mathcal{B}}_{it}} h_{ab} x_{ab} - \sum_{a \in \hat{\mathcal{A}}_{it}} \sum_{j \in \mathcal{C}} c_{aj} y_{aj} +$$

$$\sum_{a' \in \hat{\mathcal{A}}_{it+1}} \nu_{a'} w_{a'} \Bigg),$$

$$= \sum_{i \in \mathcal{C}} \tilde{F}_{it}(\hat{S}_t, u_t, \nu_{t+1}).$$

Consequently, the subproblem for terminal i at time t, DRAP$_{it}$, is to solve:

$$\tilde{F}_{it}(\hat{S}_t, u_t, \nu_{t+1})$$

subject to:

Physical constraints:

$$\sum_{j \in \mathcal{C}} y_{aj} + \sum_{b \in \hat{\mathcal{B}}_{it}} x_{ab} = w_a \quad \forall a \in \hat{\mathcal{A}}_{it}$$

$$\sum_{a \in \hat{\mathcal{A}}_{it}} x_{ab} = z_b \quad \forall b \in \hat{\mathcal{B}}_{it}$$

Control constraints:

$$z_b \leq q_b \quad \forall b \in \hat{\mathcal{B}}_{it}$$

$$y_{aj} \leq u_{aj} \quad \forall a \in \hat{\mathcal{A}}_{it}, \quad \forall j \in \mathcal{C}$$

Nonnegativity:

$$x_{ab} \geq 0 \quad \forall a \in \hat{\mathcal{A}}_{it}, \quad \forall b \in \hat{\mathcal{B}}_{it}$$

$$y_{aj} \geq 0 \quad \forall a \in \hat{\mathcal{A}}_{it}, \quad \forall j \in \mathcal{C}$$

An important assumption underlying this spatial decomposition is that we are only directly assigning resources to tasks already queued at the resource's current location. If a resource needs to reposition itself first before it can cover a task, this must be accomplished using the y vector. The need for such a move should always be captured by the gradient of a resource at the task's current location.

Unlike Powell and Carvalho, who use a simple greedy heuristic to solve the subproblems, we solve the subproblem for each node optimally using the network simplex algorithm. For each subproblem, DRAP$_{it}$, we construct a small network as depicted in Figure 2. In such a network, we build a single node for each resource (with supply w_a) and task (with demand 0) in $\mathcal{R}_{it}$ and $\mathcal{L}_{it}$, respectively. Each arc in the network is labeled $[m : n]$, where m represents the profit of the arc and n represents the upper bound on flow on the arc. Arcs from resource nodes to task nodes

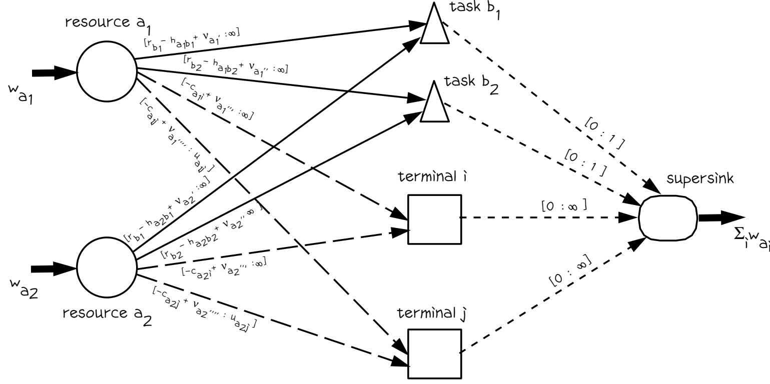

Figure 2: Network Subproblem for DRAP_{it}

represent loaded movements, so the solid arcs in the figure represent the assignment of resources to cover tasks. Arcs from resource nodes to the terminal i node represent resources that are idle. These arcs must be given an upper bound of ∞ to ensure feasibility (i.e., we are assuming infinite waiting room at each terminal). Finally, we build a node for each terminal $j \neq i$ to which at least one resource can be repositioned. An arc from a resource node to a node for some terminal $j \neq i$ represents a repositioning movement, and is given an upper bound of u_{aj}. All flow in the network must terminate in the supersink node, which is given a node demand of $\sum_{a \in \hat{\mathcal{A}}_{it}} w_a$.

A byproduct of solving the network problem is the dual solution. Let π_a be the dual variable associated with resource node a, produced by the network code. As a result of degeneracy, we found that this dual was not sufficiently accurate. Instead, we found the right derivative by finding a flow augmenting path from each node to the supersink (see [19]). This path gives the value of an additional resource of each type. Denote the cost of this path by π_a^+.

Next, in STEP 3, the global update, we compute the subgradient vector ν_t. We use a smoothed estimate for ν, calculated using:

$$\bar{\nu}^{k+1} \quad = \quad (1-\alpha)\bar{\nu}^k + \alpha\pi^+$$

where α is a stepsize less than 1.0. We then used $\bar{\nu}^k$ as the estimate of ν at iteration k. Using the smoothed estimates of the gradients $\bar{\nu}^k$, we would then adjust the flow bounds u_t. This logic is the same as that used in [22], and is summarized in the appendix.

In contrast to Powell and Carvalho's algorithm, we do *not* have a backward pass. We have eliminated the backward pass for a practical reason. A backward pass requires that the subproblem

data structures for all $i \in \mathcal{C}$ and $t \in \mathcal{T}$ be kept in memory simultaneously, so that they can be analyzed after the forward pass. This process worked well on smaller fleet management problems, but proved intractable for a driver scheduling problem of this size.

4 Computational Experiments

The purpose of this section is to study two performance issues of our methodology:

Solution quality: We want to measure the quality of solutions generated by the algorithm. How near to optimality are the solutions that are generated?

Computational tractability of problems of this scale: The ultimate question of this research is whether we can actually optimize, in the sense of providing good quality solutions to these very large scale problems using reasonable memory requirements and CPU times.

We investigate solution quality in three ways. In section 4.1, we consider relatively simpler single and multicommodity flow problems that can be formulated and solved in a reasonable time using linear programming (this yields only a bound, since the solutions are noninteger). These experiments indicate that the method works well on fleet management problems (confirming the work of [22]), but does not test the quality of the linear approximation in the context of a large, sparsely populated state space.

Section 4.2 looks at the real problem, which involves the dynamic management of 6,000 drivers covering approximately 35,000 loads over a four day horizon. Since this problem is too large to be solved by an LP solver (even for the noninteger case), we struggled with the challenge of evaluating the quality of our solution. We devised two methods. The first is a static approximation where all movements are modeled over a static graph. Since this solution ignores timing constraints on loads, as well as resource constraints on drivers, this model should represent an upper bound. However, it is only an approximate upper bound, since all drivers are required to complete full cycles (due to flow conservation), whereas in a dynamic model, there is not a hard constraint that all drivers return to their starting location at the end of the horizon.

Our second method of comparison for the large dataset is the more standard comparison between our solution, and the solution actually achieved in the field based on a historical dataset. This comparison should be the "acid test" and is the manner by which most approximations are

evaluated. The challenge that we faced, which often appears to be ignored in comparisons of this type, is the presence of data errors that makes the solution "in history" infeasible. For example, a load will move from A to B, when in fact the dataset of drivers indicated that there were no drivers at A to move the load. Our model would have to generate an empty to A to accomplish the same move. Other data errors include moves that in history moved by rail, but in the dataset were recorded as truck moves.

Thus, there is no single method of comparison which answers unambiguously the question of the quality of the solution. We feel that the combination of all three sets of experiments represents a compelling case that indicates that our method shows genuine promise as an optimizing tool.

4.1 Solution quality for problems with static attributes

We first measure the quality of our solutions on relatively simpler problems with static attributes that can be formulated as linear programs and solved using a commercial solver. This solution is an upper bound because it is not integer, but it does capture the dynamic nature of the problem, and avoids any aggregation errors. For comparison purposes, we use the problems given in Powell and Carvalho in [22] and [21]. This work is intended to show that our solution method for this problem class produces solutions that are within a few percent of optimality.

Powell and Carvalho study two classes of problems which they call "single commodity" and "multicommodity" problems in reference to their similarity to single and multicommodity network flow problems. In our context, their single commodity problems would be classified as having homogeneous attributes. Their multicommodity problems possess a single static attribute, a resource 'type' (i.e., commodity), that is monitored in addition to space and time. Because this attribute is static, resources remain the same type across all space and time. Task revenues are stationary, and tasks have hard time windows at both their origin and their destination. Repositioning costs for resources vary by resource type. Furthermore, certain types of tasks may only be covered by certain types of resources.

There are two important differences between the original LQN algorithm in [22] and our implementation. First, we solve a network subproblem and use dual information to obtain gradients. Powell and Carvalho used a heuristic sort and performed numerical derivatives (adding a resource and then resolving the subproblem, heuristically). We would expect that the explicit formulation

of the subproblem would produce higher quality results. Second, the smaller state space in [22] and [21] allowed this earlier work to explicitly generate a bucket for each resource state, which would then prompt the algorithm to calculate derivatives for each type of resource. As a result, if we assign a resource with attribute a_r to a task with attribute b_ℓ which produces a resource with modified attribute vector a_r' (specifically, a different location), then we will have an estimate of the value of this modified resource $\nu_{a'}$. In our larger problem, we had to generate resource buckets on the fly. As a result, if we wanted to assign a resource to a task which creates a modified resource a_r' which we have never encountered before, then we had to initially use a "best estimate" of the gradient $\nu_{a'}$. Thus, there is no guarantee how our implementation of this algorithm would work, even on the same problem class as that posed in [22] and [21].

Problem sets from [22] are labeled "SC" while those from [21] are labeled "MC". We present a summary of the test problems in Table 1. The numbers in the second column are the problem set numbers as used in [22] and [21]. The length of a time period was 4 hours in each case.

Problem	Problem Set	Task Types	Resource Types	Tasks	Resources	T
SC1	8	1	1	2000	200	40
SC2	16	1	1	4000	400	70
SC3	17	1	1	6000	400	100
MC1	6	5	5	2000	510	40
MC2	9	5	5	2000	420	40
MC3	16	9	9	2000	700	39
MC4	24	10	10	6000	2000	40
MC5	25	20	20	6000	2100	40

Table 1: Summary of Powell and Carvalho's Data Sets from [22, 21] used in the evaluation of solution quality

As for any heuristic, our algorithm is sensitive to parameter values. We have chosen to use parameter settings which were shown to work well by Powell and Carvalho, as given in Table 8 in the appendix.

Two strategies are used for updating the resource empty bounds, u. The first strategy, strategy S1, performs both subgradient search (SS) and coordinate search (CS) on u using the gradient of the control function H (defined in (16)) with respect to u. As suggested by Powell and Carvalho [22], we use SS for the first $K \in \mathbb{N}$ iterations, followed by CS for any remaining iterations. The second strategy, strategy S2, makes u identically 1 and does not modify its value during the run. Strategy

S2 is based on the observation that in many DRAP's, the majority of the tasks can be moved without the need for repositioning moves, hence allowing at most one between each OD pair per time period should be sufficient.

Tests were performed on an SGI Indigo 2 Impact with a 250 MHz R4400 processor and 320 MB RAM. We present the results of our test runs in Table 2. The figure "OPT Ratio" refers to the ratio (in percent) of the best objective value we obtained to the known LP bound computed by Powell and Carvalho in [22, 21]. The "Gap" figure refers to the difference, in percentage points, between Powell and Carvalho's best solution (their gradient method) and our OPT Ratio:

$$\text{Gap} = OPTRatio_{P\&C} - OPTRatio_{P\&S}.$$

Results are presented for both bounds adjustment strategies, S1 and S2. We believe that gaps of

Problem	Strategy S1			Strategy S2		
	OPT Ratio	*Gap*	*CPU Time (s)*	*OPT Ratio*	*Gap*	*CPU Time (s)*
SC1	92.4	0.1	536	92.4	0.1	784
SC2	94.1	1.4	862	96.5	-1.0	1346
SC3	94.8	-1.4	1352	97.1	-3.7	1863
MC1	97.1	-0.1	1357	96.2	0.8	2837
MC2	95.8	0.9	2583	95.3	1.4	2891
MC3	96.1	-0.1	1967	95.5	0.5	4991
MC4*	97.6	-1.4	3125	97.9	-1.7	7368
MC5*	97.7	-0.8	6528	97.5	-0.6	18442

*Powell and Carvalho give results for this problem using the LAMA algorithm, which violates nonanticipativity.

Table 2: Test results for evaluation of solution quality, where negative gaps indicate that our algorithm produced a better result than Powell and Carvalho

0.5 % and less between two results can be explained by noise (this is the approximate magnitude of the fluctuation in the OPT ratio from one iteration to the next), and such situations will be called a 'tie'.

Strategy S1 outperformed Powell and Carvalho on three of the eight problems tested (SC3, MC4 and MC5), and tied them on another three (SC1, MC1 and MC3). It is not clear from these results whether there is an advantage to solving each subproblem by the network simplex as opposed to Powell and Carvalho's greedy sort.

Surprisingly, strategy S2 seems to have performed just as well as the more sophisticated strategy S1. Strategy S2 bettered Powell and Carvalho's results on four of eight problems (SC2, SC3, MC4

and MC5) and tied them on another two problems (SC1 and MC3). Strategy S2 is clearly superior to strategy S1 on the single commodity problems but seems worse on the multicommodity problems. The reasons for this discrepancy are not immediately obvious.

In conclusion, we can say that both strategies S1 and S2 perform similarly to Powell and Carvalho's gradient method. Because strategy S1 is faster on multicommodity problems, like the problems with multiple dynamic attributes that we study in the next section, we choose it as our preferred method. CPU times are given with our results to indicate that our algorithm's speed is competitive with that of Powell and Carvalho in terms of the amount of time required to solve each problem. However, Powell and Carvalho's method uses less memory since they have hard-coded their software for a fixed number of static attributes.

4.2 Solution quality for problems with dynamic attributes

We next wish to test our method on an ultra-large scale problem. We use the data of a large motor carrier which must manage the flows of 6,000 drivers between 550 terminals, observing union workrules and federal constraints on driving time. This problem can be represented as a crew scheduling problem and solved using column generation methods, but at this point the number of resources (drivers) and other problem characteristics (such as wide time windows for moving loads) put this problem completely outside the scope of the largest problems that have been solved to date with this technology.

For our study, we use four attributes to describe a driver. The resource attribute vector a, then, is shown in table 3. The size of this system can be measured in two ways: the size of the driver

Index i	Attribute $a_r(i)$	Number of Possible Values
1	current location	550
2	domicile	60
3	clock	6
4	sleeper status	3
5	layovers out-of-home	2

Table 3: Base Driver Attributes

state space, and the size of the system state space. For this attribute vector, there are a possible $550 \times 60 \times 6 \times 3 \times 2 = 1,188,000$ kinds of drivers at a given point in time. Thus, we would say that the size of our aggregate attribute space is given by $|\hat{\mathcal{A}}_t| = 1,188,000$.

If we had only one driver, the number of states in our system would be $|\hat{\mathcal{A}}_t|$. When we have n drivers, each of which can take on one of $|\hat{\mathcal{A}}_t|$ states, then the number of states in our *system* is much larger. It can be shown that for n resources, each of which can occupy $|\hat{\mathcal{A}}_t|$ states, the number of states in the system state space for any time $t \in \mathcal{T}$ is given by

$$\binom{n + |\hat{\mathcal{A}}_t| - 1}{|\hat{\mathcal{A}}_t| - 1}.$$

So for $n = 6000$ and $|\hat{\mathcal{A}}_t| = 1,188,000$, we can use Stirling's approximation to estimate the number of states in our system as approximately 10^{16390}. Furthermore, this state space is based on an *aggregation* of the original problem. Clearly, this rules out any direct application of dynamic programming methods.

We can easily apply our algorithm to a problem of this size. The challenge, then, is estimating the quality of the solution produced by the approximation. We are unable to formulate a tractable linear program that appropriately captures the characteristics of the problem. Even if we approximate the problem as a dynamic, multicommodity network flow problem (using driver domicile as the commodity attribute) the resulting linear program is too large for current technology. For example, experiments in [21] show run times of over 15 hours using CPLEX for problems with only 10 commodities and 2000 tasks. Thus, we cannot even get a solution to the LP relaxation to use as a bound.

As an alternative, we developed a static formulation of the problem whereby all the flows were aggregated into a single commodity. This optimization problem, which is trivial to solve (it is a pure network) effectively estimates empty repositioning costs, but otherwise ignores all timing constraints and union rules. We view the static approximation as an upper bound, since we have dropped every constraint except flow conservation. However, it is not a strict upper bound. The reason is that the static model enforces flow conservation aggregated over all points in time, while the dynamic model only enforces flow conservation at individual instants in time. Just the same, we feel our static model provides a useful yardstick of solution quality.

We performed two sets of runs of our algorithm against the static approximation. In the first set, we model all the attributes of a driver, but ignore all constraints relating to workrules. The goal is to create a problem that reasonably approximates the assumptions in our static approximation. Our intent is to show that when both models use basically the same constraints with respect to work rules, our algorithm produces high quality solutions. The problem with running such

an unconstrained problem is that the effective size of the state space will be much larger than would arise in practice. The second set of runs enforces work rules, producing a lower objective function but much faster run times. We believe that the drop in the objective function can be attributed almost entirely to the enforcement of additional constraints, although this cannot be proved. However, the faster run times will demonstrate the computational tractability of our solution algorithm.

In our first set of runs, we ignore work rules but still capture the driver attributes. Since there are no constraints related to driver attributes, it is interesting to see what effect the inclusion of driver attributes has on solution quality. For this analysis, we performed a series of runs using three variations of the attribute vector specified in Table 3: driver location only,driver location plus driver hours ($a_r(1)$), and driver location plus driver domicile ($a_r(2)$). Without the constraints imposed by work rules, the size of the effective state space grows rapidly. As a result, we were unable to perform a run which combined driver hours and domiciles (by contrast, we had no difficulty including driver hours and domiciles when work rules were present).

We stop the algorithm whenever five iterations have passed without an improvement in the objective function value or 50 iterations, whichever comes first. Furthermore, we use the parameter settings given in Table 8 in the appendix, except that under strategy $S1$ we perform only 20 iterations of subgradient search (K=20). In this set of runs, we have selected only the most basic rules to govern the feasibility of movements. Any task may be moved by any resource after the time of first availability of the task (heuristic pruning rules were used to reduce the number of possible empty movements). The model is allowed to reposition a driver empty from one location to another.

Figure 3 shows the progression of the objective function using strategy S2 over the first 50 iterations. Notice that the algorithm continues to improve the objective value quite rapidly for the first 10–20 iterations. After this point, the algorithm essentially stalls and undertakes a great deal of computational effort to produce very little improvement in the remaining iterations (there is little room for improvement though, with the high OPT Ratios).

The results of our first run of experiments are presented in Table 4, where the “OPT Ratio” is the ratio of the best objective value achieved to the optimal solution of the static approximation (which represents an approximate upper bound). The “CPU Time” column gives first the total CPU time for each run followed by the CPU time per iteration (since some runs may have been stopped before

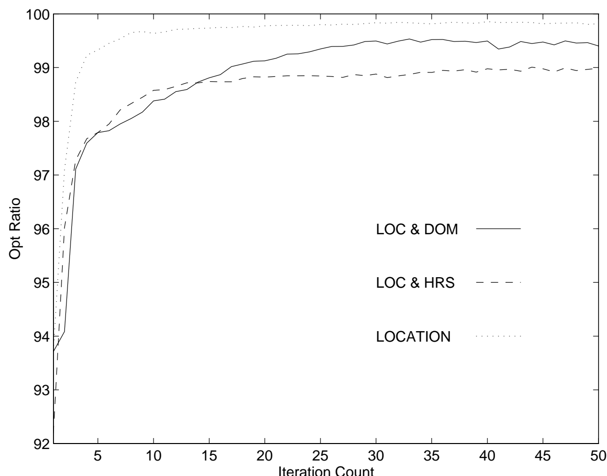

Figure 3: Opt Ratio vs Iteration Count

50 iterations). The results of LOCATION, where we use only the driver's current location, indicate

Problem	Attributes	Strategy S1		Strategy S2	
		OPT Ratio	*CPU Time*	*OPT Ratio*	*CPU Time*
LOCATION	—	99.5	1438/50	99.9	2429/65
LOC & HRS	$a_r(2)$	96.1	914/91	99.0	6501/144
LOC & DOM	$a_r(1)$	99.0	23068/461	99.5	79311/2266

Table 4: Effect of increasing dimensions of driver attribute vector on solution quality and execution times (total time/time per iteration given in seconds)

that our bound on the optimal objective is *very* tight for this relatively unconstrained version of the DRAP. The bound remains tight even for LOC & HRS where we track and restrict resource road and duty hours. We believe these results indicate that we are producing very high quality solutions. At the same time, the runs indicate that the needless addition of resource attributes (that is, attributes that are not reflected in the constraint set) can increase run times dramatically.

We next performed a run with the full vector of attributes, and a complete set of work rules, using the same parameters as for Table 4. However, all runs were left to complete exactly 50 iterations, regardless of whether the solutions continued to improve or not. A variety of performance measures on the results are given in Tables 5 through 7. These runs were performed on an SGI Challenge 194 MHz R10000 system with 1152 MB of RAM.

The most important question at this stage concerns the quality of our solution. Table 5 shows that our "Opt ratio" gap, where we compare our results to the static approximation, are slightly

Data Set	OPT Ratio	CPU Time	$\hat{\mathcal{A}}$ *Density (%)*
MON	96.5	3838/77	0.3
WED	96.9	3758/75	0.3
FRI	98.2	3197/64	0.2

Table 5: Percent of Upper Bound Achieved (Total/Per Iteration CPU Times in Seconds)

worse than with the unconstrained problem. At least some of this reduction is due to the imposition of work rules, that increases empties and reduces the capacity of the system to cover loads (our static bound relaxes work rule constraints, hence we expect to see a larger gap). Table 4 suggests that we can obtain results for this problem class that are within one percent of optimality. Our previous results on single and multicommodity problems indicate that our results are within five percent (of a linear relaxation). We can infer that our solution quality on the ultra-large problem is in the same range. However, at this point, further refinement of our performance bound will require additional research. The column "$\hat{\mathcal{A}}$ *Density (%)*" gives the attribute space density (in percent) computed by the dividing the observed number of states by the theoretically possible number of states. These results show that the size of the state space, and the execution times, drop dramatically. This occurs primarily as a result of the rule that restricts the ability of drivers from a particular domicile to move anywhere around the country.

A different measure of solution quality is obtained by comparing our results to historical performance. This provides an indication of the quality of our solution, although it, like the static bound, has both strengths and weaknesses. We took three datasets, each covering four days, one starting on Monday (MON), the second starting on Wednesday (WED) and the last starting on Friday (FRI). Table 6 shows the degree to which the model actually moved all the loads in the system, with numbers ranging from 96 to 97 percent. We were unable to move 100 percent of the loads within the allowed time parameters in part because of errors in the dataset (in some cases, the carrier would move a load, even though the dataset did not show the presence of a driver to move a load). In these cases, our model was forced to run a driver empty, if possible, to move a load. A second source of discrepancy arises because of model truncation problems. If the carrier moved a load at the end of the four days, and the movement required sending a driver empty, it is possible that at the end of the horizon that the model would decide that the move was uneconomical.

From history, we only have statistics on total empty miles. Table 7 shows the performance of the model relative to history. The reduction in empties is substantial. Some of this reduction may

Data set	History	Model
MON	100.0	96.2
WED	100.0	96.6
FRI	100.0	97.9

Table 6: Percent of Freight Bills Covered

be attributed to our inability to cover all the loads, but the results are still highly favorable. We note that our greatest reduction in empty miles occurs in the FRI dataset, where we were able to cover almost 98 percent of all the loads.

Data set	History	Model
MON	7.6	4.3
WED	6.3	4.3
FRI	9.2	5.1

Table 7: Percent of Total Miles Driven Empty

It is not possible, for a problem of this size, to obtain a provably optimal solution in a reasonable time. Instead, we have used several measures to indicate that our approach is providing high quality solutions. First, we have applied the method to problems that are sufficiently small that can be solved using a commercial LP solver. Second, we used a static approximation, which proved to be reasonably tight, as a measure of solution quality for the larger dataset. Finally, we compared our results to historical performance. None of these methods provides a perfect measure of the quality of our solution, but together offer convincing evidence that the technique is providing high quality solutions to this ultra-large problem class.

5 Conclusions

In this paper we have developed a new approach for solving ultra large-scale dynamic resource allocation problems. The approach is very flexible, and allows practitioners to capture levels of real-world detail that were previously only attainable in non-optimizing simulations. Real-world constraints and issues are easily captured in a simple way, without complex mathematics. The LQN algorithm that we have developed has been shown to perform as well as other algorithms recently proposed in the literature on a set of standard test problems. Most importantly, and unlike standard linear programming packages (even when we do not require integer solutions), our method is scalable to problems that require optimizing thousands of resources over tens of

thousands of tasks, over long planning horizons. The algorithm converges relatively quickly to a solution that appears to be of high quality, based on indirect evidence. All this is achieved with reasonable CPU and memory requirements, and can be run on low cost workstations. It has been shown that the aggregation of similar resources can produce significant savings in computation times and memory requirements without unreasonable losses in solution quality. Without question, our approach is ideally suited for producing the real-time capacity management decisions that are required by nearly all of today's larger, progressive transportation firms.

References

[1] S. Axsäter. State aggregation in dynamic programming—an application to scheduling of independent jobs on parallel processors. *Operations Research Letters*, 2(4):171–176, November 1983.

[2] E. Balas. Solution of large-scale transportation problems through aggregation. *Operations Research*, 13:82–93, 1965.

[3] J. C. Bean, J. R. Birge, and R. L. Smith. Aggregation in dynamic programming. *Operations Research*, 35(2):215–220, March-April 1987.

[4] D. P. Bertsekas and D. A. Castanõn. Adaptive aggregation methods for infinite horizon dynamic programming. *IEEE Transactions on Automatic Control*, 34(6):589–598, June 1989.

[5] D.P. Bertsekas and J.N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, Belmont, MA, 1996.

[6] R.K. Cheung and W.B. Powell. An algorithm for multistage dynamic networks with random arc capacities, with an application to dynamic fleet management. *Operations Research*, 44(6):951–963, 1996.

[7] T. G. Crainic and J.-M. Rousseau. The column generation principle and the airline crew scheduling. *INFOR*, 25(2):136–151, 1987.

[8] J. Desrosiers, M. Solomon, and F. Soumis. Time constrained routing and scheduling. In C. Monma, T. Magnanti, and M. Ball, editors, Handbook in Operations Research and Management Science*, Volume on* Networks. North Holland, 1995.

[9] J. Desrosiers, F. Soumis, and M. Desrochers. Routing with time windows by column generation. *Networks*, 14:545–565, 1984.

[10] J. R. Evans. A network decomposition/aggregation procedure for a class of multicommodity transportation problems. *Networks*, 13:197–205, 1983.

[11] L.F. Frantzeskakis and W.B. Powell. A successive linear approximation procedure for stochastic dynamic vehicle allocation problems. *Transportation Science*, 24(1):40–57, 1990.

[12] W.C. Jordan and M.A. Turnquist. A stochastic dynamic network model for railroad car distribution. *Transportation Science*, 17:123–145, 1983.

[13] S. Lavoie, M. Minoux, and E. Odier. A new approach for crew pairing problems by column generation with an application to air transport. *European Journal of Operational Research*, 35:45–58, 1988.

[14] W. B. Powell. A review of sensitivity results for linear networks and a new approximation to reduce the effects of degeneracy. *Transportation Science*, 23(4):231–243, 1989.

[15] W. B. Powell, T. A. Carvalho, G. A. Godfrey, and H. P. Simão. Dynamic fleet management as a logistics queueing network. *Annals of Operations Research*, 61:165–188, 1995.

[16] W. B. Powell, P. Jaillet, and A. Odoni. Stochastic and dynamic networks and routing. In C. Monma, T. Magnanti, and M. Ball, editors, Handbook in Operations Research and Management Science*, Volume on* Networks, pages 141–295. North Holland, 1995.

[17] W. B. Powell, P. Jaillet, and A. Odoni. Stochastic and dynamic networks and routing. In *Handbook in Operations Research and Management Science*. North Holland, 1995.

[18] W.B. Powell. A comparative review of alternative algorithms for the dynamic vehicle allocation problem. In B. Golden and A. Assad, editors, *Vehicle Routing: Methods and Studies*, pages 249–292, New York, 1988. North Holland.

[19] W.B. Powell. A review of sensitivity results for linear networks and a new approximation to reduce the effects of degeneracy. *Transportation Science*, 23(4):231–243, 1989.

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

[21] W.B. Powell and T. A. Carvalho. Dynamic control of multicommodity fleet management problems. *European Journal of Operations Research*, 98, 1997.

[22] W.B. Powell and T. A. Carvalho. Dynamic control of logistics queueing network for large-scale fleet management. *Transportation Science*, 32(2):90–109, 1998.

[23] D. F. Rogers, R. D. Plante, R. T. Wong, and J. R. Evans. Aggregation and disaggregation techniques and methodology in optimization. *Operations Research*, 39:553–582, 1991.

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

[25] S. E. Wright. Primal-dual aggregation and disaggregation for stochastic linear programs. *Annals of Operations Research*, 19:893–907, 1994.

[26] P. H. Zipkin. Bounds for aggregating nodes in network problems. *Mathematical Programming*, 19:155–177, 1980.

[27] P. H. Zipkin. Bounds on the effect of aggregating variables in linear programs. *Operations Research*, 28(2):403–418, March-April 1980.

[28] P. H. Zipkin. Aggregation and disaggregation in convex network problems. *Networks*, 12:101–117, 1982.

Appendix A LQN in Detail

In this section we present additional details on the workings of our LQN approach. The reader should consult Powell and Carvalho [22, 21] for a more thorough discussion of the technical details presented. We begin in Section A.1 with an explanation of the forward pass. Following this, in Section A.2, we describe the global update procedure.

A.1 The LQN Forward Pass

The principal purpose of the forward pass is to solve $\tilde{F}_t(\hat{S}_t, u_t, \mu_{t+1})$ for $t \in \mathcal{T}$. It turns out that $\tilde{F}_t$ decomposes into independent components $\tilde{F}_{it}$ for each terminal $i \in \mathcal{C}$. Since the constraints (2) through (10) are also separable by terminal, it follows that instead of solving DRAP$_t$ directly, we can solve a sequence of subproblems of DRAP$_t$, say DRAP$_{it}$ for each $i \in \mathcal{C}$, independently of DRAP$_{jt}$ for all $j \neq i$. To show this we define the mappings $G_x : \hat{\mathcal{A}} \times \hat{\mathcal{B}} \mapsto \hat{\mathcal{A}}$ with

$$G_x(a,b) \quad = \quad \text{the future state of a resource in state } a \in \hat{\mathcal{A}}_{it} \text{ upon completion of an assignment to a task in state } b \in \hat{B}_{it} \text{ at node } (i,t),$$

and $G_y : \hat{\mathcal{A}} \times \mathcal{C} \mapsto \hat{\mathcal{A}}$ with

$$G_y(a,j) \quad = \quad \text{the future state of a resource in state } a \in \hat{\mathcal{A}}_{it} \text{ upon completion of an unassigned move from node } (i,t) \text{ to terminal } j,$$

and $G_z : \hat{\mathcal{A}} \times \hat{\mathcal{B}} \mapsto \hat{\mathcal{B}}$ with

$$G_z(a,b) \quad = \quad \text{the future state of a task in state } b \in \hat{\mathcal{B}}_{it} \text{ at node } (i,t) \text{ after its coverage by a resource in state } a \in \hat{\mathcal{A}}_{it} \text{ is completed,}$$

$$G_z(\emptyset,b) \quad = \quad \text{the future state of a task in state } b \in \hat{\mathcal{B}}_{it} \text{ at node } (i,t) \text{, given that it goes uncovered during time period } t.$$

We need these mappings to help us compute w_{t+1} and q_{t+1}. Writing the set indicator function as

$$1_{\mathcal{D}}[x] = \begin{cases} 1 & \text{if } x \in \mathcal{D}, \\ 0 & \text{otherwise} \end{cases}$$

then for $a' \in \hat{\mathcal{A}}_{it+1}$,

$$w_{a'} \quad = \quad \sum_{j \in \mathcal{C}} \sum_{a \in \hat{\mathcal{A}}_{jt-\tau_{ji}+1}} \left(y_{ai} 1_{\{a'\}}[G_y(a,i)] + \sum_{b \in \hat{\mathcal{B}}_{jt-\tau_{ji}+1}} x_{ab} 1_{\{a'\}}[G_x(a,b)] \right). \tag{18}$$

Similarly, we update the task balances by

$$q_{b'} = \sum_{j\in\mathcal{C}} \sum_{b\in\hat{\mathcal{B}}_{jt-\tau_{ji}+1}} \left((q_b - z_b) 1_{\{b'\}}[G_z(\emptyset, b)] + \sum_{a\in\hat{\mathcal{A}}_{jt-\tau_{ji}+1}} x_{ab} 1_{\{b'\}}[G_z(a,b)] \right), \tag{19}$$

for $b' \in \hat{\mathcal{B}}_{it+1}$.

Recall the optimality recursion for time $t \in \mathcal{T}$:

$$\begin{aligned}\tilde{F}_t(\hat{S}_t, u_t, v_t) &= \max_{x_t, y_t, \hat{S}_{t+1}} \sum_{i\in\mathcal{C}} \sum_{b\in\hat{\mathcal{B}}_{it}} r_b z_b - \sum_{i\in\mathcal{C}} \sum_{a\in\hat{\mathcal{A}}_{it}} \sum_{b\in\hat{\mathcal{B}}_{it}} h_{ab} x_{ab} - \sum_{i\in\mathcal{C}} \sum_{a\in\hat{\mathcal{A}}_{it}} \sum_{j\in\mathcal{C}} c_{aj} y_{aj} + \\ &\quad \sum_{i\in\mathcal{C}} \sum_{a'\in\hat{\mathcal{A}}_{it+1}} \nu_{a'} w_{a'} + \sum_{i\in\mathcal{C}} \sum_{b'\in\hat{\mathcal{B}}_{it+1}} \mu_{b'} q_{b'}.\end{aligned}$$

Using the expressions for w and q given in (18) and (19), respectively, we may rewrite the recursion as

$$\begin{aligned}\tilde{F}_t(\hat{S}_t, u_t, v_t) &= \max_{x_t, y_t, \hat{S}_{t+1}} \sum_{i\in\mathcal{C}} \left(\sum_{b\in\hat{\mathcal{B}}_{it}} r_b z_b - \sum_{a\in\hat{\mathcal{A}}_{it}} \sum_{b\in\hat{\mathcal{B}}_{it}} h_{ab} x_{ab} - \sum_{a\in\hat{\mathcal{A}}_{it}} \sum_{j\in\mathcal{C}} c_{aj} y_{aj} + \right. \\ &\quad \sum_{a'\in\hat{\mathcal{A}}_{it+1}} \sum_{j\in\mathcal{C}} \sum_{a\in\hat{\mathcal{A}}_{jt-\tau_{ji}+1}} (y_{ai} 1_{\{a'\}}[G_y(a,i)] + \sum_{b\in\hat{\mathcal{B}}_{jt-\tau_{ji}+1}} x_{ab} 1_{\{a'\}}[G_x(a,b)]) \nu_{a'} + \\ &\quad \left. \sum_{b'\in\hat{\mathcal{B}}_{it+1}} \sum_{j\in\mathcal{C}} \sum_{b\in\hat{\mathcal{B}}_{jt-\tau_{ji}+1}} ((q_b - z_b) 1_{\{b'\}}[G_z(\emptyset, b)] + \sum_{a\in\hat{\mathcal{A}}_{jt-\tau_{ji}+1}} x_{ab} 1_{\{b'\}}[G_z(a,b)]) \mu_{b'} \right).\end{aligned}$$

Assuming that all summands are finite, and that the index sets are finite, we may pass the max operator inside the first sum over $i \in \mathcal{C}$. It is then obvious that $\tilde{F}_t$ decomposes into a separate component, $\tilde{F}_{it}$ for each terminal $i \in \mathcal{C}$:

$$\begin{aligned}\tilde{F}_t(\hat{S}_t, u_t, v_t) &= \sum_{i\in\mathcal{C}} \left(\max_{x_t, y_t, \hat{S}_{t+1}} \sum_{b\in\hat{\mathcal{B}}_{it}} r_b z_b - \sum_{a\in\hat{\mathcal{A}}_{it}} \sum_{b\in\hat{\mathcal{B}}_{it}} h_{ab} x_{ab} - \sum_{a\in\hat{\mathcal{A}}_{it}} \sum_{j\in\mathcal{C}} c_{aj} y_{aj} + \right. \\ &\quad \sum_{a'\in\hat{\mathcal{A}}_{it+1}} \sum_{j\in\mathcal{C}} \sum_{a\in\hat{\mathcal{A}}_{jt-\tau_{ji}+1}} (y_{ai} 1_{\{a'\}}[G_y(a,i)] + \sum_{b\in\hat{\mathcal{B}}_{jt-\tau_{ji}+1}} x_{ab} 1_{\{a'\}}[G_x(a,b)]) \nu_{a'} + \\ &\quad \left. \sum_{b'\in\hat{\mathcal{B}}_{it+1}} \sum_{j\in\mathcal{C}} \sum_{b\in\hat{\mathcal{B}}_{jt-\tau_{ji}+1}} ((q_b - z_b) 1_{\{b'\}}[G_z(\emptyset, b)] + \sum_{a\in\hat{\mathcal{A}}_{jt-\tau_{ji}+1}} x_{ab} 1_{\{b'\}}[G_z(a,b)]) \mu_{b'} \right), \\ &= \sum_{i\in\mathcal{C}} \tilde{F}_{it}(\hat{S}_t, u_t, v_t),\end{aligned}$$

where

$$\begin{aligned}\tilde{F}_{it}(\hat{S}_t, u_t, v_t) &= \max_{x_t, y_t, \hat{S}_{t+1}} \sum_{b \in \hat{\mathcal{B}}_{it}} r_b z_b - \sum_{a \in \hat{\mathcal{A}}_{it}} \sum_{b \in \hat{\mathcal{B}}_{it}} h_{ab} x_{ab} - \sum_{a \in \hat{\mathcal{A}}_{it}} \sum_{j \in \mathcal{C}} c_{aj} y_{aj} + \\ & \sum_{a' \in \hat{\mathcal{A}}_{it+1}} \sum_{j \in \mathcal{C}} \sum_{a \in \hat{\mathcal{A}}_{jt-\tau_{ji}+1}} (y_{ai} 1_{\{a'\}}[G_y(a,i)] + \sum_{b \in \hat{\mathcal{B}}_{jt-\tau_{ji}+1}} x_{ab} 1_{\{a'\}}[G_x(a,b)]) \nu_{a'} + \\ & \sum_{b' \in \hat{\mathcal{B}}_{it+1}} \sum_{j \in \mathcal{C}} \sum_{b \in \hat{\mathcal{B}}_{jt-\tau_{ji}+1}} ((q_b - z_b) 1_{\{b'\}}[G_z(\emptyset, b)] + \sum_{a \in \hat{\mathcal{A}}_{jt-\tau_{ji}+1}} x_{ab} 1_{\{b'\}}[G_z(a,b)]) \mu_{b'}.\end{aligned}$$

Or, in more compact form,

$$\begin{aligned}\tilde{F}_{it}(\hat{S}_t, u_t, v_t) &= \max_{x_t, y_t, \hat{S}_{t+1}} \sum_{b \in \hat{\mathcal{B}}_{it}} r_b z_b - \sum_{a \in \hat{\mathcal{A}}_{it}} \sum_{b \in \hat{\mathcal{B}}_{it}} h_{ab} x_{ab} - \sum_{a \in \hat{\mathcal{A}}_{it}} \sum_{j \in \mathcal{C}} c_{aj} y_{aj} + \\ & \sum_{a' \in \hat{\mathcal{A}}_{it+1}} \nu_{a'} w_{a'} + \sum_{b' \in \hat{\mathcal{B}}_{it+1}} \mu_{b'} q_{b'}.\end{aligned}$$

The independence of DRAP_{it} from DRAP_{jt}, for all $j \neq i$ suggests that we solve these terminal-based subproblems in parallel. Hence, by solving a collection of subproblems of DRAP_t, each with fewer constraints than DRAP_t, the net computational effort may be reduced.

As we explain in the next section, we obtain the subproblem duals ν_t from network duals computed on the subproblem network in Figure 2. In practice, we use the right-hand gradient ν_t^+ in place of ν_t. Since we use both ν_t^+ and ν_t^- for updating the empty upper bounds u, we do not bother to compute ν_t as well. Nevertheless, we know that

$$\nu_t^- \geq \nu_t \geq \nu_t^+$$

holds $\forall t \in \mathcal{T}$ since $\hat{F}_t$ is a concave function. Hence we feel justified in using ν_t^+ as a conservative estimate of ν_t. Since no subproblems are solved for $t \geq T$, we assume that $\nu_t^- = \nu_t = \nu_t^+ = 0$ for all $t \geq T$. One may substitute ν^+ for ν throughout what follows. When it is important to distinguish between ν^+ and ν^-, we will do so.

A summary of the sequence of steps needed in the forward pass is given in Figure 4. Evidently most of the computational effort will be devoted to STEP 1, so it is important to make this step as efficient as possible.

STEP 0 Initialization:

- Set $t = 0$ and get $\mathcal{R}_0$ and $\mathcal{L}_0$ as input data. Compute w and q from $\mathcal{R}_0$ and $\mathcal{L}_0$, respectively.

STEP 1 Subproblem Solution:

- Solve DRAP$_{it}$ for each $i \in \mathcal{C}$.

STEP 2 System Update:

- Update $\hat{S}_t$ to give $\hat{S}_{t+1}$.

STEP 3 Termination Check:

- If $t < T$, let $t = t + 1$ and GOTO STEP 1. ELSE, STOP.

Figure 4: The LQN Forward Pass

A.2 The LQN Global Update

The global update is responsible for computing and then smoothing the gradient vector ν, as well as updating the upper bound limits on the flow of unassigned resources.

Gradient Computation: We begin each global update by computing the gradient vectors ν_t for each forward pass. We compute these gradients from the node duals of each resource node in a terminal's subproblem at time $t \in \mathcal{T}$. We obtain the right-hand gradient ν_a^+ as the value (right-hand dual) of a flow-augmenting path from resource a's node to the supersink. The left-hand gradient ν_a^- is obtained as the value (left-hand dual) of a flow-decreasing path from resource a's node to the supersink. Both gradients can be calculated very quickly, using algorithms outlined in [14]. Computing duals in this fashion is much faster than obtaining numerical gradients by re-solving the subproblem with a perturbed state vector, as is required in a greedy solution. We avoid using straight network simplex duals since they include the effects of the Big M start used to obtain an initial basic feasible solution for each subproblem.

Gradient Smoothing: Given that we use the duals ν in the forward pass, we unavoidably introduce a degree of randomness into the estimate of the gradient of $H(u, \nu)$ as we revise the duals from one iteration to the next (regardless of whether the problem is deterministic or stochastic; we are not introducing any random variables here). To correct for this effect we propose to use a smoothing scheme similar to that used in stochastic linearization algorithms. Let ν^n be the duals

calculated in the nth iteration. We propose to use

$$\bar{\nu}^n = \gamma^n \nu^n + (1-\gamma^n)\bar{\nu}^{n-1}$$

instead of ν^n, where $\gamma^n \in (0,1]$ is the smoothing coefficient (to be determined experimentally). We use the strategy $\gamma^n = \gamma^0/n$, although cutting γ^n in half when no improvement is found after a certain number of iterations is another possibility.

Updating the upper bound vector: Upon completing the forward pass, we are still faced with the problem of updating the upper bound vector. We would like to do this in a manner that improves the function $H(u,\nu)$. For the resource upper bound vector, the gradient

$$\eta_{aj} = \frac{\partial H}{\partial u_{aj}}$$

indicates how a change in u_{aj} would affect H. Powell and Carvalho [22, 21] derive the following expression for computing η^+_{aj}, where $a' = G_y(a,j)$:

$$\eta^+_{aj} = \begin{cases} -c_{aj} + \bar{\nu}^+_{a'} - \bar{\nu}^-_{a} & \text{if } -c_{aj} + \bar{\nu}^+_{a'} - \bar{\nu}^-_{a} > 0, \\ 0 & \text{otherwise.} \end{cases}$$

The expression for computing η^-_{aj} is:

$$\eta^-_{aj} = \begin{cases} c_{aj} - \bar{\nu}^-_{a'} + \bar{\nu}^+_{a} & \text{if } c_{aj} - \bar{\nu}^-_{a'} - \bar{\nu}^+_{a} > 0, \\ 0 & \text{otherwise.} \end{cases}$$

We denote the vector of gradients of H with respect to u^n by ω^n, where

$$\omega^n_{aj} = \begin{cases} \eta^+_{aj} & \text{if } \eta^+_{aj} > \eta^-_{aj}, \\ -\eta^-_{aj} & \text{if } \eta^+_{aj} \leq \eta^-_{aj}. \end{cases}$$

One method for modifying u is to use subgradient optimization. We form the updated upper bound vector u^{n+1} using

$$u^{n+1} = u^n + \delta^n \omega^n,$$

where δ^n is a step size which depends on our proximity to the optimal solution. We propose to use

$$\delta^n = \frac{\lambda^n (H^n_{upper} - H^n_{lower})}{\|\omega^n\|^2}$$

where $\lambda^n > 0$, and the values H^n_{upper} and H^n_{lower} are, respectively, the best-to-date upper and lower bounds on the optimal value of H. We set $\lambda^1 > 0$, and cut it in half whenever 5 iterations pass without improvement in H^n_{lower}. It is easy to determine H^n_{lower}—it is the best objective function value of all feasible solutions examined so far. We use the LP upper bounds which were computed by Powell and Carvalho in [22, 21] for H^n_{upper} in Section 4.1 (to give our algorithm the same environment as theirs). We denote the subgradient optimization method of updating u by SS.

A second approach to updating u is to use simple coordinate search. Although we compute ω^n as above, we update at most $M \in \mathbb{N}$ components of u^n. Picking the M largest values of ω^n, we then set $u^{n+1}_{aj} = u^n_{aj} + d$, where d is $+1$ or -1 according as ω^n_{aj} is η^+_{aj} or η^-_{aj}. We refer to this approach as CS for "Coordinate Search". Larger values of M will typically promote more rapid change in solution quality from iteration to iteration. However, when M becomes too large, the algorithm may behave in an unstable fashion, since ω is meant to evaluate marginal changes in u. We typically use $M = 1$.

A third approach is to keep u fixed, say $u = 1$. This is simple and requires no computation or updating of upper bounds.

A.3 LQN Parameter Settings

The parameter settings that we used in the computational experiments of Section 4.1 are given in Table 8. Ideally we would use statistical experiments to determine the best values of these

Parameter	Setting
Iterations	150
γ	0.15
λ^1	0.25
K	50
M	1

Table 8: Parameter Settings for Evaluation of Solution Quality

parameters, but these values have been shown to work well by Powell and Carvalho.