

Adaptive Nonlinear Approximation Algorithms for Multiattribute Resource Scheduling Problems

Warren B. Powell
Karthik R. Sarma

Department of Civil Engineering and Operations Research,
Princeton University, Princeton, NJ 08544
Statistics and Operations Research

May 30, 1998

1 Introduction

The problem we are considering is the Dynamic Multiattribute Resource Management problem. The objective of the problem is to manage a set of resources subject to a set of assignment costs with the goal of maximizing profits. In this paper, we focus on the problem as faced in transportation systems. Here, we have a dispatcher making decisions on the assignments of drivers to loads. Demands arrive continuously throughout the day. The dispatcher is required to assign drivers to handle specific demands, move empty or stay idle in anticipation of future loads. We index the time at which decisions are made by $t \in \{0, 1, 2, \dots, T - 1\}$ where T denotes the planning horizon of the problem and we model the problem as a dynamic program.

Different methods have been used to solve this problem depending on whether the resource types involved are homogeneous or heterogeneous. By homogeneous resources, we refer to resources whose only attributes are location and time. That is, all resources at Atlanta at 8am on December 5, 1997 are equivalent. They can all cover the same loads. These problems are similar to single commodity network flow problems. Whereas in a problem involving heterogeneous resources, resources have attributes in addition to space and time that distinguish them. For example, all drivers at Atlanta at 8am on December 5, 1997 with their domicile as Atlanta and 4hrs of service on their clock are equivalent. That is a driver with 4 hrs of service may not be able to move a load that one at Atlanta at 8am on December 5, 1997 with his domicile as Atlanta with 2hrs on his clock can move. Problems involving homogeneous resources have been solved using linear approximations of dynamic programs by [?] and nonlinear approximations of the recourse function by. [?] [?] have solved problems involving heterogeneous resources by employing linear approximations. In this paper, we extend these ideas and provide a solution technique to problems involving both homogeneous and heterogeneous resources by using nonlinear approximations of the recourse function.

The problem involving heterogeneous resource types is of interest because it involves taking separable approximations of nonseparable functions. We wish to see if it is possible to get high quality solutions by approximating nonseparable functions by means of piecewise linear separable functions. To obtain better approximations and for greater computational efficiency, our solution technique makes use of multiple levels of aggregation of resources.

In solving complex real world problems, there is always a tradeoff between solution quality and computational efficiency. To achieve computational tractability the set of resources of the large

problem are partitioned, and each subset is replaced by a single aggregated resource bucket. These buckets may further be aggregated on the basis of some resource attribute. Figure 1 illustrates an example of multiple levels of aggregation of resources.

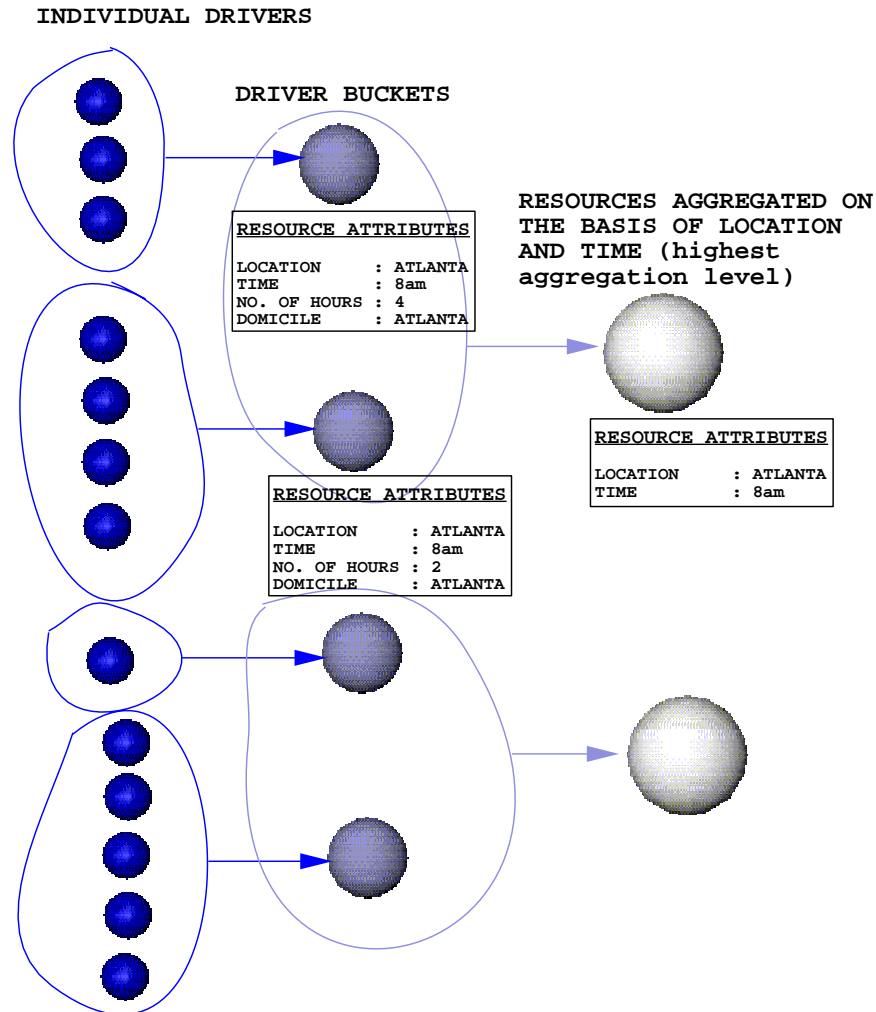


Figure 1: Different levels of Resource Aggregation

The small, dark spheres represent individual drivers and the larger spheres represent resources - aggregated on the basis of one or more attributes. Clearly as we move to higher levels of aggregation, the number of spheres goes down significantly thereby decreasing the computational effort required. Another very important feature is that as we move towards more aggregated resources, there is a considerable increase in the degree of separability of resources buckets. For instance, while multiple drivers (small, dark spheres) might be capable of moving a given load, it is unlikely that drivers in Atlanta can move the same load that drivers in Cleveland can move even if they are both available at the same time. Hence, there is a much greater degree of separability between the tasks

that the aggregated resource buckets (large spheres) can perform. As a result, the error term due to approximating the recourse function by means of separable approximations at the level of the aggregated resource is low.

Our solution approach is as follows. We model the resource management problem as a dynamic program. We propose an algorithm for solving problems involving multiattribute resources making use of aggregation techniques and nonlinear approximations of the recourse function in our model. We illustrate that our algorithm is capable of solving real world problems resulting in high quality solutions.

The paper is organized as follows. In Section 2, we present notation with which the problem is represented and the solution algorithm discussed. The notation closely follows [?]. Section 3 discusses the solution methodology that we employ. We demonstrate that the optimality recursion is computationally intractable. We present an algorithm involving aggregation of resources and nonlinear approximations of the recourse function to increase computationally efficiency while achieving high solution quality. In section 4, we present computational results representing the performance of our algorithm in solving both homogeneous and heterogeneous resource management problems.

2 Problem Notation

In this section we introduce notation using which we formulate the problem and discuss the solution methodology throughout the paper. The system of notation used closely follows that described in [?].

The Dynamic Resource Transformation Problem, DRTP, consists of three fundamental elements:

1. Resources
2. Processes
3. Controls

2.1 Resources

Resources correspond to the set of drivers and loads that are available in the system for assignment. To represent resources moving through the network, we define:

\mathcal{R}_t = Set of individual resources at time t .

\mathcal{A} = Space of possible attributes of each resource in the system.

\mathbf{a}_r = Attribute vector of the r^{th} resource $r \in \mathcal{R}_t$.

\mathcal{Q} = Set of dimensions of \mathbf{a} ; a_{q_r} is a particular element of a_r

The attribute vector a_r of a resource r describes the state of the resource. \mathcal{Q} defines the individual fields of the attribute vector for both drivers and loads. Example 1 shows a set of possible driver attributes.

Example 1 Driver Attribute Vector:

- a_{1r} = resource type - driver
- a_{2r} = the domicile terminal of driver r
- a_{3r} = the current/next terminal of driver r
- a_{4r} = the ETA of driver r at his current/next terminal
- a_{5r} = the number of hours of service of driver r since his last rest

The five elements of the attribute vector mentioned in the above example are different fields of the set \mathcal{Q} .

Example 2 specifies a set of typical elements that a load attribute might contain.

Example 2 Load Attribute Vector:

- a_{1r} = resource type - load
- a_{2r} = the origin of the load r
- a_{3r} = the destination of load r
- a_{4r} = the scheduled due time of load r at its destination
- a_{5r} = the driver who is coupled to load r

To make the problem solution more computationally tractable, we might aggregate the resources on the basis of certain attributes. For this purpose, we define an aggregation function,

$$\mathcal{G} : \mathcal{A} \rightarrow \mathcal{A}'$$

where \mathcal{A} and \mathcal{A}' are successive levels of aggregation of the attribute space, such that $\mathcal{A}' \subseteq \mathcal{A}$.

Example 3 illustrates one possible definition for the aggregation function acting on operator attributes.

Example 3 Aggregation Function: We specify the aggregation function as follows: If, a is defined as:

- a_1 = resource type - driver
- a_2 = driver domicile
- a_3 = the current/next terminal
- a_4 = the ETA of driver at his current/next terminal
- a_5 = hours on clock

Then, $a' = \mathcal{G}(a)$ has the following attributes:

- a'_1 = resource type - driver
- a'_2 = the current/next terminal
- a'_3 = the ETA of driver at his current/next terminal

We call this function a location time aggregation function for drivers.

As a result of aggregation, multiple resources can aggregate into a single element. To account for flow, we define,

$R_{a,t}$ = number of resources with attribute a present at time t

$R_{a,t}^e$ = number of resources with attribute a present at time t who arrived as a result of empty moves

$R_{a,t}^l$ = number of resources with attribute a present at time t who arrived as a result of loaded moves

2.2 Processes

The processes component of a DRTP illustrates how the system evolves over time. This component of the representation shows the evolution of information that comes to the system. It also describes the constraints that the system must observe.

2.2.1 Exogenous Information Processes

We model the information arriving to the system as data packets entering the system. Let:

e = An information event representing the arrival of a data packet.

t_e = The time of arrival of data packet e .

t_e^I = The time at which the information gets implemented into the system. ($t_e^I \geq t_e$)

In our case, we solve the problem with the assumption that as data arrives into the system, it is incorporated into the solution process. That is,

$$t_e^I = t_e$$

2.2.2 System Dynamics

We define a sample space Ω such that:

Ω = The set of possible outcomes of all events, $t \geq 0$

We also define a function called the modify function which describes the evolution of the system.

The function, \mathcal{M}_t is represented by the following mapping:

$$\mathcal{M}_t(a, d_t, K_t) \rightarrow (a', c, \tau) \tag{1}$$

where:

a = The attribute vector of the resource being modified.

d_t = A decision taken at time t .

K_t = The knowledge base of the system at time t .

The outputs of a modification are:

a' = The attributes of the modified resource.

c = The costs (or contributions) resulting from the action.

τ = The time required to complete the action.

An element $\omega \in \Omega$, then represents a particular outcome of a sequence of events. Hence, ω captures an instance of the information sequence as a result of which (1) can be equivalently written as:

$$\mathcal{M}_t(a, d_t, \omega) \rightarrow (a', c, \tau)$$

To account for flow conservation, we need a counting variable which we define as follows:

$x_{a,t}(d_t)$ = Flow variable that represents the number of times decision d_t is taken by resources at state a at time t

Also, to evaluate the rewards, we represent the contribution (or cost) of a transition as:

$c_{a,t}(d_t) =$ Reward obtained if a resource with attribute a takes decision d_t at time t

2.2.3 Constraints

The system has two primary physical constraints:

- Conservation of flow
- Capacity constraint

Flow conservation at each point in time is easily represented by

$\sum_{d_t \in \mathcal{D}} x_{a,t}(d_t) = R_{a,t} \quad \forall a \in \mathcal{A} \quad \text{duals} : \pi_a \quad \pi_a = [\pi_a^+ \ \pi_a^-]$ This constraint ensures that the total number of resources with attribute vector a being transformed at time t is equal to the number of resources with attribute vector a available in the system.

Capacity constraints for the system are expressed as:

$$x_{a,t}(d_t) \leq u_{a,t}(d_t)$$

where $u_{a,t}(d_t)$ represents a physical constraint in the transition $a \rightarrow a'$. It could represent for instance the capacity of a truck.

2.3 Controls

Controls describe the decision to be made and the method of evaluating decisions. We represent decisions that we make by the following notation. Let:

$\mathcal{D} =$ Complete set of possible decisions.

$\mathcal{D}^l =$ Set of decisions corresponding to loaded moves.

$\mathcal{D}^e =$ Set of decisions corresponding to empty moves.

$d_t =$ Individual decisions taken at time t , $d_t \in \mathcal{D}$

In our case, the individual decisions are very simple. For drivers, the two possible decisions are 'move empty' or 'move a load'. For loads, the two possible decisions are to 'couple a load to a driver' or 'do nothing'.

Using this notation, the number of resources expected to have resource attribute a' at some

time in the future as a result of loaded moves at time t , $R_{a',t+\tau}^l$ is given by:

$$R_{a',t+\tau}^l = \sum_{a \in \mathcal{A}} \sum_{d_t \in \mathcal{D}^l} x_{a,t}(d_t) 1_{\{\mathcal{M}_t(a,d_t,\omega)=(a',\tau)\}}$$

For evaluating our decisions, we have to define a value function. This value function captures the value of a set of resources at time t until the end of horizon. We define:

$V_t(R_t)$ = Value of the set of resources at time t until the end of horizon.

$\hat{V}_t(R_t)$ = An approximation of the value function.

In our case, we use a piecewise linear concave approximation of the value function.

3 Solution Approach

In this section, we present the dynamic programming formulation and discuss the approximations that we employ to make our algorithm computationally tractable. We then present our solution algorithm.

3.1 Dynamic Programming Formulation

We model the problem as a dynamic program by breaking it up into a sequence of smaller subproblems each representing a particular time period. We start with the standard dynamic program,

$$V_t(R_t) = \max_{x_t} \{c_t x_t + E(V_{t+1}(R_{t+1}))\} \quad (2)$$

where R_{t+1} represents the state of the system at time $t + 1$. The state, R_{t+1} is expressed as a function of the state at time t , R_t by the modify function, \mathcal{M}_t as,

$$R_{t+1} \leftarrow \mathcal{M}_t(R_t, d_t, \omega)$$

We solve the optimality recursion subject to the following constraints:

Flow Conservation

$$\sum_{d_t \in \mathcal{D}} x_{a,t}(d_t) = R_{a,t} \quad \forall a \in \mathcal{A} \quad \text{duals} : \pi_a \quad \pi_a = [\pi_a^+ \ \pi_a^-]$$

Capacity Constraint

$$x_{a,t}(d_t) \leq u_{a,t}(d_t)$$

The value of resources at time t is captured as a function of the value of resources in the future namely V_{t+1} . The objective of the problem is to determine $V_0(R_0)$ - the maximum value of all resources at time $t = 0$ till the end of horizon and the set of actions x_t that lead to this value. The problem is solved successively for every time period starting from $t = T - 1$ (where T represents the length of the planning horizon) and going down until $t = 0$. At any given time t , we are required to evaluate the value function $V_t(R_t)$ for every possible system state R_t . We observe that if we attempt to solve this problem by the dynamic programming optimality recursion, then it would be unsolvable. This is because evaluating the function $V_{t+1}(R_{t+1}) \forall R_{t+1}$ would mean evaluating it for every possible system state. If we have a problem involving say 5000 operators, 30000 loads and 500 terminals over 50 time periods, then at any given time the number of possible states the system can have is an extremely large number. This clearly suggests that we should pursue alternate routes. To get around this, we attempt to approximate the recourse function to obtain high quality solutions. The success of the algorithm then depends largely on the accuracy of our approximations.

3.2 Solution Algorithm

We replace the dynamic programming recursion (2) by:

$$\tilde{V}_t(R_t) = \max_{x_t} \{c_t x_t + \hat{V}_{t+1}(R_{t+1})\}$$

where $\hat{V}_{t+1}(R_{t+1})$ is an approximation of the recourse function. We calculate the approximation, $\hat{V}_{t+1}(R_{t+1})$ as a sum of separable functions, each function representing the value of an individual resource bucket at time $t + 1$. Let a' be the attribute vector of the drivers at time $t + 1$, $a' \in \mathcal{A}$. We evaluate the value function as the sum of separable approximations, $\hat{V}_{a',t+1}(R_{a',t+1})$ each representing the value of resources at node a' . That is,

$$\hat{V}_{t+1}(R_{t+1}) = \sum_{a' \in \mathcal{A}} \hat{V}_{a',t+1}(R_{a',t+1})$$

The function $\hat{V}_{a',t+1}$ is a piecewise linear concave function. We make use of the Concave Adaptive Value Estimation (CAVE) algorithm described in [?] to get this approximation. We intend to solve the problem iteratively and improve on the accuracy of our estimate of $\hat{V}_{a',t+1}$. As a result, we replace $\tilde{V}_t(R_t)$ with $\tilde{V}_t^k(R_t)$ to represent the value function at time t during the k^{th} iteration. The resulting dynamic programming recursion becomes:

$$\begin{aligned}\tilde{V}_t^k(R_t) &= \max_{x_t} \{c_t x_t + \hat{V}_{t+1}^k(R_{t+1})\} \\ \hat{V}_{t+1}^k(R_{t+1}) &= \sum_{a' \in \mathcal{A}} \hat{V}_{a',t+1}^k(R_{a',t+1})\end{aligned}$$

After solving the subproblem at time t , the CAVE algorithm makes use of the gradient information of the network subproblem and the flow into node a to get better approximations of $\hat{V}_{a,t}$ $\forall a \in \mathcal{A}$. That is,

$$\hat{V}_{a,t}^{k+1} \leftarrow f(\hat{V}_{a,t}^k, \pi_a^+, \pi_a^-, R_{a,t})$$

Without going into the details of the CAVE algorithm, we illustrate how we make use of the piecewise linear concave function that the algorithm generates. Figure 2 represents the concave function obtained at an aggregate node using CAVE. The first two drivers at the aggregate node

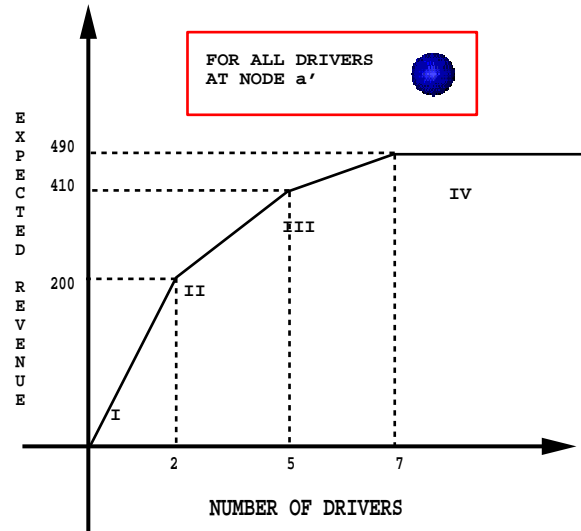


Figure 2: Value Function at a resource node as a function of the number of drivers present

are worth \$200 each, the next three are worth \$70 each, the next two are worth \$40 each and the rest are worth \$0.

These values are translated into the network as shown in figure 3. An important feature of

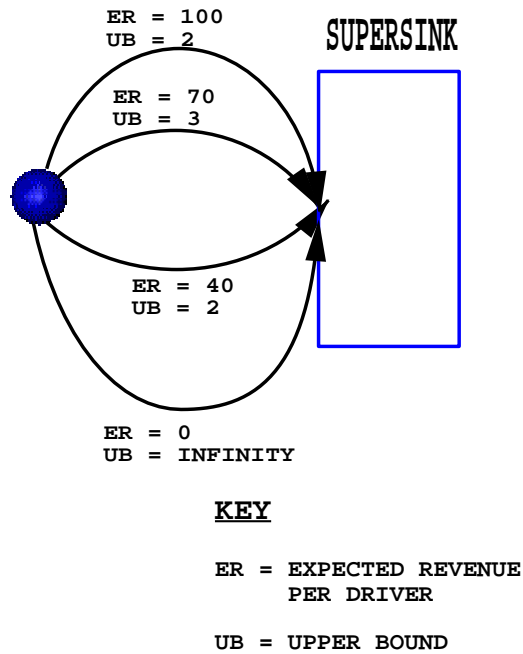


Figure 3: Equivalent Network Representation of figure 2

using the piecewise linear concave approximation of the value function is that we retain the network structure of the subproblem. The concave nature of the function ensures that by generating all the arcs shown in figure 3, flow goes through arcs with greater revenues first before going through those with lower revenues.

We now present the solution algorithm. The algorithm is outlined in the figure below. The

Solution Algorithm

Initialization

Step 1: Set time $t \leftarrow 0$, value function approximation $\hat{V}_{a,t}^0 \leftarrow 0 \quad \forall a, t$, iteration count $k \leftarrow 1$

Forward Pass of the Dynamic Program

Step 2: Construct the network subproblem at time t

Step 3: Solve the time t network subproblem

Step 4: Update $\hat{V}_{a,t}^k$, the CAVE approximation associated with time t nodes at the k^{th} iteration

Step 5: Set $t \leftarrow t + 1$ *Step 6:* Repeat steps 2-5 while $t < T$

Iterative process

Step 7: Set $k \leftarrow k + 1$, $t \leftarrow 0$

Step 8: Repeat steps 2-7 while $k \leq K$

first step involves initialization of the value function approximation at all nodes, \hat{V} to 0. Then we solve the forward pass of the dynamic program. At each time period t , this amounts to solving the network problem in figure 4.

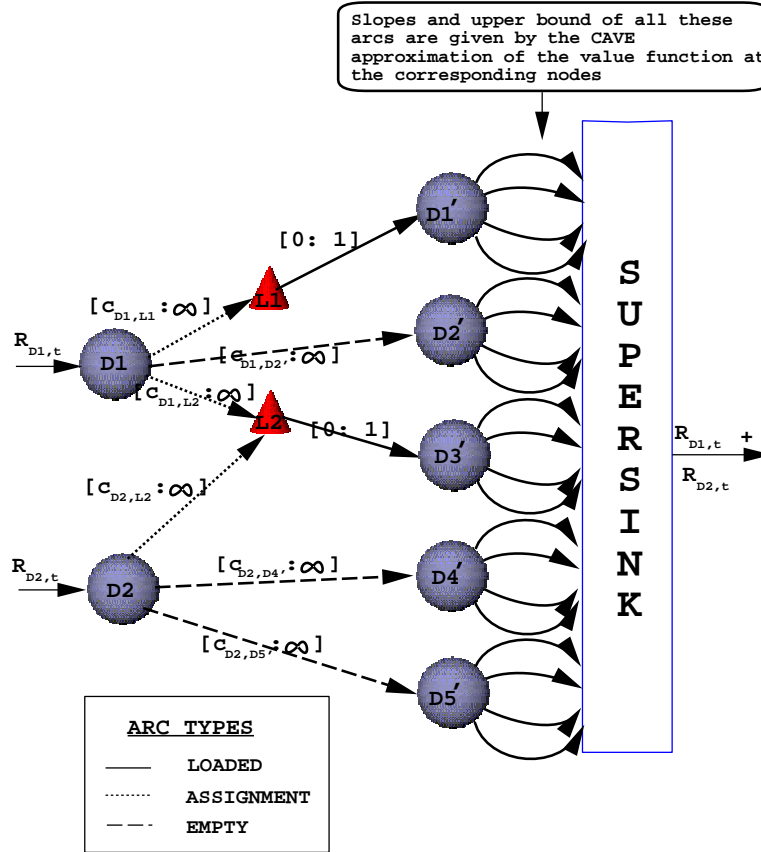


Figure 4: Network subproblem at time t

The arcs in our network are represented by $[m:n]$ where m represents the value of having flow on the arc and n represents the upper bound on the flow on the arc. The nodes $D1$ and $D2$ have supplies $R_{D1,t}$ and $R_{D2,t}$ respectively. The supersink has a demand equal to the sum of all the supplies in the network, which in this case is equal to $R_{D1,t} + R_{D2,t}$. As a result, to achieve flow balance, all flows must terminate at the supersink.

We observe again that the concave nature of the value function approximation ensures that we retain the network structure. As a result, we solve any time t subproblem by means of a network simplex algorithm that provides us with the π^+ and π^- values at each node. Using these values, we update the CAVE approximations at the corresponding nodes and move on to the next time period. We continue with this process until we hit the end of the planning horizon T . This entire

process corresponds to one simulation and we repeat the process for K simulations, using the newly updated value function approximations at each simulation.

In the above case, we have evaluated the value function approximations at the level of each driver bucket. However, it is not necessary that this has to be the level at which to calculate the approximations. We can calculate them at a higher level of aggregation of the state space, $\mathcal{A}' \subseteq \mathcal{A}$. There are two reasons for choosing a higher level of aggregation. First, by increasing the level of aggregation, we reduce the number of aggregate nodes as shown in figure 1, thereby decreasing the number of value function approximations that need to be evaluated. This considerably reduces the computational complexity involved. Second, as we move to higher levels of aggregation, the inherent separability between the loads that can be covered by different aggregate nodes increases considerably. This reduces the error involved in taking separable approximations of nonseparable functions.

The two key questions faced by us in adopting a solution scheme that incorporates state space aggregation are:

- What is the right level of aggregation of the state space at which we should calculate the value function approximation?
- Given the level of aggregation, \mathcal{A}' how do we update the value function approximation, $\hat{V}_{a',t+1} \forall a' \in \mathcal{A}'$?

The next section attempts to answer these questions.

4 Different Levels of Aggregation of the State Space

In this section we discuss the issues related to taking value function approximations at different levels of aggregation of the state space. In our problem, we have considered three levels of aggregation of the state space:

1. Highest level of aggregation (location, time)
2. Lowest level of aggregation (location, time, hours, sleeper status, driver domicile, etc.)
3. Intermediate aggregation (location, time, driver domicile)

Given a level of aggregation, there are two issues that are of importance:

- Constructing the network subproblem
- Updating the value function approximation, $\hat{V}_{a',t+1}$

If we aggregate at the highest level, then the construction of the subproblem is relatively straightforward whereas updating the value function becomes an important issue. If we aggregate at the lowest level, then the construction of the network subproblem becomes complicated. Finally, if we aggregate at an intermediate level, then we are compelled to deal with both issues. We now discuss this in greater detail.

4.1 Highest level of aggregation

Aggregation of the state space at the level of space and time represents the highest level of aggregation. Before we discuss the issues associated with different aggregation levels, let us first examine the structure of the network subproblem at some time t under aggregation of the state space at the level of location and time. A small network is shown in Figure 5. Each arc in the network is characterized by the nodes it connects, the value of having a unit of flow on the arc and the upper bound on the flow on the arc. The arcs connecting the driver to the load nodes are arcs representing loaded moves. They have positive flow if the given driver covers the load.

Now let us take a closer look at the aggregate nodes. If driver D1 or D2 stay idle at his location at time t , then he ends up at his current location at time $t + 1$ - this combination of location and time is represented by the aggregate resource node A2. Aggregate node, A4 represents the location and time of a driver D2 if he moves empty to another location. Aggregate nodes A1 and A3 represent the location and time of the destination of anyone who covers load L1 or L2 respectively. Deciding on the aggregate nodes A1 and A3 prior to solving the subproblem is a key issue related to aggregation. In this case, since we aggregate the state space by location and time, it is reasonable to expect that the location and time of the destination upon covering a given task at time t is independent of the driver covering it. However, this may not have been true if we had aggregated on the basis of location, time and some other driver attribute. Then, for example, it would have been difficult to identify the destination on covering load L2. The aggregate node at the destination would then depend on whether driver D1 or driver D2 covers the load.

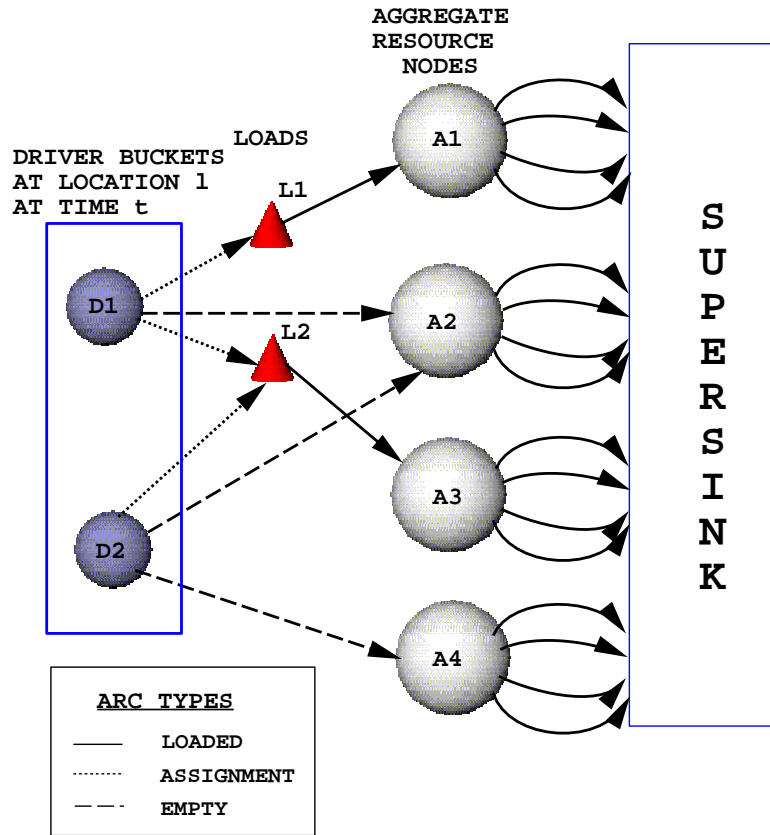


Figure 5: Sample Network

The slope and upper bound on the arcs connecting the aggregate resource nodes to the supersink correspond to the value function approximation at these nodes. These arcs are generated in the same way as shown in figure 3. They represent the future value of having drivers at the aggregate nodes. We use the CAVE algorithm to obtain this approximation. We initially start the problem assuming that the future value of having operators at all aggregate nodes is zero. After solving a problem the subproblem at time period t , we update the value function associated with all aggregate nodes whose time index is t . The inputs that CAVE requires at each iteration to update the value function associated with an aggregate node are:

1. Current supply at the aggregate node ($R_{A,t}$)
2. Value of having one additional driver at the aggregate node (π_A^+)
3. Value of having one less driver at the aggregate node (π_A^-)

Figure 6 gives an example of how we calculate the inputs for CAVE.

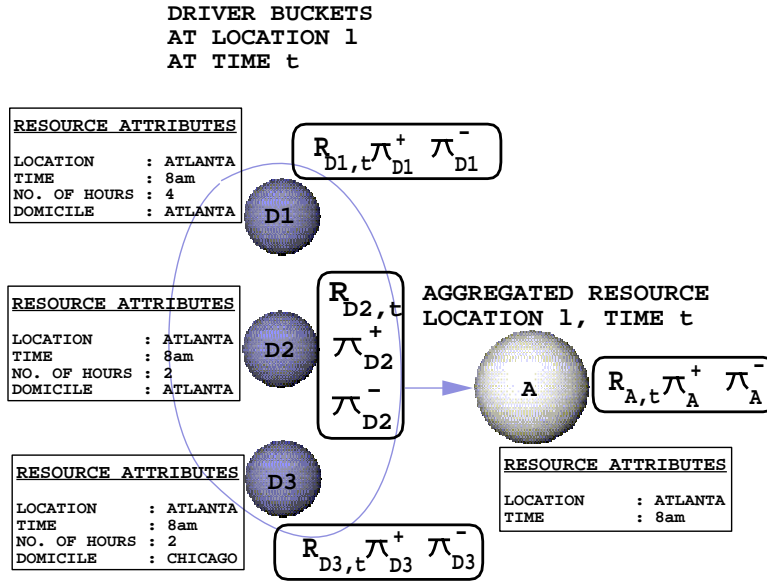


Figure 6: Calculation of the value function at an aggregate node

The values $R_{D_i,t}, \pi_{D_i}^+, \pi_{D_i}^- \forall i$ are obtained when we solve the network subproblem at time t . The question now is how do we use these values to calculate the corresponding values for the aggregate node A?

The first value, $R_{A,t}$ is easily calculated. It is clearly the sum of the individual supplies. But it is not clear how to calculate the dual values. This is another important issue related to aggregation. We approximate the dual value of the aggregate node as a weighted average of the individual duals, weighted by the supply at the the individual nodes. We feel that this gives a reasonable estimate of the dual value at the aggregate node. The equations below represent the method we used to calculate the CAVE inputs.

$$R_{A,t} \leftarrow \sum_i R_{D_i,t}$$

$$\pi_A^+ \leftarrow \sum_i \pi_{D_i}^+ R_{D_i,t} / \sum_i R_{D_i,t}$$

$$\pi_A^- \leftarrow \sum_i \pi_{D_i}^- R_{D_i,t} / \sum_i R_{D_i,t}$$

Other possible estimates for the dual values of the aggregate node could be the maximum of the individual duals, or the dual at the node with the largest supply.

An important feature of this aggregation is that there is a large degree of separability between the loads that drivers at different aggregated resource buckets can cover. As a result, there is a

smaller error term in approximating the recourse function as a sum of separable approximations of the value function at the level of the aggregated resource.

4.2 Lowest level of aggregation

Aggregation at the level of each driver bucket represents the lowest level of aggregation or the most disaggregate level. The network is constructed in much the same way as it is in the previous case. However, we have one added complication, we do not know the destination of a driver upon covering a load prior to solving the network subproblem. This is because the destination may vary depending on the driver covering the load. We overcome this difficulty by connecting the loads directly to the supersink and solving the subproblem. Now let us take the example of an aggregate node a' . Let us say that after solving the subproblem, we find that it was the destination of $R_{a'}^l$ loaded moves and $R_{a'}^e$ empties. When we had solved the subproblem, we only had the empties terminate at this node (because we had connected the loads to the supersink). This led to the value of each of the empties being perceived to be higher than it actually was thereby leading to extra empties being moved. This discrepancy is illustrated in figure 7. We attempt to capture this

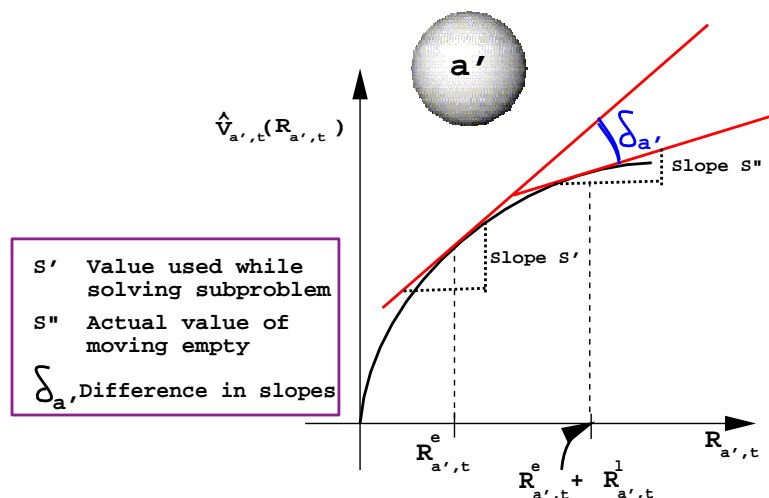


Figure 7: Discrepancy between the value function approximation used and the actual value difference by means of the iterative $\bar{\delta}$ solution scheme.

To account for loaded moves that might arrive at a particular destination prior to solving the subproblem, we present the iterative $\bar{\delta}$ solution scheme.

Iterative $\bar{\delta}$ solution scheme

We solve the network subproblem iteratively. At each iteration, we adjust the cost of moving empty to account for loads arriving into the aggregate nodes. Let us consider the example of the aggregate node a' described in figure 7. We compute $\delta_{a'}$, the difference in slopes S' and S'' . We perform some smoothing on this value to obtain $\bar{\delta}_{a'}$. To all inbound arcs of aggregate node a' , we add $\bar{\delta}_{a'}$ to the arc cost (or subtract from the revenue) and resolve the network. The algorithm is shown below.

Algorithm for solving the Network Subproblem

- Step 1:* Construct the network as before, but connect all loads directly to the supersink
- Step 2:* Solve the network subproblem
- Step 3:* For each load, determine the aggregate node into which the driver covering the load would end in
- Step 4:* For each aggregate node do:
 - Step 4a: Calculate the number of loaded moves terminating in the node $R_{a'}^l$
 - Step 4b: Calculate the number of empty moves terminating in the node $R_{a'}^e$
 - Step 4c: Total flow into the node, $R_{a'}^l + R_{a'}^e$
 - Step 4d: $\delta_{a'}^n = \partial \hat{V}_{a'}(R_{a'}^e) - \partial \hat{V}_{a'}(R_{a'}^l + R_{a'}^e)$, where $\hat{V}_{a'}$ = value function approximation at a'
 - Step 4e: $\bar{\delta}_{a'}^n = \gamma^n \delta_{a'}^n + (1 - \gamma^n) \bar{\delta}_{a'}^{n-1}$ by performing some smoothing on ϵ^n
 - Step 4f: For all inbound arcs into the aggregate node, subtract their values by $\bar{\delta}_{a'}^n$
- Step 5:* Repeat steps 2-4 while number of iterations, $n < N$, where N is a constant

At each time t , the problem is solved iteratively, each iteration corresponds to solving a network subproblem. This increases the computational work. However, the important point to note is that at each iteration, we do not create a new network. We merely modify a few arc costs in the old network. Since the majority of the computational time is taken up in creating the network and solving the network is only a small fraction of this time, we are not increasing the computational effort significantly.

We now discuss the calculations required to update the value function approximation. Since, we are aggregating the state space at the level of the driver bucket, each aggregate node is comprised of exactly one node. As a result, the calculation of the supply and dual values is straightforward. We directly take the value of the supply and the duals at the driver bucket and there is no need for aggregating it.

From the above discussion, it becomes apparent that while solving the network subproblem has become more complicated under this level of aggregation, updating of the value function approximation has become simpler.

However, at this level of aggregation there is a larger error term in approximating the recourse function as a sum of separable approximations of the value function at the level of the aggregated resource. This is because the loads that drivers at different aggregated resource buckets can cover have considerable overlap. This is one of the shortcoming of this form of aggregation.

4.3 Aggregation of the state space at an intermediate level

We choose an intermediate level of resource aggregation as the level of location, time and driver domicile. Figure 8 gives an example of this form of aggregation. The reason we have chosen this

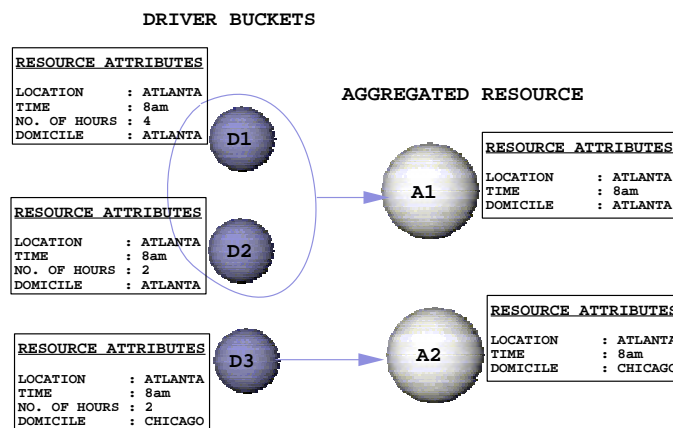


Figure 8: Aggregation of the state space at the level of location, time and driver domicile

form of aggregation is that in our problem, the loads tend to separate themselves out at the level of location, time and driver domicile. That is, at this level of aggregation, loads that can be covered by drivers at one aggregate node typically can't be covered by drivers at a different aggregate node and vice-versa. This underlying separability in our problem motivates this intermediate level of aggregation.

We note that if we aggregate at the intermediate level of aggregation, then we have to get around both the hurdles we encountered earlier, namely

- Solving the network subproblem iteratively
- Aggregating individual dual values to obtain the dual value at the aggregate node

5 Computational Results

In this section, we attempt to answer two key questions:

- *How good is the nonlinear approximation of the recourse function?*
- *How do different levels of aggregation of the state space affect solution quality?*

To answer the first question, we consider a problem involving homogeneous resources. The problem under consideration involves 850 drivers and 3100 loads distributed over 100 locations. We solve the problem for a planning horizon of 1 week. Decisions are taken at the end of every 4 hour period. As a result, we have 6 decision epochs/day, leading to a total of 42 decision epochs ($t \in \{0, 41\}$). We apply our algorithm and one involving linear approximations of dynamic programs to the problem and compare their performance. The algorithms have been coded in C and tested on a SGI INDY R5000 workstation.

We also formulate the problem by aggregating all flows into a single commodity. This optimization problem estimates the cost of moving empty. But it ignores all timing constraints and union rules. The only constraint that we adhere to is flow conservation. This problem is solved as a pure network. The solution to this problem gives us an upper bound to the objective function value. However, it is only a “pseudo upper bound” since this model requires flow conservation over all points in time. But, it serves a useful measure on which to evaluate solution quality. Using this value, we calculate the OPT ratio which is the ratio of our objective function value to that of this upper bound.

We compare solutions on the basis of the following parameters:

- **Solution quality:** We compare the objective function values that both algorithms attain, which is equivalent to the comparison of OPT ratios.
- **CPU time:** We compare the processing time of both algorithms. We also wish to see if the algorithm can be used to solve real-world problems in a reasonable amount of time.

For the purpose of comparing CPU times, we define hitting times, T_α ,

$T_\alpha =$ Time taken for the algorithm to get to a solution that has an OPT ratio of $\alpha\%$.

The results are summarized in Figure 9.

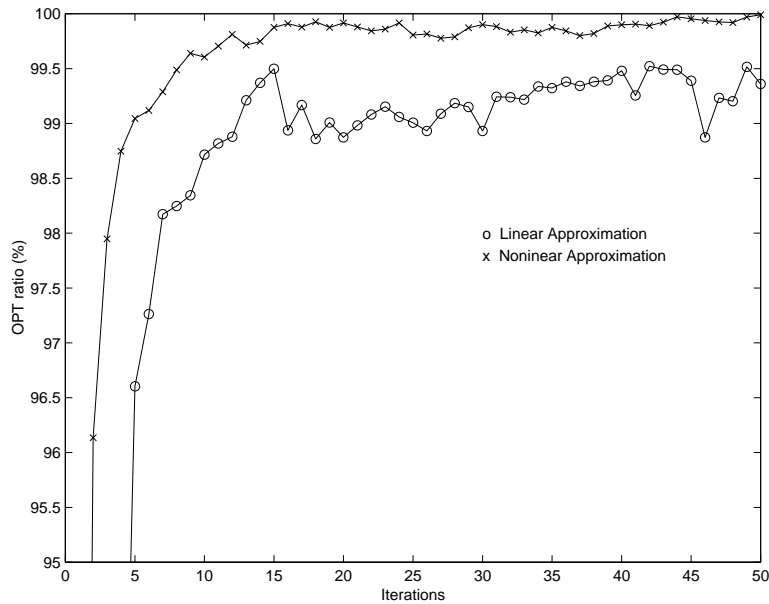


Figure 9: Linear Approximation vs. Nonlinear Approximations for problems involving Homogeneous Resources

It can be clearly seen that the solution obtained by using a nonlinear approximation for the recourse function dominates the linear approximation. From the graph, it is clear that our algorithm has a much faster rate of convergence than the linear approximation. After around 15 iterations of the nonlinear approximation, the graph flattens out. In the case of the linear approximation, we have several peaks and valleys even after 30 iterations.

Table 1 shows the comparative performance between the linear and nonlinear approximations over a set of performance measures.

<i>Parameter</i>	<i>Linear Approximation</i>	<i>Nonlinear Approximation</i>
OPT ratio	99.51%	99.99%
CPU time/iteration (sec)	10.9	11.4
Loaded moves	3044	2934
Empty moves	956	651
<i>Hitting Times (sec)</i>		
T_{90}	49.1	27.3
T_{95}	60.7	27.3
T_{98}	81.5	49.1
T_{99}	144.8	60.0
$T_{99.5}$	457.6	104.3
$T_{99.9}$	DNR(Did Not Reach)	182.2

Table 1: Comparative performance of Linear and Nonlinear Approximations

Although the average CPU time/iteration is greater for the nonlinear approximation, the number of iterations needed to achieve a high quality solution is far less. As a result, the nonlinear approximation gives us better solutions as well as faster solutions. This is reflected by the lower T_α values of the nonlinear approximation.

We also note that while the linear approximation results in a solution that covers more loads than the nonlinear approximation, it requires a lot more empty repositioning of drivers to cover the loads, leading to lower profits.

Now that we have established that the nonlinear approximation is a good approximation, we focus our attention on answering the second question. For this purpose, we consider a problem in a more general setting involving multiattribute resources. The problem under consideration involves approximately 5000 drivers and 15000 loads distributed over 550 locations. The components of the attribute vector for each driver are given in table 2.

i	Attribute a_{ir}
1	Location
2	ETA of driver at his current/next terminal
3	Hours of service
4	Domicile
5	Sleeper status
6	Layovers out of home

Table 2: Attribute Vector of Driver r

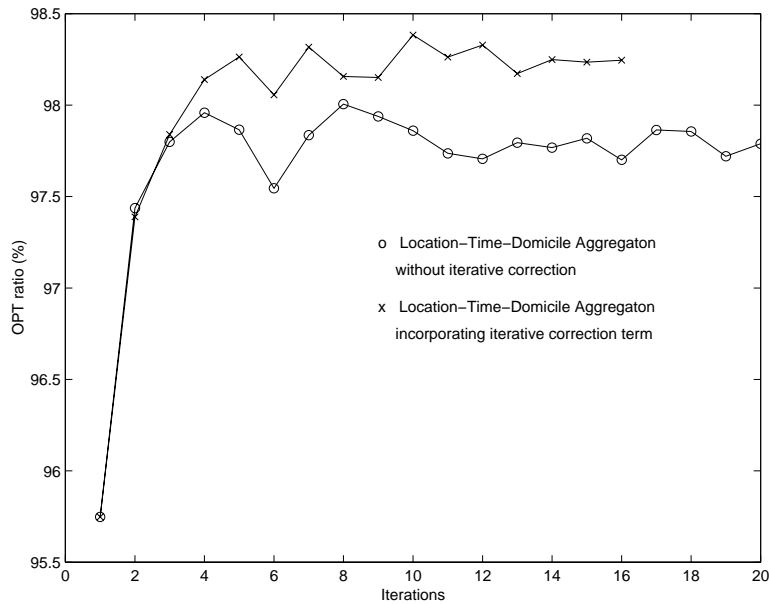
For this problem, we focus on the different levels of aggregation of the state space and compare the corresponding solutions. We consider three levels of aggregation of the state space. The first level involves aggregation at the level of location and time, the second at the level of location, time and driver domicile and the third at the level of each individual driver bucket. We first consider the effect of the iterative $\bar{\delta}$ scheme proposed in section 4.2 on solution quality. Let,

N = number of iterations of the network simplex at each decision epoch t

γ^n = smoothing factor used in smoothing δ at iteration n

Figure 10 shows the effect of the iterative scheme for the case of aggregation at the level of location, time and driver domicile.

Clearly, there is a substantial improvement in the objective function value in this case. Table 3 lists the parameters used in the δ correction and compares the two solutions. We applied the solution scheme for different values of γ^n and N . We observed best results for $N=3$ and $\gamma^n = 1/n$

Figure 10: Effect of δ correction on solution quality

<i>Parameter</i>	<i>Without δ correction</i>	<i>Using δ correction</i>
N	1	3
γ^n	1	$1/n$
OPT ratio	98.00%	98.33%
CPU time/iteration (sec)	501.0	506.0

Table 3: Comparison of solutions with/without iterative $\bar{\delta}$ scheme

We observe that there is hardly any increase in CPU time due to the additional iterations of the network simplex. This is due to the fact that each iteration corresponds to solving the network with some of the costs altered. It does not require the creation of a new network from scratch. Network creation is the step which takes up the bulk of the time. We applied the iterative scheme to the aggregation at the level of each driver bucket. In this case, we did not observe a sizable improvement in solution quality. Since there is hardly any increase in solution time, we performed all experiments (for the more disaggregate cases) incorporating this scheme.

We then evaluate the value function approximations at the aggregate resource levels. Figure 10 shows the comparative rates of convergence of the algorithm under the three different aggregation levels.

As the graph indicates, we get better solutions as we move towards aggregation levels with more attributes. However, as we move towards higher levels of aggregation (aggregate resources

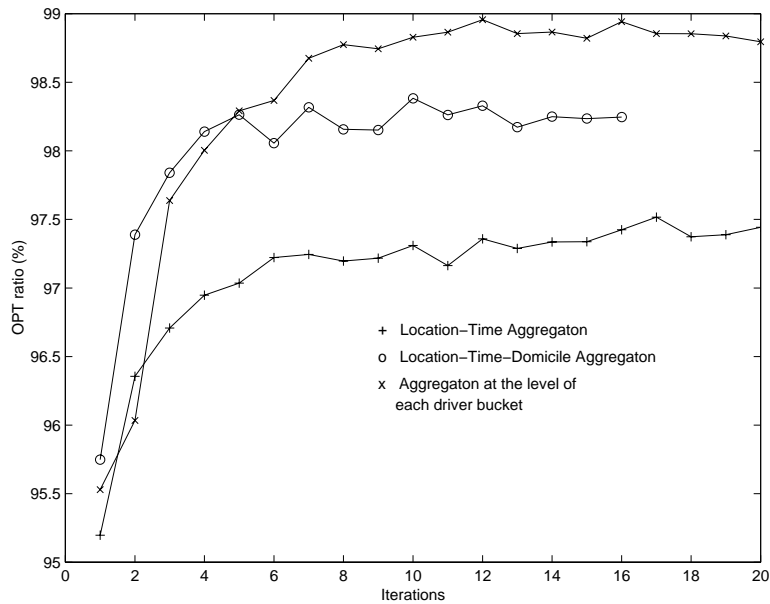


Figure 11: OPT ratio vs. Iteration Count at different levels of aggregation of the state space

with fewer attributes), there is a significant decrease in the number of aggregate nodes. As a result, fewer value function approximations need to be evaluated, leading to a significant decrease in computational complexity for each iteration of the algorithm. This tradeoff between higher solution accuracy (suggesting a fine aggregation) and reduced computational effort (suggesting a coarse aggregation) is a key issue associated with aggregation.

Table 4 gives a comparison of some performance measures for the solutions obtained at the different aggregation levels.

<i>Parameter</i>	<i>Aggregation Levels</i>		
	Location-Time	Location-Time-Domicile	Driver bucket
OPT ratio	97.5%	98.3%	99.0%
CPU time/iteration (sec)	190.1	506.0	530.0
Loaded moves	14127	14041	14197
Empty moves	1682	1334	1786
<i>Hitting Times (sec)</i>			
T_{90}	195.2	378.6	309.4
T_{95}	195.2	378.6	309.4
T_{97}	822.5	757.3	1060.7
$T_{97.5}$	3192.1	1193.7	1060.7
T_{98}	DNR(Did Not Reach)	1655.7	1497.8
$T_{98.5}$	DNR	DNR	3059.7
$T_{99.0}$	DNR	DNR	5723.2

Table 4: Comparative performance at different levels of state space aggregation

It is clear from the table that the choice of the appropriate level of aggregation of the state space represents a tradeoff between solution quality and solution time.

6 Conclusions

We have introduced an algorithm for multiattribute resource scheduling problems. The algorithm incorporates nonlinear approximations of the value function. We have established that the approximation method gives high quality solutions. The algorithm has been tested for real world problems involving multiattribute resources for different levels of state space aggregation. The tradeoffs that exist between solution quality and computational efficiency have been clearly demonstrated. Our algorithm produces high quality solutions within a reasonable amount of time for real world problems.