Computer-Aided Civil and Infrastructure Engineering **13** (1998) 315–327

Traffic Prediction and Management via RBF Neural Nets and Semantic Control

S. Massoud Amin*

Strategic Science and Technology, EPRI, 3412 Hillview Avenue, Palo Alto, CA 94304, USA

E. Y. Rodin, A-P. Liu, K. Rink

Center for Optimization and Semantic Control, Department of Systems Science and Mathematics, Washington University, St. Louis, Missouri 63130, USA

&

A. García-Ortiz

Advanced Development Center, Systems & Electronics, Inc., 201 Evans Avenue, St. Louis, Missouri 63121, USA

Abstract: *The objective of this work has been to develop layers of control and optimization modules for the purpose of urban traffic management. We utilize the semantic control paradigm to model both the macrolevel (traffic control) and the microlevel (vehicle path planning and steering control). A semantic controller consists of three modules for identification, goal selection, and adaptation, respectively. This hierarchical structure has been used successfully at the Center for Optimization and Semantic Control to solve complex, nonlinear, and time-varying problems. In our previous work we have used a judicious combination of artificial intelligence, optimization, and control systems.*

The focus of this paper is the identifier module, which performs "system identification," i.e., determines the road network congestion level. Traffic flow can be characterized as a nonlinear stochastic process where linear prediction models such as linear regression are not suitable. However, neural network techniques may provide an effective tool for data-based modeling and system identification. The radial basis function neural network (RBFNN) is an attractive tool for nonlinear time-series modeling and traffic-flow prediction. The goal selector module that finds the shortest path is also discussed in some detail.

A model of the highway system, based on historical data provided by Missouri Highway and Transportation Department (MoHTD), has been developed. The prediction and planning system is evaluated using the traffic-flow data from nine sensors located on the highway in the St. Louis metropolitan area.

* To whom correspondence should be addressed.

1 INTRODUCTION

The emergence of the various thrusts of Intelligent Transportation Systems (ITS)[1,15] presents numerous new theoretical and practical challenges; many of these deal with the modeling, prediction, cause-and-effect relationships, analysis, optimization, and control of an overall transportation system. In view of these, an advanced traffic management system (ATMS) will require a dynamic traffic model that can

operate in real time and reliably predict traffic congestion. In general, the solution methodologies available for this problem can be grouped into five categories[1,15]: computer simulation, mathematical programming, optimal control, artificial intelligence, and intelligent control. Our methodology, one of intelligent/hybrid control, utilizes algorithms and tools from all the preceding approaches. Such hybrid systems for ATMS attempt to cope with the nonlinear and stochastic nature of traffic flow and incidents.

There are two approaches to a general prediction problem: explanatory and time series. *Explanatory forecasting* evaluates a cause-and-effect relationship between inputs to the system and its outputs. Usually, the inputs and outputs can be expressed as equations. On the other hand, *time-series forecasting* treats a system as a black box. The system is neither "fully" understood nor explicitly represented; therefore, the causes and effects at the output are not clearly explained. It relies on the discovery of strongly empirical regularities in the observation of the system. Traffic-flow forecasting can be viewed as a time-series prediction problem: Given a sequence $x(1), x(2), \ldots, x(N)$, predict the continuation $x(N+1), x(N+2), x(N+3), \ldots$. In the past three decades, several methods have been applied to traffic-flow prediction problems. Among these are the Kalman and adaptive filtering methods, as well as the Box-Jenkins method. They typically provide a one-step ahead prediction. The radial basis function neural network (RBFNN) has been proposed by different authors,[7,29] and it is an attractive tool for time-series prediction and system identification problem. Our goal here is to design a system identifier via RBFNN for every sensor station in the St. Louis area. Each system identifier uses the its own past traffic-flow data and other sensors' past traffic flow as input. The output of the system is the predicted traffic flow.

1.1 Traffic management via semantic control

In this approach, utilizing the semantic control paradigm, we implement a hybrid prediction/routing/control system to model both the macro level (traffic control) and the micro level (in-vehicle path planning and steering control). A semantic controller consists of three modules for identification, goal selection, and adaptation, respectively. This hierarchical structure has been used successfully at the Center for Optimization and Semantic Control to solve complex, nonlinear, and time-varying problems.[3,16,24–26,34,36] A semantic controller (Figure 1) consists of

Identifier: Processes traffic data and interprets the available information for travel times and incidents.
Goal Selector: Generates and evaluates candidate paths and provides a traveler advisory.
Adapter: Implements vehicle steering control laws and provides driver's support.

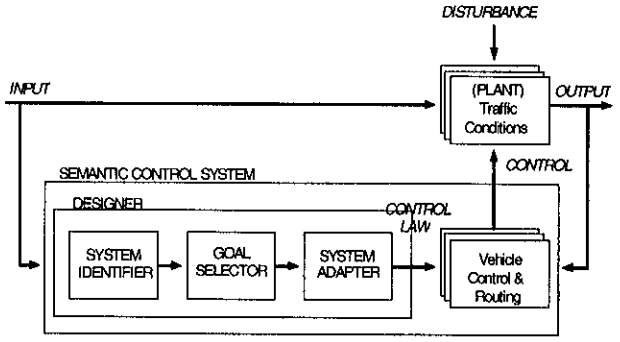

Fig. 1. A semantic control system consists of a system identifier, a goal selector, a control system adapter, and one or more control systems/laws.

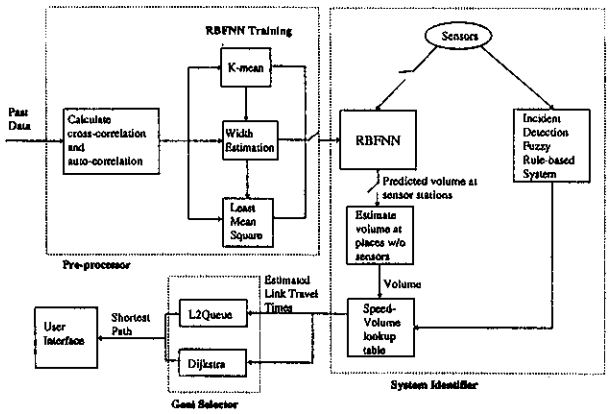

Fig. 2. Functional block diagram of the identifier and the goal selector modules.

A functional block diagram of our methodology is depicted in Figure 2; the system consists of three main parts: preprocessor, system identifier, and goal selector.

The preprocessor consists of codes for calculation of correlation coefficients and the radial basis function neural network (RBFNN) training algorithm. RBFNNs are trained to predict traffic volumes at nine sensor stations located along the major highways in the St. Louis area. As a part of the training algorithm, the auto-correlation coefficient and the cross-correlation coefficient between sensors are used to select inputs to the RBFNN for each sensor. After the input data have been selected, the K-means algorithm, width estimation, and least-mean-square algorithm are performed to complete the RBFNN training. The system identifier consists of an incident detection rule–based system, a volume estimator for nodes and/or links without sensors, and a speed-volume lookup table for speed estimation.

The goal selector handles traffic routing for maximum use of available road network capacity; the goal selector consists of two submodules, one for traffic management in signaled

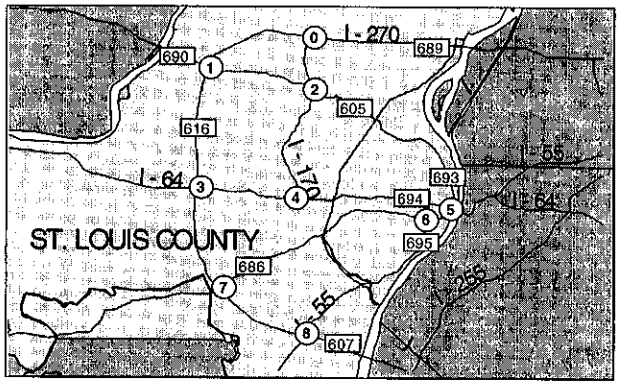

Fig. 3. The graphic user interface for the St. Louis highway system.

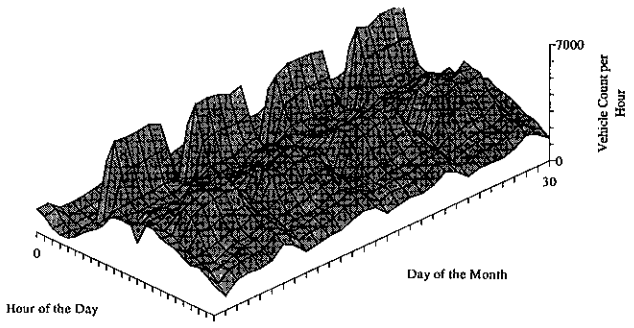

Fig. 4. The periodicity of traffic flow evidences itself when vehicle counts obtained through the use of roadside sensors are graphed as surface plots.

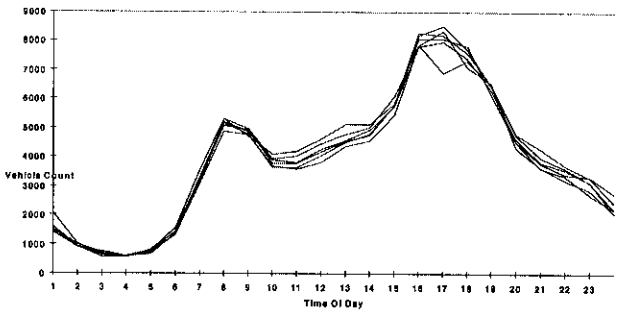

Fig. 5. I-70 westbound traffic, Thursdays.

streets and another for highway traffic control (advisory). The adapter copes with changes as well as providing micro-level driver's support for shortest-path routing and steering control (an in-vehicle driver's assistance). In summary, the identifier has a stochastic prediction module, rules for incident detection, and a radial basis function neural network–based estimator. The goal selector consists of shortest-path algorithms and a discrete-event dynamic system simulation of signaled intersections. The control laws in the adapter are developed for two situations: (1) vehicle control within a platoon and (2) neurocontrol of a single vehicle. In this paper we focus mostly on the first module, the identifier, which is more closely tied to the use of neural networks for system identification and parameter estimation; additionally, we will discuss the goal selector module in Section 4 of this paper.

2 IDENTIFIER: TRAFFIC FLOW ESTIMATION AND PREDICTION

There are currently nine permanent sensors deployed in several locations on some of the major highways in the area. Figure 3 depicts a facsimile of the St. Louis highway network with marked locations for the sensor stations; it is also our graphic user interface.

From a modeling point of view, the problem of dynamic traffic-flow assignment can be viewed as the prediction of the behavior of traffic flow where the flow varies both temporally and spatially. The temporal nature of the flow is evident in data shown in Figures 4 and 5. For example, Figure 4 corresponds to eastbound traffic along I-70 near the city of St. Louis. Two temporal patterns can be discerned from the graph: a daily flow pattern and a weekly flow pattern. Both of these are being explored by Department of Transportation (DOT) personnel for road-maintenance scheduling and by ITS researchers for trip planning and vehicle routing. A third pattern, not yet addressed in the literature until now, is the seasonal or annual pattern, which reflects vacation travel during holidays and the summer and winter seasons. This last pattern is incorporated in the ITS research being conducted at Systems & Electronics, Inc., and Washington University in St. Louis. *Temporal* traffic flow is due to work-related travel, i.e., travel to and from work from Monday through Friday between the hours of 8 A.M. and 4 P.M. *Spatial* traffic flow is due to the population and workplace distributions in a given area. As a result of this, some fundamental issues arise when attempting to model traffic flow, namely:

- Where to place traffic flow sensors?
- How to characterize the traffic flow?
- How to manage the traffic load?
- What on-line estimators to use for system identification and parameter estimation?
- How to perform vehicle routing and adjust traffic signals?
- How to formulate traffic flow (e.g., as interconnected pipes with adjustable valves or as discrete event dynamics, a system approach versus a mathematical programming and combinatorial optimization)?

2.1 A brief introduction to radial basis function neural networks (RBFNNs)

The input-output of a RBFNN (see, for example, ref. 15) is characterized by

$$\begin{aligned} x_i &= g\left(\frac{|u - c_i|}{\sigma_i}\right) \qquad i = 1, 2, \ldots, n_1 \\ y &= Cx \end{aligned} \tag{1}$$

where $u \in \Re^M$ represents the input, $x \in \Re^{n_1}$ is the output of the hidden layer, $y \in \Re^P$ is the network output, $C \in \Re^{P \times n_1}$ is the weight matrix, and $c_i \in \Re^M$ and $\sigma_i > 0$ are the center and variance of the i^{th} unit. Examples of radial basis function include $\Phi(r) = r^2 \log r$, $\Phi(r) = (r^2 + c^2)^{1/2}$, and $\Phi(r) = \exp[-(r^2/\sigma^2)]$. It has been stated by Park and Sandberg[32] that the radial basis function is capable of universal approximation. The approximation capability of a gaussian RFBNN is stated next.

Theorem (RBFNN Approximation). *Let K be a compact subset of the $\Re^M$ and $f\colon K \to \Re^P$ be a continuous mapping. Then, for any $\varepsilon > 0$, there exists an integer n_1, a matrix $C \in \Re^{P \times n_1}$, vectors $c_i \in \Re^M$ and scalars $\sigma_i > 0$ for $I = 1, \ldots, n_1$ such that $\max_{x \in K} \|f(x) - \hat{f}(x)\| < \varepsilon$, where $\hat{f}\colon K \to \Re^M$ is given by Eq. (1) with gaussian activation functions.*

Radial basis functions were first introduced as a technique for real-valued multivariate interpolation problems. The work is surveyed by Powell.[33] Since then, radial basis function (RBF) (e.g., gaussian function) approximation has become a very popular technique for interpolation in multidimension space. Radial basis function neural networks have attracted a great deal of interest because of their fast learning and simple network structure. Instead of applying stochastic approximation techniques to neural network design problems, RBFNNs use a curve-fitting approach. In the context of neural networks, the hidden layers provide a combination of a "function" that constitutes an arbitrary basis for the input data when they are passing through the hidden layer. These functions are called *radial basis functions*. In this paper, all functions in the hidden layer are gaussian functions. Therefore, it is also called *gaussian basis function neural network*. Thus a radial basis function neural network can be regarded as a special fully connected two-layer feedforward neural network with radial activation functions in the hidden layer. Besides the traditional weight adjusting for neural networks, the training of RBFNNs also consists of center and width parameter estimation for the RBF. The performance of a RBFNN critically depends on the chosen centers and widths. RBFNNs have a fast learning (or training) speed that makes them a powerful tool for on-line problems. They have been used in system identification, modeling, classification, load forecasting, and bankruptcy-prediction problems.

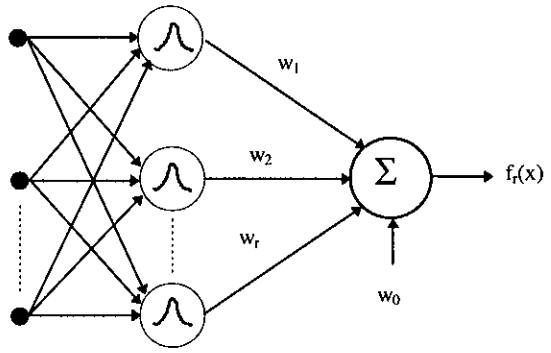

Fig. 6. A radial basis function neural network.

2.2 Radial basis function neural network structure

Figure 6 shows the basic structure of a RBFNN with n inputs and 1 output. A RBFNN consists of two layers: a hidden layer and an output layer. The output nodes form a linear combination of the basis functions computed by hidden layer nodes. The input nodes pass each input to every hidden layer without weights. Therefore, each hidden node receives input data unaltered; the hidden nodes contain the gaussian function, whose purpose is similar to that of the sigmoid function (normally used in backpropagation networks). It is nonmonotonic in contrast to the monotonic sigmoidal function. It performs a nonlinear transformation of the input data, using a parameter vector called *center*. The euclidean distance between the center and the input is calculated, and the result is passed through the nonlinear function to generate the output. The second layer of connection is weighted. The final output of the network is just a linear combination of the output of the hidden layer. This function produces the largest output when the input variables are closest to the center, and its output decreases as the distance from the center increases. The width of the gaussian function controls the rate of decrease, with a smaller width giving rise to a rapidly decreasing function. It serves the same purpose as the standard deviation in the standard normal probability distribution, although it is not estimated in the same way. The learning phase in the radial basis function neural network can be divided into three steps and two phases. The three steps are (1) find the centers, (2) find the widths, and (3) weight training on the outer layer. The two phases are (1) unsupervised learning (centers and widths) and (2) supervised learning (weight training). The performance of an RBFNN critically depends on the chosen centers and widths. The selection of values for centers and widths is discussed in the following subsections.

2.2.1 *Learning algorithm*

Training an RBF neural network consists of determining the location of the centers and widths for the hidden layer and the

Table 1
K-means algorithm

```
Procedure K-means
    Initialize a group of clusters c_j whose centers
      are w_j      j = 1, 2, ..., N
    Repeat
        for all training pattern x_j do
            Assign x_j to c_j if d(x_j − w_j) is minimum
              for all j = 1, 2, ..., N
        end
        for all w_j do
            New w_j = (1/m_j) Σ_{x_j ∈ j} x_i
        end
    Until cluster centers do not change from one
      iteration to the next
```

Table 2
LMS algorithm

```
Procedure LMS
    Initialize the weights
    Repeat
        choose training pair (x, d)
        for all j do
            y_j = w_j x
        end
        for all j do
            e_j = y_j − d_j
        end
        for all j do
            w_j(k + 1) = w_j(k) + μ e_j x
        end
    until termination condition reached
```

weights of the output layer. It is trained using a two-phase approach: In the first phase, unsupervised learning occurs, whose main objective is to optimize the location of center and width. In the second phase, the output layer is trained in a supervised mode using the least-mean-square (LMS) algorithm to adjust the weight so as to obtain the minimum mean square error at the output. The following are the three steps of the hybrid learning method for an RBFNN, and they are discussed in more detail in the next three subsections:

1. Find the cluster centers of the radial basis function; use the K-means clustering algorithm.
2. Find the width of the radial basis function.
3. Find the weights; use LMS.

2.2.2 *Calculation of centers*

There are numerous clustering algorithms that can be used to find the center of the hidden layer. The most popular is the K-means clustering algorithm. The simplicity of the K-means clustering algorithm, combined with its capability to produce good results, makes it the most widely known clustering method. The purpose of applying the K-means clustering algorithm is to find a set of clustered centers and a partition of training data into subclasses. The center of each cluster is initialized to a randomly chosen input datum. Then each training datum is assigned to the cluster that is nearest to itself. After training data have been assigned to a new cluster unit, the new center of a cluster represents the average of the training data associated with that cluster unit. After all the new centers have been calculated, the process is repeated until it converges. Another popular way to find centers is based on the orthogonal least squares method[7]; a pseudocode for the algorithm is given in Table 1.

2.2.3 *Width calculation*

After the RBF centers have been found, the width is calculated. The width represents a measure of the spread of data associated with each node. The most common way is to make them equal to the average distance between the cluster centers and the associated training data. Other ways to calculate the width is the P-nearest neighbor method and genetic algorithm,[22] a robust stochastic search technique.

$$\sigma_j^2 = \frac{1}{M_j} \sum_{x \in \theta_j} (x - c_j)^T (x - c_j)$$

where θ_j is the cluster j
M_j is the number of members in θ_j
c_j is the center of θ_j

2.2.4 *Weight estimation*

Learning in the outer layer is performed after calculation of the centers and widths of the RBF in the hidden layer has been completed. The objective is to minimize the error between the observed output and desired one. It is commonly trained using the LMS algorithm (Table 2). Another popular method for the weight training is the pseudoinverse algorithm.

3 AN ILLUSTRATIVE EXAMPLE

Our radial basis function neural network consists of a total of 72 neurons. There are six input nodes (previous traffic flow) and one output (predicted traffic flow). For illustration, let us consider the sensor 605 EB (I-70 eastbound in Figure 3). After all the auto-correlations and cross-correlations between the 605 EB and other sensors have been calculated, the highest 6 correlation coefficients were used as input to the neural network. The six centers for each neuron were then calculated using the K-means clustering algorithm. After the centers

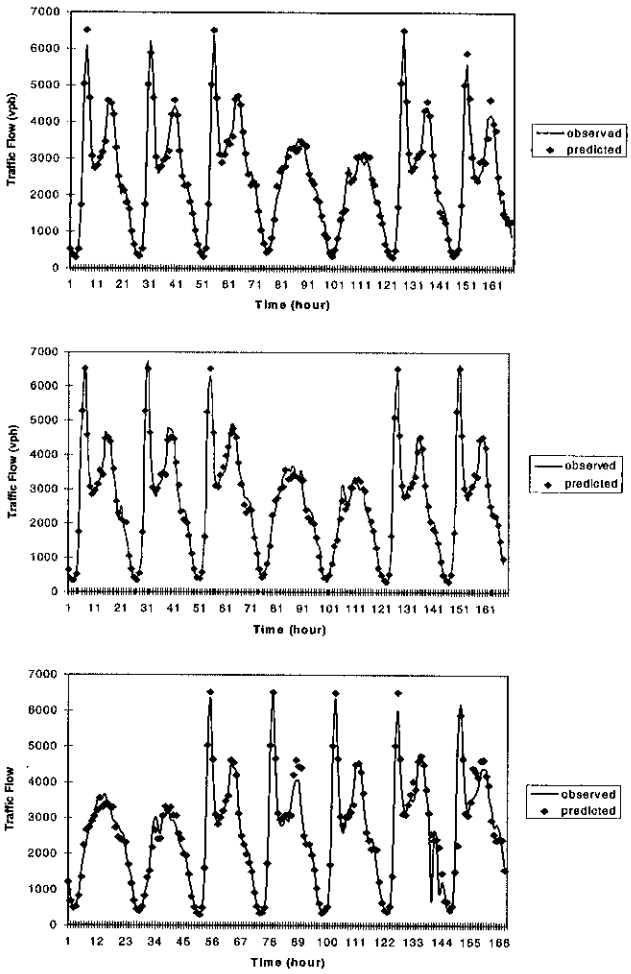

Fig. 7. Traffic flow (actual compared with the predicted value) at sensor station 605 EB for (top) the first week of March 1995, (middle) the third week of March 1995, (bottom) the second week of April 1995.

were computed, the width of each neuron was calculated, and then we proceeded with the weight training via the LMS method (see Figure 7).

Box-Jenkins autoregression integrated moving average (ARIMA) is a traditional regression technique for time-series forecasting.[6,27] It consists of three basic components: autoregressive [AR(p)] component, moving average [MA(q)] component, and integration (I) component. An AR of order p, AR(p), can be expressed as $x_t = a_1 x_{t-1} + a_2 x_{t-2} + \cdots + a_p x_{t-p} + e_t$. An MA of order q, MA(q), can be expressed as $x_t = -(b_1 e_{t-1} + b_2 e_{t-2} + \cdots + b_q e_{t-q}) + e_t$, where e_t is the error term at time t. Integration is the number of differences needed to take in a time series to achieve stationarity. An ARIMA(p, q, d) includes both AR and MA terms in the time series, where p is the order of the autoregression, q is the order of the moving average, and d is the number of differences taken.

The task was then to use identification tools to identify the appropriate order for each of the three basic ARIMA model components. The identification task is usually complex and time-consuming. We compared the results of the RFBNN prediction versus those of the Box-Jenkins method; the RMS error for the RBFNN was one order of magnitude smaller than that of the ARIMA model. As illustrated in Figure 7, the RBFNN yields a better result than the ARIMA model. The training time took less than 5 minutes on a Pentium 100 computer[11,12]; the code is implemented in Borland C++.

4 GOAL SELECTOR: TRAFFIC MANAGEMENT AND ROUTE PLANNING

Dynamic traffic prediction and control provide a promising area for research, development, and the application of intelligent controllers, i.e., hierarchical controllers that use artificial intelligence, mathematical programming, and optimal control theory. The approach calls for a judicious combination of pattern recognition, prediction, system identification, goal selection and optimization, and adaptation and regulation. A tutorial on these technologies as well as reference to traffic management is provided in refs. 1 and 15.

Vehicle path planning is central to the development of advanced traveler information systems (ATIS). An enabling technology for the navigation system is the availability of and improvements to vehicle location systems. These include global positioning systems (GPSs), mobile communication networks, dead-reckoning technology, Argos, Loran-C, Omega, and Lojack.[21] A typical vehicle navigation system contains several modules: sensors for heading, distance traveled, and geographic location; digital map database; data transceiver; user inputs and control commands; and information display and audio/graphic outputs.[13,21] Examples of such GPS routers include Blaupunkt's Berlin RCM 303A mobile information center (designed by Blaupunkt, Robert Bosch GmbH, Hildesheim, Germany), General Motor's TravTek, Oldsmobile's Guidestar, Telepath 100 by Delco Electronics, and Sony's NVX-F160, which uses a GPS with digital street maps developed by Etak, Inc. In this subsection we provide a very brief overview of our approach, methodologies, theoretical formulations, and implementation of route-planning algorithms that provide the driver with "optimal" routes in a timely and memory-efficient manner. These have been extracted from network optimization and search and planning applications in artificial intelligence.

From a representation point of view for a given a list of **m** vertices and **n** edges, and estimated flows on each edge, there are several ways to represent such a road network for computer processing. These include adjacency lists, edge lists, forward star, vertex adjacency matrix, and a vertex-edge incidence matrix.[1] Excellent reviews, bibliographies, and surveys on shortest-path problems and related subjects have been

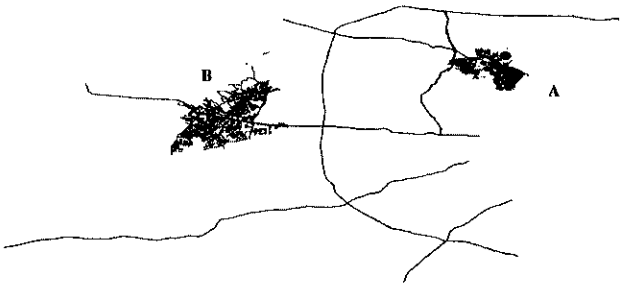

Fig. 8. In searching for a shortest path between A and B, hierarchical segmentation of the network reduces a time-consuming SP problem into three smaller SP problems.

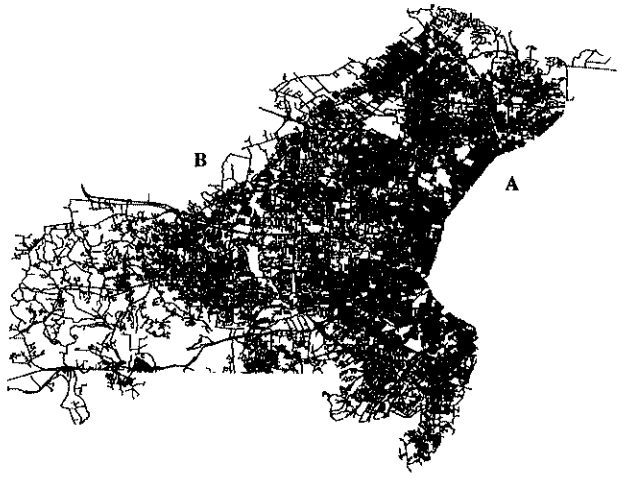

Fig. 9. Finding a path from A to B in the typical metropolitan area represents an SP problem of horrendous magnitude due to the large number of streets involved.

published.[4,10–12,14,18] For this problem, the street/highway databases are typically organized as a set of nodes (vertices) and arcs (edges). The roads can be viewed as arcs in a graph, their intersections being the nodes in the graph. Depending on the problem, the weights associated with the arcs can be either distance, travel time, or scenic utility. Once the graph is defined, several techniques can be applied to find the "best" route to follow. Among them are Dijkstra's Shortest Path, L-Deque, L-2Queue, and Hierarchical Segmentation. How quickly a solution can be found is more directly related to the size of the graph than to the particular shortest-path (SP) algorithm used. Consider a metropolitan area such as St. Louis County, MO. For this locale, the number of arcs in the graph is about 35,000. Any one of the first three SP algorithms mentioned earlier will take on the order of tens of minutes to find a solution; judicious segmentation of the graph and concatenation of several solutions reduce the graph size to several hundred arcs and the overall computation time to a few seconds. Figure 8 shows the road map associated with a routing problem when the Hierarchical Segmentation approach is used. Compare this with the "raw" map shown in Figure 9.

4.1 Shortest-path algorithms for street/highway traffic management

Street/highway databases are typically organized as a set of nodes (vertices) and arcs (edges). A computational complexity summary of several exact solution methods for shortest-path problems (SPP) is given in Table 3. The number of vertices in the graph is n, and the number of edges is m.

The main objectives of our work in this area have been implementation, simulation, testing, and performance evaluation of several algorithms for inter- and intracity dynamic routing. Three shortest-path algorithms were intially coded into MODSIM on the Sun workstation. These algorithms were a partitioning shortest path (PSP) model, an L2-Queue (L2Q) model, and Dijkstra's shortest path (DSP) model. The PSP algorithm is presented in ref. 4; the L2Q algorithm and the DSP algorithm are given in ref. 28. The PSP algorithm is an example of a label-correcting algorithm that was originally proposed by Glover et al. in 1985.[4,12] Let s represent the root node, x is any node, $d(x)$ is the label for node x, $\mathrm{FS}(x)$ is the forward star for node x, and $a(x_1, x_2)$ is the arc length from x_1 to x_2. The pseudocode is as follows:

```
Step 1: d(s) := 0
        d(x) := ∞ for all x ≠ s
        Set the predecessor function p(x) = 0
        for all x
Create two mutually exclusive and
collectively exhaustive lists of unlabeled
nodes call NOW and NEXT.
        Initialize NOW = {s} and NEXT = { }
Step 2: Select any node in NOW, call it node
        u. If NOW is empty, go to step 4.
Step 3: Delete node u from NOW.
        For all v ∈ FS(u)
            if d(u) + a(u, v) < d(v)
            then d(v) := d(u) + a(u, v)
                 p(v) := u
        Add node v to NEXT if it is not
        already in NEXT or NOW. Go to step 2.
Step 4: If NEXT is empty, stop.
        Else k := k + 1. Transfer all nodes
        from NEXT to NOW. Go to step 2.
```

The L2Q algorithm makes use of linked lists in order to implement the set of candidate nodes Q. In particular, Q is implemented by means of a 2-queue. In the following pseudocode, let $P[x]$ be the predecessor of node x, $D[x]$ the label

Table 3
Computational complexity summary for various exact solution methods

Dijkstra's SP	Requires $3n(n-1)/2$ operations; $O(n^3)$ running time. A label-setting method.
Partitioning algorithm	$O(nm)$, with modification $O(n^2)$. A label-correcting method.
Dijkstra's two-tree algorithm	A bidirectional search; Dijkstra's algorithm generated about 50 percent of all vertices to find a SP from s to t; the two-tree method generated 6 percent of the vertices. Good for problems where repeated solutions of SPP are needed.
L-2Queue (Pape-All Dest)	$O(n^2m)$; storage $4n + 2m$. A label-correcting method.
S-Heap (Heap-All Dest)	$O(m \log n)$; storage $5n + 2m$. Uses a binary heap, label-correcting method.
Heap-Select-Dest	A modification of S-Heap; single origin, multiple destination.
PSA (primal sequential all pairs)	$O(n^3)$; finds shortest distances between all pairs of vertices. Performed well for transportation problems: took 20 to 35 percent less time than L-Deque or S-Dial.

of node x, and *pntr* and *last* both represent pointers to the candidate 2-queue Q.

```
For i = 1 to n
    P[i] := s;  D[i] := ∞;  Q[i] := 0
P[s] := D[s] := 0;  Q[n + 1] := last := s;
  Q[s] := pntr := n + 1;
Repeat
    u := Q[n + 1]; Q[n + 1] := Q[u]; Q[u] := 0;
    if pntr = u, then pntr := n + 1;
    if last = u, then last := n + 1;
    For all v, FS(u)
        if D[u] + a(u, v) < D[v], then
            if Q[v] = 0, then
                if D[v] = ∞, then
                    Q[last] := last := v;
                    Q[v] := n + 1;
                else
                    Q[v] := Q[pntr]; Q[pntr] := v;
                    if last = pntr, then last := v;
                    pntr := v
            P[v] := u; D[v] := D[u] + a(u, v);
until Q[n + 1] = n + 1
```

The DSP algorithm represents Q as an unordered linked list. The following pseudocode uses the same notation as in L2Q.

```
For i = 1 to n
    P[i] := s;  D[i] := ∞;  Q[i] := 0
P[s] := D[s] := 0;  Q[n + 1] := s;  Q[s] := n + 1;
Repeat
    min := ∞;  i := n + 1;  x := i;
    While Q[i] ≠ n + 1
        if D[Q[i]] < min, then x := i;
          min := D[Q[i]];
        i := Q[i];
    u := Q[x];  Q[x] := Q[u];  Q[u] := 0;
    For all v ∈ FS(u)
        if D[u] + a(u, v) < D[v], then
            if Q[v] = 0, then Q[v] := Q[n + 1];
              Q[n + 1] := v;
            P[v] := u;  D[v] := D[u] + a(u, v);
until Q[n + 1] = n + 1
```

All programs use the same data structure to represent nodes, arcs, and distances, namely, a forward star. A node is represented as an object. The fields are length (the label length at the node), pred (the node's predecessor), and nodenumber (the number of the node). Because of the way MODSIM treats objects, ASK methods are necessary to change these values at any point in the program. There is then an array of n nodes. An arc is also represented as an object. The fields are length (the distance from start node to end node), arcnum (the number of the arc), startnode, endnode, and roadtype (which may be employed later to use in a more stochastic version). ASK methods are necessary to change the values of each arc when read in from a data file. There is then an array of m arcs.

The data file is arranged as follows. The beginning of the first colunm must be preceded by a tab; otherwise, errors in reading the data will occur. For similar reasons, every subsequent colunm must be separated by a tab. The first colunm is the start node, followed by the end node, length, arc number, and finally the road type. Because the arcs are meant to be undirected, whereas the algorithms assume a directed graph, the data file must be read in twice. At the second reading the first colunm is then interpreted as the end node and the second colunm is the start node. An example of an ArcInfo-formatted street database for St. Louis is given in Table 4.

There are also several common procedures among the programs. Because of the restrictions on an array, the procedure CountNum is required to count the number of arcs in the forward star of node u. Then the array for the forward star of node u can be established, and the forward star can be set up. The disadvantage to this method of setting up the forward star is that it involves a large number of calculations and comparisons, which dominate the preprocessing time. However, it is preferable to have the dynamic array sizes, which allow for less memory use. Procedure SetupFS is very similar to

Table 4
Sample input data structure from a St. Louis city (ArcInfo formatted) GDT street database

FNODE	*TNODE*	*LENGTH*	*DB29189S*	*NAME*	*CFCC*
4375	4374	0.000999	5380	I270	A11
4374	4388	0.001400	5393	I270	A11
4626	4375	0.012890	5714	I270	A11
4388	4635	0.007723	5726	I270	A11
4635	4645	0.000188	5738	I270	A11

Table 5
St. Louis street highway network: Execution times in seconds for various SPP algorithms

	Preprocessing	*L2Q*	*PSP*	*DSP*
Subset of St. Louis city				
$n = 379, m = 996$	9.8139	0.1077	1.009	0.179
St. Louis city				
$n = 9105, m = 28702$	5776.2	4.094	582.04	13.56

CountNum, except that it actually assigns arc to the forward star of each node. FS is a two-dimensional array with integral indices. The first index refers to which node the forward star belongs, and the second index simply numbers the arcs contained in the forward star.

Each program uses a timer to determine how long the program actually takes to complete. The start timer is the first command made, and then the stop timer and time calculation are the last commands made. The timer is used to time both how long the preprocessing takes and how long the actual program runs. This involves starting the timer at the beginning of the program, stopping it at the end of the preprocessing, computing the run time, and then using that end time as the new start time and stopping the timer again at the end of the program for the final run time.

The PSP program required a minor adjustment to allow it to work with undirected arcs. It was necessary to add a third set, USED, to mark which nodes' forward stars had already been examined. The lists NOW, NEXT, and USED were represented by a RankedObj. This is the same as a QueueObj, except that the items in the list are ranked according to some field of the object. The ASK method Rank of the object LabeledNodes was written to order the nodes by their length fields. The ASK method Add(Node) is used to add Node to the list, and the ASK method Remove() returns the top Node off the list while also deleting it from the list. The field Includes(Node) returns a Boolean value to signal whether or not Node is included in the list. The rest of the PSP program, as well as the L2Q and DSP programs, follows the algorithms very closely.

The performance of these algorithms was evaluated for the city of St. Louis with the number of vertices n equal to 9105 and the number of edges m equal to 28,702. For example, for a portion of St. Louis with 379 vertices and 996 edges, the preprocessing took 9 to 10 seconds, whereas computation of the path using the L2Q algorithm took an average of 0.107 second. For the complete graph of St. Louis, with 9105 vertices and 28,702 edges, the preprocessing took about 1.6 hours, and the L2Q algorithm took only 4.1 seconds to compute the shortest path (Table 5). As seen in Table 5, the preprocessing of the street data was the most time-consuming part of the path planning. The actual algorithm itself is considerable quicker. There are several things that can be done to improve the preprocessing performance. The preprocessing does not need to be performed at every run of the program, because none of the forward star information should change. Therefore, the preprocessing could be done a priori, the values saved to a separate file, and then those values are read in at the beginning of each run. This would still involve additional time, but it would be considerably less time. Also, the FS data structure could be changed to a simpler array of integers. The indices would remain the same, but instead of referring to an ArcObj, it could refer to the arc number i. Then any information needed on that arc could be referenced through Arc[i]. This would save considerable memory space, as well as time in reading in the values for FS once the former idea was introduced.

4.2 St. Louis highway traffic management: Routing algorithms

A related problem is to develop a method for routing traffic through the St. Louis highway network given predicted travel times and incident data. There are many system-optimal algorithms available to use. However, we were constrained by hardware and software requirements, which led us to use the L2-Queue method and Dijkstra's shortest-path method. Although these are user-optimal methods, while the problem is a system-optimal problem, these methods provide us with traffic routing information that will be useful in optimizing traffic flow. They will use the estimated travel times on each arc to find the shortest paths from each travel advisory point to each desired destination. This estimate can be updated however often is desired. This will allow for clustering of drivers in such a way that they should not overflow the assigned routes.

The performance was evaluated for the city of St. Louis with the number of vertices n equal to 9 and the number of edges m equal to 26. A picture of this network is shown in Figure 3. Both programs used the same data structures to represent vertices, edges, and distances, namely, a Forward Star. A timer was again used to determine how long each algorithm actually took to complete. Another timer was used to determine how long the street data preprocessing took.

Table 6
St. Louis highway network: Execution times, in seconds, for various SPP algorithms*

Preprocessing	*DSP*	*L2Q*
0.06	0	0

*Corresponding to the major highways depicted in Figure 7 with $n = 9$ and $m = 26$.

The run times are given in Table 6; due to small network size ($n = 9$ and $m = 26$), preprocessing took only 0.06 second, and computation of the fastest (shortest) route took practically 0 seconds. The host platform used for the evaluation was a 486 PC; the algorithms were implemented using C++.

5 CONCLUSIONS

In this paper we have provided an overview of a semantic control approach to traffic management. In our approach to ATMS, all three major functions are incorporated into the semantic controller. The *identifier* measures and/or predicts the road congestion levels and travel times and performs incident detection/localization. The *goal selector* handles street/highway traffic routing for maximum use of available network capacity. The *Adapter* provides driver's support for shortest path/time routing and steering control.

More specifically, in road traffic prediction via RBFNNs, our results show that radial basis function neural networks provide an efficient tool for traffic-flow prediction. The performance of the radial basis function neural network depends on the selection of centers and widths. The simplicity of the K-means clustering algorithm, width calculation, and the least-mean-squared algorithm for weight training make the radial basis function neural network training and learning more efficient and faster than both the backpropagation neural network with a tapped delay line and the Box-Jenkins algorithm. In addition to the use of artificial neural networks (ANNs) in system identification and parameter estimation, there are several other areas that can benefit from ANNs. For example, neurocontrol of uncertain nonlinear dynamic systems is one such area with potential applications to advanced vehicle control systems (AVCSs). For a comprehensive discussion of references and applications of neural networks to the control of nonlinear systems, we refer readers to refs. 19, 38, and 40. Some of the key issues that remain are stability, validation, and verification of hybrid systems resulting from incorporation of the preceding modules into AVCS. In the past these issues have been addressed through the extensive use of simulations, more recently, mathematical analysis has been used to investigate the stability, observability, and controllability of closed-loop systems with neurocontrollers. Good surveys and reference material on ANNs with fuzzy logic controllers can be found in refs. 9, 20, 28, 30, 35, 37, and 39. Used in conjunction with classic synthesis methodologies and multiresolutional algorithms, ANNs and FLCs hold great promise for the future of ATMS and AVCS.

ACKNOWLEDGMENTS

We gratefully acknowledge the helpful suggestions and comments of the guest editor Dr. Azim Eskandarian and anonymous referees.

REFERENCES

1. Amin, S. M., Garcia, A. & Wootton, J. R. (eds.), Network, control, communications and computing technologies for ITS, *Mathematics and Computer Modeling*, **22** (4–7) (1995), 454.
2. Amin, S. M., Rodin, E. Y., Lui, A.-P., Rink, K., Cusick, T., Gosh, B. K., Gerhard, V., García-Ortiz, A. & Wootton, J., A semantic control approach to ITS, in *Proceedings of Intelligent Vehicles '95*, Detroit, MI, Sept. 25–26, 1995, pp. 430–5.
3. Amin, S. M., Intelligent prediction methodologies in the navigation of autonomous vehicles, Ph.D. thesis, Washington University, January 1990.
4. Bertsekas, D. P., *Linear Network Optimization*, MIT Press, Cambridge, MA, 1991.
5. Billings, S., Identification of nonlinear system: A survey, *IEEE Proceedings*, part D (1980).
6. Box, G. E. P. & Jenkins, G. M., *Time Series Analysis: Forecasting and Control*, Holden-Day, Oakland, CA, 1976.
7. Chen, S., Cowan, C. F. N. & Grant, P. M., Orthogonal least squares learning algorithm for radial basis function network, *IEEE Transaction on Neural Network*, **2** (2) (1991).
8. Chen, S. & Billings, S., Representation of nonlinear systems: The NARMAX model, *International Journal of Control*, (1989), 1013–32.
9. Cox, E., *The Fuzzy Systems Handbook*, Academic Press, New York, 1994.
10. Deo, N. & Pang, C., Shortest-path algorithms: Taxonomy and annotations, *Networks*, **14** (1984), 275–323.
11. Dreyfus, S. E., An appraisal of some shortest-path algorithms, *Operations Research*, **17** (1969), 395–412.
12. Evans, J. R. & Minieka, E., *Optimization Algorithms for Networks and Graphs*, 2d ed., Marcel Dekker, New York, 1992.
13. French, R. L., *Automotive Electronics Handbook*, McGraw-Hill, New York, 1995.
14. Gallo, G. S. & Pallotino, S., Shortest-path methods: A unified approach, *Mathematical Programming Study*, **26** (1986), 38–64.
15. Garcia, A., Amin, S. M. & Wootton, J. R., Intelligent transportation systems: Enabling technologies, *Mathematical and Computer Modeling*, **22** (4–7) (1995), 11–81.
16. Geist, D., Semantic control in continuous systems applications to aerospace problems, Ph.D. thesis, Washington University, December 1990.

17. Girosi, F. & Poggio, T., A theory of networks for approximation and learning, also Networks and the best approximation property, technical reports, MIT, 1989.
18. Golden, B. & Magnati, T. L., Deterministic network optimization: A bibliography, *Networks*, **7** (1977), 149–83.
19. Hunt, K., Sparaboro, D., Zibowski, R. & Gawthrop, P., Neural networks for control systems: A survey, *Automatica*, **28** (6) (1992), 1083–1112.
20. Jang, J.-S. R. & Sun, C.-T., Neurofuzzy modeling and control, in *Proceedings of IEEE*, (1995).
21. Jurgen, R. K. (ed.), *Automotive Electronics Handbook*, McGraw-Hill, New York, 1995.
22. Kuo, L. E. & Melsheimer, S. S., Using genetic algorithms to estimate the optimum width parameter in radial basis function networks, in *Proceedings of American Control Conference*, June 1994.
23. Leontaritis, I. & Billings, S., Input-output parametric models for nonlinear systems, *International Journal of Control*, **41** (2) (1985), 303–28.
24. Lirov, Y., Artificial intelligence methods in decision and control systems, Ph.D. thesis, Washington University, August 1987.
25. Liu, A.-P., Amin, S. M. & Rodin, E. Y., Traffic flow estimation via RBFNN, technical report 2-6-95, Center for Optimization and Semantic Control, Washington University, 1995.
26. Liu, A.-P., Amin, S. M. & Rodin, E. Y., Traffic flow modeling via radial basis function neural network and Box-Jenkins method, presented at the Fourth IFORS Specialized Conference on Operations Research and Engineering Design, St. Louis, October 24–27, 1995.
27. Ljung, L., *System Identification: Theory for the User*, Prentice-Hall, Englewood-Cliffs, NJ, 1987.
28. Lubin, J. M., et al., Lateral control of an autonomous road vehicle in a simulated highway environment using adaptive resonance neural networks, technical report, Department of Civil Engineering and Operations Research, Princeton University, 1992.
29. Moody, J. & Darken, C. J., Fast learning in networks of locally-tuned processing units, *Neural Computation*, **1** (1989), 281–94.
30. Neusser, S., et al., Neurocontrol for lateral vehicle guidance, *IEEE Micro*, (1993), 57–66.
31. Pallottino, S., Shortest-path methods: Complexity, interrelations and new propositions, *Networks*, **14** (1984), 257–67.
32. Park, J. & Sandberg, I. W., Universal approximation using radial-basis-function networks, *Neural Computation*, **3** (1991), 246–57.
33. Powell, M. J. D., Radial basis functions for multi-variable interpolation: A review, presented at the IMA Conference on Algorithm for the Approximation of Functions and Data, RMCS, Shrivenharn, U.K.
34. Rodin, E. Y., Semantic control theory, *Applied Mathematical Letters*, **1** (1) (1988), 73–8.
35. Ross, T. J., *Fuzzy Logic with Engineering Applications*, McGraw-Hill, New York, 1995.
36. Weil, R. D., AI methods in utilizing low dimensional models of differential games, Ph.D. thesis, Washington University, September 1990.
37. White, D. A. & Sofage, D. A. (eds.), *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, Van Nostrand Reinhold, New York, 1992.
38. Yager, R. R. & Zadeh, L. A. (eds.), *Fuzzy Sets, Neural Networks, and Soft Computing*, Van Nostrand Reinhold, New York, 1994.
39. Yager, R. R. & Filev, D. P., *Essentials of Fuzzy Modeling and Control*, John Wiley and Sons, New York, 1994.
40. Zbikowski, R., et al., A review of advances in neural adaptive control systems, technical report ES-PRIT III, Project 8039: NACT, Daimler-Benz AG and University of Glasgow, 1994.

APPENDIX A: BACKGROUND ON NEUROIDENTIFICATION AND PREDICTION OF UNCERTAIN NONLINEAR SYSTEMS

In general, traffic modeling and prediction are a special case of system identification in which the traffic flows/densities represent the state of a nonlinear and uncertain dynamic system. In this appendix we provide a brief overview of application of neural networks in modeling and prediction of nonlinear systems.

In the past, classic methods for identifying nonlinear systems often have used static polynomial or sinusoidal nonlinearities with linear dynamical blocks (e.g., see Hammerstein[27] or Weiner operators and Volterra functional series[5]). These provide an adequate representation for a wide class of nonlinear systems, but often several hundred parameters are required to characterize even simple problems. The disadvantages in utilizing such functional series techniques are (1) excessive computational effort spent in estimating unknown parameters, (2) difficulty in interpreting the results, and (3) the need to use special input signals. On the other hand, theoretical and practical applications of artificial neural networks (ANN) have had an enormous revival in recent years. Their use for approximation and modeling of "static" systems has been studied extensively. From a theoretical point of view, it has been proved that even with one layer, feedforward ANNs with an appropriately chosen number of units can approximate any continuous function over a compact domain. These networks can approximate arbitrary nonlinear mappings, generalize, attenuate noise, cluster data, handle both quantitative and/or qualitative data, and are applicable to multivariable systems. Neural networks have been implemented in many real-world applications ranging from economics to control engineering utilizing their celebrated properties for estimation, system identification, and control engineering.

Although a closed mathematical theory for linear time-invariant systems does exist, a comparable overall theory for nonlinear systems is not available. Due to lack of models or mathematical difficulty of suitable identification and control design procedures, nonlinear systems are often linearized around their operating points in order to apply the linear methods. In general, nonlinear systems have been studied on a system-by-system basis. In recent years, a number of authors have addressed issues such as controllability, observability,

feedback linearization, and observer design for a class of affine nonlinear systems (see works of Isidori, Nijmeijer, and Knobloch). The theory is based on a differential geometric approach and consists of generic analysis and design methods. Despite such promising work, general constructive procedures, similar to those available for linear systems, have not been achieved for general nonlinear systems. A considerable progress in nonlinear systems theory will be needed to obtain rigorous solutions to the identification problem. In addition to the analytical difficulties encountered with uncertain and nonlinear dynamic systems even if their mathematical models are known, there are numerous systems in diverse fields that have been only partially modeled or do not have a reasonable model at all. Design of "universal" modules and structures that are capable of identification, prediction, and control remains a fundamental issue. On the other hand, nonlinear relationships can be learned with relative ease by ANNs if sufficiently "interesting" measured data and enough computing power are available. The latter requirement in the applications of ANNs has decreased progressively as a result of fairly rapid development of powerful computer systems in recent years. The utilization of the learning ability of ANNs may eliminate difficult mathematical analysis in solving system identification and control problems where the underlying system dynamics are complex and highly nonlinear.

Since 1990, a few papers not only have demonstrated satisfactory results in applying the approaches of neurocontrol and neuroidentification but also have begun to address fundamental issues such as system approximation and identification, controllability, observability, and stability theory. Although major results in approximation and identification of systems using ANNs are available, only a small group of people is actually familiar with them. Perhaps the most popular ANN structure has been the static multilayer feedforward neural network trained via the backpropagation learning algorithm. Such applications involve approximation of static nonlinear maps; in order to deal with time-varying mappings such as the input-output mappings of a dynamic system, static ANNs were augmented with so-called tapped delay lines at their inputs. With the incorporation of feedback connections and delay elements between the units, static ANNs are made recurrent by construction. Recurrent neural networks (RNNs) are characterized by their internal memory and thus are very suitable for imitating the behavior of dynamic systems. In addition to feedback connections between units, several approaches have distributed dynamic elements throughout the network. The activation of each unit is a time-varying internal state or output of a dynamic systems. Such networks were first proposed by Hopfield and have been rediscovered recently as dynamic neural networks (DNNs) in the context of identification and control of dynamic systems.

While the approximation results for the above-mentioned networks only guarantee the existence of a set of network parameters that allow the approximate realization of a given dynamic system, the question of the uniqueness of this set and whether there exists an algorithm that can actually determine the desired parameter values has to be clarified. In the context of dynamic systems, this involves the problem of neural identifiability and the design of appropriate parameter-estimation algorithms. The parameter-adaptation laws not only have to ensure the convergence of parameters to their desired values but also guarantee the performance of the overall system.

A general single-input, single-output (SISO) nonlinear system can be represented by the deterministic (i.e., process noise and incidents are disregarded) discrete-time input-output description:

$$y(k+1) = f[y(k), y(k-1), \ldots, y(k-n+1); u(k), u(k-1), \ldots, u(k-m+1)].$$

This description is referred to as the *nonlinear autoregressive moving-average model* (NARMAX).[23] According to ref. 8, the NARMAX model is a natural representation for sampled nonlinear continuous-time system. It is assumed that f is continuous, but not given explicitly. In fact, for many real sampled nonlinear systems, their NARMAX models are very difficult to determine, and in general, the nonlinear structure of f is unknown. Given a finite number of measurements, i.e., the current and delayed samples of inputs and outputs of a system (a traffic network), one may pose the question of whether there exists a representation consisting of a finite number of known functions and real parameters that approximate the mapping f. Notice, however, that the approximation capability of a given parametrized ANN does not imply the existence of a convergent identification algorithm for the parameters of this model. Thus approximation theory provides a pure existence result, whereas identification mainly deals with a constructive procedure. Nevertheless, before considering the identification problem, i.e., how to find the appropriate parameter values for a given set of data, the fundamental representation problem must be addressed. Answers must be given to the fundamental questions such as Which class of mappings f can be approximated by which models, and how well?

As mentioned earlier, ANNs provide a promising approach, where fast parallel computation using relatively inexpensive hardware is hoped to offer substantial benefits. To reconstruct dynamic behavior, it is necessary to store past information about the system inputs, states, and outputs in the network. A network must have a memory; this can be realized in one of several ways via

1. Use of tapped delay lines in feedforward ANNs (TDNNs)
2. Use of general recurrent ANNs as well as DNNs and recurrent higher-order NNs (RHONN)
3. Use of RBFNNs

In our past work in identification and control of uncertain

nonlinear systems arising in aerospace and process-control applications, we have made use of the first two classes of ANNs. For problems of traffic estimation and prediction, we have adopted the third approach. This approach, discussed in Section 2 of this paper, was motivated partially by the "gaussian-like" behavior of the data.