International Journal of Fuzzy Logic and Intelligent Systems, vol. 5, no. 4, December 2005 pp. 360-366

Real Time Traffic Signal Plan using Neural Network

Myeong-Bok Choi, You-Sik Hong

Department of Admin. Computer, Wonju National College, Wonju, Korea
Department of Computer Science, SangJi University, Wonju, Korea
TEL: (033) 742-1121 / FAX (033) 730-0480

Abstract

In the past, when there were few vehicles on the road, the T.O.D.(Time of Day) traffic signal worked very well. The T.O.D. signal operates on a preset signal cycling which cycles on the basis of the average number of average passenger cars in the memory device of an electric signal unit. Now days, with increasing many vehicles on restricted roads, the conventional traffic light creates startup-delay time and end-lag-time. The conventional traffic light loses the function of optimal cycle. And so, 30-45% of conventional traffic cycle is not matched to the present traffic cycle. In this paper we proposes electro sensitive traffic light using fuzzy look up table method which will reduce the average vehicle waiting time and improve average vehicle speed. Computer simulation results prove that reducing the average vehicle waiting time which proposed considering passing vehicle length for optimal traffic cycle is better than fixed signal method which doesn't consider vehicle length.

Key Words : Real Time Traffic Signal, Optimal Green Time, Spillback, Neural Network, Predictive Filter

1. Introduction

These days, the role of the traffic signal is very important when the volume of traffic can't be predicted. When there are a lot of running vehicles at an intersection, the signal cycle should be extended and when there are few running vehicles the signal cycle should be shortened. Most research has focused on low-saturated traffic conditions [1,2,3,4].

Only a few studies have investigated traffic control for high-saturated traffic conditions [5,6,7]. In order to produce traffic optimal signal cycle we must first check how many waiting cars are in the lower intersection. If there are a lot of cars between the two intersections there may not be enough space for cars to pass through the lower intersections. T.O.D. traffic signal systems simply repeat the fixed preset traffic signal cycle. Creating end-lag-time and startup-delay loses time when queue length at the lower traffic intersection is bigger than the capacity of the upper traffic intersection.

Electro sensitive traffic systems can not consider passenger car unit. Because of this, it causes start up delay time and passenger waiting time. In this paper, it antecedently creates optimal traffic cycle of passenger car unit at the bottom traffic intersection. Mistakes can be made due to different lengths, car speed and width of road. However, it continues to reduce the car waiting time and start-up delay time using fuzzy control of feed-back data.

Moreover, to prevent spillback, it can adapt control even though upper traffic intersection has a different vehicle length, road slope and road width. In this paper we used fuzzy membership function vary between 0 and 1 which estimate uncertain length of vehicle, vehicle speed and width of road. Fuzzy neural networks can accommodate uncertain traffic conditions very easily.

If we improve average traffic speed by 10-15%, it will save 2 million dollars per year. Since traffic congestion has increased steadily in urban traffic networks, route guidance systems have been proposed to avoid traffic congestion links and inform the shortest travel time route of traffic networks. Moreover, cycle lengths and green time must be adaptively controlled according to the variation of incoming traffic volume, and change more drastically than actual measurement values. To solve this problem, this paper proposes a new concept of optimal green time using fuzzy neural network. Using computer simulation, we prove that the spillback phenomenon generated under highly saturated traffic condition is improved using fuzzy logic and neural networks. This paper is organized as follows: Section 2 briefly explains the problems of conventional traffic lights. Section 3 discusses the determination of optimal traffic cycles using a neural network and fuzzy logic computer simulation. Section 4 describes simulation results. Finally, Section 5 will give our conclusions.

2. Problems Using Conventional Traffic Light

Looking at Fig. 1. there are 6 vehicles waiting at the lower traffic intersection. Fig. 1. with associated table 1 shows that the waiting automobiles consist of 4 small vehicles and 2 medium size vehicles. From the calculations, all 6 vehicles may pass to the upper traffic intersection during the green time, since the avaliable distance is 30 meters and only 28.5

Manuscript received Aug. 31, 2005; revised Oct. 3, 2005.
This research was supported by SangJi University research fund, 2005.

meters are required to passing. Fig. 2. shows the waiting automobiles consist of 3 large vehicles and 3 medium sized vehicles. In this case only the first 3 vehicles may pass to the upper traffic intersection during the green time since the total distance required for 3 vehicles is 29 meters. If all 6 vehicles were to pass to the upper intersection, a spillback would be created, since a minimum distance of 48.5 meters would be required to hold the vehicles. A detailed description is presented in Section 3 which describe how to prevent this spill back phenomenon from occurring.

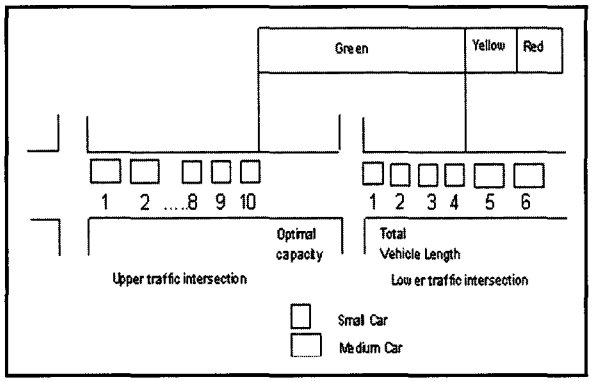

Fig. 1. Greentime depending on waiting vehicle queue 1.

$$Optimal\ Capacity < Upcap - Occv \quad (1)$$

$$Optimal\ Capacity : \sum_{n=1}^{6} q(i) \quad (2)$$

$q(i)$: $(4+4+4+3.5+6+7) = 28.5 meter$

Upcap: Maximum upper intersection capacity
Total waiting vehicles length
Upcap equals 100 *meter*

Occv: Total occupied distance at the upper intersection
Occv equals 70 *meter*

(4+4+4+3.5+6+7)=28.5 METER < 30 METER

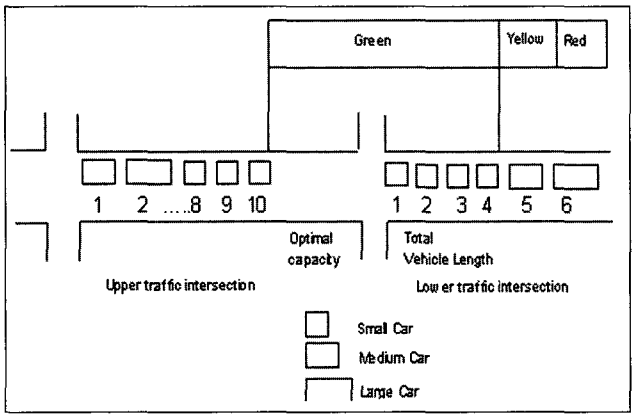

Fig. 2. Green time depending on vehicle waiting queue 2.

We present a system for coordinating green time which controls 10 traffic intersections. For instance, if we have a baseball game at 8 pm today, traffic volume toward the baseball game will be increased 1 hour or 1 hour and 30 minutes before the baseball game. At that time we can not estimate optimal green time. Therefore, we used fuzzy neural network to estimate uncertain optimal green time and reduce vehicle waiting time. Fuzzy neural networks can accommodate uncertain traffic conditions very easily.

Table 1. Waiting queue length consisting of small-medium-large vehicles at the lower traffic intersection

passing cars	length	passenger car unit
1 (small)	4 meter	1.3
2 (med)	6.5 meter	1.5
3 (med)	6 meter	1.5
4 (med)	7 meter	1.6
5 (Large)	12 meter	1.7
6 (Large)	13 meter	1.8

$$Optimal\ Capacity < Upcap - Occv \quad (3)$$

$$Optimal\ Capacity : \sum_{n=1}^{6} q(i) \quad (4)$$

$q(i)$: $(4+6.5+6+7+12+13) = 48.5 meter$

Upcap: Maximum upper intersection capacity
Total waiting vehicles length
Upcap equals 100 *meter*

Occv: Total occupied distance at the upper intersection
Occv equals 70 *meter*

(4+6.5+6+7+12+13)=48.5 METER > 30 METER

In this paper, it antecedently creates an optimal traffic cycle of passenger car units at the bottom traffic intersection. Mistakes are possible due to different car lengths, car speed and length of intersection. Therefore, it consequently reduces the car waiting time and start-up delay time using fuzzy control of feed-back data.

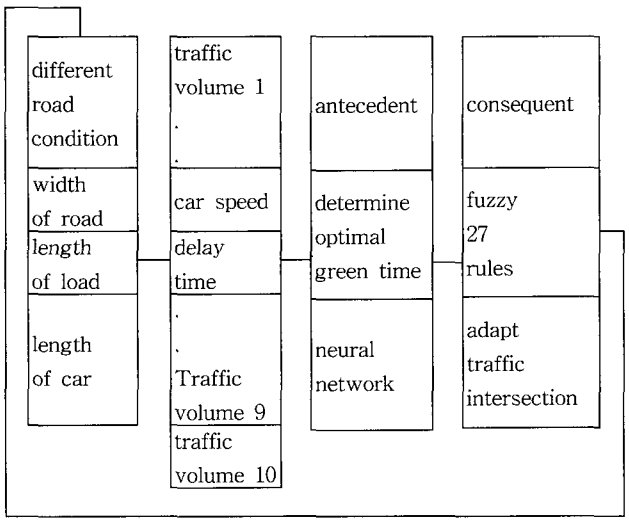

Fig. 3. Block diagram of optimal traffic cycle light using

fuzzy neural network

Moreover, to prevent spillback, it can adapt control even though upper traffic intersection has a different vehicle length, road slope and road width. Fig. 3. shows a block diagram of an optimal traffic cycle light using fuzzy neural network. The diagram holds the network's ability to reduce vehicle waiting time and to determine optimal green time, adapting to any different type of traffic intersection.

In order to solve spillback problems, we must determine which car is big or small. However, traffic intersection length, width of lane and number of lanes in the intersection is different. It adapts to the different traffic intersection types and sizes, while using the fuzzy 27 rules.

3. Predictive Filter for Software Reliability Prediction

An adaptive filter is created by combining a tapped delay line (TDL) with an ADALINE (Adaptive Linear Neuron Network) [8]. The ADALINE network is similar to the perceptron, but their transfer function is linear rather than hard-limiting. This network can only solve linearly separable problems. To solve a nonlinear problems, we need a new component, the TDL, to make full use of the ADALINE network. The adaptive filter can automatically adapts in the face of environment changes. Fig. 4. shows a simple description of the adaptive filter, in which x_i is the input signal at time i, $\hat{y}_i = \sum_{k=0}^{d} w_k x_{i-k} + b$ is the corresponding output, d is the number of delays, w_k are adaptive weights, and b is the bias (or threshold).

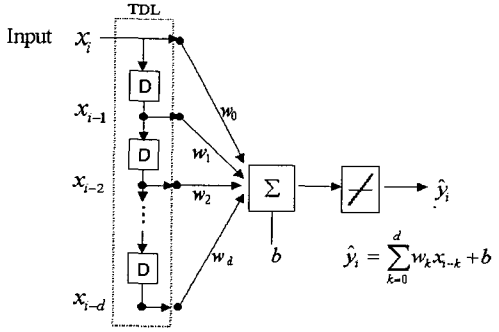

Fig. 4. Adaptive Filter

This network is usually referred to, in the digital signal processing field, as a finite impulse response (FIR) filter (also known as input-delay neural network). This architecture has been used quite successfully in a number of practical applications including speech recognition and time series prediction. If we adjust the adaptive weights in a way that the corresponding output $\hat{y}_i$ matches as best as possible a desired signal (target) y_i, the adaptive filter can predict the future values of the signal. We thus use the network described in Fig. 5. for the failure count data. This network may be called the predictive filter, which is a modified adaptive filter. Specific modifications are : (1) m_i is not connected to the linear neuron filter; (2) errors are computed and backpropagated to adjust connection weights.

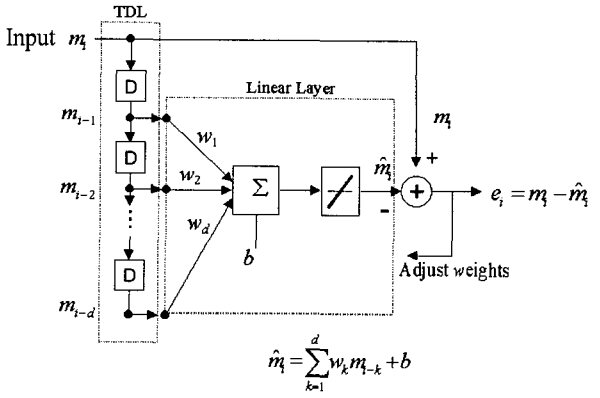

Fig. 5. Predictive Filter

The cumulative number of failures found up to t_i, is then predicted by $\hat{m}_i = \sum_{k=1}^{d} w_k m_{i-k} + b$, which is a linear combination of m_{i-k}, $k = 1, 2, \cdots, d$. This implies that the predictive filter primarily aims the next-step prediction. However, if we use $\hat{m}_i$ as the input signal at time i and m_{i-k}'s as the corresponding delayed signals, we can obtain $\hat{m}_{i+1}$ from the predictive filter. Repetition of this procedure enables us to predict $\hat{m}_{i+k}$, $k = 0, 1, \cdots$ at time $i-1$. At this time we should note that the observation time intervals, $t_i - t_{i-1}$, are required to be constant for the application of the above predictive filter. On the other hand, we need to replace m_i in Fig. 5. with s_i for applying the predictive filter to the failure time data. That is, $\hat{s}_i = \sum_{k=1}^{d} w_k s_{i-k} + b$ and $e_i = s_i - \hat{s}_i$.

4. Prediction Experiments and Results

This section empirically studies predictive accuracy of the predictive filter suggested in the previous section. Three distinct approaches that are very common in software reliability research community are goodness-of-fit, next-step predictability, and variable-term predictability [9]. Using these approaches, a two-component predictability measure consisting of average relative prediction error (AE) and average bias (AB) was proposed by Malaiya, Karunanithi, and Verma [9]. AE is a measure of how well a model predicts through the test phase, and AB is the general bias of the model. In this paper, we use the AE measure for next-step predictability. For 14 different software projects, the next-step AE of predictive filter is computed and compared with those of other well-known neural networks and SRGMs simulated by Karunanithi, Whitley, and Malaiya [10]. Generally, we assume that the data set size is sufficiently large when its size is larger than 30 (referred to statistical field). Data set size, n, of Data1, Data3-6, and Data10 are not large enough for the neural net-

work experiments. Furthermore the observation time intervals $t_i - t_{i-1}$ of Data4 and Data5 are not constant. Therefore, the predictive accuracy of the predictive filter evaluated for these data sets is expected to be low. Nevertheless, we also performed experiments for these data sets for the sake of reference.

Let l denote the training data set size. And r is the prediction distance. For example, $r=1$ for the next-step prediction and $r=(n-l)$ for the end-point prediction. For given number of delays d, the next-step prediction experiments proceed as follows :

Step (1) Set $l=d$.

Step (2) Train each neural network model by using the LMS (Least Mean Square error) algorithm and the training data set $m_1, m_2, \cdots, m_l$.

Step (3) For each trained network, predict j step ahead cumulative number of failures $\hat{m}_{l+j}$ for $j=1,2,\cdots,r$ and then compute corresponding relative prediction errors, $e_{lj}=(\hat{m}_{l+j}-m_{l+j})/m_{l+j}\cdot 100$.

Step (4) Increase l by 1 and repeat Steps (2) and (3) until $l=n-r$.

Step (5) Compute the average of e_{lj} over l.

The average of e_{lj} over l is denoted by $\bar{e}_{\cdot j}$ and referred to as the AE. Karunanithi, Whitley and Malaiya [10] considered the averages of e_{l1} and $e_{l(n-l)}$ over l as the predictability measures, which correspond to the next-step and end-point prediction errors, respectively. Since the predictive filter primarily aims the next-step prediction, we will use $\bar{e}_{\cdot 1}$. For SRGMs $\hat{m}_{l+j}$ is the estimate of the expected number of failures up to t_{l+j} obtained from m_j, $j=1,2,\cdots,l$. Then e_{lj} and $\bar{e}_{\cdot 1}$ for SRGMs can be similarly obtained.

The most critical problem in implementing the predictive filter is to determine the appropriate value of d. To find a suitable value of d, we follow a trial-and-error approach. That is, Steps (1)-(5) are performed for $d=0,1,\cdots$ and the resulting $\bar{e}_{\cdot 1}$'s are plotted against d. An appropriate value of d is selected by examining the plot. For example, Fig. 6. shows the plot of $\bar{e}_{\cdot 1}$ against d for Data 7.

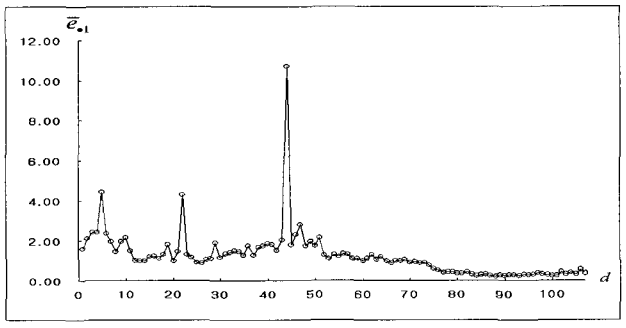

Fig. 6. Plot of $\bar{e}_{\cdot 1}$ against d for Data7

Examining Fig. 6., we find that $\bar{e}_{\cdot 1}$ achieves its minimum when $d > 70$. However, it is apparent that overfitting occurs when d is too large. We thus consider the cases where d is small. $\bar{e}_{\cdot 1}$'s for $d=1,8,11-18,\cdots$ are practically comparable to each other. Since a simpler model is usually desirable, we select 1 as an appropriate value of d. Fig. 7. shows m_i and $\hat{m}_i$ for $d=1$. Table 2 shows selected values of d for the 10 failure count data sets. Using the values of d in Table 2, Steps (1)-(5) were performed for each data set.

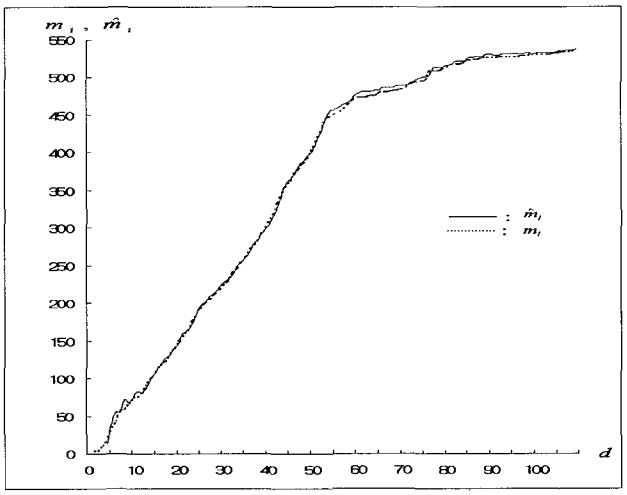

Fig. 7. Plot of m_i and $\hat{m}_i$ for Data7 ($d=1$)

Table 2. Selected values of d for 10 failure count data sets.

Data Set	Data 1	Data 3	Data 4	Data 5	Data 6	Data 7	Data 8	Data 9	Data 10	Data 14
Number of Delays	1	1	4	2	1	1	1	1	3	1

The resulting $\bar{e}_{\cdot 1}$'s were presented and compared with those of Karunanithi, Whitley and Malaiya [10] in Table 3. The numbers in parentheses are the corresponding ranks. Experiment results for the failure count data suggest the followings :

(1) The predictive filter performs well when the data set is large. (Data7-9) That is, the predictive filter is efficient for software reliability prediction especially when the data set size is large;

(2) When the data set size is small, all models seem to have almost similar next-step prediction accuracy. The neural network models and SRGMs are either too pessimistic or too optimistic. Therefore, there is no model that works best for all software projects when the data set is small.

(3) The number of hidden units in the neural network models simulated by Karunanithi, Whitley and Malaiya [10] varied from 0 to 4 depending on the size of the training data set and the nature of the data set. But in the predictive filter, the number of delays is 1 for 7 data sets out of 10 data sets and does not exceed 4. That is, one or two delays is enough for good predictability;

(4) Most of the SRGMs have only 2 or 3 parameters. The number of parameters in the predictive filter is $d+1$.

Since $1 \le d \le 4$ for the 10 data sets, the number of parameters for good prediction is almost the same with those for SRGMs.

Table 3. $\bar{e}_{.1}$ for 10 Failure Count Data Sets

Model		Next-step Average Prediction Error (Rank)									
		Data 1	Data 3	Data 4	Data 5	Data 6	Data 7	Data 8	Data 9	Data 10	Data 14
Neural Networks	Predictive Filter	7.53 (6)	10.56 (10)	6.27 (7)	8.48 (5)	4.47 (7)	1.61 (1)	2.47 (1)	3.54 (1)	9.59 (10)	3.53 (5)
	FFN-Generalization	9.37 (9)	8.44 (9)	5.28 (2)	10.00 (8)	4.33 (6)	3.66 (9)	6.56 (8)	11.82 (9)	5.90 (7)	4.56 (8)
	FFN-Prediction	5.61 (2)	6.84 (5)	4.64 (1)	6.95 (4)	4.51 (8)	3.74 (10)	5.24 (7)	6.72 (3)	7.45 (9)	4.80 (9)
	JordanNet-Generalization	6.79 (4)	6.84 (5)	8.84 (9)	5.09 (1)	5.25 (10)	2.97 (6)	9.11 (10)	9.72 (5)	4.08 (4)	4.86 (10)
	JordanNet-Prediction	4.66 (1)	6.03 (2)	6.11 (5)	8.67 (6)	5.24 (9)	2.90 (4)	3.73 (3)	6.41 (2)	3.22 (1)	4.50 (7)
Statistical SRGMs	Logarithmic	7.33 (5)	7.78 (7)	5.93 (3)	6.42 (3)	3.47 (1)	2.88 (3)	4.47 (6)	10.20 (6)	3.75 (3)	3.24 (3)
	Inverse Polynomial	9.21 (8)	6.17 (3)	7.95 (8)	9.71 (7)	3.59 (4)	2.92 (5)	4.45 (5)	9.09 (4)	4.99 (5)	3.13 (1)
	Exponential	6.67 (3)	7.85 (8)	6.01 (4)	6.15 (2)	3.51 (2)	2.50 (2)	4.26 (4)	13.06 (10)	3.28 (2)	3.52 (4)
	Power	11.15 (10)	6.55 (4)	9.40 (10)	23.36 (10)	3.51 (2)	3.19 (8)	7.06 (9)	11.36 (8)	5.53 (6)	3.20 (2)
	Delayed S-shape	8.03 (7)	5.99 (1)	6.25 (6)	10.90 (9)	3.63 (5)	3.07 (7)	3.39 (2)	10.86 (7)	6.57 (8)	3.55 (6)

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.util.*;
import java.io.*;
import javax.microedition.io.*;

public class Weather extends MIDlet implements
CommandListener {
  private Command exitCommand, goCommand,
          backCommand;
  private Display display;
  private Form locationScreen;
  private TextField cityField, stateField;
  private Form conditionsScreen;
  private StringItem locationItem, conditionsItem,
          temperatureItem, humidityItem, windItem;

 public Weather() {
    // Get the Display object for the MIDlet
    display = Display.getDisplay(this);

    // Create the Exit, Go, and Back commands
    exitCommand = new Command("Exit",
    Command.EXIT, 2);
    goCommand=new Command("Go", Command.OK,  2);
    backCommand  =  new  Command("Back",
Command.BACK, 2);

    // Create the location screen form
    locationScreen = new Form("Enter Location");
    cityField = new TextField("City", "", 25, TextField.ANY);
    locationScreen.append(cityField);
    stateField = new TextField("State", "", 2, TextField.ANY);
    locationScreen.append(stateField);

    // Set the Exit and Go commands for the location screen
    locationScreen.addCommand(exitCommand);
    locationScreen.addCommand(goCommand);
    locationScreen.setCommandListener(this);

    // Create the conditions screen form
      conditionsScreen = new Form("Current Conditions");
    locationItem = new StringItem("", "");
    conditionsScreen.append(locationItem);
    conditionsItem = new StringItem("", "");
    conditionsScreen.append(conditionsItem);
    temperatureItem = new StringItem("", "");
    conditionsScreen.append(temperatureItem);
    humidityItem = new StringItem("", "");
    conditionsScreen.append(humidityItem);
    windItem = new StringItem("", "");
    conditionsScreen.append(windItem);

    // Set the Back command for the conditions screen
    conditionsScreen.addCommand(backCommand);
    conditionsScreen.setCommandListener(this);
  }

  public void startApp() throws MIDletStateChangeException
{
    // Set the current display to the location screen
    display.setCurrent(locationScreen);
  }

  public void pauseApp() {
  }

  public void destroyApp(boolean unconditional) {
  }

  public void commandAction(Command c, Displayable s) {
    if (c == exitCommand) {
      destroyApp(false);
      notifyDestroyed();
    }
    else if (c == goCommand) {
      // Get the conditions for the city and state
      getConditions(cityField.getString().toUpperCase(),
        stateField.getString().toLowerCase());
    }
    else if (c == backCommand) {
      // Clear the location fields
      cityField.setString("");
      stateField.setString("");

      // Set the current display back to the location screen
      display.setCurrent(locationScreen);
    }
  }
```

```
private void getConditions(String city, String state) {
  StreamConnection conn = null;
  InputStream in = null;
  StringBuffer data = new StringBuffer();

  try {
    // Open the HTTP connection
    con= (StreamConnection)Connector.open
         ("http://iwin.nws.noa a.gov/iwin/" +
           state + "/hourly.html");

    // Obtain an input stream for the connection
       in = conn.openInputStream();

    // Read a line at a time from the input stream
    int ch;
    boolean done = false;
    while (((char)(ch = in.read()) != '\003') &&
            (ch != -1) && !done) {
      if (ch != '\n') {
        // Read the line a character at a time
        data.append((char)ch);
      }
      else {
        // Make sure the line is long enough
        if (data.length() >= city.length()) {
          // See if the line starts with the city name
          if ((city.length() > 0) &&
            (data.toString().substring(0,city.length()).
            compareTo(city) == 0)) {
            // Fill in the conditions string items
            locationItem.setText(city + ", " +state.toUpper
            Case());
            conditionsItem.setText("Cond.: " +
              data.toString().substring(15, 22));
            temperatureItem.setText
             ("Temperature: " +
             data.toString().substring(25+27)+ \260');
             humidityItem.setText
             ("Rel. Humidity: " +
             data.toString().substring(33,35) + "%");
             windItem.setText("Wind: " +
             data.toString().substring(36,44)+
             " mph");

            // Set the current display to the conditions
            screen
            display.setCurrent(conditionsScreen);

            // We're done, so bail out of the outer while
            loop
            done = true;
          }
        }
        // Clear the string for the next line
        data = new StringBuffer();
      }
    }

    // The done flag tells us if there was a problem
    if (!done)
      display.setCurrent(new Alert("Weather",
      "The location is invalid. Please try another.", null,
      AlertType.ERROR));
  }
  catch (IOException e) {
    System.err.println("The connection could not be
established.");
  }
 }
}
```

5. Conclusion

With traffic constantly increasing at lighted intersections, neural networks in conjunction with fuzzy logic will fit extremely well into today's traffic conditions. Remember that the T.O.D. method mentioned relies solely on a predetermined cycling time which remains constant. This means that the T.O.D. system can not adjust the green time to the current traffic conditions for optimal traffic flow. An electro-sensitive traffic light system was shown to extend the traffic cycle when there are many vehicles passings on the road or reduce the cycle if there are few vehicles. However, it can not determine which vehicle is long or short. When this happens overflows or the spill back phenomenon occur and waiting time is increased. On the other hand, we saw that a fuzzy neural network analyzes the number of passing vehicles to predict the P.C.U. and to determine the optimal traffic cycle.

The conventional method was shown to produce a much longer waiting time as well as create spill back since it can not adjust for traffic conditions. In summary, not only will neural networks with fuzzy logic dramatically reduce vehicle waiting time and increase overall traffic efficiency, but it will also make a dramatic dent in decreasing energy costs. In this paper, we implemented an agent-oriented multiple crossroad simulator with n × n intersections to evaluate the performance of the traffic signal control algorithms. We utilized agent-oriented paradigm to extend or maintain developed system easily. Since our real traffic situation is suitable to design agents on urban road networks, we chose agent-oriented scheme to develop traffic simulator. The developed simulator can be tested specific traffic situations by changing input parameters like traffic volume sets. The proposed FLC has been applied to crossroads with various traffic flow rates. The simulation results indicate that the proposed FLC yields lower average delay and lower average cost than conventional controllers. This verifies that the proposed method fits the real needs at traffic junctions.

Further research is needed to develop more coordinated approach to solve cooperation and negotiation between neighboring traffic junctions. Also, protecting spillback phenomenon of

multiple crossroads, we have to consider car size such as small automobile and large car, i.e., bus or trailer. In this paper, we have simulated on the crossroads, but real road has several traffic conditions including crossroads, 3-branch street, 10-branch street, and so on. So the experiments are needed for various practical traffic conditions.

References

[1] Allsop, R. E., "Delay at a Fixed Time Traffic Signal. I : Theoretical Analysis", Transp. Sci., 6(3), pp. 260-285, 1972.

[2] K. G. Courage and S. M. Parapar, "Delay and Fuel consumption at Traffic Signals", Traffic Engineering, Vol.45, Nov. pp. 23-27, 1975.

[3] Werner Brilon and Ning Wu, "Delay at Fixed Time Traffic Signals under Time Dependent Traffic conditions", Traffic Engineering Control, 31(12), pp. 623-631, 1990.

[4] C. P. Pappis, E. H. Mamdani, "A Fuzzy Logic Controller for a Traffic Junction", IEEE Trans. Syst., Man, Cybern., 7(10), pp. 707-717, 1977.

[5] M. Jamshidi, R. Kelsey, K. Bisset, "Traffic Fuzzy Control: Software and Hardware Implementations", Proc. 5th IFSA, pp. 907-910, Seoul, Korea, 1993.

[6] R. Hoyer, U. Jumar,"Fuzzy Control of Traffic Lights", Proc. 3rd IEEE International Conference on Fuzzy Systems, pp. 1526-1531, Orlando, U.S.A., 1994.

[7] Hong, YouSik and Park, ChongKug, "Considering Passenger Car Unit of Fuzzy Logic", Proc. of the sixth international fuzzy system association, IFSA, pp.461-464, 1995.

[8] B. Widrow and M. E. Hoff, Jr., "Adaptive switching circuits", In 1960 IRE Western Electric Show and Convention Record, Part 4, pp. 96-104, August 23, 1960.

[9] Y. K. Malaiya, N. Karunanithi, and P. Verma, "Predictability of Software Reliability Models", IEEE Trans. on Reliability, Vol. 41, No. 4, pp. 539-546, 1992.

[10] N. Karunanithi, D. Whitley and Y.K. Malaiya, " Applying Neural Networks to Software Reliability Prediction", IEEE Software, pp. 53-59, July 1992.

Myeong-Bok Choi
was born in CheongJu, Korea, in 1966. He received the B.S. degree in Computer engineering from the Hoseo University, ChungNam, Korea, in 1992, the M.S. degree in Computer engineering from the Ajou University, Suwon, Korea, in 1994, and the Ph.D. degree in Computer engineering from the Ajou University, Suwon, Korea, in 2001. He is currently Associate professor in the Department of Administration Computer Science, National Wonju College, Wonju, Korea. His research interests are in the areas of Artificial Intelligence, Fuzzy and Neural Applications, Intelligent Information Retrieval, Intelligence Traffic System and Thesaurus.

You-Sik Hong
was born in Seoul, Korea, in 1959. He received the B.S. degree in Electronic engineering from the KyungHee University, Seoul, Korea, in 1983, the M.S. degree in electronic engineering from New York Institute of Technology , New York, U.S.A., in 1989, and the Ph.D. degree in electronic engineering from the KyungHee University, Seoul, Korea, in 1997. From 1989-1990, he was a research engineer at the Samsung company in Korea. He is currently Full professor in the Department of Computer Science, Sangji University, Wonju, Korea. His research interests are in the areas of Fuzzy Rule, Expert System, Fuzzy Expert System, Intelligence Traffic System and Neural Network Problems.