TD100: AZ 94-383

ARIZONA DEPARTMENT OF TRANSPORTATION

REPORT NUMBER: FHWA-AZ94-383

RHODES PROJECT: PHASE II (A)

Final Report

Prepared by:
Larry Head
Pitu Mirchandani
Systems and Industrial Engineering Department
University of Arizona
Tucson, Arizona 85721

July 1994

Prepared for:
Arizona Department of Transportation
206 South 17th Avenue
Phoenix, Arizona 85007
in cooperation with
U.S. Department of Transportation
Federal Highway Administration

129984

Technical Report Documentation Page

1. Report No. FHWA-AZ-94-383	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle RHODES Project:Phase II (a)		5. Report Date July 1994
		6. Performing Organization Code
7. Author(s) Larry Head and Pitu Mirchandani		8. Performing Organization Report No. SIE Dept., University of Arizona
9. Performing Organization Name and Address Systems and Industrial Engineering Department University of Arizona Tucson, AZ 85721		10. Work Unit No.
		11. Contract or Grant No. HPR-PL-1(43)383
12. Sponsoring Agency Name and Address **ARIZONA DEPARTMENT OF TRANSPORTATION** **206 S. 17TH AVENUE** **PHOENIX, ARIZONA 85007**		13.Type of Report & Period Covered Final Report 5/92 -1/93
		14. Sponsoring Agency Code

15. Supplementary Notes

Prepared in cooperation with the U.S. Department of Transportation, Federal Highway Administration

16. Abstract

This report documents the work performed on the *RHODES* Project Phase II(a). This research effort was the continuation of the *RHODES* Project Phase I. Phase I explored concepts for models and algorithms for a real-time traffic-adaptive control systems for street networks, referred to as the *RHODES* System. Phase II(a) focused on further development of some of these algorithms and on performing some preliminary laboratory experiments with these algorithms using simulation models.

The control architecture of *RHODES* is based on a hierarchical decomposition of the overall traffic control problem. In an aggregate sense, there are three levels in the control hierarchy: *network load control*, *network flow (platoon) control*, and *intersection (vehicular) control*. *RHODES* architecture allows for a modular implementation of many of the subsystems within the control structure and the incorporation of IVHS technologies (e.g. new vehicle sensors) when they become available. In Phase II(a), the decision problems at each of the hierarchical levels were further analyzed and the decision model/algorithm at the intersection level was explicitly formulated, solved, and evaluated using simulation models.

17. Key Words Traffic Control Systems, intersection Control, Real-Time TrafficAdaptive Control, Simulation, Optimization.		18. Distribution Statement Document is available to the U.S. public through the National Technical Information Service, Springfield, Virginia 22161		23. Registrant's Seal
19. Security Classification Unclassified	20. Security Classification Unclassified	21. No. of Pages 170	22. Price	

METRIC (SI*) CONVERSION FACTORS

APPROXIMATE CONVERSIONS TO SI UNITS

LENGTH

Symbol	When You Know	Multiply By	To Find	Symbol
in	inches	2.54	centimetres	cm
ft	feet	0.3048	metres	m
yd	yards	0.914	metres	m
mi	miles	1.61	kilometres	km

AREA

Symbol	When You Know	Multiply By	To Find	Symbol
in^2	square inches	645.2	centimetres squared	cm^2
ft^2	square feet	0.0929	metres squared	m^2
yd^2	square yards	0.836	metres squared	m^2
mi^2	square miles	2.59	kilometres squared	km^2
ac	acres	0.395	hectares	ha

MASS (weight)

Symbol	When You Know	Multiply By	To Find	Symbol
oz	ounces	28.35	grams	g
lb	pounds	0.454	kilograms	kg
T	short tons (2000 lb)	0.907	megagrams	Mg

VOLUME

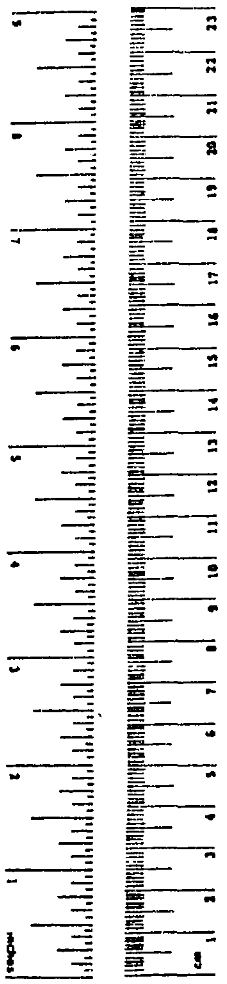
Symbol	When You Know	Multiply By	To Find	Symbol
fl oz	fluid ounces	29.57	millilitres	mL
gal	gallons	3.785	litres	L
ft^3	cubic feet	0.0328	metres cubed	m^3
yd^3	cubic yards	0.0765	metres cubed	m^3

NOTE: Volumes greater than 1000 L shall be shown in m^3.

TEMPERATURE (exact)

Symbol	When You Know	Multiply By	To Find	Symbol
°F	Fahrenheit temperature	5/9 (after subtracting 32)	Celsius temperature	°C

* SI is the symbol for the International System of Measurements



APPROXIMATE CONVERSIONS TO SI UNITS

LENGTH

Symbol	When You Know	Multiply By	To Find	Symbol
mm	millimetres	0.039	inches	in
m	metres	3.28	feet	ft
m	metres	1.09	yards	yd
km	kilometres	0.621	miles	mi

AREA

Symbol	When You Know	Multiply By	To Find	Symbol
mm^2	millimetres squared	0.0016	square inches	in^2
m^2	metres squared	10.764	square feet	ft^2
km^2	kilometres squared	0.39	square miles	mi^2
ha	hectores (10 000 m^2)	2.53	acres	ac

MASS (weight)

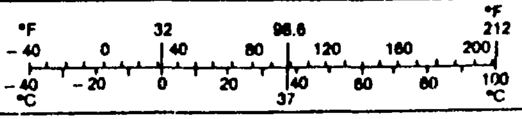
Symbol	When You Know	Multiply By	To Find	Symbol
g	grams	0.0353	ounces	oz
kg	kilograms	2.205	pounds	lb
Mg	megagrams (1 000 kg)	1.103	short tons	T

VOLUME

Symbol	When You Know	Multiply By	To Find	Symbol
mL	millilitres	0.034	fluid ounces	fl oz
L	litres	0.264	gallons	gal
m^3	metres cubed	35.315	cubic feet	ft^3
m^3	metres cubed	1.308	cubic yards	yd^3

TEMPERATURE (exact)

Symbol	When You Know	Multiply By	To Find	Symbol
°C	Celsius temperature	9/5 (then add 32)	Fahrenheit temperature	°F



These factors conform to the requirement of FHWA Order 5190.1A.

PREFACE

This report documents the work performed on the *RHODES* Project Phase II(a). This research effort was the continuation of the *RHODES* Project Phase I. Both phases were funded by the Arizona Department of Transportation (ADOT) through the Pima Association of Governments (PAG). Essentially, Phase I explored concepts for models and algorithms for real-time traffic-adaptive control systems for street networks. Phase II(a) focused on further development of some of these algorithms and on performing some preliminary laboratory experiments with these algorithms using simulation models.

This report was written by the principal investigator, **Pitu B. Mirchandani**, and co-principal investigator, **Larry Head**, both of the Systems and Industrial Engineering (SIE) Department at the University of Arizona. It is based on the compilation of research efforts and results of various individuals who have been involved in the *RHODES* Project. In particular, the efforts of the following individuals are acknowledged:

Julia Higle	Associate Professor, SIE Department
Suvrajeet Sen	Associate Professor, SIE Department
Douglas Gettman	Undergraduate Assistant, SIE Department
Srikanth Nagarajan	Graduate Assistant, SIE Department
Ranjit Rebello	Graduate Assistant, SIE Department
Douglas Tarico	Graduate Assistant, SIE Department
Gregory Tomooka	Graduate Assistant, SIE Department
Michael Whalen	Graduate Assistant, SIE Department
Paolo Dell'Olmo	Visiting Scientist, SIE Department (from the Instituto di Analisi dei Sistemi ed Informatica, Rome).

In addition, the principal investigators wish to acknowledge the following individuals whose technical input, interactions with key investigators, and general support on the project has been invaluable:

<u>Arizona Department of Transportation</u>

Harry A. Reed	Director, Transportation Planning, ADOT
Larry A. Scofield	Manager, Transportation Research, Arizona Transportation Research Center (ATRC), ADOT
Sarath Joshua	Senior Research Engineer, ATRC, ADOT
Louis Schmitt	Assistant County Manager, Maricopa County Transportation and Development Agency, (formerly with ADOT)

<u>Pima Association of Governments</u>

Thomas Buick	Chief, Transportation Planning Division, Maricopa County Department of Transportation, (formerly with PAG)
David Wolfson	Senior Transportation Planner, PAG

<u>City of Tucson</u>

Benny Young	Director of Transportation, City of Tucson
Richard Nassi	Traffic Engineer, City of Tucson.
Dennis Sheppard	Assistant Traffic Engineer, City of Tucson.

Comments and support of **Harvey Friedson** and **James Decker** of the Traffic Engineering Department of the City of Tempe, AZ, are also gratefully acknowledged.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (continued)

LIST OF TABLES

1. INTRODUCTION

Over a period of 15 months, since June 1, 1991, the Arizona Department of Transportation supported the R&D efforts on the development of the RHODES street traffic control system within the Department of Systems and Industrial Engineering at the University of Arizona. PHASE I and PHASE II(a) of this effort have been completed. During these phases the University of Arizona has worked closely with the City of Tucson and the Pima Association of Governments (PAG) in the development of the RHODES concept, some preliminary algorithms, and a simulation model.

PHASE I of the RHODES project consisted of the following tasks:

Task 1(a): Develop RHODES concept
Task 1(b): Develop analysis/simulation tools
Task 1(c): Select demonstration test grid
Task 1(d); Hold traffic control workshop

Task 2(a): Refine RHODES concepts
Task 2(b): Investigate flow optimization models
Task 2(c): Investigate intersection dispatching schemes
Task 2(d): Coordinate modeling efforts.

Tasks 1(a) and 2(a) concentrated on developing a technically sound concept for real-time traffic adaptive control and identifying the key research problems that need to be solved. Task 1(b) consisted of specifying the requirements for a simulation model for demonstrating, testing and evaluating real-time control. It was decided, at least for the short term, that modification of the TRAF-NETSIM model would provide a suitable simulation environment. In the longer term, more advanced simulation models that allow dynamic vehicle routing and have the ability to assess ATIS and other IVHS technologies would be more appropriate. To this end, an investigation of object oriented traffic simulation has been initiated.

Task 1(c) addressed the long-term project goal of implementing RHODES for the Tucson street network. A potential test grid has been selected. The test grid offers several interesting traffic characteristics, such as having a variety of traffic volumes, and a mix of residential and commercial zones. Early selection of the test grid provides a source of real-world data for the traffic simulations and a measure against which to validate the developed simulation model.

Task 1(d) provided a forum of noted experts on real-time traffic control to discuss research issues and comment on the RHODES concept. The workshop was very valuable to the research team. It led to the refinement of the RHODES concept and identified several new key issues.

Tasks 2(b) and (c) focused on the investigation and development of some preliminary algorithms for intersection and network flow control. An algorithm was developed, called COP, based on a dynamic programming formulation of the intersection control problem. The COP algorithm provides the necessary planning horizon, approximately 5 minutes, for integration with network flow control methods.

In addition to these tasks, a major goal of PHASE I was the development of a proposal to FHWA on the design of a real-time traffic-adaptive signal control system. The RHODES team led a strong consortium, that included JHK, SRI, TASC, RPI, and Hughes, and submitted a consortium proposal to FHWA in January 1992. The proposal was not selected for funding; the contract was awarded to Farradyne Systems in June 1992. However, the RHODES team plans to respond to an anticipated FHWA-RFP that will call for alternative prototype developments.

Phase II(a) is concentrated on (1) the development of some RHODES component models and algorithms and (2) a demonstration of these algorithms, using the modified TRAF-NETSIM simulation model. PHASE II(a) consisted of three task:

Task A: Develop algorithms for network loading and control
Task B: Demonstrate controller interface and network control concepts
Task C: Reporting and planning

Task A addressed the investigation and development of algorithms at several levels of the hierarchy as identified in PHASE I. The purpose of Task B is to demonstrate the proof of concept that the RHODES approach can be implemented using existing controller technology.

The research progress on the RHODES project has been significant. A simulation model has been developed for testing and demonstrating real-time traffic control algorithms, and several algorithms have been developed. Concurrently with the further development of the RHODES system for street network control, it is now appropriate to extend the RHODES concept for developing a traffic control system for an integrated freeway/street network. This final report contains the detailed results of the PHASE II(a) effort.

The RHODES concept is depicted in Figure 1. At the highest level of RHODES is the *"dynamic network loading"* model that captures the slow-varying characteristics of traffic. These characteristics pertain to the network geometry (available routes including road closures, construction, etc.) and the typical route selection of travelers. Based on the slow-varying characteristics of the network traffic loads, estimates of the load on each particular link, in terms of <u>vehicles per hour</u>, can be calculated. These load estimates then allow RHODES to allocate "green time" for each different demand pattern and each phase (North-South through movement, North-South left turn, East-West left turn, and so on). These decisions are made at the middle level of the hierarchy, referred to as *"network flow control"*. Traffic flow characteristics at this level are measured in terms of <u>platoons of vehicles</u> and their speeds. Given the approximate green times, the *"intersection control"* at the third level selects the appropriate phase change epochs based on observed and predicted arrivals, of <u>individual vehicles</u>, at each intersection. The RHODES architecture is modular; it allows the accommodation of new modeling methodologies and new technologies as they are developed.

A significant difference between RHODES and other "real-time" traffic control systems is that RHODES is being designed to accommodate real-time measurements of traffic and to become an integral component of IVHS. For example, integration of *Advanced Traveler Information Services* within IVHS will result in (1) improved prediction and estimation of network loads, (2) will allow the ATMS system to provide drivers with real-time information about traffic conditions, and (3) advise the travelers of alternate routes. Priority and accommodation of public and private transit, emergency vehicles, and commercial vehicles, can be easily integrated into the decision-making structure of RHODES.

At the highest *network loading* level of the hierarchy we envision the decision time horizons to be in hours, days and weeks. This model allows for integration of historical

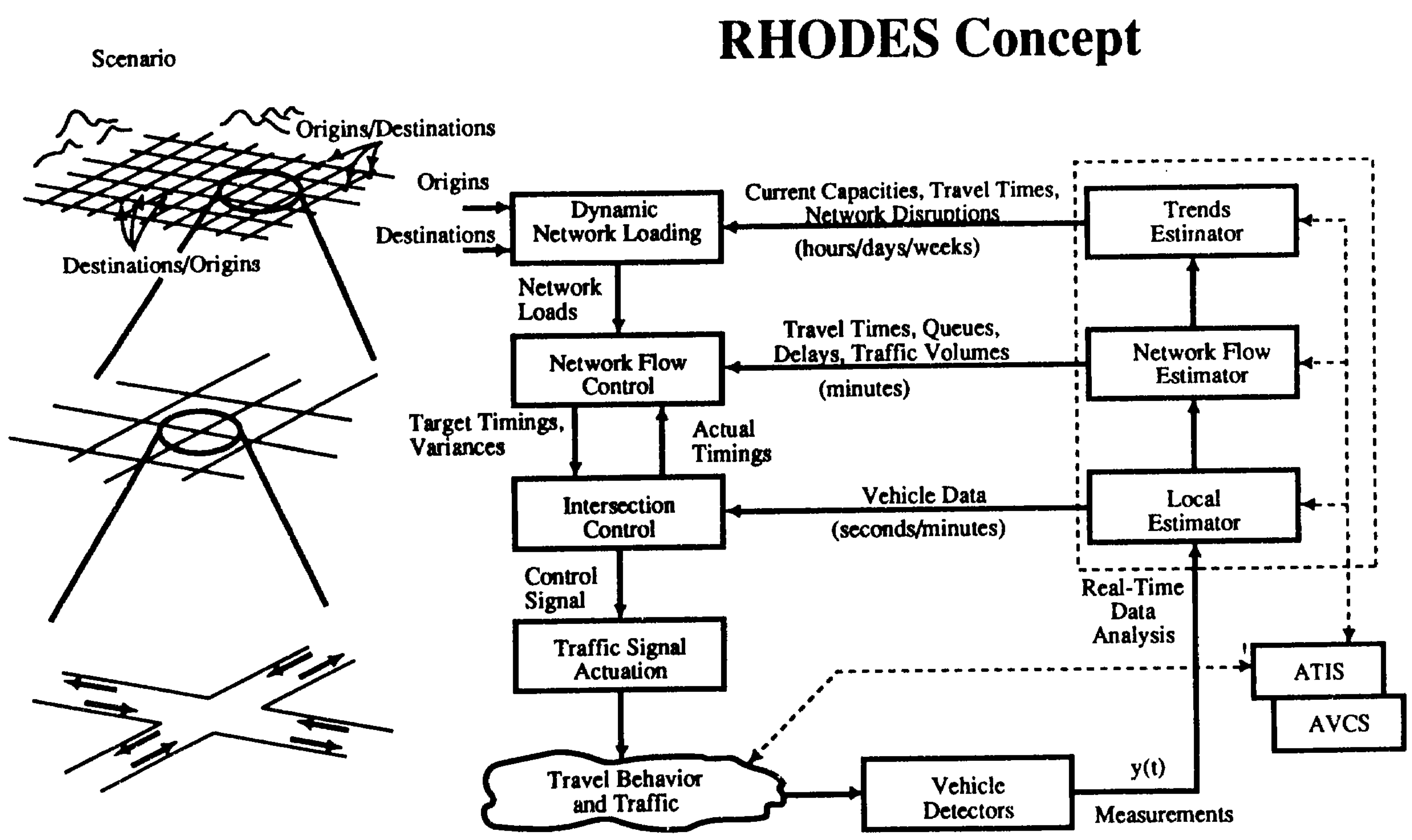


Figure 1. The Rhodes Hierarchical Control System Architecture.

data (a priori information), observed traffic flows (posterior information) and potential[1] ATIS information about IVHS suggested routes and traffic conditions (congestion, accidents and other network events) to allow prediction of near future loads and hence exercise real-time proactive traffic control. The next level of the hierarchy utilizes the predicted and estimated network loads to control traffic on a network wide basis. At this level the *network flow controller* will integrate the network load information with observations of actual volumes and flow profiles to select appropriate phase sequences and phase lengths as well as the allowable variances to accommodate for the stochastic nature of traffic flow on the network level. These timing decisions will be passed to the *intersection controller* where decisions to shorten or extend the current phase will be made (in a decentralized distributed fashion) based on actual observations of the current traffic arrival pattern at each intersection. The lowest level of the hierarchy, referred to as *traffic signal actuation*, is responsible for implementation of the intersection controller decision on the signal control hardware.

[1]The scope of this effort does not include development of an ATIS system. It does, however, include the consideration of potential information available from an ATIS in the design of a proactive traffic control system.

2. NETWORK LOADING

2.1 A Statistical Network Loading Model

In this section, a method that uses historical data to estimate network loads is described. The method is similar in spirit to the dynamic Bayes procedure described by Higle and Nagarajan (1992), although it has been adapted to context of network load estimation. The method is an empirical procedure where the amount of data used to obtain load estimates is determined by the quality/accuracy of the estimates being produced. Higle and Nagarajan show that the procedure is well suited to identifying and reacting to changes in the underlying traffic trends, specifically turning flow probabilities. Thus, it is believed that this method will also be well suited to estimating network loads.

2.1.1 Empirical Bayes Estimates

The primary objective is to estimate the number of vehicles traveling on a particular link during a particular interval of time on a particular calendar day. Let $N_{ij}(t,d)$ be the number of vehicles traveling on link (i,j) during time period t on calendar day d, as observed using vehicle detectors. Assume that $N_{ij}(t,d)$ has a Poisson distribution with a mean $\lambda_{ij}(t,d)$. There are several points implicit in this simple assumption.

First, note that the average vehicle load on link (i,j), denoted by $\lambda_{ij}(t,d)$ need not be presumed constant over time or over calendar day. This rate typically varies by "time of day" and "day of week". Figure 2 depicts the time varying vehicular flow on a particular day of the week for various calendar days. It is assumed that $\lambda_{ij}(t,d)$ for each day d is for a collection of calendar days that have essentially the same characteristics. Second, the interval of time, Δt, associated with the estimation/prediction task need not be held constant throughout the day, but may also vary by time of day. Thus, there is sufficient flexibility in the estimation/prediction method to allow for longer time intervals during low use periods, and shorter time intervals during high use periods. Third, as discussed before, the vehicular rate, $N_{ij}(t,d)$, is generally assumed constant for a particular time period on a particular day. Finally, note that $\lambda_{ij}(t,d)$, the average vehicle loads, are the quantities is to be estimated.

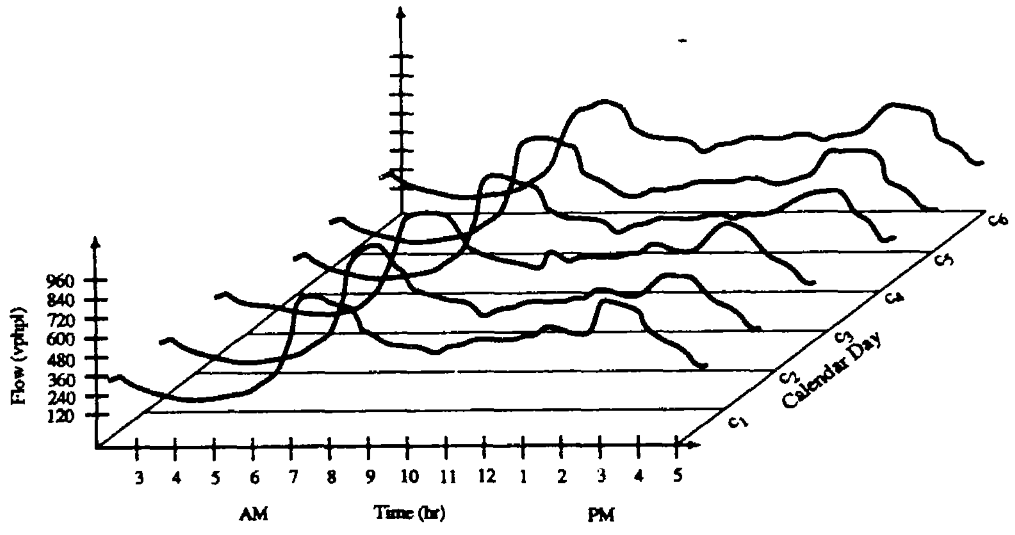


Figure 2. Typical day-of-week loads on link [i, j] over time of day and on different calendar days.

Since the mean, $\lambda_{ij}(t,d)$, of the Poisson distribution is unknown, a Bayesian viewpoint is adopted and it is modeled as a random variable. For the purposes of mathematical convenience and computational ease, assume that $\lambda_{ij}(t,d)$ has a gamma distribution with parameters α_0 and β_0. As data is collected the parameters of the gamma distribution will be updated to describe $\lambda_{ij}(t,d)$. The mean of the resulting distribution will be used as a point estimator for $\lambda_{ij}(t,d)$. That is, if $\lambda_{ij}(t,d) \sim \text{Gamma}(\alpha_0,\beta_0)$, then its mean

$$\hat{\lambda}_{ij}(t,d) = \frac{\alpha_0}{\beta_0} \tag{1}$$

will be used as the point estimate of the vehicle flow rate.

The estimation procedure evolves over time in a manner that follows readily from well known properties of the gamma and Poisson distributions. Specifically the key property is:

If $\{N_k\}_{k=1}^{\delta}$ are independent and identically distributed observations of a random variable whose conditional distribution <u>given</u> μ is Poisson(with mean μ), and μ has a gamma distribution with parameters α_0 and β_0, then the conditional distribution of μ, given the observations $\{N_k\}_{k=1}^{\delta}$ is a gamma distribution with parameters α_δ and β_δ, where

$$\alpha_\delta = \alpha_0 + \sum_{k=1}^{\delta} N_k \text{ and } \beta_\delta = \beta_0 + \delta, \tag{2}$$

(see, for example (DeGroot, 1977), chapter 11).

When translated to the context of traffic flow estimation, this result leads to a simple recursive procedure for estimating traffic flows over time. Here, μ corresponds to $\lambda_{ij}(t,d)$. Thus, given initial values of the parameters of the gamma distribution, α_0 and β_0, these values are updated to reflect the observations $N_{ij}(t,l), l = 1,\ldots,d$ as follows:

$$\begin{aligned} \alpha_d &= \alpha_0 + \sum_{l=1}^{d} N_{ij}(t,l), \\ \beta_d &= \beta_0 + d. \end{aligned} \tag{3}$$

This may be accomplished recursively as:

$$\begin{aligned} \alpha_d &= \alpha_{d-1} + N_{ij}(t,d-1), \\ \beta_d &= \beta_{d-1} + 1. \end{aligned} \tag{4}$$

Given α_d and β_d the predicted load on the forthcoming day is given by

$$\hat{\lambda}_{ij}(t,d+1) = \alpha_d / \beta_d. \tag{5}$$

To ensure that the resulting estimator, α_d/β_d, is capable of responding to changes in the underlying trends and responds adaptively to the quality of the estimates being produced, it is necessary to be able to use different amounts of data for obtaining α_d and β_d. That is, when the estimate is "good", additional data should be included so that estimates with lower error variances will result. However, when the estimate appears to be persistently poor, less, but newer, data should be used so that the estimators will be more responsive to apparent changes in the underlying trends. In the next section an adaptive method for determining the amount of data used in the estimation process is discussed.

2.1.2 Dynamic Bayes Estimation

The estimators obtained using the updated parameters specified in (3) and (4) will be most accurate when the underlying flow rate, $\lambda_{ij}(t,d)$, is constant over calendar day, d. However, even when the data is normalized for time of day and calendar day cycles, there are still likely to be changes in the flow rate (e.g., seasonal tendencies, construction obstructions, special events, etc.). In this case, it is necessary to allow the estimators to respond dynamically to the errors observed. This can be accomplished using the dynamic Bayes estimates proposed by Higle and Nagarajan (1992), adapted to the context of network load estimation.

Note first that the estimate of the anticipate flow for calendar day d is based on the observed flows in the previous calendar days. Thus, if α_d and β_d denote the parameters of the gamma distribution used to describe $\lambda_{ij}(t,d+1)$ after having observed $\{N_{ij}(t,l)\}_{l=1}^{d}$, then the point estimate of $\lambda_{ij}(t,d+1)$ is given by

$$\hat{\lambda}_{ij}(t,d+1)=\frac{\alpha_d}{\beta_d}. \tag{6}$$

The quality of this estimate depends on the extent to which it appears to be approximately equal to the observed flow in that period, $N_{ij}(t,d+1)$. If $N_{ij}(t,d+1)$ is consistent with $\hat{\lambda}_{ij}(t,d+1)$, then the vehicle flow rate over time appears to be stable enough to allow the simple update procedure described in (4). If $N_{ij}(t,d+1)$ is inconsistent with $\hat{\lambda}_{ij}(t,d+1)$, then steps must be taken to allow subsequent estimates to adapt to a potential change in the underlying trend. One approach is to discard the observations used early in the estimation process, so that the more recent observations influence the estimate more significantly.

To determine whether or not the observation is inconsistent with the estimate, it is necessary to obtain probabilistic statements from the Poisson distribution. Let an error probability ε, $0<\varepsilon<1$, be given and let quantities $\overline{N}$ and $\underline{N}$ be defined so that if $N_{ij}(t,d)\sim \text{Poisson}(\hat{\lambda}_{ij}(t,d))$ then

$$P\{N_{ij}(t,d) \le \underline{N}\} = \frac{\varepsilon}{2}, \quad P\{N_{ij}(t,d) \ge \overline{N}\} = \frac{\varepsilon}{2}. \tag{7}$$

Therefore, under the hypothesis that $N_{ij}(t,d) \sim \text{Poisson}(\hat{\lambda}_{ij}(t,d))$,

$$P\{\underline{N} \le N_{ij}(t,d) \le \overline{N}\} = 1 - \varepsilon. \tag{8}$$

In standard statistical tests of hypothesis, $N_{ij}(t,d)$ is said to be consistent with $\hat{\lambda}_{ij}(t,d)$ if $\underline{N} \le N_{ij}(t,d) \le \overline{N}$, and inconsistent otherwise. The inconsistency is said to be persistent if there is at least one inconsistency in the previous γ days, where γ is a specified estimation threshold. In this dynamic Bayes approach, α_d and β_d are determined in one of three ways, depending on the detection and persistence of an inconsistency.

Exhibit A outlines the dynamic Bayesian network loading algorithm. Note that at the start of this procedure, α_{d-1} , β_{d-1}, and therefore $\hat{\lambda}_{ij}(t,d)$ are available from the previous estimate. Parameters ε , δ and γ are chosen (by the designer/user of the algorithm) to facilitate the responsiveness of the algorithm to real-time data. As discussed above, ε and γ are used to identify inconsistancy in the estimates being produced, and δ denotes the time-window of the data used for the estimation. $\delta_{\min}$ and $\delta_{\max}$ are the minimum and maximum allowable sizes for the time window δ . The parameter δ is updated within the algorithm, depending on the quality of the estimates being produced. In steps 3 and 4, estimated α_d and β_d are derived using observed data $\{N_{ij}(t,l)\}_{l=d-d+1}^{d}$ during the current time window.

A brief discussion of Step 2 of this procedure is in order. Entering Step 2, $\hat{\lambda}_{ij}(t,d)$ has been determined from $\{N_{ij}(t,l)\}_{l=d-\delta}^{d-1}$ and is compared to the observed value $N_{ij}(t,d)$ to determine if δ should be adjusted. Note that δ is increased in Step 2 (a), decreased in Step 2 (b), but at all times $\delta_{\min} \le \delta \le \delta_{\max}$. These limits are intended to prevent the use of "too many" or "too few" observations in the computation of $\hat{\lambda}_{ij}(t,d)$ and can be specified at the user's discretion. In step 2 (a), the estimate is consistent with the observation, so δ is increased. In step 2 (b), the persistance of the inconsistancy is tested. If it is determined that the inconsistancy is persistant, δ may be decreased. Each time it is decreased, the oldest observation is discarded and a new estimate of $\hat{\lambda}_{ij}(t,d)$ is computed using the most recent observations (excluding, of course, $N_{ij}(t,d)$). The reduction of δ

Exhibit A. Dynamic Bayesian Network Loading Algorithm

```
                    Procedure: Compute $\hat{\lambda}_{ij}(t,d+1)$

Step 0:. Given estimates $\alpha_{d-1}$, $\beta_{d-1}$, $\hat{\lambda}_{ij}(t,d)$, parameters $\gamma$, $0<\varepsilon<1$, $\delta$, $\delta_{\min}$, $\delta_{\max}$ and
    the observations $\{N_{ij}(t,l)\}_{l=d-\delta+1}^{d}$ for time period $t$ and caledar day $d$,

Step 1: Define $\overline{N}$ and $\underline{N}$ according to (7).

Step 2: Determine $\delta$, the amount of data to be used in computing $\alpha_d$ and $\beta_d$:
    (a) If $\underline{N} \le N_{ij}(t,d) \le \overline{N}$
            then
                $\delta = \mathrm{Min}(\delta+1, \delta_{\max})$,

        (b) else if $N_{ij}(t,l) \notin [\underline{N}, \overline{N}]$ for at least one observation $l \in [d-\gamma, d-1]$,
            then while $N_{ij}(t,d) \notin [\underline{N}, \overline{N}]$ and $\delta > \delta_{\min}$
                $\delta \leftarrow \delta - 1$
                $\alpha_{d-1} \leftarrow \alpha_{d-1} - N_{ij}(t,d-\delta)$
                $\beta_{d-1} \leftarrow \beta_{d-1} - 1$
                $\hat{\lambda}_{ij}(t,d) \leftarrow \frac{\alpha_{d-1}}{\beta_{d-1}}$
                $\alpha_0 \leftarrow \alpha_{d-\delta}$
                $\beta_0 \leftarrow \beta_{d-\delta}$
    define $\underline{N}$, $\overline{N}$ according to (7) using $\hat{\lambda}_{ij}(t,d)$ and repeat Step 2 (b)
    (c) else $\delta$ remains unchanged.
Step 3: Compute
                $\alpha_d \leftarrow \alpha_0 + \sum_{l=d-\delta+1}^{d} N_{ij}(t,l)$,
                $\beta_d \leftarrow \beta_0 + \delta$.
Step 4:Compute
                $\hat{\lambda}_{ij}(t,d+1) \leftarrow \frac{\alpha_d}{\beta_d}$.
```

terminates when either $N_{ij}(t,d)$ becomes consistent with the recomputed estimate, $\hat{\lambda}_{ij}(t,d)$ or when δ reaches its lower limit, $\delta_{\min}$. When the algorithm process is in step 2 (c), inconsistency has been detected, but it is too early to tell if it is persistent. In this case, δ is neither increased nor decreased. Once δ has been set, $\hat{\lambda}_{ij}(t,d+1)$ is computed using the δ most recent observations, including $N_{ij}(t,d)$. By monitoring the quality of the estimates produced and adaptively responding to errors when they are detected, the estimated load for the forthcoming time period, $\hat{\lambda}_{ij}(t,d+1)$, should more closely approximate the load that will be observed , $N_{ij}(t,d+1)$.

2.2 A Network Loading Example

To demonstrate both the statistical network loading model and the capacity allocation model discussed above, a small traffic network was simulated using the modified TRAF-NETSIM model. Figure 3 shows the layout of the traffic network. This network was selected because it contains a long arterial (Campbell Avenue) near the University of Arizona football stadium[1]. The primary nodes of interest, those that will be used for testing control algorithms, are numbers 335, 369, 401 and 483. The remainder of the nodes are included to provide realistic traffic flows, i.e. platoons and non-uniform arrivals, into the controlled area. The location of vehicle detectors in the simulation model is consistent with the existing detector locations in the actual network. For the purposes of this example, the conditions on the network were simulated between 11 AM and 1 PM, a period of moderate to heavy usage.

To demonstrate the statistical network loading algorithm, the dynamic Bayes algorithm, detectors on each major links of the network are included in the estimation. For the purposes of presentation in this paper, the results from a single detector will be discussed in detail. This detector is located 130 feet north of the intersection of Speedway Blvd. and Campbell Ave (intersection 335). It includes all vehicles in all three lanes that approach the intersection. To represent both time periods and calendar days several runs of the simulation model were made. Each run utilized a unique random number with all other parameters (source input rates, turning probabilities, and signal timing parameters) held constant.

[1]This network selection is intended to allow the RHODES team to be prepared for the FHWA Real-time Traffic Adaptive Signal Control RFP for alternative algorithms due to be announced in 1993. This type of network/arterial will be the basis for the testing and performance competition for real-time traffic-adaptive control algorithms.

Figure 4 shows the observed number of vehicles for the eight time periods of fifteen minutes each over thirty days (simulation runs). From these observations it can be seen that the number of vehicles crossing the detector is essentially constant over the days and not constant over time. To test the estimation procedure the initial estimates, α_0 and β_0, were selected so that the estimate would beinitially be inaccurate and the responsiveness of the algorithm could be validated. Figure 5 shows the estimated loads over time and day. From the Figure it appears that the estimator can overcome the large initial estimate error and closely estimate the loads.

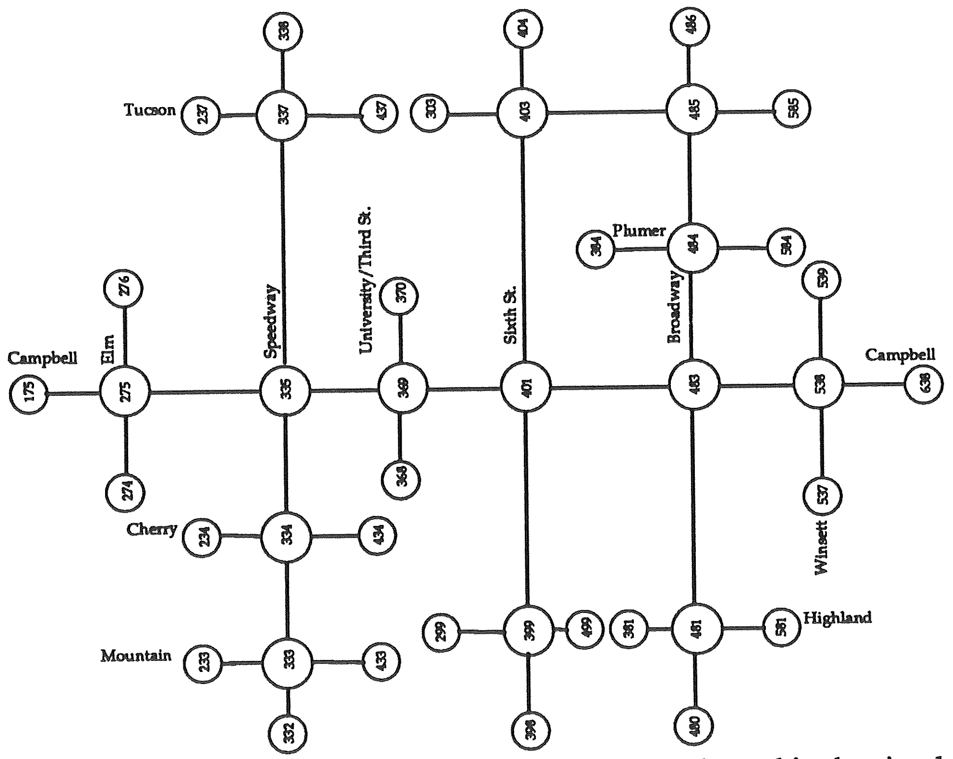


Figure 3. Topological layout of the traffic network used in the simulation studies.

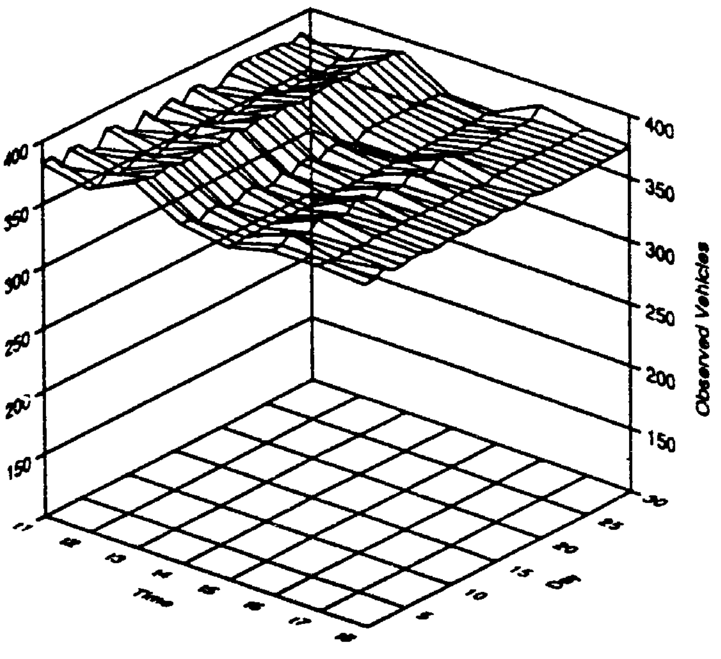


Figure 4. Observed number of vehicles for eight time periods of 15 minutes each over thirty days (simulation runs)

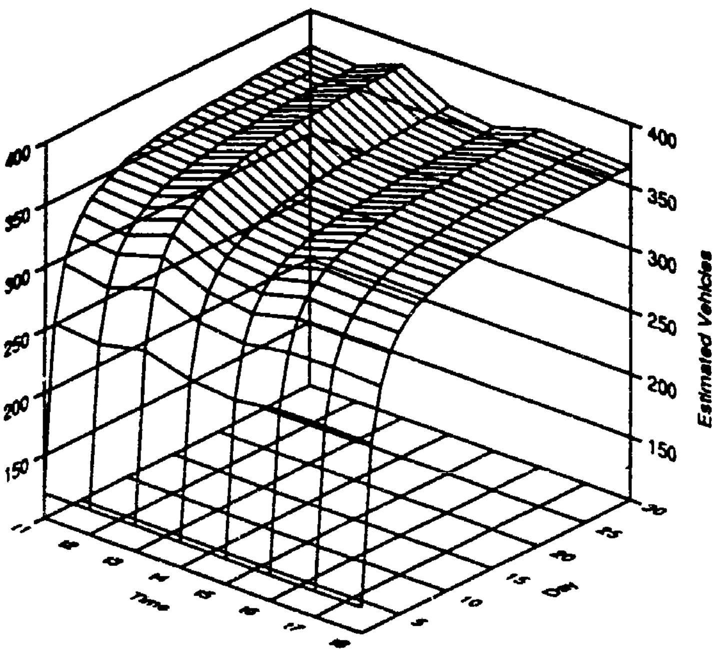


Figure 5. Estimated loads over time and day.

The performance of the method can be more closely studies by considering either a single time period or a single day. Figures 6, 7 and 8 show the observed and estimated loads over time on calendar days 0, 5 and 30, respectively. The large initial estimate error, approximately 100% on day 0, is reduced to less than 10% afer only five days and less than 1% ofter 30 days. Figure 9 shows the observed and estimted loads over calendar day at a single time period, t_3. This figure demonstrates the ability of the method to correctly estimate the load.

The statistical network loading model presented here is useful for estimating the expected link volumes based on existing loop detector data. It is important to note that this model is not based on known, or approximated, origin-destination data and hence is not an equilibrium or assignment model. This model does address the need for a statistical method of estimating link volumes based on loop detector data that will allow for the statistical classification of anomalies such as non-recurrent congestion due to events such as accidents. In these cases, alternative historical data sets can be used for the prediction purpose. Based on this statistical foundation this model can be extended to include equilibrium or assignment data, as well as other information that will be available through the deployment of IVHS.

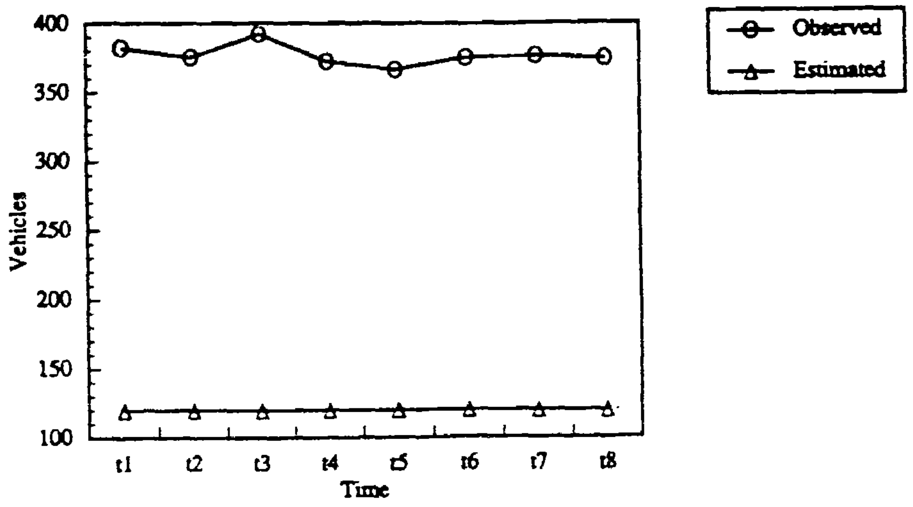


Figure 6. Observed and estimated load over time on calendar day 0.

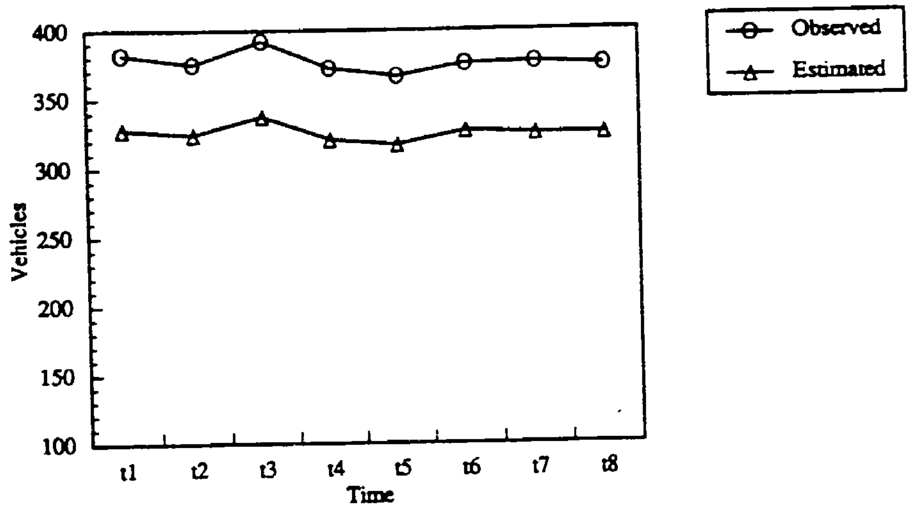


Figure 7. Observed and estimated load over time on calendar day 5.

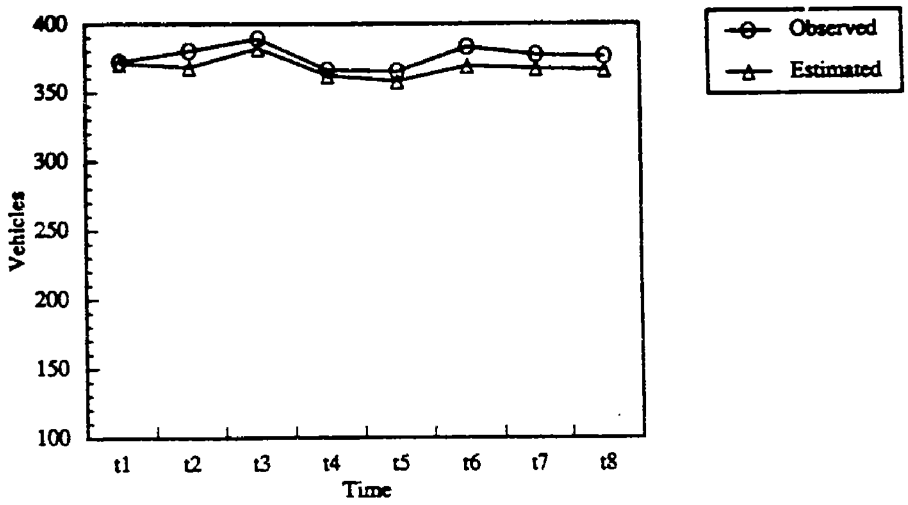


Figure 8. Observed and estimated load over time on calendar day 30.

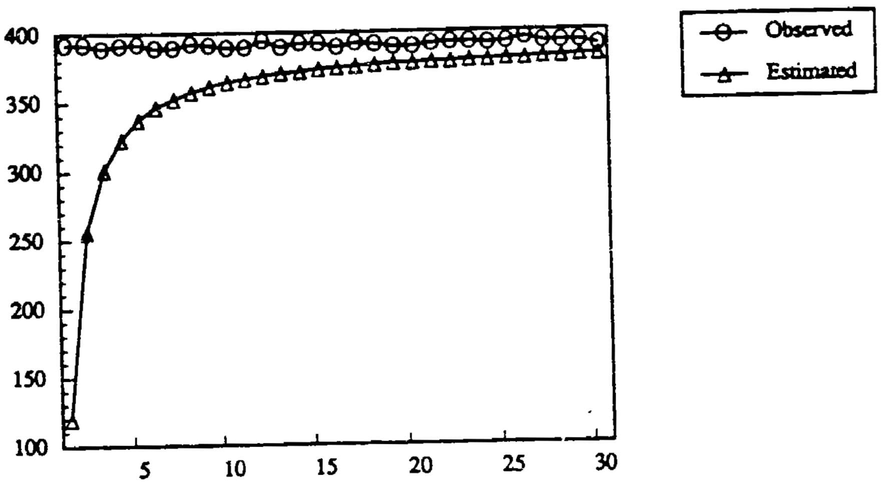


Figure 9. Observed and estimated load over calendar day at a fixed time.

3. CAPACITY ALLOCATION

3.1 The Capacity Allocation Model

The network loading model provides estimates of the expected link loads on the network. These estimates are used by the capacity allocation model to determine the fraction of time that should be allocated to each phase in order to satisfy the network demand. At this level of the hierarchy, a uniform, fluid flow viewpoint of traffic is assumed. The solution to the capacity allocation problem does not consider the flow of individual vehicles or platoons between signalized intersections. It establishes general fractions of time that must be allocated to different phases to satisfy the average demand over extended periods of time. These fractions serve as constraints to the network coordination model and the intersection scheduler.

Let v_i denote the demand (arrival rate) for movement i at some intersection. This demand can be derived from the predicted loads generated by the network loading model and estimated turning probabilities[1]. The quantity $\hat{\lambda}_{ij}(t,d+1)$ represents the estimated vehicular load on link (i,j) during time period t and calendar day d. If p_{ij}^{m} denotes the probability of a vehicle on link (i,j) demanding movement m then

$$v_m = p_{ij}^{m}\hat{\lambda}_{ij}(t,d+1) \tag{9}$$

is the estimated demand for movement m.

Figure 10 shows the standard labels of 8 possible movements at an intersection. Let $\phi_{ij} = \{i,j\}$ denote the signal phase where movements i and j are allowed. For the purpose of this development, assume that the only possible phases at this intersection are $\phi_{26}, \phi_{15}, \phi_{48}$ and ϕ_{37} (as shown in Figure 10). Let x_{26}, x_{15}, x_{48}, and x_{37} denote the fractions of the intersection capacity (green time) allocated to each phase. Then, assuming a uniform arrival rate, the delay (Hurdle, 1984) (uniform delay per vehicle) associated with phase ϕ is

[1]Here it is assumed that the turning probabilities are known. Estimation of these turning probabilities has been address by Higle and Nagaragan (1992) for the case of a fully instrumented intersection. Their approach has still to be adapted to a partially instrumented intersection.

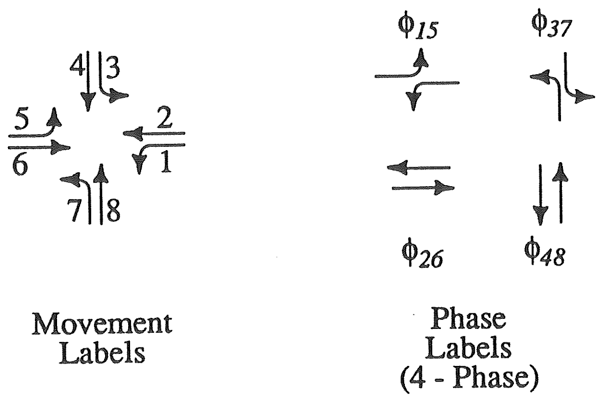


Figure 10. Standard labels of 8 possible movements and the associated 4-phases.

$$D(x_\phi) = \frac{(1-x_\phi)^2}{2} \sum_{m\in\phi} \frac{1}{(1-\frac{v_m}{s_m})} \tag{10}$$

where s_m denotes the saturation flow rate associated with movement m. Note that the saturation flow rate must be selected to reflect the appropriate number of lanes and other traffic considerations (grade, lane width, etc.) associated with each movement.

Given these definitions of delay, arrival rates and saturation flow rates, the capacity allocation problem can be stated as

$$\begin{aligned} &\text{Minimize } D(x) = \sum_{\text{all }\phi} D(x_\phi) \\ &\text{subject to} \\ &\qquad \sum_{\text{all }\phi} x_\phi = 1 \\ &\qquad x_\phi \geq 0, \quad \text{all } \phi. \end{aligned} \tag{11}$$

For the 4-phase case depicted in Figure 10 the capacity allocation problem can be stated as

$$\text{Minimize } D(x)=\frac{(1-x_{15})^2}{2}\left[\frac{1}{(1-\frac{v_1}{s_1})}+\frac{1}{(1-\frac{v_5}{s_5})}\right]+\frac{(1-x_{26})^2}{2}\left[\frac{1}{(1-\frac{v_2}{s_2})}+\frac{1}{(1-\frac{v_6}{s_6})}\right]$$
$$+\frac{(1-x_{37})^2}{2}\left[\frac{1}{(1-\frac{v_3}{s_3})}+\frac{1}{(1-\frac{v_7}{s_7})}\right]+\frac{(1-x_{48})^2}{2}\left[\frac{1}{(1-\frac{v_4}{s_8})}+\frac{1}{(1-\frac{v_8}{s_8})}\right] \qquad (12)$$

subject to

$$x_{15}+x_{26}+x_{37}+x_{48}=1,$$
$$x_{15}\geq 0,\quad x_{26}\geq 0,$$
$$x_{37}\geq 0,\quad x_{48}\geq 0.$$

The capacity allocation problem, as stated here, has a quadratic objective function with a single linear equality constraint and the usual non-negativity (inequality) constraints on the decision variables. This form of a quadratic mathematical programming problem can be solved using the quadratic structure of the objective function and an *active set method* to manage the inequality constraints.

The capacity allocation problem (12) can be written in the form

$$\begin{aligned}&\text{Minimize } \frac{1}{2}x'Qx+c'x+K\\&\text{subject to}\\&\qquad e'x=1,\\&\qquad x_i\geq 0,\quad i=1,\ldots N,\end{aligned} \qquad (13)$$

where $e'=(1 \ \cdots \ 1)$ and N is the number of allowable phases. Given this formulation the only mathematical condition necessary for a quadratic programming active set method to be applicable is that the matrix Q be positive definite.

The capacity allocation algorithm is outlined in Exhibit B. It is assumed that an initial starting solution is given (step 0). It is important that the initial solution satisfy the equality constraint $e'x^0=1$. A good candidate is

$$x_i^0=\frac{\sum_{m\in\phi_i} v_m}{\sum_{\text{all }\phi} v_m}. \qquad (14)$$

Exhibit B. Capacity Allocation Algorithm.

Procedure: Solve Quadratic Program Using Active Set

Step 0:. Assume x^0 , a starting solution, is given and feasible.

Step 1: Define W^k as the set of active inequality constraints, $W^0 = \varnothing$.

Step 2: Solve the equality constrained quadratic program:

$$\begin{aligned} \text{Minimize} \quad & \frac{1}{2} y^{k\prime} Q y^k + c' y^k \\ \text{subject to} \quad & \\ & e' y^k = 1, \\ & y_n^k = 0, \quad n \in W^k. \end{aligned}$$

Step 3: Let $x^{k+1} = x^k + \alpha^k (y^k - x^k)$ where

$$\alpha^k = \min\left\{\alpha\colon \alpha \in [0,1],\, x_n^k + \alpha(y_n^k - x_n^k) = 0,\ n = 1,\ldots,N\right\}.$$

Step 4: Update the working set: Let $j \notin W^k$ denote the index of the constraint(s) such that

$$x_j^k + \alpha^k (y_j^k - x_j^k) = 0$$

then

$$W^{k+1} = W^k \cup \{j\}.$$

Step 5: Release a binding constraint: Let $i \in W^k$ denote the index of a constraint such that $\mu_i < 0$, then

$$W^{k+1} = W^k - \{i\}.$$

If $\mu_i > 0$ for all $i \in W^k$ STOP, else GO TO Step 2.

In step 1, the working set, the set of active inequality constraints, is initially defined to be the empty set. It is feasible of course that one or more $x_i^0 = 0$. In this case it would be necessary to define the working set as

$$W^0 = \left\{i\colon x_i^0 = 0, i = 1,\ldots N\right\}.$$

In step 2, the equality constrained quadratic optimization problem is solved. This problem can be solved using the first-order optimality conditions (Luenberger, 1984) for equality constrained problems by solving the following system of linear equations:

$$\begin{bmatrix} Q & -e & -E^k \\ e' & 0 & 0 \\ (E^k)' & 0 & 0 \end{bmatrix} \begin{pmatrix} y^k \\ \lambda \\ \mu \end{pmatrix} = \begin{pmatrix} -c \\ 1 \\ \underline{0} \end{pmatrix} \tag{15}$$

where E^k is the matrix whose rows are formed by the vectors e_n' that are all zeroes except for a 1 in the n^{th} position for $n \in W^k$. It is not necessary that the solution to (15) yield a solution, y^k that is optimal or feasible to the inequality constrained problem.

In step 3, a line search is conducted from the current candidate solution, x^k, towards y^k. Movement along this direction occurs only until either one of the non-tight inequality constraints becomes tight or until the point y^k is reached (in this case $\alpha^k = 1$). If one or more of the non-tight constraints becomes tight in step 3, they are added as binding constraints in step 4.

In step 5 a constraint that was tight is released if the associated Lagrange multiplier is less than zero. (This condition is based on the first-order optimality conditions for inequality constrained optimization). If none of the Lagrange multipliers are negative and no new constraints are added to the working set, the algorithm is stopped at the optimal solution. If the algorithm is not stopped, steps 2-5 are repeated until an optimal solution is found.

3. 2 A Capacity Allocation Example

This example will utilize the results of the statistical network loading model from Section 2.2 at the intersection of Campbell Avenue and Speedway Boulevard (node 335). Figure 11 shows the layout of the intersection including the proportion of vehicles that turn left, right or proceed though the intersection and the associated phase definition. These values we determined by the City of Tucson's traffic engineering department. Each approach has one turning lane and three through lanes. A saturation flow rate of 1800 vphpl was assumed.

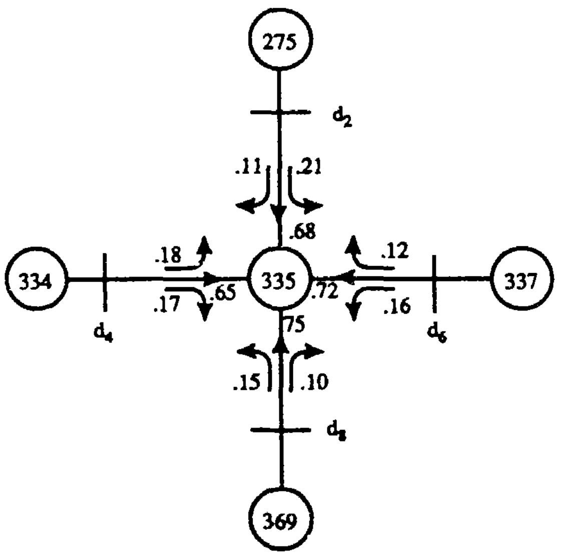


Figure 11. Layout of the Campbell Avenue and Speedway Boulevard intersection.

Table 1 shows the approach volumes for each approach for thirty 15 minute time intervals. These approach volumes are used to derive the movement demands, assuming a 4-phase control, during each time interval. The capacity allocation algorithm is used to find the percent green allocation for each time interval. The results of the capacity allocation algorithm are shown in Table 1.

The results of the capacity allocation algorithm must be carefully interpreted. The numbers in the two tables cannot be directly compared since the approach volumes are related to the traffic volumes using the turning probabilities (see Equation (9)) and the delay is computed using Equation (10). It is also important to note into that the capacity allocation results are to be used for providing estimates and not as the exact signal timing parameters.

Table 1. Approach volumes and percent green allocation from the capacity allocation example. Each approach volume represents the number of vehicles arriving during a 15 minute interval. The percent green allocation is computed to minimize total vehicle delay.

Time	Approach Volume (per 15 minute period)			
	d_2	d_4	d_6	d_8
1	270	416	382	224
2	270	417	384	239
3	266	390	373	231
4	270	433	383	226
5	270	416	382	234
6	266	391	373	212
7	266	391	372	235
8	270	340	384	236
9	270	393	382	239
10	266	382	373	217
11	266	329	371	251
12	270	416	382	241
13	266	447	372	224
14	270	341	379	221
15	270	415	382	235
16	266	392	373	231
17	270	415	383	228
18	270	409	382	236
19	266	392	373	222
20	266	391	373	219
21	270	416	380	247
22	270	414	384	232
23	270	414	384	233
24	270	462	383	212
25	270	418	383	229
26	270	416	383	207
27	270	383	382	217
28	270	468	379	229
29	270	395	384	231
30	266	455	372	221

Time	Percent Allocation of Green (per Phase)			
	Phase 15	Phase 26	Phase 37	Phase 48
1	0.2546	0.2990	0.2104	0.2360
2	0.2533	0.2981	0.2106	0.2379
3	0.2517	0.2945	0.2138	0.2400
4	0.2559	0.3010	0.2086	0.2345
5	0.2536	0.2982	0.2108	0.2375
6	0.2535	0.2963	0.2130	0.2372
7	0.2513	0.2941	0.2140	0.2406
8	0.2473	0.2895	0.2183	0.2449
9	0.2512	0.2950	0.2133	0.2405
10	0.2523	0.2948	0.2141	0.2388
11	0.2447	0.2854	0.2209	0.2490
12	0.2530	0.2975	0.2110	0.2385
13	0.2571	0.3018	0.2077	0.2334
14	0.2485	0.2902	0.2181	0.2432
15	0.2534	0.2979	0.2109	0.2377
16	0.2518	0.2947	0.2136	0.2398
17	0.2542	0.2987	0.2105	0.2366
18	0.2528	0.2971	0.2116	0.2385
19	0.2527	0.2955	0.2133	0.2385
20	0.2529	0.2957	0.2132	0.2382
21	0.2523	0.2967	0.2115	0.2396
22	0.2538	0.2984	0.2107	0.2372
23	0.2537	0.2983	0.2107	0.2373
24	0.2597	0.3058	0.2050	0.2295
25	0.2543	0.2990	0.2103	0.2364
26	0.2562	0.3007	0.2096	0.2335
27	0.2525	0.2958	0.2135	0.2383
28	0.2584	0.3045	0.2054	0.2317
29	0.2523	0.2962	0.2126	0.2389
30	0.2581	0.3030	0.2068	0.2321

4. NETWORK FLOW CONTROL

At the second level of control in the control hierarchy are decisions/actions that allow for coordination of flow of traffic on the network. Prototypical off-line approaches to network coordination are TRANSYT, MAXBAND, and PASSER II, the latter two being predominately for arterial coordination. (Although the original MAXBAND model [Little et al., 1981] allowed the optimization of signal timings in a network, the model has been predominately used for coordinating arterials. However, the recent enhancements of MAXBAND, as embodied in MAXBAND-86 [Chang et al., 1988] and PASSER IV [Chaudhary and Messer, 1993], have made possible its implementation to grid networks, but applications to actual networks are still lacking.)

The basic ingredients within these methods are (1) a traffic flow model and (2) an algorithm for optimizing a specified performance criterion (this criterion could be a weighted sum of several performance indices). For example, in TRANSYT, vehicles are "loaded" on to the network at given *origins* and are propagated through the network in accordance with a traffic flow model. Traffic controls affect the movement of these vehicles, and numerical optimization (gradient search) is performed to find controls that optimize the specified performance criterion. In MAXBAND and PASSER II, vehicles are loaded on an arterial and traffic signals on that arterial are coordinated to optimize a performance criterion, which often relates to the number of stops. Since these are off-line methods, assumptions on the traffic loads are based on historical average volumes and these are uniformly loaded on to the arterials. This results in an assumption of platoons of uniform size and identical speeds.

One may use TRANSYT in an on-line fashion and compute signal settings every few minutes and download those settings to the field. In a way, this is exactly what SCOOT does. However, the current versions of SCOOT have the disadvantage that arterials within the network may not have sufficient platoon progression. Band-aid approaches have been suggested to enhance SCOOT to consider this; however, to our knowledge, it has not been implemented. On the other hand, TRANSYT has been modified to include progression opportunities [Hadi and Wallace, 1992] but has not been implemented for real-time applications. It is not obvious that this approach is amenable for real time applications due to its excessive computational requirements. Furthermore, TRANSYT, and for that matter SCOOT, do not explicitly consider the current traffic flows (i.e., platoons and their speeds) but rather takes the current data and assumes a uniform flow of the current volumes.

In RHODES Phases I and II(a), A concept for network flow control was investigated. Although a preliminary network simulation model for evaluating network control was developed, no algorithms for network control were designed. The concept studied explicitly considers available real-time information on computing signal timings. It first identifies platoons in the network -- the sizes and their speeds. These are identified by the fusing and filtering the traffic data obtained, from various sources, in the last few minutes. Then, using an approximate traffic model, these platoons are propagated through the network, for a given time horizon, say, T. The signals are set so that the identified platoons are provided appropriate green times to optimize the performance criterion. It is obvious that two platoons demanding conflicting movements may arrive at an intersection at the same time. In that case, then either one or the other will be given priority on the green time, or it may be necessary to split one of the platoons, to maximize the given measure of performance. Resolving such conflicts in the heart of the optimization process within the algorithm of the proposed system.

4.1 REALBAND for Real-time Network Coordination

Consider the time-distance diagram on a single arterial as shown in Figure 12. The goal of arterial progression algorithms, such as MAXBAND and PASSER II, is to set the signal timings such that number of vehicles that can traverse the arterial in either direction without stopping (other similar criteria may be incorporated) is maximized. The figure shows these bands of green times. Note that following drawbacks: it is assumed (1) that platoons are of equal size, (2) that platoons travel at the same constant speed, and (3) that streets intersecting with this arterial have small traffic volumes and therefore movements in those directions have low priorities (that is the reason why MAXBAND and PASSER II are predominately used to coordinate signals on an arterial rather than on a network).

Now consider the time-distance diagram as shown in Figure 13. Here we have platoons of different sizes and different speeds. The green times required for these platoons is different than those required for the uniform case shown in Figure 12, and therefore the smooth anticipated progression is disrupted. By slightly adjusting the red times, we now reinstate the green bands for the given platoons with their own sizes and speeds, as shown in Figure 14. In this manner we can also include platoon dispersion and compression but for

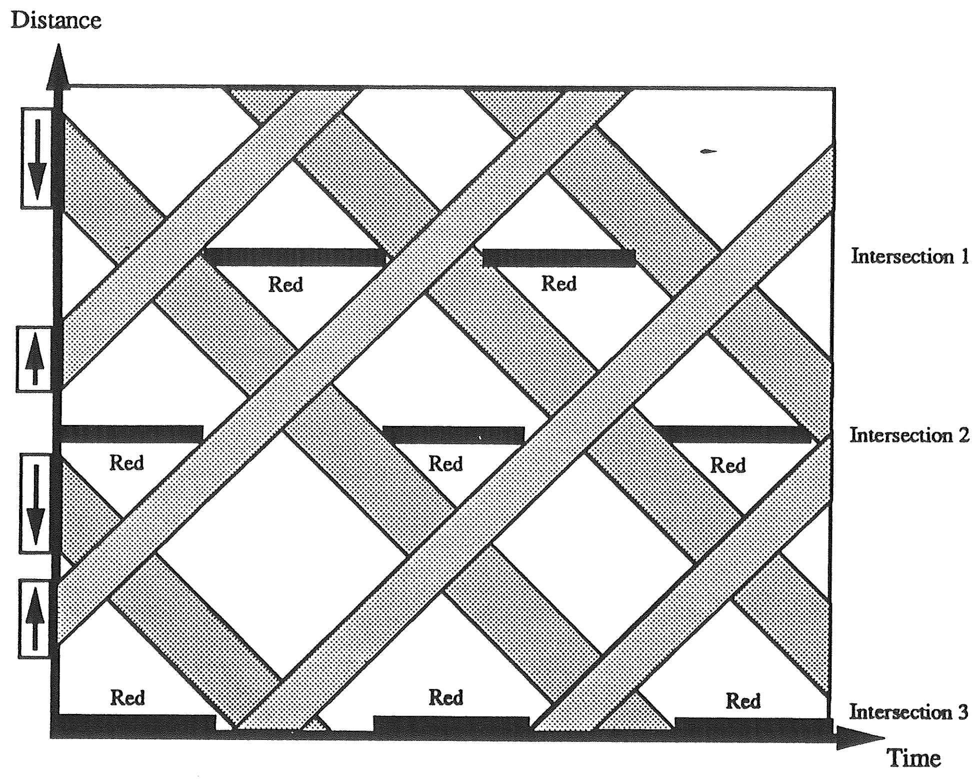


Figure 12. The MAXBAND Concept

convenience we haven't shown this in the figure. In fact, in the illustrations to follow, we have not shown turning vehicles which could grow or decrease platoon sizes and have purposefully exaggerated the speed differences in the figures. This is so that proposed concept for network flow control is visualized easier. The platoon flow prediction algorithm (to be developed in the future) should take into account these underlined characteristics.

The platoons, if any, on intersecting arterials have still not been considered above. If the intersecting platoons fitted exactly within the red times shown here, then we do not need to resolve green-time demand of conflicting movements. On the other hand, if flows at an intersection produced a concurrent green-time demand for conflicting movements, then the conflict has to be resolved in determining to which movement the green time must be allocated.

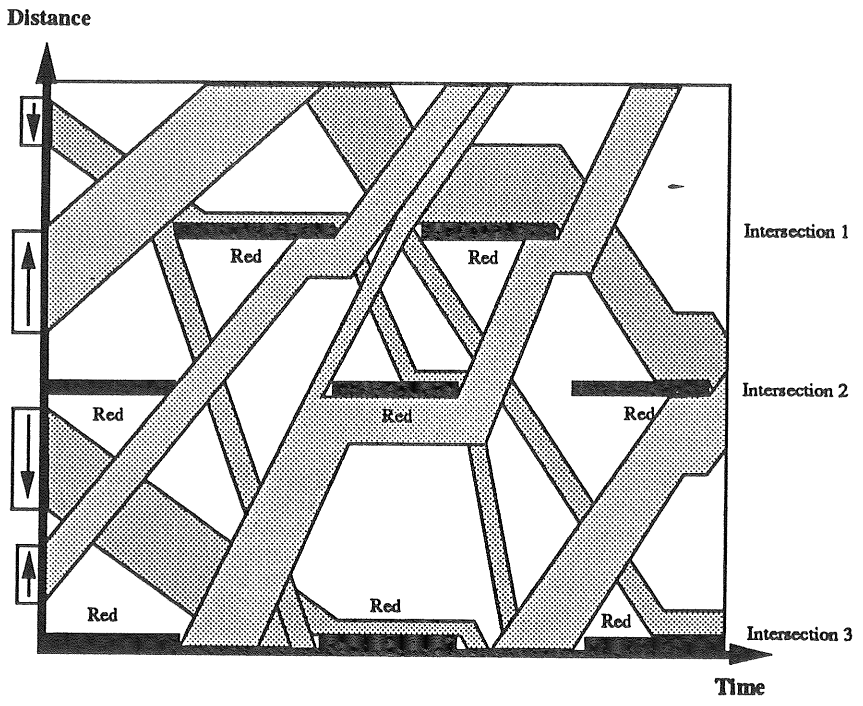


Figure 13. Actual MAXBAND Performance

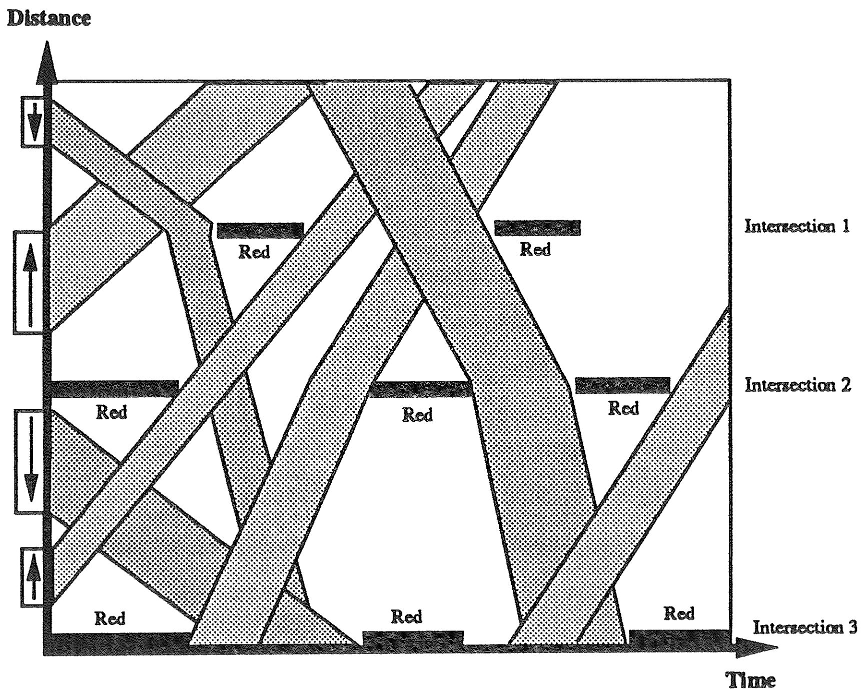


Figure 14. The REALBAND Concept for a Single Arterial

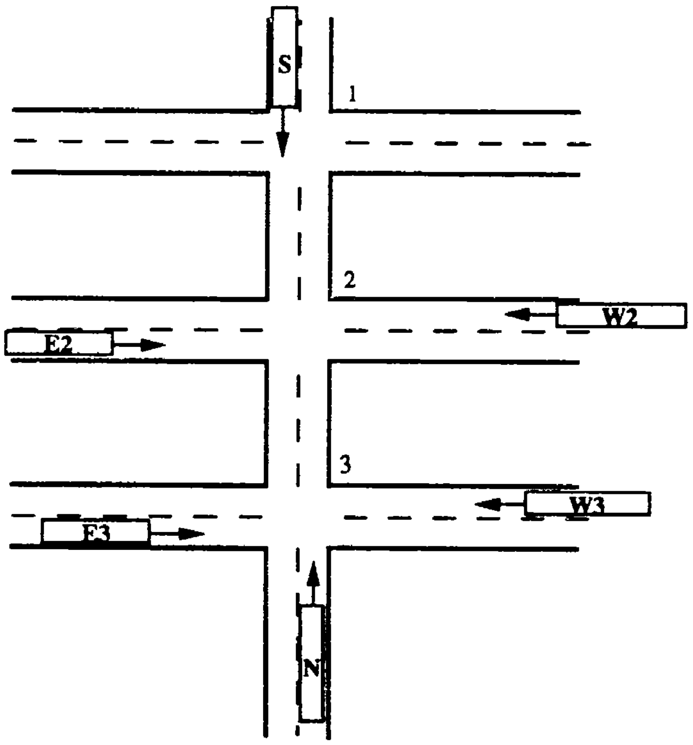


Figure 15. REALBAND Example Network

Figure 15 now shows platoons on two other perpendicular arterials , at intersections 2 and 3. Our approach -- for brevity we refer to it as *REALBAND* -- makes a forward pass in time. When a conflict arises, a decision node in a tree is formed; the types of decisions at this node are (1) give green time to platoon A, (2) give green time to platoon B, (3) split platoon A (or platoon B, since only one or the other platoon needs to be split). Each branch of the tree is propagated over time, keeping track of the total performance up to the decision node plus the performance on the link associated with the potential decision. Note that we use an implicit approximation on the additive nature of the performance measure as we propagate from node to node in the decision tree.

Figure 16 is the current prediction of the movement of the platoons shown in Figure 15. The first demand conflict arises between platoon N and W3 at Intersection 3. At this point we either split platoon N (Figure 17) or stop platoon W3 (Figure 18). Considering the resulting predictions shown in Figure 17, the next conflict arises between platoon S and E3. Here the decision is to either stop platoon S (Figure 19) or stop E3 (Figure 20). In this way, a decision tree is formed which keeps track of various candidate decisions as demand conflicts arise. For this illustration, the predictions that arise for various decisions are given in Figures 17-27 and the decision tree that is formed is given in Figure 28.

When the time horizon is reached, associated with each end node will be the total cost of the cumulative decisions made up to the node, on the path from the root of the decision tree to the end node (leaf) of the decision tree. Choosing the one with minimum cost provides the least cost trajectory of conflict resolution decisions. A final backward pass provides a phase plan, within the time horizon considered, for the identified platoons. This is passed to the third level of the hierarchical traffic control system (intersection control logic) as an <u>initial solution</u> for further optimization as more detailed data is gathered at the intersection level. Choosing the root to end-node path with optimum performance (in this case minimum total delay), the optimal decisions from the decision tree for this illustrative problem, for the platoons shown in Figure 15, are given in Figure 25. Figure 27 shows the red and green times for the N-S arterial for the decisions implied by Figure 25. It indicates that at Intersection 3 platoon N should not be stopped but platoon W3 should be stopped and, later, platoon E3 should not be stopped but platoon S should be stopped, when the corresponding demand conflicts arise.

Two advantages of the REALBAND approach should be noted:

(1) Using real-time data, it explicitly identifies the platoons in the network and their sizes and speed, and responds to these real platoons.

(2) It does not necessarily require a pre-determined sequence of phases. The output provides an initial cut at a sequence of phases for further optimization at the lower intersection/interchange level.

A final issue that needs to be resolved in this method is the computation of performance measures, for example, the total number of stops, total delay, etc. This is where concepts from TRAF NETSIM and TRANSYT are utilized to come up with a *quick-and-dirty* simulation to evaluate the performance of a set of signal settings. Again, for real time

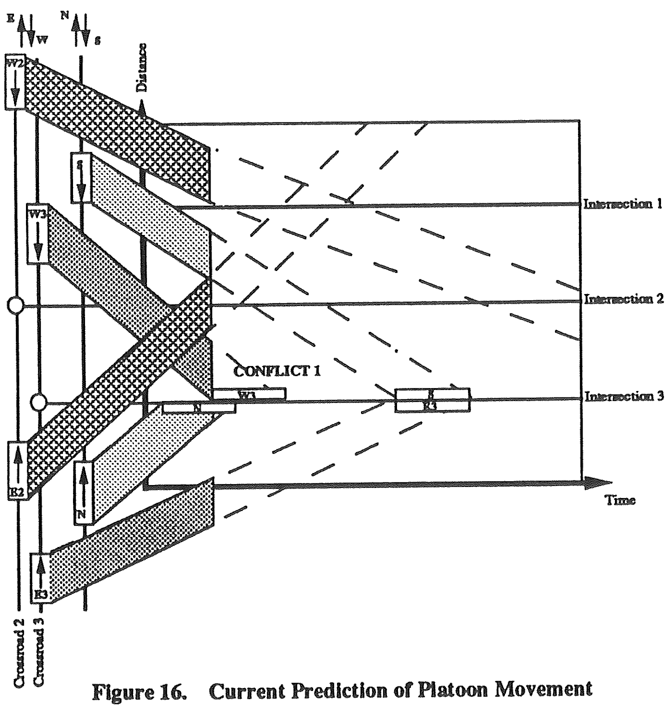


Figure 16. Current Prediction of Platoon Movement

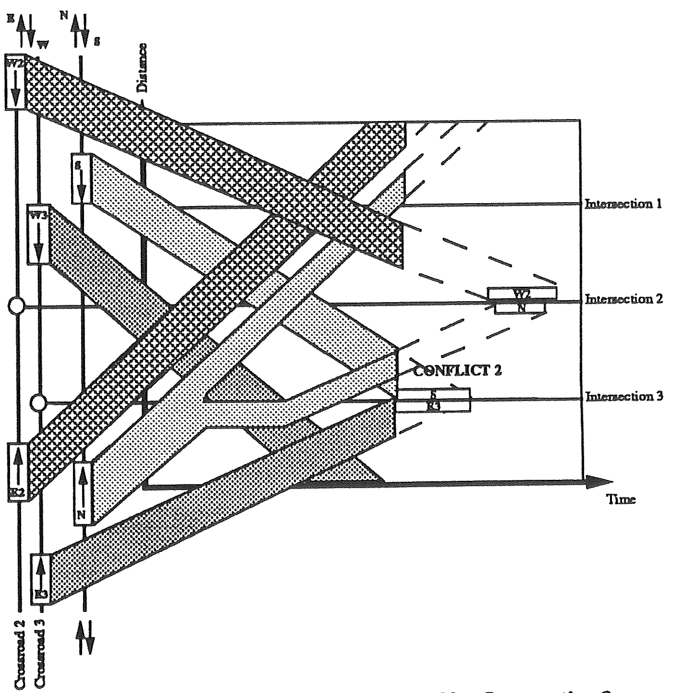


Figure 17. Decision to split Platoon N at Intersection 3

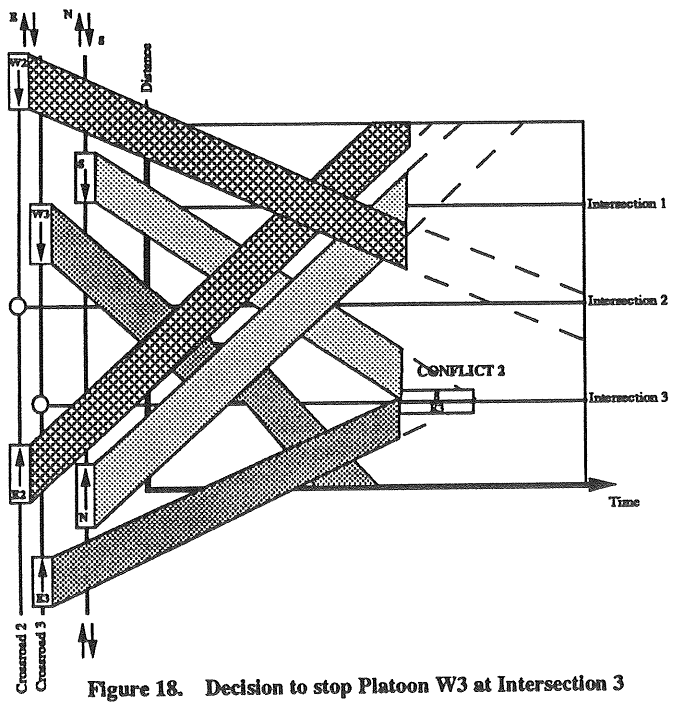


Figure 18. Decision to stop Platoon W3 at Intersection 3

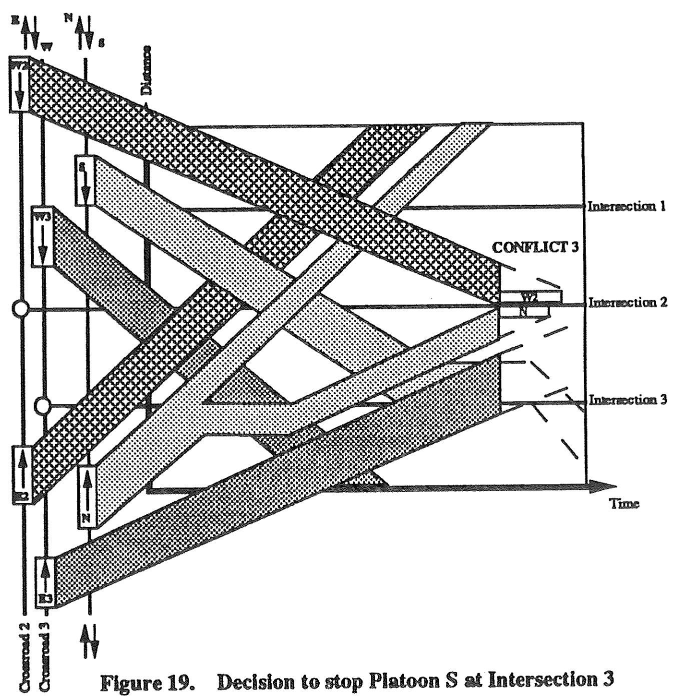


Figure 19. Decision to stop Platoon S at Intersection 3

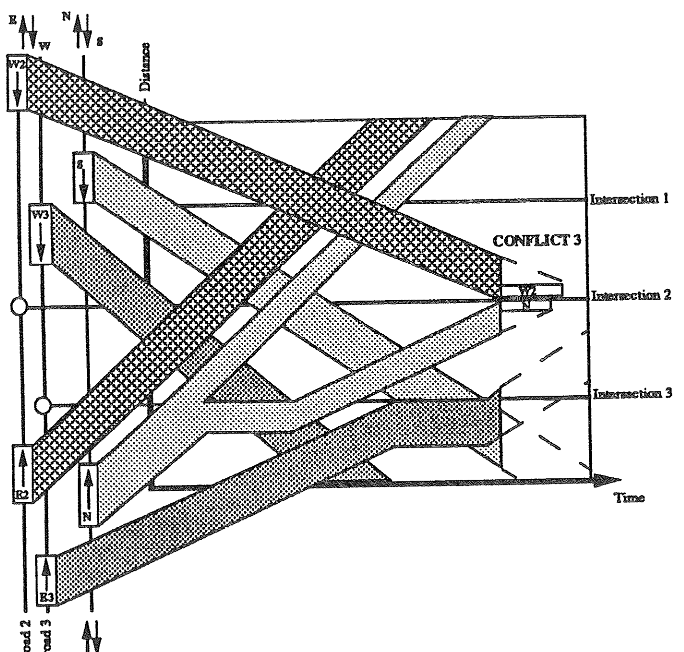


Figure 20. Decision to stop Platoon E3 at Intersection 3

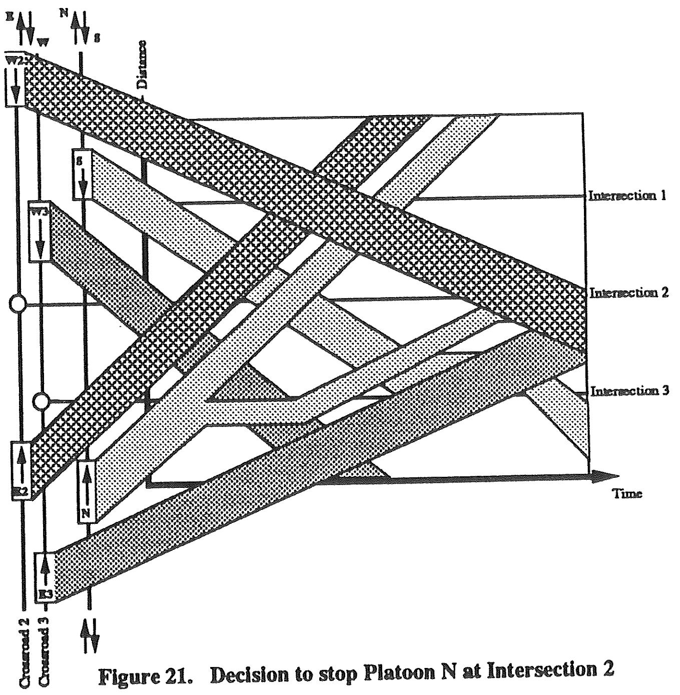


Figure 21. Decision to stop Platoon N at Intersection 2

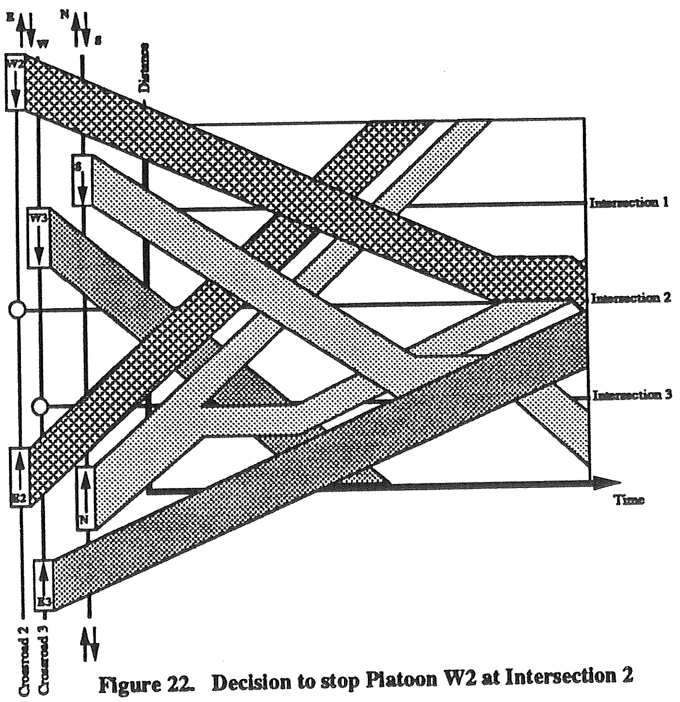


Figure 22. Decision to stop Platoon W2 at Intersection 2

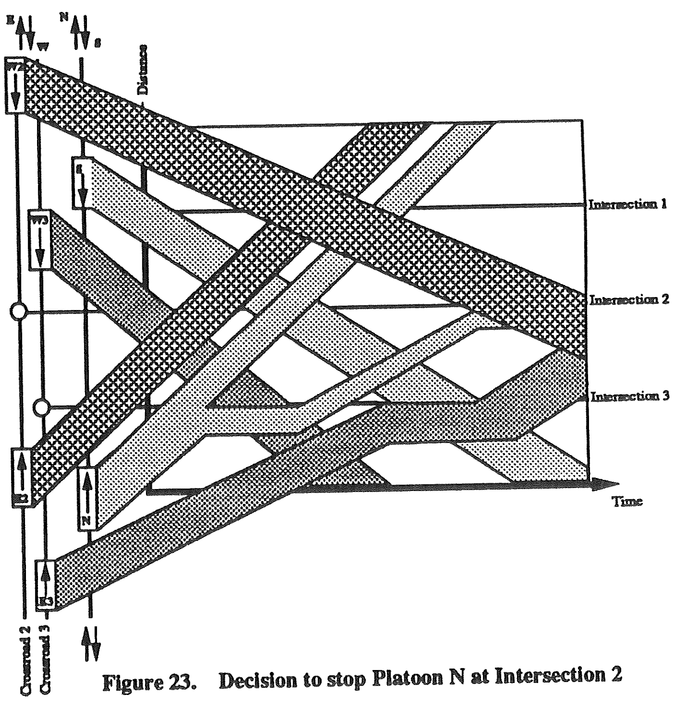


Figure 23. Decision to stop Platoon N at Intersection 2

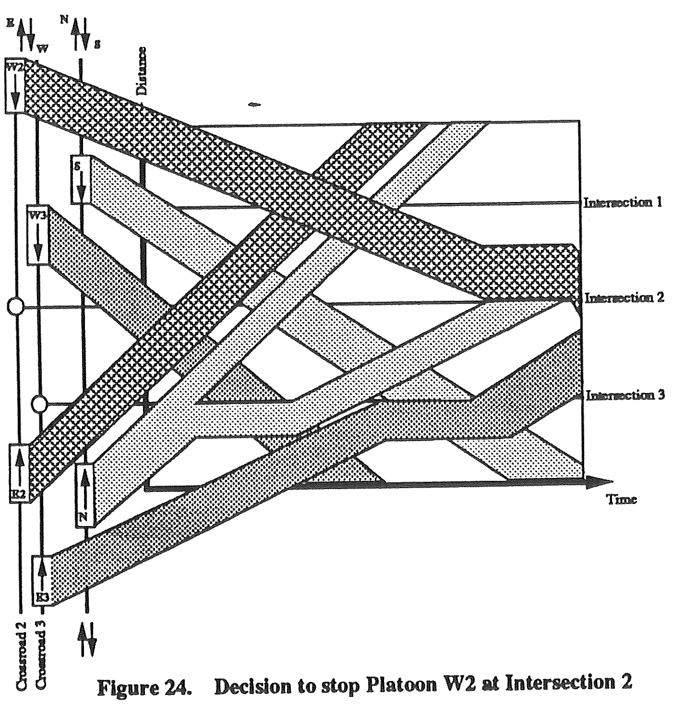


Figure 24. Decision to stop Platoon W2 at Intersection 2

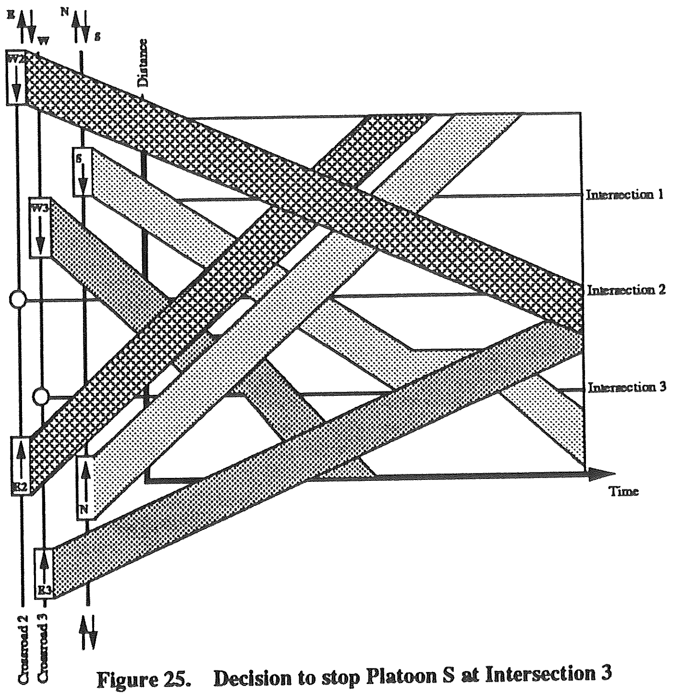


Figure 25. Decision to stop Platoon S at Intersection 3

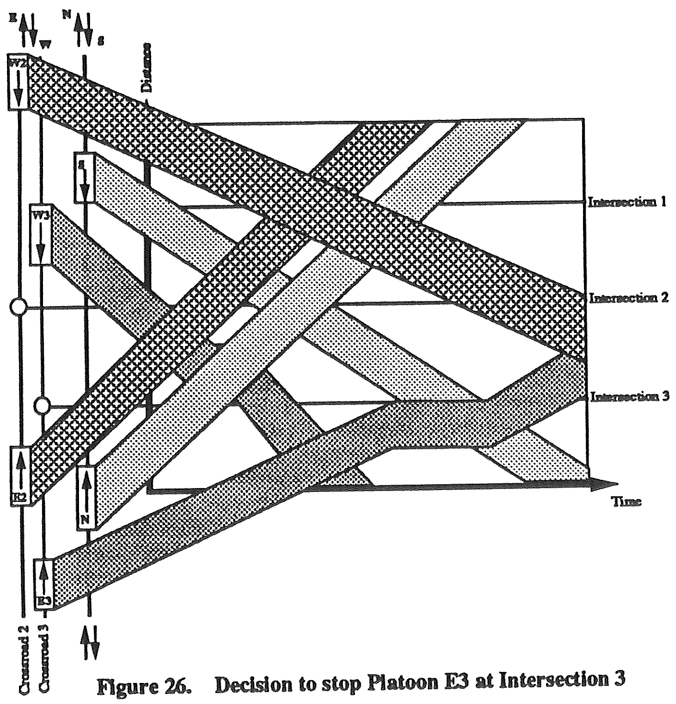


Figure 26. Decision to stop Platoon E3 at Intersection 3

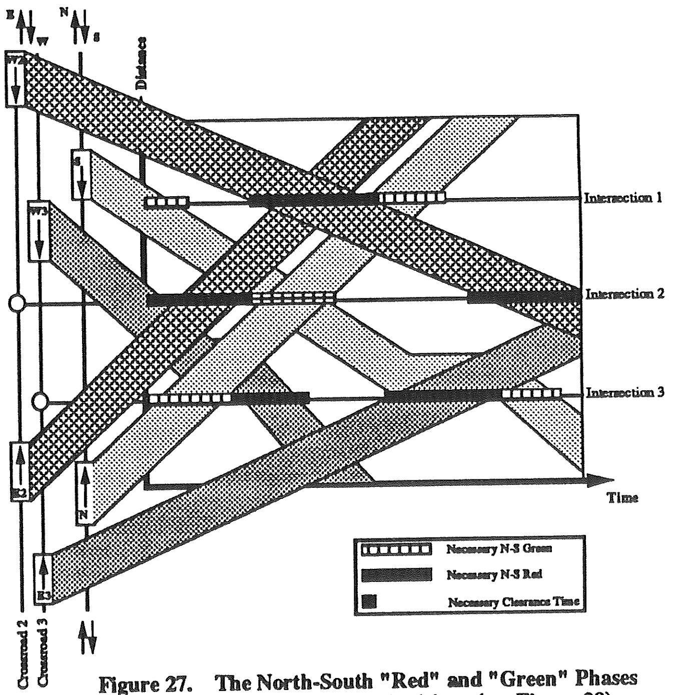


Figure 27. The North-South "Red" and "Green" Phases for the Optimum Decisions (see Figure 28)

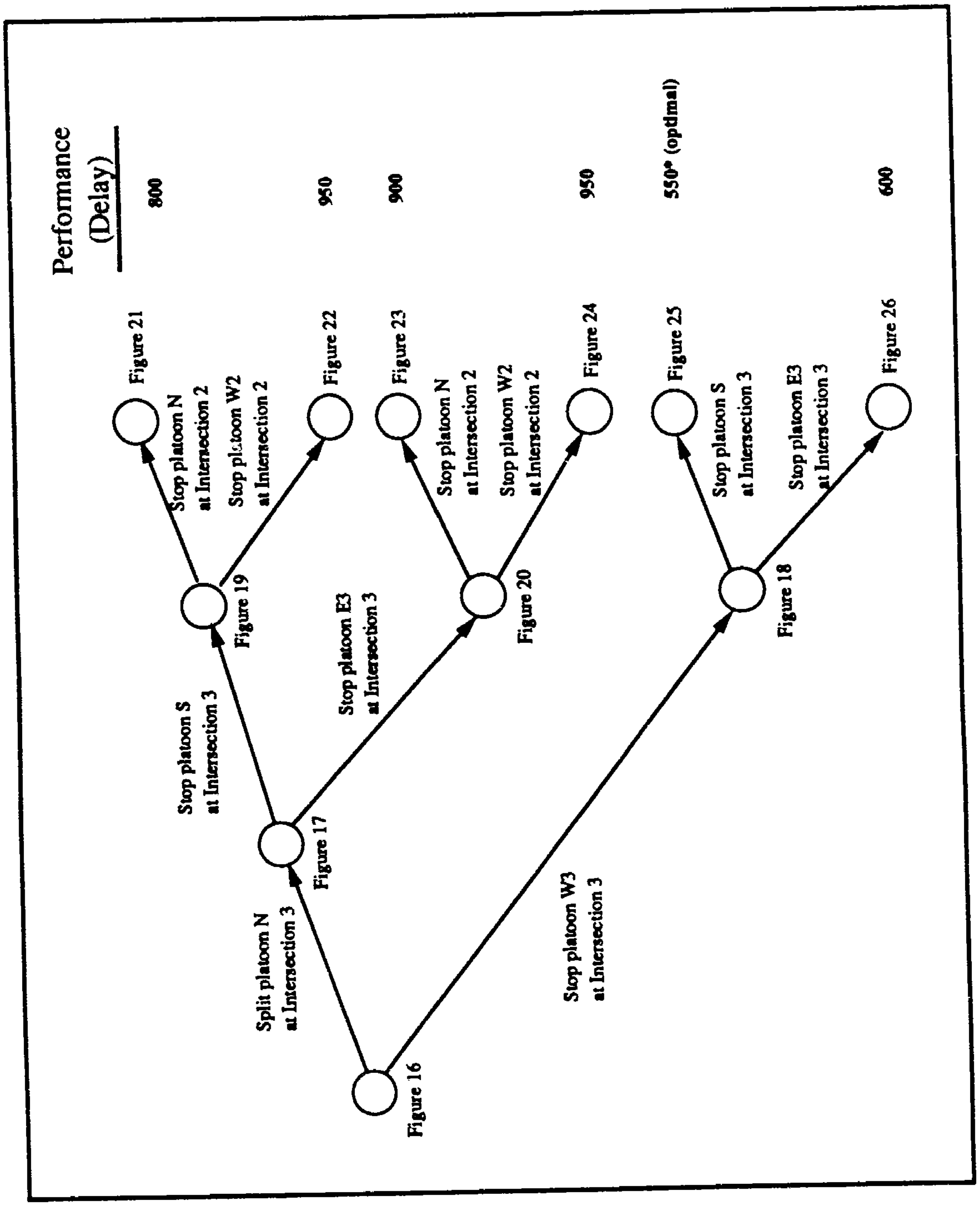

Performance (Delay)
Figure 16
Split platoon N at Intersection 3
Stop platoon W3 at Intersection 3
Figure 17
Stop platoon S at Intersection 3
Stop platoon E3 at Intersection 3
Figure 18
Figure 19
Figure 20
Stop platoon N at Intersection 2
Stop platoon W2 at Intersection 2
Stop platoon N at Intersection 2
Stop platoon W2 at Intersection 2
Stop platoon S at Intersection 3
Stop platoon E3 at Intersection 3
Figure 21
Figure 22
Figure 23
Figure 24
Figure 25
Figure 26
800
950
900
950
550* (optimal)
600

Figure 28. Decision Tree for Illustrative Problem

applications, these performance measures are needed quickly so that the performance criterion may be optimized in real time. A detailed simulation becomes computationally unwieldy when one uses the simulation model as a *function evaluator* (i.e., for evaluating the performance function for each candidate signal setting) in an optimization routine. We remark that at this second level of hierarchy only approximate values for optimal signal timings are necessary so that we start in the right "ball park" for the optimizations being performed at the intersection level. We refer to this simulator as *Platoon Flow Fast Prediction* model, (the current version developed jointly with Mr. Paolo Dell'Olmo of Instituto di Analisi dei Sistemi ed Informatica, Rome, Italy, is called *OPTINET*).

The flow chart for the algorithmic process for network flow control optimization is given in Figure 29. Note that to begin the recursion it is suggested that an initial signal plan be given. This is so that the network flow control optimization itself begins in the right "ball park". The initial plan may be obtained from an off-line method such as TRANSYT [Hadi and Wallace, 1992] with data from the dynamic network loading function, if available, or from historical data.

In relation to the use of the fast simulation model for function evaluation, the spatial region for the simulation model needs to be bigger than the area of control for network coordination, so that one can predict the movements of all real-time platoons within the region of control for several minutes in the future. As an illustration, Figure 30 shows the region of control for the network flow control logic and Figure 31 shows the area simulated using the *Platoon Flow Fast Prediction* simulator.

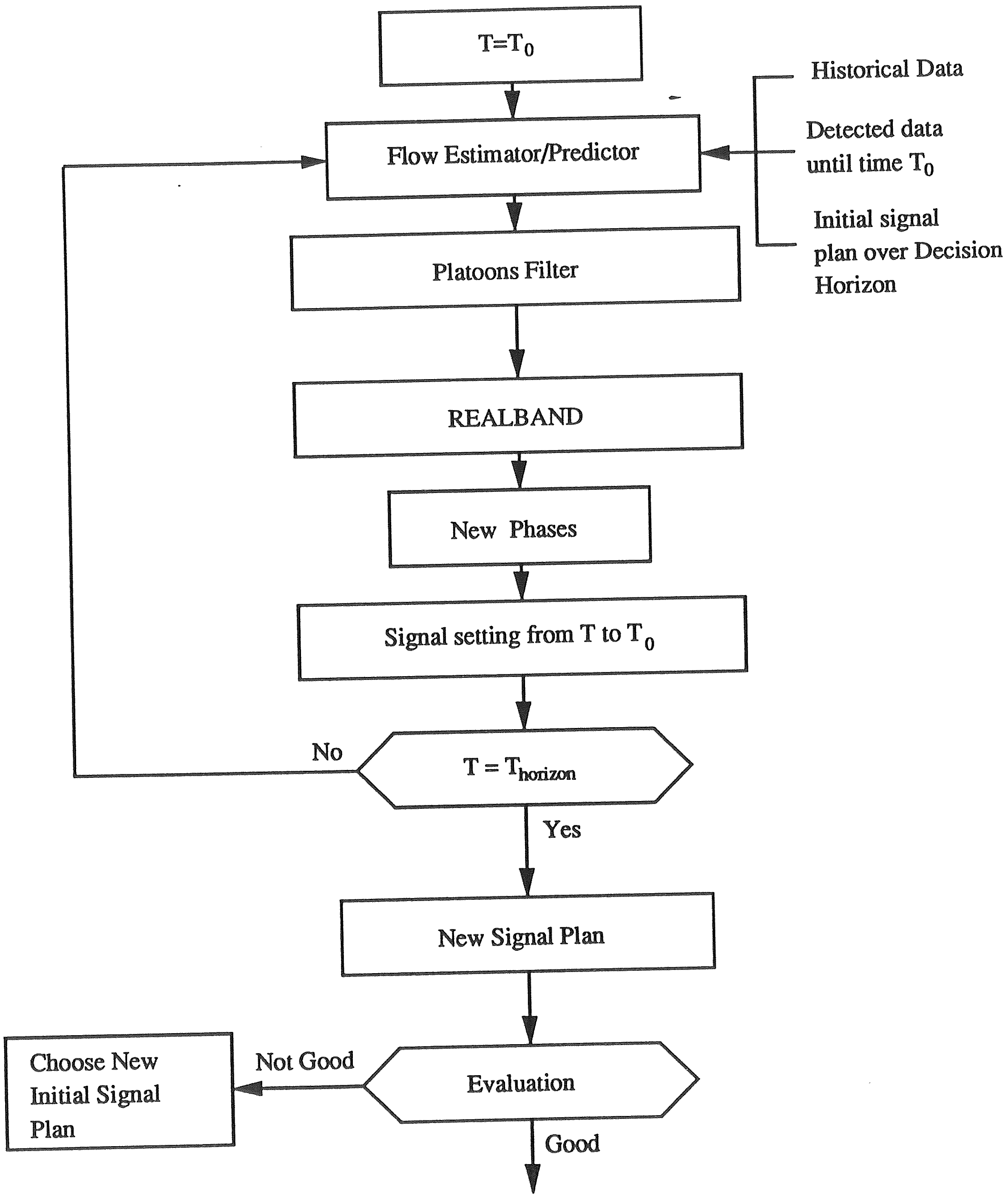


Figure 29. Flow Chart for Network Flow Control Optimization

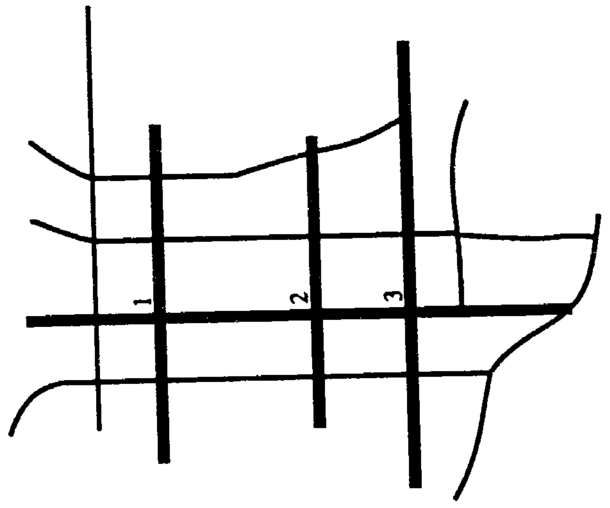


Figure 30. Region for Network Flow Control

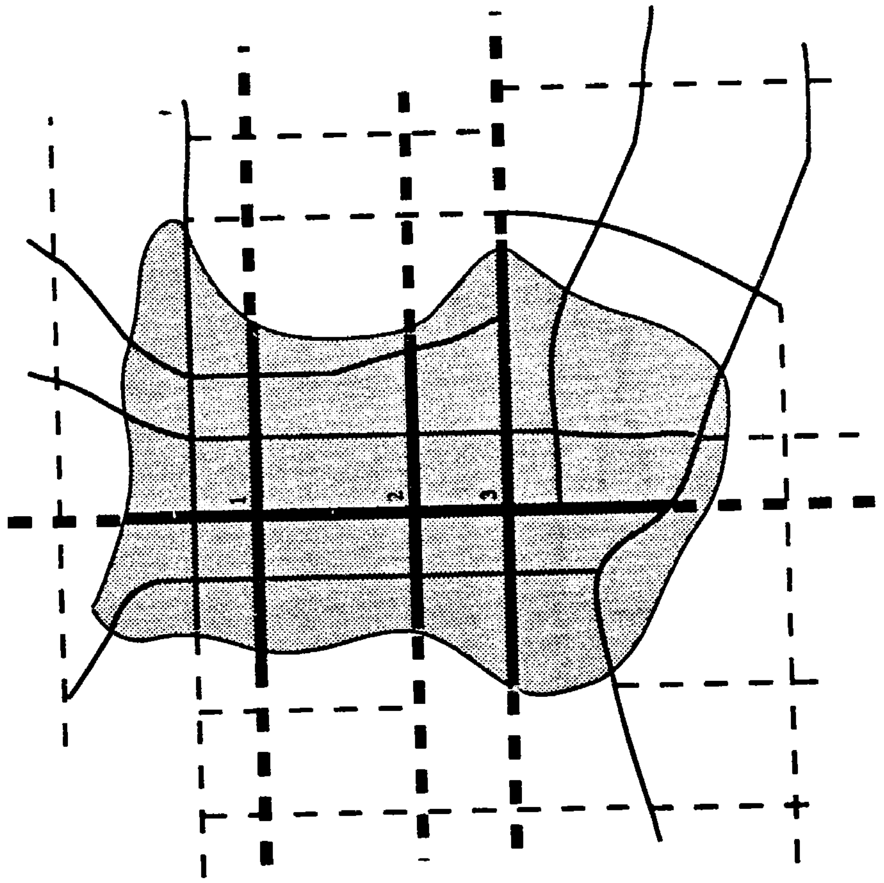


Figure 31. Region for Simulation

5. INTERSECTION CONTROL

5.1 Existing Intersection Control Algorithms

In this section we discuss intersection control algorithms. First, we review existing control logic, including fixed-time, semi- and fully-actuated, and traffic responsive. We then present two new intersection control algorithms, DISPATCH 1.0 and COP.

Existing control algorithms provide a framework for discussing issues related to network/intersection control adaptivity. For example, fixed-time signal timing plans (cycle length, phase sequence, phase splits, and offset) can be developed and coordinated to provide a simple form of network flow control, but do not provide any distributed intersection adaptivity. On the other hand, fully-actuated control provides distributed intersection adaptivity, but does not provide any adaptive network flow coordination. Semi-actuated control incorporates both network flow coordination and intersection control with limited adaptivity.

Each of these control algorithms, or strategies, are based on a signal timing plan that provides the fundamental operating parameters for signal control. These parameters are generally developed based on traffic studies and standard procedures, such as the Highway Capacity Manual, or signal timing software such as TRANSYT, PASSER, SOAP, etc. The traffic studies result in estimates of traffic conditions, link volumes and turning percentages, for specified time periods. Signal timing plans are developed for each of these time periods and, typically, implemented on a time-of-day basis with no real consideration of current traffic conditions. Such approaches have been somewhat successful when sufficient resources are available to update the traffic studies to reflect changes in long term traffic patterns, or when the operating personnel heuristically adjust the signal timing plans to improve performance. In many cases the use of standard procedures for the development of signal timing plans is abandoned and traffic engineers operate in an ad-hoc fashion with moderate levels of success.

Traffic responsive systems attempt to address the problem of responding to current traffic conditions by switching signal timing plans based on current wide-area traffic conditions rather than time of day. This requires that signal timing plans be developed for a variety of possible traffic conditions.

Methodological and technological advances are now available that allow traffic signal control systems to utilize real-time traffic information and to provide traffic-adaptive control to improve the performance of the signal control system. In this section several existing signal control strategies are reviewed and some new strategies within the RHODES framework are discussed.

5.1.1 Fixed-Time Control

Fixed-time control utilizes a very simple logic based on predetermined fixed cycle length, phase sequences, and durations. Fixed-time control requires no information about existing traffic conditions on the network.

Fixed-time control logic relies on historical data to prepare timing plans for a signalized area. These timing plans may be developed off-line using, for example, the TRANSYT optimization program, which optimizes coordinated traffic signal systems to reduce delay, stops, and fuel consumption. Three to four plans, representing the a.m. peak, p.m. peak, and off-peak conditions, are commonly used. A particular plan is generally switched into operation according to time-of-day.

Vehicle detectors are not required with this method. Coordination of intersections is achieved by linking local controllers to a master controller by a communication network. A fixed-time system can be implemented in the form of a cableless-linked system with the use of crystal clocks in local controllers. After synchronization with a master clock, local crystal clocks are left to run on their own timings. Timing plans are stored in programmable ROMs (Read-Only Memory) which are installed in each controller. (Modern controllers use microprocessors for supervisory functions and semi-conductor memories for the storage of timing plans.) Without laying cables, a resulting drawback is the lack of feedback from the field signals.

A fixed-time control system is simple in structure and does not require vehicle detection, which is usually the most unreliable component in an area or urban traffic control (ATC or UTC) system. An ATC system consists of a number of traffic signals which are linked in such a way that any signal timing change is in some way dependent upon conditions (i.e., vehicle detections) at any of the other intersections. The system of signals may consist of a single linked pair, a linear group, or a complete network. The control system is concerned with the selection and implementation of three control elements for every signalized intersection in the network: cycle time, phase splits, and offset (an offset is the time difference in the starting times of the green phases at adjacent intersections). As a set, they constitute the timing plan for the intersection (Luk, 1984).

Fixed-time control systems are, however, inflexible in their operations. Unforeseen incidents such as bad weather conditions, illegal parking, or accidents would create traffic patterns that could not be matched by any of those predetermined timing plans. Therefore, a fixed-time method with optimized signal setting is commonly used as a standard for the evaluation of other traffic control methods.

5.1.2 Semi- and Fully-Actuated Control

In semi- and fully-actuated control, timing plans are selected by time-of-day as in fixed-time control, but vehicle-actuated tactics are allowed at each intersection. These tactics depend on the structure of the controller, but usually include gapping due to expiry of gap or waste-time timers, phase skipping in the absence of demand, and green time transfer. Semi-actuated control allows for traffic actuation on one or more, but not all, approaches to an intersection. Whereas, in fully-actuated control traffic actuation occurs on all approaches. Also, the goal of semi-actuated control is to maintain some level of system-wide coordination.

5.1.3 Traffic Responsive Control

In traffic responsive control, the timing plans are selected to match current flow patterns by using vehicle detectors to monitor traffic volumes and occupancies at strategic locations. There are many forms of traffic responsive control implementation which have various levels of traffic adaptability.

5.1.4 Real-time Intersection Control - OPAC

The Optimization Policies for Adaptive Control (OPAC) is the most notable example of a real-time isolated intersection control strategy. OPAC is based on a dynamic optimization algorithm that provides the computation of signal timing without requiring fixed cycle time in response to the quantity of traffic with the objective on minimizing vehicle delays constrained by minimum and maximum green times. OPAC has been developed and field tested (Gartner, 1989).

OPAC has matured through four major versions: (1) OPAC-1; (2) OPAC-2; (3) OPAC-RT Version 1.0; and (4) OPAC-RT Version 2.0. The original formulation of OPAC, OPAC-1, was based on dynamic programming. The formulation assumed that time is divided into time intervals, or stages, of 5 seconds and that there are only two possible phases for the 4 approaches, each phase is either North-South through or East-West through. The initial queues and initial signal phase are specified. The decision at each stage is to either leave the signal in its current phase (indicated by "0") or to change the phase (indicated by "1"). The number of vehicle arrivals at each stage is assumed to be known. The performance measure is delay, which is calculated as the delay associated with the current-stage decision (change or no change) plus the previous stages delay and queues.

OPAC-1 is useful only for evaluation purposes, since the algorithm assumes the arrival pattern is known exactly and every possible decision for every possible initial queue length is evaluated. To simplify the computational burden of OPAC-1, OPAC-2 was developed using a fixed decision horizon (approximately one cycle) and the decision intervals were held at 5 seconds. During each decision horizon at least one phase change is required but as many as three are allowed. The performance measure for the decision horizon is calculated as the sum (over all intervals) of performances for each interval. The performance for each interval computed as the initial queue plus the arrivals minus the departures. Instead of evaluating every possible decision at each interval, an optimal sequential constrained search (OSCO) is used to solve the optimization problem of minimizing delay. OSCO limits the search to only valid phase switching times constrained by minimum and maximum phase times and the limit of no more than three, and at least one, phase change during the decision horizon.

OPAC-2 is better suited to real-time implementation, but still requires knowledge of the number of arrivals in each interval. To address this problem, a "rolling horizon" approach is employed that only assumes knowledge of arrivals over three or four intervals and an average number of arrivals in the future intervals. The knowledge of arrivals over the first three or four intervals is obtained by a detector placed upstream from the intersection, and therefore, the actual arrival pattern is used during the initial three or four intervals. The optimization is performed every three or four intervals using this rolling horizon approach. (The above description of OPAC-2 is summarized from "Evaluation of the Optimized Policies for Adaptive Control Strategy," Farradyne Systems, Inc., May 1989.)

OPAC-RT Version 1.0 was developed based on the rolling horizon approach of OPAC-2 with the proper interfaces to signal controller and detector devices. In addition, OPAC-RT Version 1.0 allowed only one or two phase changes, and hence, eliminated the possibility of three phase changes for the decsion horizon. The minimum green-time constraints were modified for pedestrian calls, according to user-specified pedestrian clearances.

OPAC-RT Version 2.0 extended Version 1.0 to include all eight phases of a dual ring controller. Actually, only the two main phases (typically the through phases) of the intersection are considered in the optimization; the other phases are treated using the gap out/max out strategy of actuated control.

OPAC has been field tested in two locations: Arlington, VA and Tucson, AZ. In both cities the two-phase version of OPAC was implemented and found to reduce delay by 3.9% (VA) and 15.9% (AZ) over actuated control. Based on the two field tests, it has been concluded that OPAC performs better in higher volume conditions. An eight-phase version of OPAC was tested in Tucson, AZ, and found to reduce delay by 7.7%. These initial field tests indicate that traffic-adaptive signal control can improve intersection performance and suggest that further algorithmic improvements, such as integration of multiple phases in the optimization and coordination with adjacent intersections, should be investigated in the future.

5.2 Traffic-adaptive Control Algorithms - DISPATCH and COP

In this section two traffic-adaptive algorithms are presented that were investigated for RHODES intersection control. The DISPATCH 1.0 algorithm uses a heuristic search routine to adjust split timings of an intersections that is operating in a coordinated system. It attempts to optimize either stops, delay, or a combination of stops and delay, by searching in the neighborhood of each phase switching time using observations of arriving traffic from a detector located several seconds upstream from the controlled intersection.

COP is a traffic adaptive algorithm based on a dynamic programming approach to intersection control that optimizes stops (and can be extended to include delay, queues, etc.) using predictions of vehicle arrivals at the controlled intersection which are based on data from detectors located at the upstream intersections and the signal timings of the upstream intersections. Currently, COP operates as an isolated intersection control algorithm. However, it has been formulated to allow integration within a coordinated system. Both of these algorithms have been tested using the TRAF-NETSIM simulation model.

5.2.1 DISPATCH 1.0

This section presents the results of implementing a heuristic which adjusts the exact phase switch epoch to adapt to the actual traffic arrival patterns. The contents of the data input file can be found at the beginning of the DISPATCH 1.0 program (Appendix A). DISPATCH 1.0 uses a dual ring controller as depicted in Figure 32. Two possible movements can occur simultaneously as a phase as follows: 1-5, 1-6, 2-5, 2-6, 3-7, 3-8, 4-7, 4-8.

The DISPATCH 1.0 program can be applied to intersections similar to the one depicted in Figure 33.(Note that the north direction is to the right. This convention was used in order to conform with TRAF-NETSIM phase scheme.)

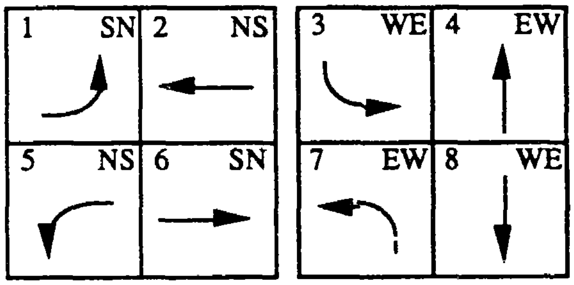


Figure 32. Dual ring controller.

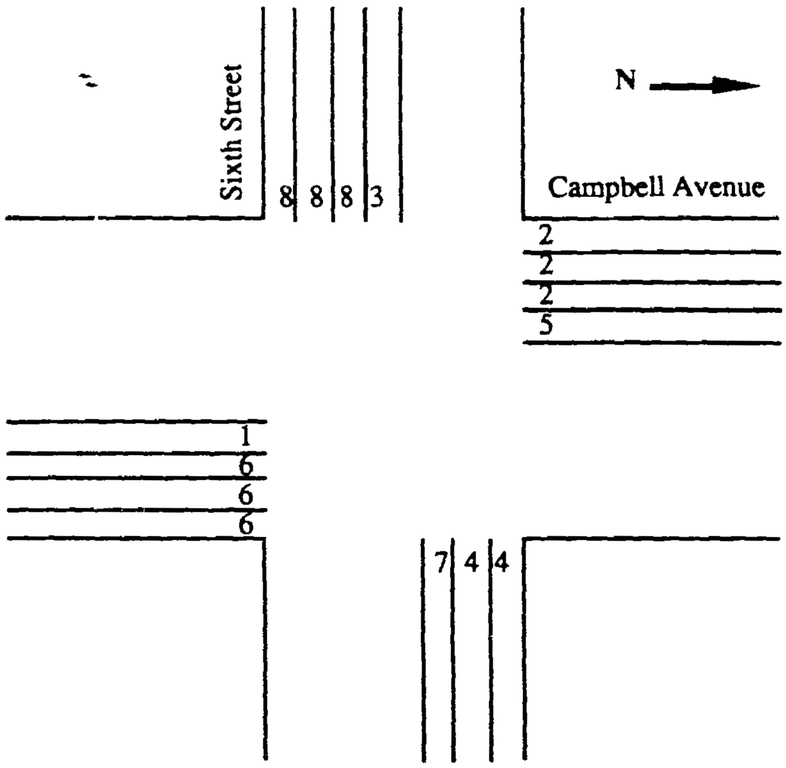


Figure 33. Campbell Avenue and Sixth Street Intersection, Tucson, Arizona.

Description of Heuristic

The initial objective of developing DISPATCH 1.0 was to model, simulate, and observe the behavior of the queues at each movement (1, 2, ..., 8). The initial model simply changed the phase at a deterministic point in time and computed the performance. However, the next step was to implement a method of evaluating the performance for range about that deterministic point in time, in order to search for the optimal time to change the phase.

For example, if the preliminary change of phase was at time t = 20 and the range was 7 (which must be less than or equal to the time required to travel from the detector to the stop-line), then the evaluation range could be from time t = 18 to time t = 24. The optimal time to change the phase could then be the time at which the best score, according to a linear combination of the total number of stops and the total delay, is obtained.

The model for dissipating the queue of each phase is as follows:

$$q(t+1) = q(t) + f_{in}(t-T) - f_{out}(t) \qquad (16)$$

where

q is the number of stopped vehicles in the queue,
$f_{in}(t)$ is the flow entering the upstream link at time t,
$f_{out}(t)$ is the flow leaving queue at time t,
t is the current time,
T is the upstream link travel time,

$$f_{out}(t) = \begin{cases} s, & \text{if } q \neq 0 \text{ and the signal is green} \\ f_{in}(t), & \text{if } q = 0 \text{ and the signal is green} \\ 0, & \text{the signal is red} \end{cases}$$

where s is the saturation flow rate (default = 1 vehicle/time unit/phase).

The DISPATCH Program

DISPATCH 1.0 has the following characteristics:

(1) uses discrete time units;
(2) has a defined time horizon (total time considered);
(3) incorporates a minimum green time for a phase to operate;
(4) incorporates a clearance time for vehicles (all phases red);
(5) is designed for eight phases;
(6) allows for up to 500 stages (the sequence of phases which can be modified);
(7) may use an intersection identification number;
(8) accepts arrival data for eight movements (this data corresponds to vehicle requests for a specific phases)as shown in Figure 32;and
(9) may easily be modified.

Furthermore, DISPATCH 1.0 requires some additional input. This information may be included in the data input file, but in the current implementation, it is requested as keyboard input. The following information is requested:

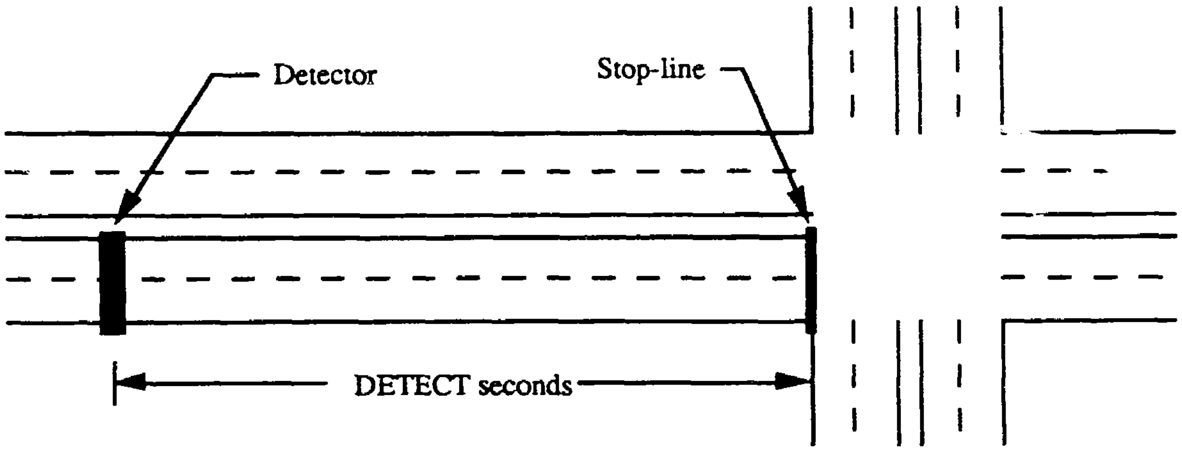


Figure 34. The time from the detector to the intersection stop-line is referred to as DETECT seconds. In current model, the time horizon, T, is equal to DETECT.

(1) total amount of time to be considered (i.e., decision horizon);
(2) saturation flow rate;
(3) travel time from detector to stop-line (Figure 34);
(4) initial point in time to change phase; and
(5) weight for total stops (used in calculating a linear combination of stops and delay, weight for delay is automatically calculated).

The initial four inputs have default settings which can be set by entering 0 (zero). The program also accounts for any possible input errors such as utilizing weights which sum up to greater than one. Unless otherwise noted, all inputs must be integer values.

5.2.2 DISPATCH 1.0 Example

Partial output of a run of the DISPATCH 1.0 program is shown in Figures 35, 36, and Table 2 (a complete listing of the output is in Appendix B). The initial information verifies the following input data: intersection identification number, number of phases, minimum green time, clearance time, total number of stages, sequence of stages, time horizon, arrival data, saturation flow rate, and time delay from detector to stop-line. The solution then lists the linear combination of total stops and total delay, the optimal score and time when the phase should be changed, the optimal queue matrix for the time horizon, and a complete analysis of all of the possible phase change points in the evaluation range, which in this case is from time t = 154 to time t = 163. This analysis includes the stops and delay for each phase, the total stops, total delay, and the score for changing the active phase at respective times.

```
---------------------------------------------------------
Information for Intersection Identification #123456789
---------------------------------------------------------
Number of Phases =      8      Minimum Green Time =   2

Clearance Time =        1   Total Number of Phases =  2

   Sequence of Phases:     37  48
   Time Horizon =          314

              Arrival Data
              -----------------------------------
              Time    1  2  3  4  5  6  7  8
              -----------------------------------
                 0    0  0  0  0  0  0  1  0
                 1    0  0  0  1  0  0  0  0
                 2    0  0  0  0  0  0  0  1
                 3    0  0  0  0  0  0  0  0
                 4    0  0  0  0  0  0  0  1
                 5    0  0  0  1  0  0  0  0
                                     .  .  .
               152    0  0  0  0  0  0  0  0
               153    0  0  0  1  0  0  0  0
               154    0  0  0  0  0  0  1  0
               155    0  0  2  1  0  0  0  0
               156    0  0  1  0  0  0  0  0
               157    1  0  0  0  0  0  0  0
               158    0  0  1  0  0  0  0  1
               159    0  0  0  0  0  0  0  1
               160    0  0  0  0  0  0  0  0
               161    0  0  0  0  0  0  0  0
               162    0  0  0  1  0  1  0  0
               163    0  0  0  0  0  0  0  0
               164    0  0  0  0  0  0  0  0
                                        .  .  .
               309    0  0  0  0  0  0  0  0
               310    0  0  0  0  0  0  0  0
               311    0  0  0  0  0  0  0  0
               312    0  0  0  0  0  0  0  0
               313    0  0  0  0  0  0  0  0

              Current saturation flow rate =      1
              Delay from detector to stop-line =  10
```

Figure 35. Input parameters and detector data for DISPATCH 1.0.

```
-------------------------------------------------------------
Optimal Performance Index = 15631.000 When Changing Phase at t=154
-------------------------------------------------------------

Performance Index Weights for Phase Change at time t∈[154,163]
-------------------------------------------------------------
Performance Index = (0.25000*Total Stops) + (0.75000*Total Delay )

Results for Changing Phase at Time:     154
-------------------------------------------------------------
 Mvmt:     1     2     3     4     5     6     7     8     Total
 Stops:    22    28    26    17    2     29    6     54    184
 Delay:    3685  3164  2163  1417  218   3927  355   5851  20780
-------------------------------------------------------------
     Performance Index =      15631.0000
```

Figure 36. Tabulation of stops and delay on each movement when switching at the optimal point.

Figure 35 is the initial output and contains an echo of the input parameters and the vehicle detections for each movement and each time unit during the entire time horizon. For example, the "2" at time 155 under movement 3 represents an aggregated total of two vehicle detections on that movement. Table 2 shows the optimal point to switch phases, and the cumulative queues for each movement resulting from the decision obtained in DISPATCH 1.0.

Figure 36 is a tabulation of the total stops and total delay on each movement as a result of switching the phases at the optimal point in time. The units for total delay are minutes. These values may appear large, but they are calculated over the entire time horizon rather than only the range considered for determining the phase switching point.

Thus, the optimal point in time to switch phases is at time 154 (i.e., the next phase begins at time 155). The switching point was initially set for time 159. DISPATCH 1.0 then evaluated the possible switching points about time 158 (i.e., from time 154 to time 163). For a more detailed listing of this output, refer to Appendix B.

To observe this adjustment in the switching point graphically, refer to Figure 37. If the times in the first column of Table 2 are divided by the cycle time (90), these remainder values become the synchronization clock times which are depicted in Figure 37. The initial point to switch from phase 37 to phase 48 was set at time t = 159 (sync time 69 in figure). Then, once DISPATCH 1.0 was executed on the data, the new switch time was

Table 2. Optimal switching point and display of queues designated by "***".

Optimal Queue Matrix

T	1	2	3	4	5	6	7	8	Phase
0	0	0	0	0	0	0	0	0	37
1	0	0	0	0	0	0	0	0	37
2	0	0	0	0	0	0	0	0	37
				•	•	•			
152	14	7	0	17	0	12	0	49	37
153	14	7	0	17	0	12	0	49	37
154	14	7	0	17	0	12	0	50	37
155	14	7	0	17	0	12	0	50	48 ***
156	14	7	0	16	0	12	0	49	48
157	14	7	0	15	0	12	0	48	48
158	14	7	0	14	0	12	0	47	48
159	14	7	0	13	0	12	0	47	48
160	14	7	2	12	0	12	0	48	48
161	14	7	3	11	0	12	0	48	48
162	14	7	3	10	0	12	0	49	48
163	14	7	3	9	0	12	0	48	48
164	14	7	3	9	0	12	0	47	48
165	14	7	3	8	0	12	1	46	48
			•	•	•				
311	22	28	26	0	2	28	6	0	48
312	22	28	26	0	2	28	6	0	48
313	22	28	26	0	2	28	6	0	48

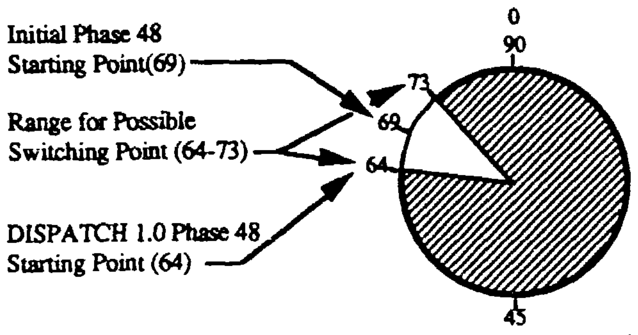


Figure 37. Graphical display of DISPATCH 1.0 example.

determined to be time t = 154 (sync time 64). Hence, DISPATCH 1.0 evaluated the possible switch points from time t = 154 to t = 163 (from sync times 64 to 73) and, based upon a combination of total stops and delay, selected the optimal switching point as t = 154.

The DISPATCH 1.0 program met the initial objectives. That is, it models, simulates, and determines the optimal point in time (in a range) to change the active phase combination for an intersection. In the near future, DISPATCH 1.0 will be implemented with fixed-time and semi-actuated control logic to determine the best time to change the phase combination based upon TRAF-NETSIM detector data and return this decision back to TRAF-NETSIM for implementation.

5.2.3 COP - Algorithm for Intersection Control

In this section the COP (Coordinated Optimization of Phases) algorithm is presented [Sen, 1991]. Let s_j represent the amount of time remaining to be allocated given that $j-1$ phase changes have occurred. The decision at stage j denoted u_j is the amount of "green" time to allocate to the phase corresponding to phase j. Then,

$$s_{j+1} = s_j - u_j. \tag{17}$$

For the purposes of this illustration, assume that the objective is to minimize the number of stops. Let $z_j(s_j,u_j)$ denote the number of stops in the time interval $(T-s_j, T-s_j+u_j)$ where T is the decision time horizon. Then the objective function may be written as

$$Z_J(s_J) = \underset{\{u_j\}}{\text{Minimum}} \sum_{j=1}^{J} z_j(s_j,u_j) \tag{18}$$

where J is the maximum number of phase changes possible in time T. Since the objective function is additive, we get the following dynamic programming recursion

$$Z_j(s_j) = \underset{u_j}{\text{Minimum}} \{z_j(s_j,u_j) + Z_{j+1}(s_{j+1})\}. \tag{19}$$

The phase duration's, u_j , are determined by the solution to the dynamic program (19). The exact phase change epochs are determined from the u_j' s.

COP with Eight Phases

A C program was written which utilized the backward recursion method of dynamic programming. The initial program only considered two possible phases, 0 or 1. In RHODE-II(a) effort, the program was enhanced to accommodate eight possible phases (Appendix C), which correspond to the phases in TRAF-NETSIM (Figure 32).

The COP algorithm is designed to minimize the number of stops as well as the total delay. The algorithm does not consider the usual decision variables such as cycle time, splits, and offset. Instead, the decision problem is to efficiently assign "green times" to the phases of a signal.

The COP formulation was motivated by the OPAC strategy (Gartner, 1983). Essentially, the dynamic program allows the allocation of zero, minimum, maximum, or any green time within the minimum and maximum green times to a phase.

In the eight-phases version of COP two movements are possible simultaneously for each phase. Thus, the possible phases are designated as follows: 1-5, 1-6, 2-5, 2-6, 3-7, 3-8, 4-7, 4-8. In the current version, no other phases are allowed. However, the mathematical formulation allows for the consideration of any arbitrary number of phases.

Objectives

There were two objectives considered in the current COP program (Appendix C). First, time was allotted for phases while minimizing the total number of vehicle stops at the intersection. Secondly, it was preferable to "skip" a phase if the algorithm determined that it was optimal to do so. The phase to be skipped would be allotted zero time units by the algorithm.

The COP Program

COP has the following characteristics:

(1) uses discrete time units;
(2) has a defined time horizon (total time considered);
(3) incorporates a minimum green time for a phase to operate,
(4) incorporates a clearance time for vehicles (all phases red);
(5) is designed for eight phases;
(6) allows for up to 500 stages (the sequence of phases which can be modified);
(7) may use an intersection identification number;
(8) accepts arrival data for eight movements (this data corresponds to the vehicle requests for specific phases); and
(9) may easily be modified.

5.2.4 COP - Example

The intersection considered was Campbell Avenue and Sixth Street in Tucson, Arizona (Figure 33)

A partial output of the COP program is given below (a complete listing can be found in Appendix D), with a description following each section of output. The initial output information verifies the following input data: intersection identification number, number of phases, minimum green time, clearance time, total number of stages, sequence of stages, time horizon, and the arrival data. The solution, which follows the input data, requires the following definitions:

j	current stage number,
state	amount of time remaining,
decision	optimal amount of time to allot for the corresponding phase, and
phase	current stage.

Figure 38 displays an echo of the input parameters required for COP as well as the vehicle detections for each movement (1-8). Note that the sequence of phases is predetermined. Therefore, to create an arbitrary sequence of phases, some phases could be assigned zero time.

Table 3 shows the optimal solution obtained using COP. The "state" represents the amount of time remaining to be assigned. The "decision" is the amount of time to be assigned to the respective phase and "stops" is the total number of stops encountered as a result of the decision.

```
Intersection Information:
-------------------------------------------------------------------
Intersection Identification Number:  9375782
Number of Phases =  8
Minimum Green Time =  2
Clearance Time =  1
Total Number of Phases =  8
Sequence of Phases:37 48 26 38 15 38 37 16
Time Horizon =  40

Arrival Data
-------------------------
 1  2  3  4  5  6  7  8
-------------------------
 0  0  1  0  1  0  0  0
 0  0  1  0  0  0  1  0
 0  0  0  0  0  0  1  1
 0  0  1  0  0  0  0  0
 0  0  1  0  0  0  1  0
 0  0  0  1  0  0  0  1
 0  0  0  1  0  0  0  0
 0  0  0  2  0  0  1  0
 0  0  0  1  0  0  0  0
 0  0  0  1  0  0  0  0
 0  0  0  2  0  0  0  1

       .   .   .

 0  0  0  0  0  0  1  0
 3  0  0  0  0  3  0  0
 0  0  0  0  0  0  0  0
```

Figure 38. Input parameters and detector data for COP.

Table 3. Optimal amount of time to assign to each phase from COP.

```
Coordinated Optimization of Phases (COP) Solution
----------------------------------------------------------
j=1       State=40       Decision=4       Phase=37      Stops=2
j=2       State=35       Decision=9       Phase=48      Stops=3
j=3       State=25       Decision=0       Phase=26      Stops=0
j=4       State=25       Decision=13      Phase=38      Stops=9
j=5       State=11       Decision=5       Phase=15      Stops=2
j=6       State=5        Decision=0       Phase=38      Stops=0
j=7       State=5        Decision=3       Phase=37      Stops=1
j=8       State=1        Decision=0       Phase=16      Stops=0
                                                  Total Stops=17
```

Table 4. Computer time to execute COP for the parameters in Figure 38.

Time Horizon	Minimum Green Time	R/W user time (sec)	B/F user time (sec)
40	2	0.030	3.380

Table 4 shows the amount of computer time required to read input files and write to output files ("R/W user time") and the computer time required to perform the backward and forward recursions in COP ("B/F user time").

As can be seen in this example, the COP algorithm is able to skip phases. Here, phases 26 and 38 (the second occurrence of this phase) are assigned zero green time. In the last stage (j=8), the state (amount of time remaining to assign) is equal to one time unit. Since the minimum green time is two units and the clearance time is one unit, a minimum of three time units are required in order to assign a phase a time duration.

5. 3 Traffic Flow Prediction

The importance of traffic flow prediction can be understood by considering the signal timing problem for a single intersection. Consider the intersection shown in Figure 39. This intersection has four approaches. Associated with each approach are several possible traffic movements: left turn, right turn and a through movement. For the purpose of signal timing, the right turn and through movements are generally considered as a single movement. Any non-conflicting combination of movements that can share the intersection at any one time can be assigned a signal phase that allows those movements protected use of the intersection. The traffic demand for a phase is determined by the approach volume and the turning probabilities associated with vehicle routes. The traffic demand is typically measured using loop detectors on the approach to each intersection and in the left-turn pockets. The intersection signal timing problem is to decide what phases, in what sequence, for what durations should be used as the signal control. Typically, these decisions are made to minimize vehicle delay or stops. Current traffic signal timing methods assign phase durations to a predetermined phase sequence based on the hourly average traffic volume and estimated turning probabilities.

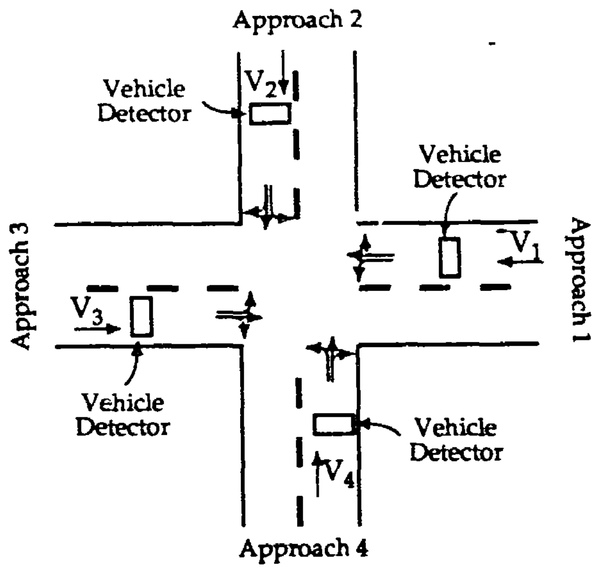


Figure 39. Basic traffic intersection showing approaches, approach volumes, movements and vehicle detectors.

The goal of real-time traffic-adaptive signal control is to choose the phase durations and phase sequences to provide optimal control for the actual traffic demand. For real-time traffic-adaptive signal control to be effective it must have an accurate view of the state of traffic conditions on the network and be able to predict, at least over short periods of time, how the current network conditions will evolve. The importance of prediction can be understood by considering the signal timing problem given two possible perfect predictions of arrivals during the planning horizon as depicted in Figure 40. Each arrival pattern represents the number of vehicles to arrive at an intersection in fixed time intervals[1]. Both arrival patterns are identical until time t_0 when the signal control has to decide whether to serve this approach or to serve another approach. In the top case, the demand occurs immediately following t_0, where as in the bottom case there is little demand immediately following t_0 and greater demand in the future. In each case the total number of vehicle arrivals are equal. However, the optimal control could be significantly different. It is of fundamental importance to know the temporal arrival distribution to build a truly real-time traffic-adaptive signal control logic.

[1]The use of "the number of vehicles" during fixed-time intervals is primarily to display the data. The arrival of vehicles can best be thought of as a point-process characterized by an instantaneous arrival rate with the additional characteristics of position, velocity and acceleration that represent the vehicle as a dynamic entity.

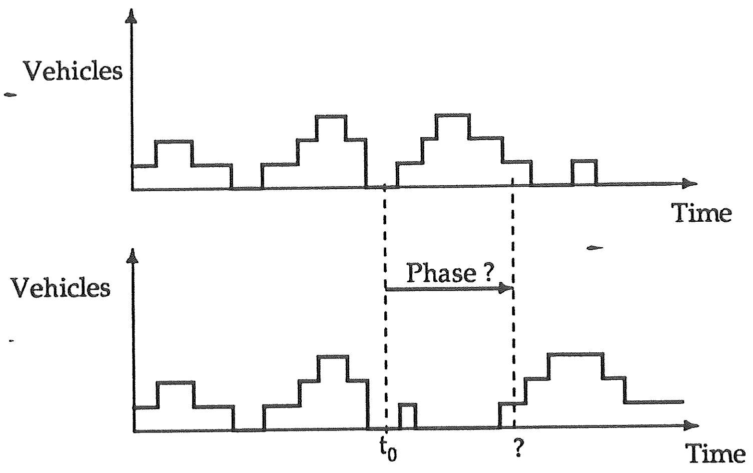


Figure 40. Graphical depiction of the effect of future arrivals on scheduling the intersection phase sequence and duration.

Three issues are important to predicting traffic flow: (1) the length of the time horizon, (2) the number of prediction points per time horizon, called the prediction frequency, and (3) the number and location of information sources. The prediction time horizon provides the real-time traffic-adaptive signal timing control logic with the ability to plan future signal timing decisions. If the prediction horizon is short, perhaps several seconds, then the signal timing decisions are restricted. For example, if the predictions are made over a 10-second horizon, the signal timing logic can only make timing decision which extend or shorten the current phase. On the other hand, if the predictions are made over a longer horizon, the signal timing decisions can include decisions on phase termination times and phase sequencing. For example, if the prediction horizon is 30-40 seconds, then the signal timing logic might schedule the next two phases and their durations based on the predicted demand instead of following a fixed phase sequence.

The prediction frequency provides information about the distribution of vehicle arrivals over time. If the predictions are made at a frequency of only one prediction for the decision time horizon, then the signal timing logic must assume that the vehicles are distributed uniformly over that time. If the predictions are made more frequently, say 10 to 30 times over the prediction horizon, then the signal timing logic will have a more accurate representation of the distribution of vehicle arrivals over time. Figure 41 depicts the information content of predictions at a frequency of once per time horizon and 10 times per horizon.

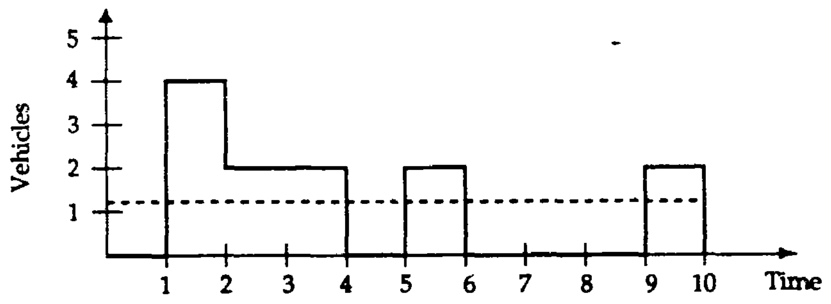


Figure 41. Illustration of the relationship between the prediction horizon (T=10) and the prediction frequency. The dashed line shows an average of 1.2 vehicles/time unit. The solid line shows the number of vehicles predicted per each time unit.

Traffic flow is, in general, a time-space phenomenon. The number and location of information sources determine the ability of any prediction algorithm to predict future conditions based on current conditions at other spatial points. For example, if a detector is located, say, 10 seconds upstream of the desired prediction point, then prediction will be easier but only for a 10-second horizon. The further away the location of other information sources, the longer the potential prediction horizon. But, the temporal information may become more distorted (e.g. platoon dispersion) and thus less valuable for prediction. In addition, the further away the information sources are located, the greater are the effects of exogenous factors, such as traffic signals and traffic sources/sinks, on prediction. Clearly, a system with many well placed detectors will provide the best information for prediction. However, the cost of such a detection system may be prohibitive.

In addition to the traditional inductance-based loop detector information discussed above, the application of advanced electronics to transportation through the national *Intelligent Vehicle Highway System* (IVHS) program has the potential to provide valuable information for traffic flow prediction and traffic control. Video image processing is already being used for vehicle detection. Vehicle identification technologies, hence information on origins-destinations, routes, speed, etc., are available and may be deployed in the near future. This application of advanced technologies is exciting, but from a real-time traffic control point of view, one must ask: "If we could apply these emerging technologies in any way possible, what is the best set of information that we should measure in order to provide the desired real-time control performance?" This question can only be answered through basic research on data fusion methods and study of information needed for effective real-time traffic-adaptive signal control.

The need for prediction was recognized in the development of the Urban Traffic Control System (UTCS) in the early 1970's. The development of second generation (UTCS-2) and third generation (UTCS-3) control logic included prediction as a primary system component. UTCS-2 based its signal timing decisions on predictions of demand for the next 5-15 minutes. UTCS-3 based its signal timing on predictions of demand over much shorter time periods of approximately a cycle length[2]. It is generally felt that these failures of UTCS-2 and UTCS-3 were due to errors in surveillance and detection [Tarnoff, 1975]. Very little research has been conducted on the development of prediction methods since the 1970's.

Stephanedes, Michalopoulos and Plum [1981] conducted a critical review of the UTCS predictors and three additional predictors. They compared the prediction accuracy of UTCS-2, UTCS-3, historical averages, current measurement, and a simpler proposed algorithm. Each predictor was compared based on mean squared error and mean absolute error for both five-minute predictions and cycle-by-cycle (90 second) predictions. They concluded that for five-minute predictions, the historical average performed better than UTCS-2, but both of these methods were superior to the others. For cycle-by-cycle comparisons, the UTCS-2 and the historical average predictors were not applicable since synchronization of cycles over historical periods was impossible. A moving average version of their proposed algorithm was superior to the UTCS-3 and current measurements. Some versions of their proposed algorithm performed better than the moving average version but the performance was sensitive to the selection of model parameters.

Each algorithm that Stephanedes, Michalopoulos and Plum studied addresses the prediction problem based on a fixed time horizon, either five-minutes or one cycle, and updates the prediction at a frequency of only once per horizon. Table 5 summarizes each of these algorithms in terms of its characteristics: prediction horizon, prediction frequency and performance. Each of these algorithms uses only a single information source (a single detector) located at the point of prediction.

[2]Generally, each signal within a coordinated system is operated on a common cycle time. The cycle time is defined to be the amount of time from the beginning of main street green until main street green begins again, where main street is arbitrary, but generally chosen to be the major traffic movement street.

Table 5. Comparison of existing traffic demand prediction algorithms.

Algorithm	Horizon	Frequency	Performance
UTCS-2	5-15 min	1	Good for 5 min
UTCS-3	5-15 min, cycle	1	Poor Overall
Historical Average	5-15 min	1	Best for 5 min
Current Measurement	5-15 min, cycle	1	Poor due to time delay
Proposed	5-15 min, cycle	1	Sensitive to Parameters

Okatani and Stephanedes (1984) utilized a Kalman filter model structure to consider information from multiple sources, i.e. detectors on a number of links. They made predictions at a frequency of once per 15-minute time horizon. Their results indicate a small improvement over the UTCS-2 prediction algorithm, but fail to address the need for higher frequency predictions required for real-time traffic-adaptive signal control logic.

In a discussion of the prediction problem, Gartner [1981] concluded that the deficiency to provide good temporally distributed predictions could only be addressed by relying on actual flows rather than average volumes. One possible method to obtain actual flows approaching an intersection would be to place detectors on the links upstream from the intersection and use the flows at these points to provide predictions. An important limitation of this approach is that the distance between the intersection and the upstream detector constrains the prediction time horizon. Another limitation is that if the detector fails, there is no other source of real-time traffic flow information.

Baras, Levine and Lin [1979] utilized point-process filter and prediction methods developed by Segal [1976] to estimate queue size at an approach to an intersection. In their research they modeled the formulation and dispersion of queues as discrete-time time-varying Markov chains that are related to the observable point processes at the detectors on the approaches to an intersection. This approach showed good performance for estimating/predicting queue size. However, it provided predictions with a time horizon of only a few seconds using a detector that was located only 20 vehicle lengths upstream of the intersection stop bar and did not directly estimate the traffic flow arrival process. The benefit of their approach is that they have incorporated a probabilistic structure into the estimation and prediction problem. Through this probabilistic structure additional information, such as the type that will be available through deployment of IVHS, can be fused into the development of traffic flow predictions.

5.3.1 The PREDICT Algorithm

We have been investigating a new prediction method, PREDICT, that is based on the use of detectors on the approach of each upstream intersection, together with the traffic state and control plan for the upstream signals to predict the future arrival. This approach allows a longer prediction time horizon since the travel distance to the intersection is longer and the delays at the upstream signal are considered. A benefit of this approach is that it couples the effects of the upstream traffic signals with the intersection control optimization problem.

This prediction approach is data driven. That is, the prediction of each arrival at an intersection depends on the event of a vehicle crossing some detector on the approach to an upstream detector and not (directly) on the traditional time averaged detection parameters of count and occupancy.

To understand how this approach works consider the scenario shown in Figure 42. It is desired to predict the flow approaching intersection A at detector d_A. Making the prediction for the point d_A is important because it is a point on link AB where the actual flow can be measured, hence the quality of the prediction can be assessed in real-time.

Traffic contributing to the flow at d_A originates from the approaches to intersection B and can be measured at detectors d_l, d_t and d_r representing the flows that will turn left, pass through and turn right, respectively, onto link AB. Other traffic that originates at sources between intersections A and B are possible, but will be considered as unmeasurable "noise". Also, it is possible that vehicles passing over d_l, d_t, and d_r will terminate their trip before arriving at d_A. This will also be considered as "noise" in the prediction.

When a vehicle passes a detection point, say d_i where $i \in \{l,t,r\}$, several factors effect when it will arrive at d_A including (1) the travel time from d_i to the stop bar at intersection B, (2) the delay due to an existing queue at B, (3) the delay due to the traffic signal at B, and (4) the travel time between B and d_A.

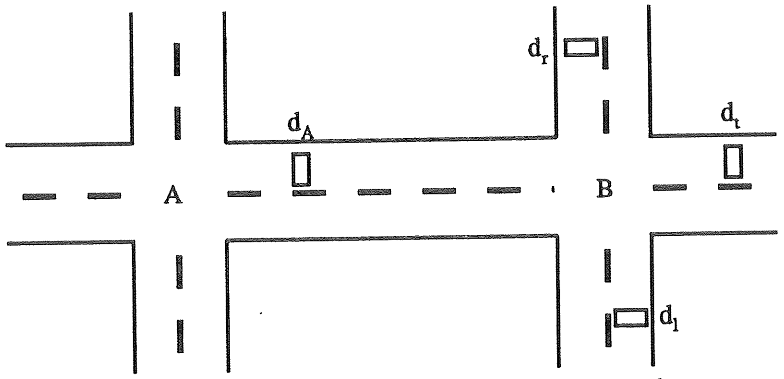


Figure 42. Prediction scenario based on detectors on the approaches to the upstream intersection (B).

Figure 43 (a)-(d) depict the delay associated with each of these factors. In Figure 43-(a) the vehicle arrives at detector d_i and passes freely to detector d_A. The arrival time, denoted t_a at d_A can be estimated as

$$t_a = t_{d_i} + T_{d_i,s_B} + T_{s_B,d_A}$$

where T_{d_i,s_B} is the travel time from d_i to the stop bar at intersection B and T_{s_B,d_A} is the travel time from the stop bar at intersection B to the detector at d_A. Each of these travel times can be estimated based on the approach speed and the link flow speed, respectively.

In Figure 43-(b) the vehicle arrives at detector d_i and is delayed by the signal at intersection B. Hence the travel time from d_i to d_A must account for the travel time from d_i to the stop bar, the delay due to the signal and the travel time from the stop bar to d_A. The arrival time at d_A can then be estimated as

$$t_a = t_{d_i} + T_{d_i,s_B} + T_{u_B} + T_{s_B,d_A}$$

where T_{u_B} is the delay until the signal timing plan advances to a phase that will serve the desired movement.

In Figure 43-(c) the arrival at d_i encounters delay for the signal as well as a standing queue, and has to travel from d_i to the stop bar at B, and from the stop bar to d_A. The signal delay can be computed based on the signal timing plan as described above. The delay due to the standing queue can be estimated using a relationship of the form

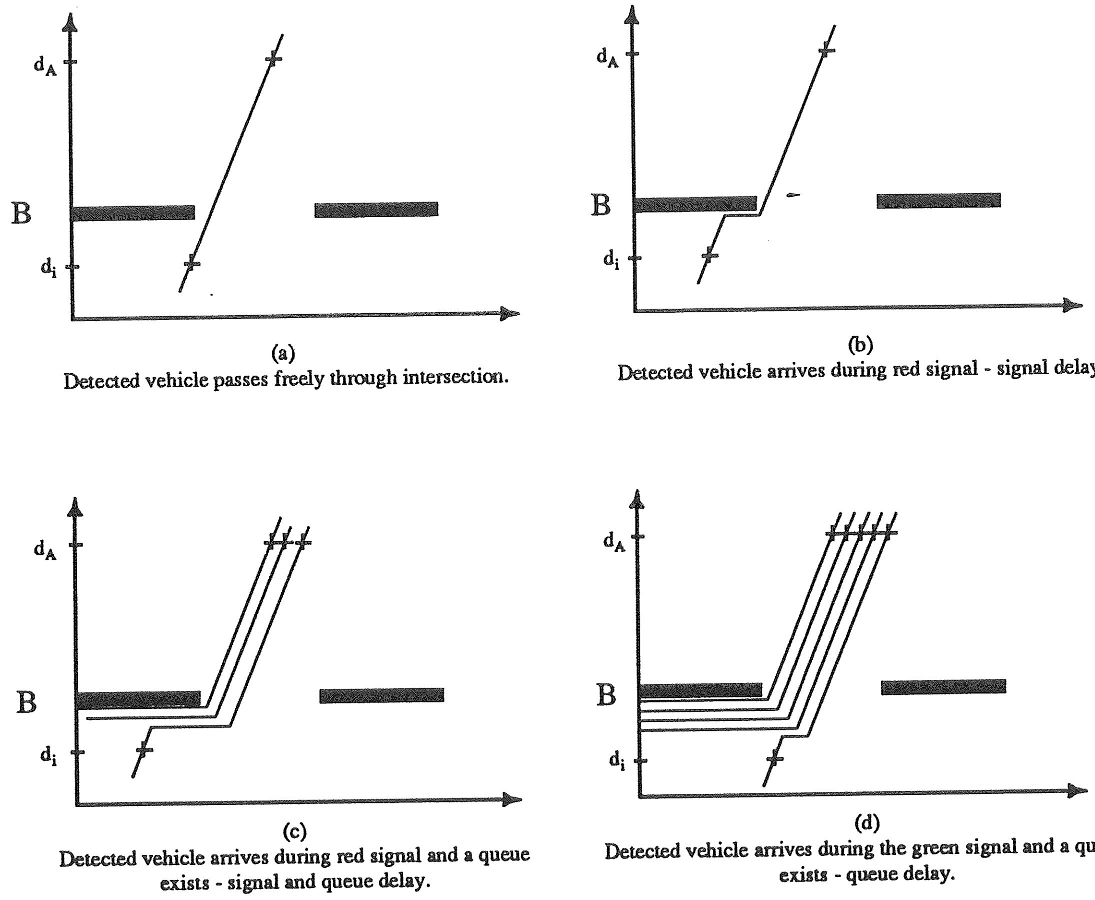

d_A
B
d_i
(a)
Detected vehicle passes freely through intersection.
(b)
Detected vehicle arrives during red signal - signal delay.
(c)
Detected vehicle arrives during red signal and a queue exists - signal and queue delay.
(d)
Detected vehicle arrives during the green signal and a queue exists - queue delay.

Figure 43. Delays associated with the prediction of arrivals at the detector d_A.

$$T_{u_B} = a_o + a_1 N_{q_i}$$

where a_0 and a_1 are parameters that can be selected based on the particular intersection and N_q is the number of vehicles in the queue. (The above equation has the form of Greensheild's equation and has been used to estimate the amount of time required to clear a queue.) Note that the T_{u_B} relationship is not motivated by Little's theorem, but is based on the dynamics of traffic queue dispersion and has been determined empirically.

Figure 43-(d) depicts the case when the arrival at d_i occurs after the signal has begun serving the desired phase, but a standing queue is present. This case is similar to the above, except that the delay due to the standing queue must be adjusted based on the amount of time that has elapsed between the onset of the signal and the arrival of the vehicle at d_i and the travel time to the back of the queue.

Once the arrival time at d_A is predicted, this model adds the prob:bility of the arrival to the current estimate of the expected number of arrivals at that time. For example, if 15% of the vehicles that pass over d_t continue on to d_A, then 0.15 will be added to the current estimate of the expected number of arrivals at the predicted arrival time t_a.

It is important to note that this predictiun model is based on processing arrival data as it evolves. Hence, at any point in time the predicted arrival flow pattern at d_A accounts for vehicles that have already passed the detectors d_r, d_t and d_l. The benefit of this evolutionary behavior of the predictor is that it constantly provides, for a given prediction horizon, partial information that can be used by the intersection control logic. Also, this model is distributed in that it can be applied for every approach of every intersection in a large urban traffic signal control network. In fact, this distributed architecture will reduce the communications required to transmit the detector information to a central computer and hence should improve the prediction speed.

We have conducted several preliminary experiments using this traffic prediction method. Figure 44 shows the actual (dotted line) and the predicted (solid line) arrival patterns at d_A for a sample network that was simulated using the FHWA's microscopic traffic simulation model TRAF-NETSIM. Although the predictions are not exact, there is clearly a strong correlation between the actual and the predicted arrival patterns.

To test the quality of the predictions the performance of a dynamic programming based real-time signal control logic [Sen, 1991] using these predictions was compared to the performance of a well-timed semi-actuated signal control logic. The performance criterion used in the real-time signal control logic was the minimization of the total number of stops. Fifty-five runs of the simulation model were made for each control strategy using paired random number sequences, varying the load on the intersection. The results of these simulations are shown in Figure 45. The results show a significant decrease in the percentage of vehicles that were stopped using the real-time control logic and the predictions generated as described above.

This preliminary results should be interpreted with both optimism and caution. Optimistically, it appears that the predicted flows and the actual flows are very similar and that this prediction method will provide the temporal distribution of arrivals at a prediction frequency that is greater than any of the currently available methods.

Cautiously, one must be aware that the simulation environment where these preliminary results were obtained is nearly ideal. That is, this study did not consider the effects of extraneous factors such as sources and sinks on the approach links as well as other traffic characteristics that may not be accurately modeled in the simulation environment.

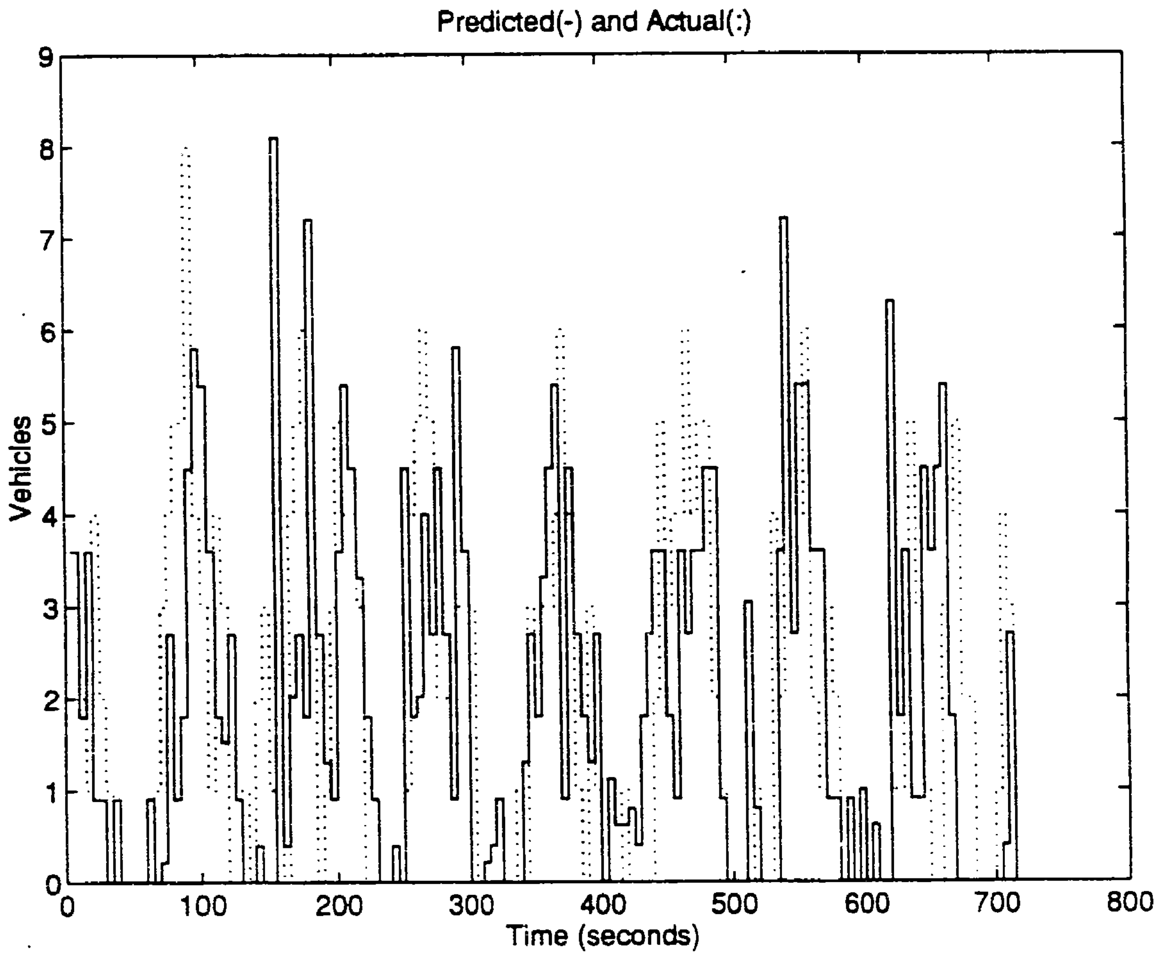


Figure 44. Link Flow Profiles: predicted (solid) and actual (dotted).

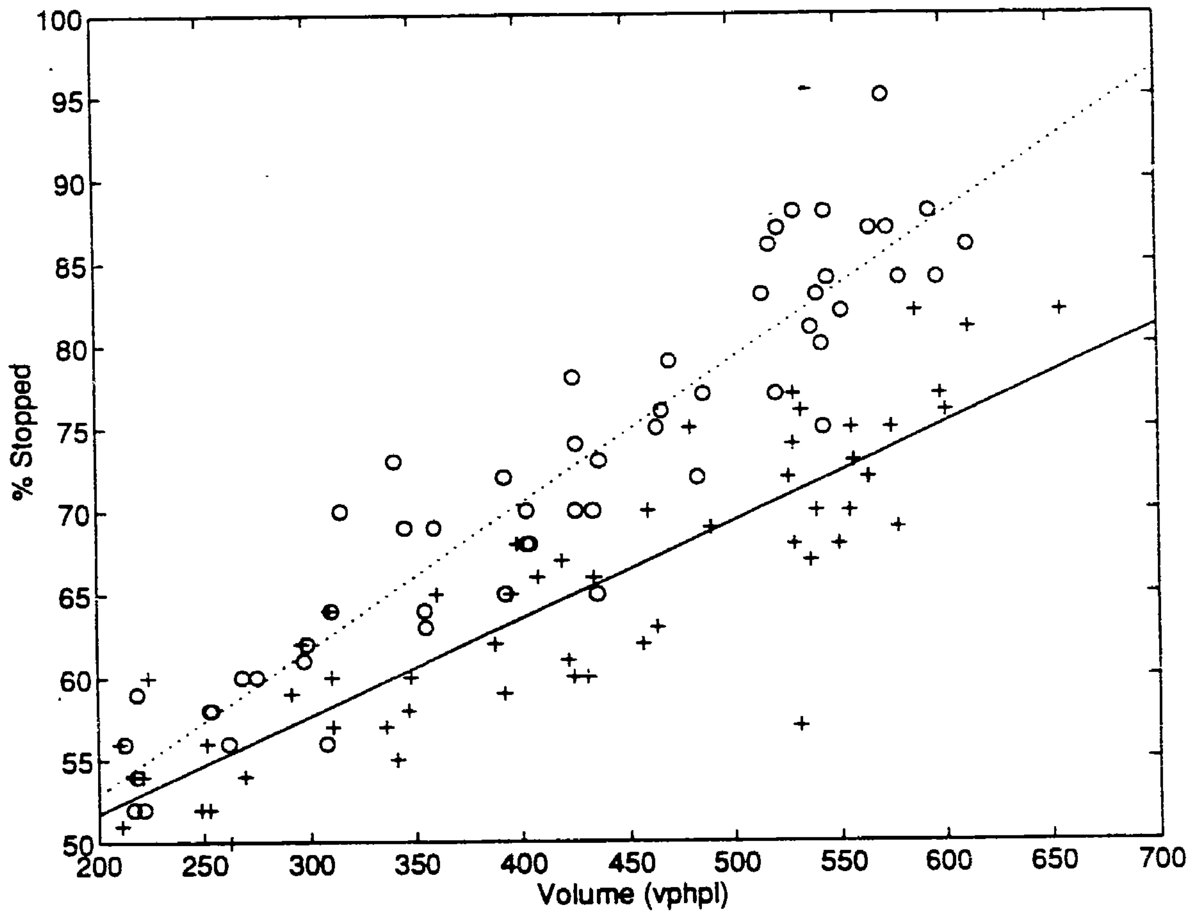


Figure 45. Comparison between well-timed semi-actuated control (represented by the 'o' and the dotted line, and a version of the proposed intersection control logic and predictions as described above.

6. SIGNAL CONTROLLER INTERFACE

In this section the traffic signal actuation level of the RHODES hierarchy is demonstrated. The purpose of this task was to demonstrate that existing signal control hardware could be used within the real-time traffic-adaptive signal control environment. This requirement is important from an economic point of view since cost to replace all existing controllers may be prohibitive. In this project we built an interface between a PC and an Eagle Signal Controller (DP9000 Series) such as those currently used by the City of Tucson.

6.1 Interface Requirements

The physical interface requires the ability to select any phase, from a prespecified set of phases, at any point in time, with the constraints that the normal yellow and all red intervals be obeyed. Although, this interface provides a very flexible control mechanism, it should be noted that safety considerations are all ready incorporated into the signal controller and parameters such as clearance intervals and minimum green interval times can also be used to maintain a safe mode of operation.

An Eagle DP9000 Series NEMA controller was reprogrammed to a fully actuated eight-phase configuration with the minimum green intervals set as small as possible. Phases were then selected by placing a CALL on the detector associated with the desired phase. For example, if the controller was resting in main street through (phase 26) and a lagging left phase was desired (Phase 15), then a CALL was placed on the detector inputs associated with phase 15. Similarly, if the controller was in phase 26 and a leading left turn for the side street was desired, a CALL was placed on phase 37 (side street left turn).

6. 2 Interface Logic Design

The interface was prototyped and demonstrated in the laboratory using an Eagle controller, a PC, a Motorola HC-11 microcontroller and a simple digital logic interface circuit. Figure 46 depicts the physical implementation. A simple communication software package (PROCOMM) is run on the PC that allows communication with the microcontroller. For the purpose of the demonstration there is no need for a special application on the PC; however, when the RHODES algorithms are implemented on the

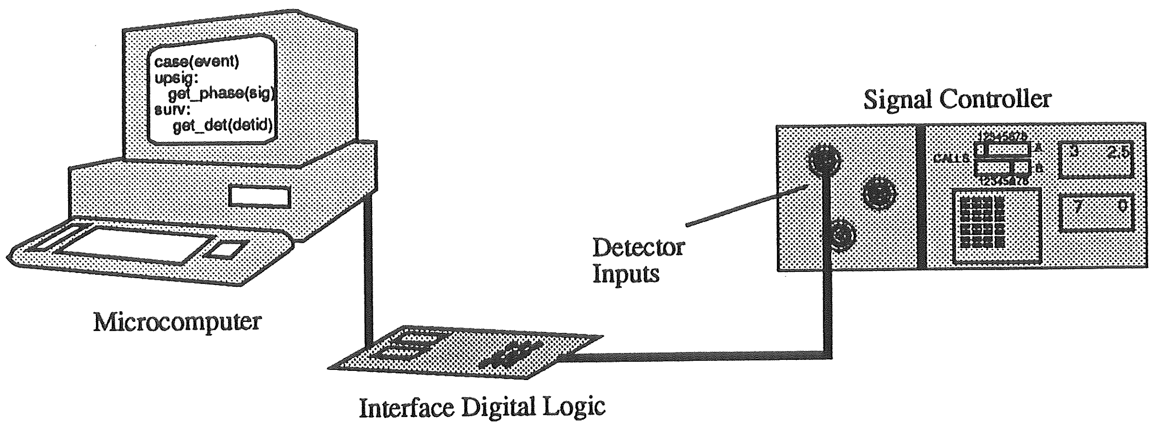


Figure 46. Physical Interface for Real-time Traffic-adaptive Signal Control.

PC the communication should be handled using software routines that address the serial communication port.

The Motorola HC-11 microcontroller (a $65 unit) allows the PC to transmit a single byte of data that represents the desired phase CALLS. This byte of data is stored in a memory location on the microcontroller that corresponds to the desired output ports where the microcontroller is interfaced to the Eagle controller. The format of the byte is shown in Figure 47. Each bit of the byte represents a CALL for one signal phase. Table 6 shows the bytes for the eight-phase dual ring controller (see Figure 32 in Section 4.2). The byte is transmitted by typing the hexadecimal equivalent on the PC.

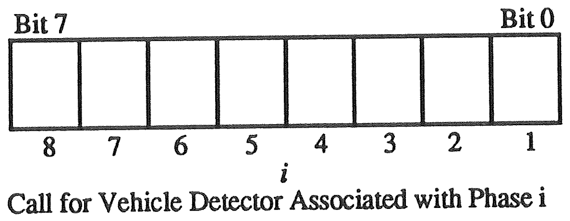


Figure 47. Format of Interface Controller Byte.

Table 6. Phase and associated byte transmitted from PC to the HC-11 microcontroller. The Hex equivalent is also shown.

PHASE	BYTE (Hex)
12	0000 0011 (03)
15	0001 0001 (11)
26	0010 0010 (22)
34	0000 1100 (0C)
37	0100 0100 (44)
48	1000 1000 (88)
56	0011 0000 (30)
78	1100 0000 (C0)

Figure 48 shows the circuit for the interface between the HC-11 microcontroller and the detector port on the Eagle controller. Eight of the microcontroller output ports are connected to an interface circuit that switches the +24 Volt to ground when a CALL is desired on a particular phase using an open collector NAND gate (SN7406). The +24 Volt power is supplied by the Eagle controller. All unconnected output ports of the microcontroller are tied to ground to ensure stability of the circuit.

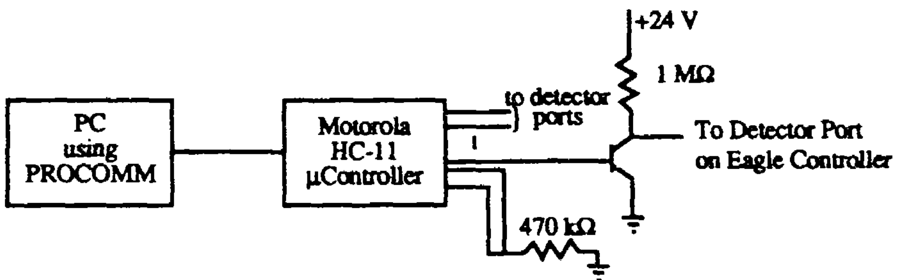

+24 V
1 MΩ
to detector ports
PC using PROCOMM
Motorola HC-11 μController
To Detector Port on Eagle Controller
470 kΩ

Figure 48. Computer-Controller Interface Circuit.

6. 3 Demonstration

The signal controller interface was demonstrated by selecting any desired phase, from the set of allowable phases, entering the associated byte on the PC and waiting until the controller transitioned through the appropriate clearance intervals and then into the desired phase. It should be noted that the Eagle controller required a start-up "cycle" where it transitioned through all phases in order to clear all CALLS that appeared when the controller was initialized. After this start-up "cycle" the controller functioned as required - - the desired phase being activated externally of the controller.

7. SIMULATION EXPERIMENTS USING TRAF-NETSIM

In this section we discuss simulation models used for studying and evaluating the effectiveness of traffic control algorithms. After a brief discussion of the issues related to simulation modeling and evaluation, we present our approach to develop a simulation model for testing real-time traffic-adaptive traffic control, which is based on the modification of the TRAF-NETSIM model. The modified model was validated by implementing external fixed-time and external semi-actuated signal control logic and comparing the performance of external control logic with the corresponding logic that is internal to TRAF-NETSIM. Having validated the simulation model, the real-time traffic-adaptive intersection control algorithm, COP, as described in Section 5.2, was interfaced to the simulation model and evaluated.

The functional requirements for simulation models for development, testing and evaluation of real-time traffic-adaptive signal control logic include:

- the ability to obtain surveillance/detector output at required frequencies;
- the ability to control traffic signals in real-time;
- the ability to generate dynamic traffic conditions, including recurrent and non-recurrent congestion such as incidents and special events;
- the ability to obtain different different types of surveillance and detector data including traditional loop detetor data, vehicle probe data (travel times, routes, etc.), and other traffic control information that will be available from the emerging IVHS technologies;
- the ability to compute various measures of effectiveness; and
- the ability to generate traffic characteristics that are not necessarily observable, such as queue lengths.

The frequency of surveillance and detector system output and the frequency of the signal control input will dictate the minimal resolution, and hence the responsiveness, of the signal control logic. The simulation model must be able to represent rates that will be achievable when the control logic is implemented for field testing.

The ability to represent dynamic recurrent and non-recurrent congestion, as well as other non-congested traffic conditions, is needed for measuring the control logic's capability to respond to real-time traffic conditions.

Simulation models used for testing must provide the same surveillance and detection information as that available in the field. When the real-time traffic-adaptive signal control logic is deployed in the future, vehicle probe data and other information may be available from emerging IVHS technologies. A valid simulation model must provide this information in order to test real-time control logic that can use such information.

It may be desirable for the signal control algorithms to optimize different measures of effectiveness (MOE), based on traffic conditions or dictated by the operating jurisdictions. Therefore, it is essential that the simulation model provide a wide variety of MOEs to evaluate the real-time traffic adaptive signal control algorithms.

The simulation model requirements from a development and testing perspective differ from the requirements for performance evaluation. Clearly, the most important requirement of a simulation model is that it accurately represent the dynamics of traffic flow and its response to dynamic signal control. This requirement dictates that the simulation model chosen for development and testing not be based on a macroscopic flow model that assumes constant cycle length and deterministic traffic flow characteristics. Rather, the model should include microscopic flow characteristics such as car-following vehicle response to actual traffic signals.

During the development and testing phase it is essential to have access to both traffic and signal control variables so that detailed behavior can be studied. One may distinguish between traffic simulation information that is needed for validation and testing and that which is available as traffic surveillance/detection data for the signal control algorithms. For example, for the purpose of testing a traffic model used in an optimization routine, it may be desirable to compare the traffic model's state-of-the-traffic parameters, such as queue length, to the corresponding parameters in the simulation model. This form of testing requires that the traffic simulation model provide accurate measurements of queue lengths despite the fact the existing traffic surveillance technology may not provide this information.

Another important consideration is the frequency at which required testing data is available. For example, the average queue length for a simulation period is insufficient for testing a routine that estimates real-time queue lengths. This information must be available as frequently as possible.

For the past two-years we have been using the TRAF-NETSIM traffic simulation model for developemnt and testing of traffic-adaptive signal control algorithms. This effort has required us to modify the TRAF-NETSIM model to (1) interface the simulated traffic surveillance equipment with the signal control logic's database, and (2) interface the simulation model's traffic controllers with the signal control logic that we have developed. In addition, we have made information accessible that is internal to the simulation model's database; this allows us to test and validate our algorithms and software.

For example, in testing the intersection control algorithms described in Section 5.2, we modified the simulation model to record the simulated signal states and we compared them with the signal states downloaded by our algorithms. We were thus able to ensure that the desired signal state was the signal state activated by the simulation model. We also modified the TRAF-NETSIM surveillance logic to allow us to record simulated traffic flow profiles on each link and comapre them with the predicted traffic flow profiles that were generated by the traffic flow prediction models described in Section 5.3.

The network that we simulated in our experiments consisted of 41 intersections within a four square-mile area in the City of Tucson (see Figure 49). The traffic characteristics, volumes and signal configurations were obtained with the assistance of the City's traffic engineers. Thus our experiments were based on realistic data.

7.1 Overall Approach

The objectives in the development of the interface was to minimize the modifications to the existing TRAF-NETSIM code and to simulate the interface between the real-time traffic-adaptive signal control logic and the NEMA controllers that are currently used in Tucson, Arizona. The same philosophy as in the design of the physical signal controller interface was used in the modification of TRAF-NETSIM. The actuated signal controller logic (software Q5 logic) was programmed, through the input data base, to have the desired set of phases with the desired minimum green intervals. Then detector information, contained in the TRAF-NETSIM internal data base, is first read, for the purposes of surveillance, and then either cleared or set to represent a CALL on the desired phase. When the signal state is updated, the CALL will be processed thereby

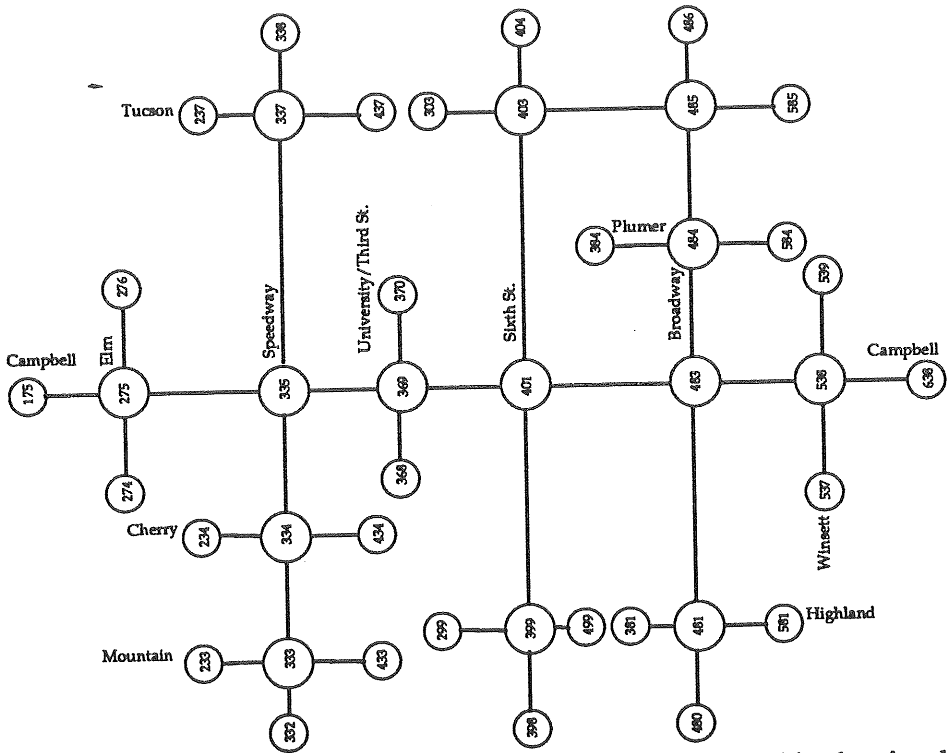


Figure 49. Topological layout of the traffic network used in the simulation studies.

forcing the signal into the desired state. Figure 50 depicts the software implementation of this approach.

This approach for interfacing of external signal control logic has been extended to support network-wide control. To accomplish this, an event-based control logic interface was developed. In the interface logic, the external event manager can implement a wide variety of network-wide control strategies by scheduling control logic processing events and signal update events as required. The external event manager is called each time a signal state requires updating. It checks the internal simulation clock to determine which events require processing. If it is the first call, for a particular time, to the external logic and an area-wide surveillance system update is required, then the external logic that performs the update is called. If the signal that initiated the call to the external event manager requires updating, the appropriate signal control logic is called. If it is desired to allow the internal simulation signal control logic to control the signal, then control is passed back to the

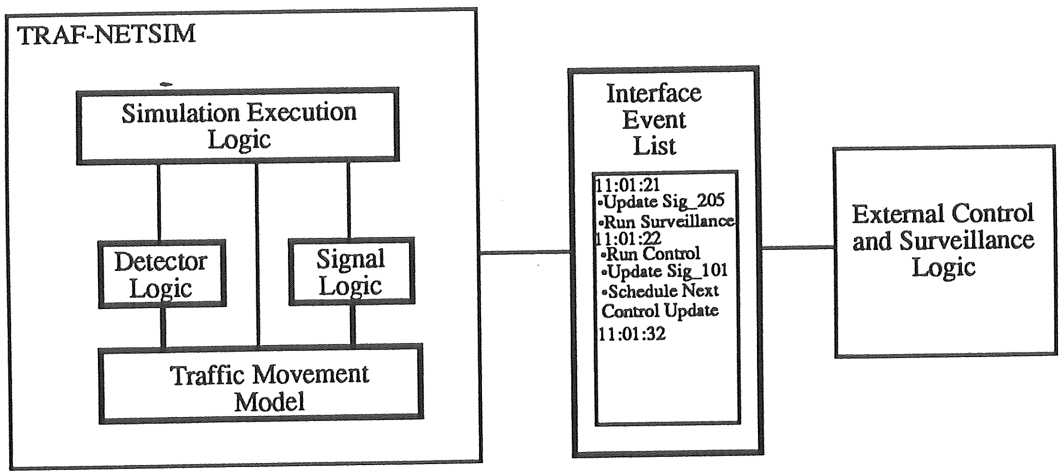


Figure 50. Software Interface for External Control and Surveillance Logic

simulation with no update taking place. This approach requires any signal that is to be controlled externally to be programmed according to the guideline discussed above as a fully actuated signal.

The interface implementation included the development of a library of interface functions that the external logic, either surveillance or control, can use to access database information, alter database information, place events on the event managers list, or remove events from the event manager list. This library facilitates the interface between the FORTRAN simulation model and the external logic that has been developed in the C programming language.

Validation testing has been conducted to ensure that this approach is both reliable and accurate. To test the interface, the actual signal states and the desired signal states are compared. These comparisons have been made on both single intersection control and on network-wide control. The interface has been found to be reliable and accurate. The only consideration that must be made is that it requires one full simulation second for the Q5 signal controller emulation logic to respond to a CALL, where as the hardware controller is capable of responding in much less time due to the underlying microprocessor clock speed.

An essential element in the development of external signal control logic, especially real-time control, is the traffic surveillance system. In our effort, we have utilized the internal surveillance detector logic of TRAF-NETSIM for the placement and processing of detector events, but we utilize an external detector signal processing logic. This approach has allowed us to estimate any traffic parameters that are desired in addition to the standard count and occupancy values. The surveillance detector logic (SUBROUTINE DETECT) is used to generate signals form each detector on a tenth-of-a-second basis. In particular, the surveillance detector logic determines the beginning and ending of detection events on tenth-of-a-second basis. These events are the externally translated into continuous binary signals. Each continuous signal represents either an occupied detector or an unoccupied detector. These signals are then processed into the proper parameter estimates for the associated signal control algorithms. The internal surveillance logic also provides other surveillance information, such as queue lengths, when appropriately placed detector have been defined.

In addition to the traditional detector surveillance data, advanced surveillance information, such as that that will be available with IVHS, can easily be obtained from the simulation model database. For example, when a vehicle triggers a detection, we can determine the vehicle identification number (VIN), its speed, and its direction of travel at the next intersection. All of this information is currently accessible using the external interface logic that has been coded and implemented in the C programming language.

It is important to note that the model enhancements made by us were intended to support our on-going research activities in real-time traffic-adaptive signal control. The primary mission in these activities was not to develop a general platform for testing IVHS components. The objective was to develop a platform for testing real-time traffic-adaptive signal control logic. We feel that we have succeeded in satisfying this objective, at least for the control features that are currently under development. Our approach is only one possible approach to the development of a general IVHS simulation model interface. It is clearly possible to optimize the computational effort required through direct modifications of the simulation model itself, or through development of a new model.

7.2 Implementation Details

This section describes, in detail, the interface which was developed and methods for linking external signal logic to TRAF-NETSIM. TRAF-NETSIM checks the states of the signals at each time step and modifies them as needed. This is implemented in a loop which updates the signal at each intersection, one intersection at a time. The routine UPSIG handles the updating of the traffic signals. If an intersection contains an actuated controller, logic is called which emulates a hybrid between a Type 170/179 and a NEMA signal controller. Within this TRAF-NETSIM controller logic, there is a routine DETQ5 which translates detector data to the format which is used by the signal controller. It is immediately before that translation that a call was inserted to a new routine, CONTROL (see Appendix E for a listing of this routine). This allows manipulation of the detector data which is sent to the internal TRAF-NETSIM signal control logic. The subroutine CONTROL serves as the primary interface between TRAF-NETSIM and any external signal logic, as shown in Figure 51.

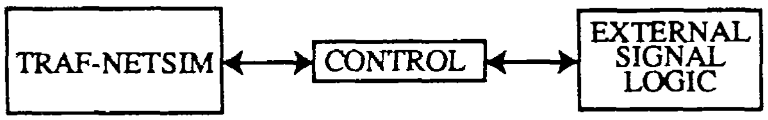


Figure 51. External Signal Control Logic Interface

The subroutine CONTROL calls the external signal logic with any necessary parameters such as detector data and a node identification number. The external signal logic returns a code to indicate the desired active phase or phases at that intersection. Based on this response from the external signal logic, CONTROL sets the TRAF-NETSIM detector data to indicate demand only on the indicated phases at the current intersection. At that point the TRAF-NETSIM simulation continues as originally designed until another intersection with an actuated signal controller is updated. The external signal logic is easily bypassed for intersections that should be controlled by the TRAF-NETSIM internal signal logic.

7.2.1 Providing Data for External Signal Control Logic

The data that TRAF-NETSIM needs to provide to the external signal logic will vary, depending on the purpose and implementation of the external signal control logic. For example, a simple fixed-time controller only needs two parameters: the network-wide synchronization time and the current node number. Other types of signal control logic will need detector data and possibly additional data. Most of the data which might be needed is generated internally by TRAF-NETSIM for use by the simulation model, and is stored in FORTRAN common blocks. That information then becomes available to any external signal logic routines through access to those common blocks. The external signal logic does not need to be coded in FORTRAN to be able to access these common blocks. In this research, all external signal logic was coded in C, which can also directly access the common blocks within the TRAF-NETSIM model (see Appendix F for further information about using FORTRAN and C together). The common blocks can be accessed by the routine CONTROL, passing the necessary information to the signal logic as parameters, or the signal logic can access the common blocks directly. It is not always the case, however, that all needed data will be stored in common blocks. One important case is the node number. Because the signals are updated one at a time (i.e. the signal logic will be called once for every actuated signal in the network) TRAF-NETSIM needs to communicate the current node number to the external logic. This is accomplished by passing the node number as a parameter from TRAF-NETSIM to the external signal control logic. The node number passed to CONTROL from DETQ5 is the node number assigned to the current intersection in the TRAF-NETSIM input file.

Detector data has been made available in common block SIN368 in the array ICOUNTS(). This data is collected from surveillance detectors (defined by card type 42) which are placed on the network. It is collected through extensions which were made to subroutine SENSOR in the TRAF-NETSIM code. This is the routine which registers all detector actuations. The data that is collected and stored in ICOUNTS is a list of counts of the number of actuations registered at each surveillance detector during the current time step. Actuations are counted as they are registered at each surveillance detector, and only if the detector was assigned a non-zero station number in the input data. Each vehicle is counted only once by any detector, at the time when the vehicle first reaches the detector. The total number of actuations registered at each detector are accumulated over the duration of the time step (one second of simulated time). After the signals have been updated, the counts are then cleared and the process is repeated during the next time step. The clearing is done in subroutine OUTDATA, a new subroutine which was created to output data and clear the detector counts (The modified version of SENSOR, as well as a listing of OUTDATA can be found in Appendix G). Through the use of surveillance detectors and the count information, it is therefore possible to determine how many vehicles arrived at any particular location on the network during the current time step.

The common block SIN368 was created to contain the detector counts in the array ICOUNTS. Each position in the array holds the count from the corresponding surveillance detector. For example, ICOUNTS[3] would contain the number of actuations that were registered at the detector with station number 3 during the current time step. The count information is collected during the "Move vehicles" phase shown in Figure 52 as the vehicle positions on the network are updated. This information is therefore available for use at the time that the signal control logic is executed during the "Update signals" phase. The counts are cleared at the end of each time step and are collected again during the next. One limitation of the current implementation is that presence information is not available from presence detectors. Either type of detector can be placed on the network, but currently both provide the same type of count, or passage, information.

For signal synchronization, the network-wide sync clock can be accessed directly from the TRAF-NETSIM simulated controller hardware. It is stored in common block SIN355 in the array LOWRAM() at location LOWRAM(38). This value may need to be incremented before being passed to the external control logic, depending upon how the sync clock is being interpreted within the external signal logic. This is because the signal states are updated before the system clock is incremented (see Figure 52). The decision whether to pass an incremented copy of the sync clock or the sync clock itself determines whether signal transitions, which are based on the sync clock, will take effect at the beginning or the end of the time step in which they occur. For example, if a phase is supposed to terminate when the sync clock reaches time 60, the transition could occur at the end of time step 59, in which case it would take effect during time step 60, or the transition could occur at the end of time step 60, in which case it would not take effect until time step 61.

During the implementation of these modifications to TRAF-NETSIM, a general goal was set to minimize changes made to the original TRAF-NETSIM code. This approach should also be heeded during the development of external signal control logic. The

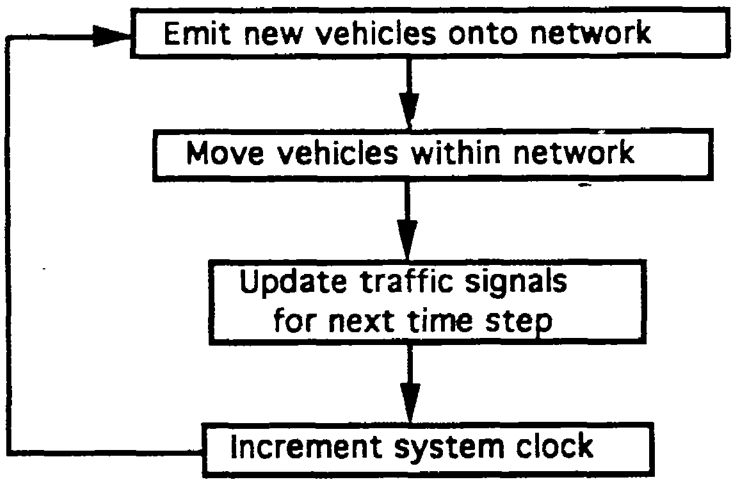

Emit new vehicles onto network
Move vehicles within network
Update traffic signals for next time step
Increment system clock

Figure 52. TRAF-NETSIM simulation logic flow.

benefits from this will be increased modularity and testability of the new signal control logic as well as maintaining the integrity of the TRAF-NETSIM simulator. Hence, the simulator should only be expected to provide basic data describing the current state of the system. Any new data which needs to be calculated or maintained should be handled in separate modules. For example, if historical data is needed, it should be maintained by the external logic and not by modifications to the TRAF-NETSIM structure. Due to the detailed implementation of the TRAF-NETSIM traffic model, most parameters that might be needed are probably available somewhere within the model. In those cases, the task is reduced to locating and properly interpreting the internal variables which contain the desired data.

7.2.2 Implementing Signal Control from External Signal Control Logic

Once the external signal control logic has been called with the proper data, it should make a decision about the desired phase, or phases, at the given intersection at the current time. It should then return that decision to CONTROL. This is encoded as a single byte return value from the signal logic. If a value of zero is returned, CONTROL will not modify the detector data for that intersection. This effectively bypasses the external signal logic and allows that intersection to be controlled by the internal TRAF-NETSIM signal logic. If a non-zero value is returned, the bits which are set to 1 will be interpreted as the desired phases. For example, if a value of 4 is returned, this is 00000100 in binary. Since the third bit is set to 1, phase 3 is indicated as the desired active phase. Multiple bits set to 1 would indicate multiple active phases. For example a return value of 34 is represented in binary as 00100010, indicating that phases 2 and 6 should be active. After receiving and interpreting the return code, CONTROL will then clear all actuated controller detector data for the current intersection (the real data) and will instead place actuations on any detectors associated with the phase or phases indicated by the external signal control logic. Note that detector actuations can only be registered by TRAF-NETSIM on detectors which have been defined to exist. It is therefore necessary to make sure that at least one detector is defined on card type 46 for each phase which may be indicated by the external signal control logic. This does not imply that real physical detectors must be placed in the road, but only that the existence of those detectors must be indicated to the internal signal control logic. By placing actuations on detectors associated with a new non-active phase, the internal signal control logic will begin the transition from the currently active phase to the new phase.

7.2.3 Modifications to TRAF-NETSIM Source Code

External signal control logic interface has been made possible by modifications and additions to the TRAF-NETSIM program. Although most of these changes were discussed in the preceding sections, it may also be useful to study Appendix H, which contains a complete listing of all of the changes and additions that were made to the original TRAF-NETSIM source code during this development. In addition to the external signal control capability, some additional data collection capabilities were added. The new data includes records of detector actuations caused by vehicles during the simulation, counts of actuations at surveillance detectors, phase decisions made by external signal control logic (if in use), and signal states throughout the duration of the simulation. This data is useful for verification of external signal control logic as well as allowing TRAF-NETSIM to produce data that can be useful for conducting other experiments. The data is collected and stored in external files. At the completion of the simulation, the data in these files can be accessed and used for any purpose desired. Appendix I contains detailed descriptions of all new data which has been made available.

7.3 Development Procedure for External Signal Control Logic

The previous sections have described the how data can be passed between TRAF-NETSIM and external signal control logic. Using this procedure and allowing subroutine

CONTROL to serve as an interface, RHODES or any other type of signal control logic can be evaluated through simulation using TRAF-NETSIM. The following steps outline the procedure for implementing external signal control and interfacing it to TRAF-NETSIM. (Appendix F is a collection of "Programmer's Notes" and provides additional information which may be helpful.)

(1) Develop and test the external control logic independently from TRAF-NETSIM. It may consist of a single or multiple subroutines, and need not be written in FORTRAN. See Appendix F for a description of how to link C routines to the FORTRAN simulation. Test the routine thoroughly to verify that is produces the desired phase decisions. Remember that it must output phase decisions at every time step, even if no change is desired.

(2) Any intersection which is to be controlled externally must be specified to have a fully-actuated signal controller. To achieve this, make sure that no card type 44 is defined for that intersection. Appendix L contains a TRAF-NETSIM input file that was used for testing the simulation model with external signal control logic. To achieve rapid switching times, it is important to specify the signal parameters to allow rapid phase termination. Specifically, the extension and gap related parameters should be set carefully. This can be accomplished by setting the parameters on card type 47 similar to those in the data file in Appendix L, although other settings can be used would also perform well.

(3) Define actuated controller detectors (card type 46) for each phase that may be specified by the external signal control logic.

(4) If detector data is to be used, define surveillance detectors with station numbers at all desired locations. Surveillance detectors are specified on card type 42 in the modified version of TRAF-NETSIM which was used for this research. Card type 42, and therefore surveillance detector capabilities, are not included in the versions of TRAF-NETSIM released for general distribution. Appendix F gives further information about the version of TRAF-NETSIM used here.

(5) Modify the subroutine CONTROL so that it will call the external signal logic and pass any needed parameters.

(6) Verify that the signal control logic returns the proper single byte return codes to CONTROL, and that it returns a value of 0 (zero) for any intersection which contains an actuated controller but is not being controlled by the external logic. Remember, the external signal logic will be called for *every* intersection which is defined in the TRAF-NETSIM input data to have an actuated signal controller. The external signal control logic must determine if the current node is to be controlled externally or internally.

(7) Assure that any variables which need to be maintained (stored) by the signal logic between invocations are saved. In a C program, for example, all such variables can be defined as static data types.

(8) Compile all signal logic and TRAF-NETSIM modules to object form and link them using the FORTRAN compiler. For example, use f77 *.o to link the object files on a UNIX system. Appendix F gives further details about how to build the executable file.

7.4 Model Validation

To verify the interfacing techniques between TRAF-NETSIM and external signal control logic, and to provide examples of how to construct the external signal control logic, two sample signal control logic programs were developed. These were the fixed-time control (Section 5.1.1), and the semi-actuated control logic (Section 5.1.2). Using the external signal control logic interface, each of the controllers was linked to the TRAF-NETSIM simulator and used during a simulation. The results from the simulation were then compared to the results using the internal TRAF-NETSIM implementation for each of these controllers.

7.4.1 Fixed-Time Control Logic

The function of the external fixed-time signal control logic developed in this section is to determine whether or not the signal at an intersection should be switched to the next phase according to a synchronized clock and a set of timing plans.

Figure 53 represents an example of a typical fixed-time control plan. Essentially, the fixed-time control logic follows a sync clock which, in this example, has a range from 0 to 89 (cycle time minus one time unit). In the figure, the Main Street Thru phase would activate at sync time 27 and complete the amber and all red durations by sync time 63. The next active phase, Main Street Left, would then begin at sync time 63. The Side Street Thru phase would be active from sync time 78 through sync time 20; this phase would pass the zero sync point.

Some assumptions made while developing the fixed-time control logic program are (1) all phases allow permissive right turns, (2) right turn calls are not specifically addressed, (3) all times and phase durations have units of seconds and are integers, and (4) phase duration is zero for all non-existent phases. The C program, fixed-time plans data file, and a sample run of the program can be found in Appendix J.

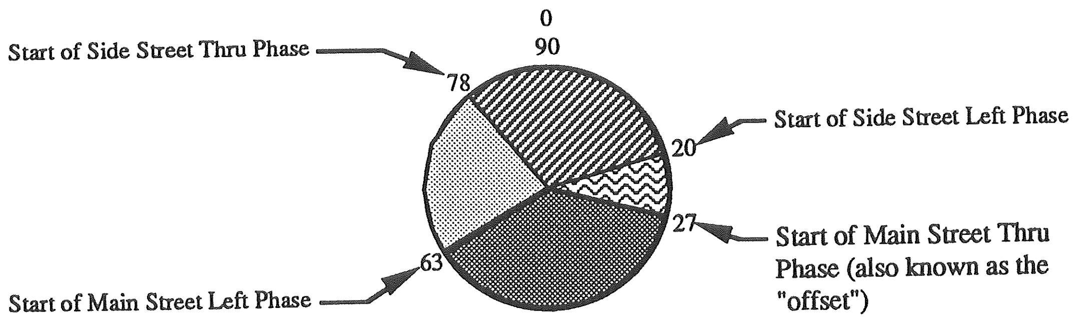

0
90
78
20
27
63
Start of Side Street Thru Phase
Start of Side Street Left Phase
Start of Main Street Thru Phase (also known as the "offset")
Start of Main Street Left Phase

Figure 53. Graphic display of fixed-time control logic.

The following inputs are required for the fixed-time control logic program:

1) A data file named "fixed.plans" containing the following information:
   - total number of timing plans (i.e., total number of intersections),
   - the cycle length (same for all intersections), and
   - intersection identification number, offset time, phase durations (in proper sequence of phases–MST, MSL, SST, SSL, where MST is Main Street Thru, MSL is Main Street Lefts, SST is Side Street Thru, and SSL is Side Street Lefts),

2) Additional information required from TRAF-NETSIM or a data file:
   - intersection identification number requested (to check status of that intersection), and
   - current synchronization time (integer value in interval [0, cycle time]).

7.4.2 Semi-Actuated Control Logic

Figure 54 shows how typical semi-actuated control logic maintains platoon progression along an artery. The shaded area in the cycle represents Main Street Thru green time. Hence, from sync time 60 to sync time 23, the signal will always have the Main Street Thru phase active. If there are vehicle detections on any other phase (Phase 2 = Main Street Left, Phase 3 = Side Street Thru, Phase 4 = Side Street Left), the control logic may only switch at the yield point. If there are no calls on any other phase, the signal will remain in Main Street Thru at least until it returns to the yield point. If there are calls on another phase besides Main Street Thru, the priority in which they are responded to are as follows: Phase 2, Phase 3, and Phase 4. If, for example, there were calls on Phase 2 and Phase 4 by the time the sync clock reached the yield point, Phase 2 would be activated at sync time 23. Then, Phase 2 would terminate at or before (if maximum green is reached or allotted green time is expended) the Phase 2 Force-off point (sync time 30) and Phase

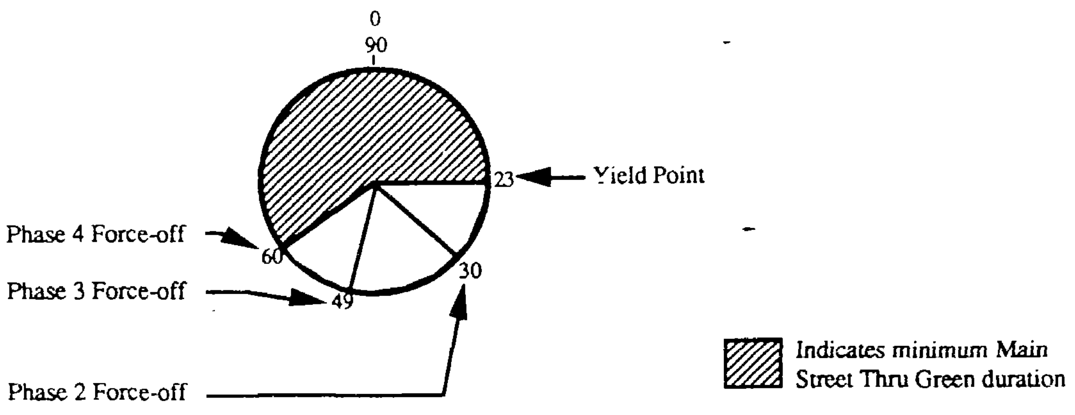


Figure 54. Graphic display of how Main Street Thru will maintain platoon progression along an artery.

4 would then become active. Once Phase 4 is terminated, the signal will return to the Main Street Thru phase.

The function of the semi-actuated signal control logic developed here is to simulate semi-actuated traffic signal control logic as accurately as possible. For the purpose of this implementation, it is assumed that semi-actuated control logic allows for four phases (Phase 1 -> MST, Phase 2 -> MSL, Phase 3 -> SST, and Phase 4->SSL), but may easily be modified to allow for eight or more phases.

Essentially, the C program will operate under the following conditions:

(1) main street through may terminate only at a specified point in the synchronization time, referred to as the yield point, and only if there is a demand for a different phase;
(2) all phases except MST may terminate due to force-off, maximum green reached, or allotted green time expended;
(3) all phases except MST may be active after the yield point, given there is demand for that respective phase, and may only switch to a phase in a certain sequence up to and including phase 1. For example, if the current phase is phase 3, the next phase could be phase 4 or phase 1. If the current phase is 2, then the next phase could be either 3, 4, or 1;
(4) to assure platoon progression, MST must always be active between the point where the synchronized clock reaches the last force-off in the cycle and the yield point (Figure 54); and
(5)if there are no calls in any phase, the control logic will rest (remain) in MST (i.e., Phase 1).

The C program for the semi-actuated signal control logic, the parameter file for the control logic, and a sample run of the program can be found in Appendix K.

7.4.3 Validation Experiments and Results

Validation was conducted using a TRAF-NETSIM network model for a real collection of streets and intersections located in Tucson, AZ. Appendix L contains the TRAF-NETSIM input file which defines this network. All input configurations and loadings represent real data from the actual street network. Figure 49 shows a link/node representation of this network. This particular area was chosen to provide an accurate representation of traffic conditions along a section of the Campbell Avenue arterial. Furthermore, this section was of interest because it contains an arterial with large signalized intersections and is located near the football and basketball stadiums of the University of Arizona, which represent major traffic sources and sinks. The simulations performed used external signal control logic to control the signals at the four of the central intersections: nodes 335, 369, 401, and 483. The external signal control logic and the interface logic were verified to perform as designed. This was accomplished by analyzing the numerical and graphical output data and by verifying that phase calls and signal changes occurred at the desired times.

As a further test, simulations were run to compare the external fixed-time and semi-actuated signal control logic with the internal TRAF-NETSIM counterparts. This was accomplished using simulation runs representing 15-minute interval between 11:00 a.m. and 11:15 a.m. on a weekday. The simulation was repeated five times for each signal control method, using different random number seeds for each of the five runs. Performance measures were aggregated across all four intersections and the interconnecting streets by defining them to be a "section" in the TRAF-NETSIM input file. These performance measures from the five runs were then averaged for each

controller. Table 7(a) and 7(b) summarizes the results comparing the internal and external implementations of the controllers.

The difference between the two implementations of fixed-time control is due to slight timing differences. Using the external signal control logic, termination of the current phase may be delayed by one or two seconds after calls are placed for a new phase. This effectively causes a switching delay. For reasons unknown at this time, the switching delay is not constant, and therefore could not be compensated for precisely. Because of

Table 7(a). Summaries of performance measures for fixed-time logic.

	TRAF-NETSIM Internal Fixed-Time Logic	External Fixed-Time Logic	Difference
Delay (Vehicle Minutes)	1424	1438	1.0 %
Stops (per Trip)	3.3	3.26	1.2 %
Average Speed (MPH)	12.9	12.5	3.1 %

Table 7(b). Summaries of performance measures for semi-actuated logic.

	TRAF-NETSIM Internal Semi-Actuated Logic	External Semi-Actuated Logic	Difference
Delay (Vehicle Minutes)	815	902	10.6%
Stops (per Trip)	2.4	2.8	16.7 %
Average Speed (MPH)	18.0	17.3	3.9 %

this, some of the switching times differed by one or two seconds between the internal and external fixed-time control implementations. If this delay was constant, it would be possible to adjust the parameters of the external fixed-time control logic to compensate for this delay and allow the two implementations to perform identically. Further investigation is required to determine the cause of the switching delay.

The semi-actuated control implementations, on the other hand, were expected to show some difference. This can be attributed primarily to the fact that the external semi-actuated control logic was operating using only passage detector information and did not receive the presence detector information that the internal controller was able to use. Because of this lack of presence information, the external controller would sometimes allocate unneeded green time to the left turn phases. Additionally, some of the disparity between the two implementations is once again due to the switching delay. Even though the external controllers did not duplicate the behavior of the internal controllers exactly, the results of these tests are nonetheless positive. The real objective was not to create perfect replicas of the internal controllers, but to prove that external signal logic can be effectively simulated and evaluated using TRAF-NETSIM and the interfacing method developed here. That goal was indeed accomplished, while identifying some areas where the interface can be improved in the future.

7.5 Integration of COP/PREDICT and TRAF-NETSIM

The validated simulation model was used to test and evaluate the traffic-adaptive control algorithm, COP (see Section 5.2.3), together with the traffic flow prediction model, PREDICT (see Section 5.3). A single intersection, number 483 on Figure 49, was used as the test intersection. The performance of the COP algorithm was compared to the existing semi-actuated control logic (internal logic to TRAF-NETSIM) based on the total number of stops. Fifty experimental runs of the simulation model were made for COP/PREDICT and fifty for semi-actuated control logic.

Figure 55 shows the results of the simulation experiment. A straight line has been fitted to the data to indicate the trend as a function of link volume. This straight line is fit to the data only for the purpose of showing the general trend and it is not proposed that stops are a linear function of volume. The results indicate a significant reduction in the number of stops using the COP/PREDICT method.

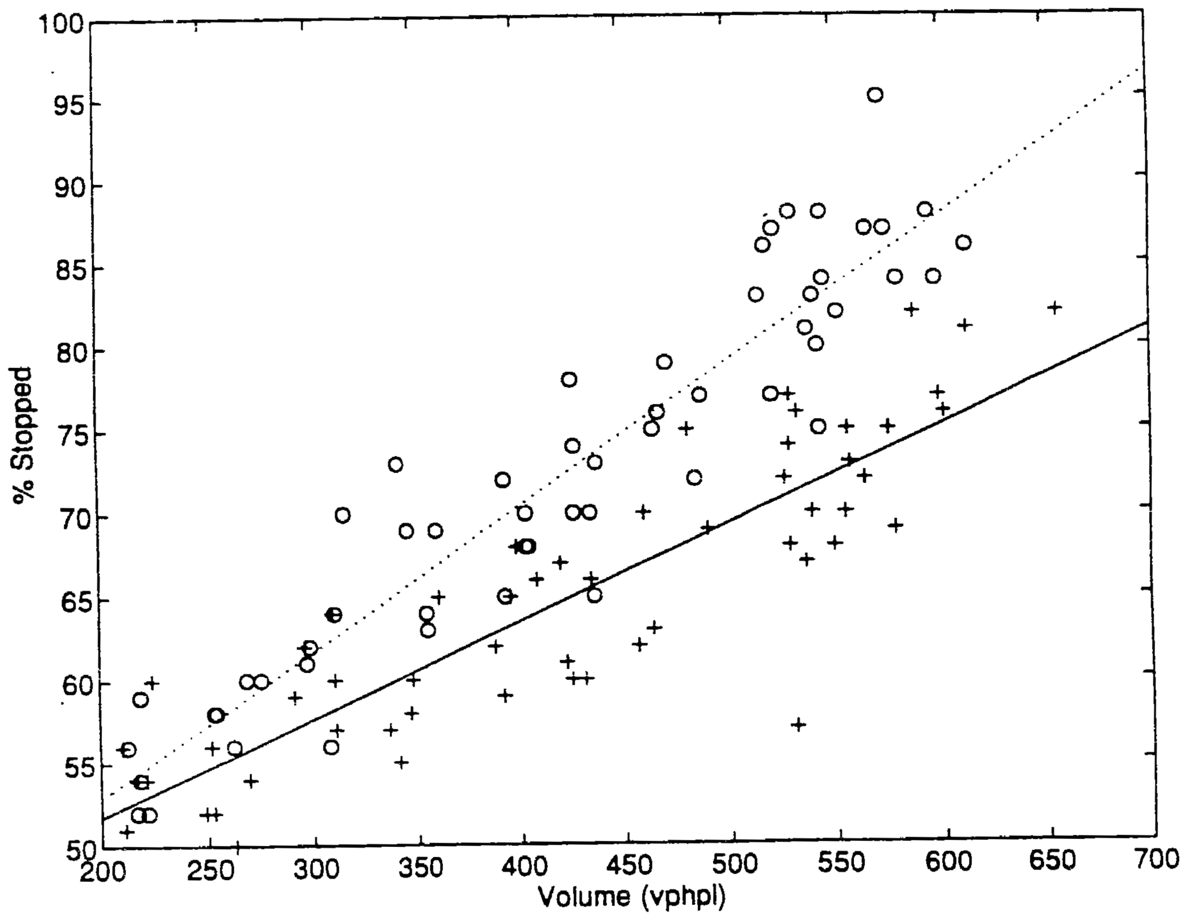


Figure 55. Comparison between well-timed semi-actuated control (represented by the 'o' and the dotted line) and the COP intersection control logic (represented by the '+' and the solid line).

8. IVHS ACTIVITIES AND PROPOSAL DEVELOPMENT

In Phase I of the RHODES Project, the research team had developed a national consortium in response to a FHWA request for proposal DTFH61-92-R-00001, that included ADOT, The Cities of Tucson and Tempe, several private companies and other academic institutions. The proposal entitled "*A Real-Time Traffic Adaptive Signal Control System*" was submitted to FHWA, a copy of the proposal was provided to ADOT. Although our consortium did not win the contract, in the sprit of partnership among the public sector, private sector, and academia, several other proposals were developed and submitted to ADOT and FHWA during RHODES Project - Phase 2a.

RHODES ITMS Proposal

Significant effort was focused on the development of problem statement and work statement for a research project to field test some of the concepts of the RHODES system at a location in Maricopa County. After several meetings with traffic engineers and technical mangers within ADOT, Phoenix, and Tempe, the research team proposed the development and field testing of real-time traffic adaptive control at an interchange along the I-17 Corridor. A draft proposal was developed and submitted to ADOT in April 1992. This was later revised to accommodate resource constraints, and resubmitted in September 1992 entitled "Real-Time Traffic Adaptive Control for Integrated Traffic Management of the I-17 Corridor". The proposal received favorable reviews but due to further institutional and budgetary considerations, the proposed scope was divided into two phases and the proposal for phase one was submitted to ADOT in June 1993. (This phase was approved for funding in December 1993.)

Advanced Traffic Management Systems Proposal

With Loral AeroSys as the lead, a consortium was developed that included TASC, KLD Associates, Ball Systems Engineering, City of Tucson, PAG, ADOT and the University of Arizona, to respond to FHWA RFP DTFH-92-R-00073 for the development of ATMS Support Systems. The proposal was submitted to FHWA in June 1992. It was awarded to this consortium in October 1992. The funding level of the contract is approximately $3 Million for five years; UA subcontract is for $274,000 for five years.

Other Proposals (not funded)

The research team also developed and submitted several proposals to FHWA during the contract period of the RHODES Phase II(a) Project and the subsequent 4-month period. Specifically, they were:

March 1992: *"Sun of RHODES"*, an equipment grant proposal to SUN Corporation for a platform for real-time traffic control, for $ 67,000

June 1992: *"Traffic Models for Testing Real-Time Traffic Adaptive Signal Control Logic: Phase I"* to FHWA in response to RFP DTFH-61-92-R-00110. The University of Arizona was the prime with subcontractors JHK & Associates, TASC, and the City of Tucson. $ 977,000 was requested; In addition, UA proposed a cost share of $82,000, ADOT of $40,000 and City of Tucson of $11,000.

May 1993: *"IVHS Research Center of Excellence"* to USDOT/FHWA in response to RFP DTFH-93-X-00017 for the establishment of a IVHS research center at the University of Arizona. A consortium was developed that included the following partners:

ADOT,
City of Tucson,,
Maricopa County,
Hughes Missiles Systems,
US West,
DEC,
TASC,
Loral AeroSys,
JHK & Associates,
Computran,
Oak Ridge National Laboratories, and
Viggin Corporation.

$ 1 Million per year was requested. In addition, the partners committed a total of $ 650,000 (including UA cost-share) if the proposal was funded.

June 1993: *"Model Enhancements for Evaluating Traffic Operations Under IVHS"* with TASC (prime contractor). Approximately $ 1.2 Million for five years was requested with UA request totaling $737,000.

IVHS related activities and meetings

The key researchers on the project participated in the following meetings during the RHODES Phase II(a) Project period and the subsequent 4-month period:

L. Head, P. Mirchandani and D. Sheppard (the presenter) gave a presentation on "Traffic Controls for IVHS" at the Spring ASCE Meeting in Tucson, April 2, 1992

P. Mirchandani gave a seminar on "Real-Time Urban arterial Traffic Control" to the Civil Engineering Department and the Florida Department of Transportation, April 30, 1992

L. Head, and P. Mirchandani participated at the ADOT/ASU/UA retreat on IVHS and Arizona Research Thrusts, April 2-4, Sedona, AZ.

P. Mirchandani organized a meeting of the Arizona Traffic Engineering Community (AZTEC) for mission planning for IVHS efforts in traffic engineering for the State of Arizona, at the University of Arizona, April 20, 1992

P. Mirchandani delivered a paper "A Hierarchical IVHS-ATMS Structure for Real-Time Traffic Control" (co-authors L. Head and D. Sheppard) at the 25th ISATA Silver Jubilee Conference, Florence, Italy, June 1-5, 1992

P. Mirchandani gave presentations on "Real-Time Advanced Traffic Control Systems" to the Faculty of Engineering at the University of Sienna, June 5, 1992; and to the Faculty of Electronics at the Warsaw University of Technology, Poland, June 10, 1992.

L. Head attended the Integrated Traffic Management Systems Workshop and the 1992 TRB Signal Systems Committee summer meeting in Newport Beach, CA, June 22-25.

P. Mirchandani visited Dr. Christopher Wrathall of MIZAR in Torino to discuss UTOPIA, a real-time traffic control system implemented in that city by MIZAR, June 1992.

L. Head and P. Mirchandani met with Dr. Larry Klien of Hughes, on July 14 in Phoenix (with individuals from ADOT, Maricopa County, and Cities of Phoenix and Tempe), and on July 15 in Tucson (with individuals from Pima County and City of Tucson) to plan field tests for state-of-the-art vehicle detectors in Arizona. (This project is funded by a contract from FHWA to Hughes Ground Systems.)

L. Head presented a progress report on RHODES PHASE-II(a) to the PAG Transportation Planning Committee on September 1, 1992.

L. Head and P. Mirchandani (presenter) gave a presentation on IVHS to the University of Arizona Phoenix Alumni Association in Phoenix, September 23,1992.

L. Head and P. Mirchandani participated and gave a presentation on RHODES at the ATRC/ADOT IVHS meeting in Tempe, AZ, October 15, 1992.

L. Head and P. Mirchandani participated and gave a presentation on RHODES at the JHK Workshop on Freeway Management Systems, October 22, 1992.

P. Mirchandani (presenter), L. Head and D. Sheppard gave an invited presentation on "RHODES: A Hierarchical Real-Time Traffic Control Systems" at the San Francisco ORSA/TIMS Meeting November, 1-4, 1992.

S. Sen gave an invited presentation on "Coordinated Optimization for Phases (COP)" at the San Francisco ORSA/TIMS Meeting , November 1-4, 1992.

P. Mirchandani visited FHWA, Washington D.C., as a member of the Technical Advisory Committee on ATMS Support Systems Project conducted by Loral AeroSys.

L. Head, P. Mirchandani and D. Sheppard attended the Transportation Research Board 72st Annual Meeting , January 10-14, 1993. D. Sheppard gave a presentation on "Real-Time Speed Advisory for Urban Traffic Management".

L. Head and P. Mirchandani gave a presentation to ADOT, City of Phoenix, City of Tempe, and Maricopa County Traffic Engineers on potential application of RHODES results to I-17 corridor, March 26, 1993.

P. Mirchandani met with Greg Shelton, Product Line manger for Technology and Commercial Programs, and Fred Smoller, Program Manger for Intelligent Sensor Systems to discuss possible research collaborations in the IVHS area with UA and the public agencies within the State of Arizona, April 12, 1993.

L. Head and P. Mirchandani visited JHK & Associates in Pasadena to find out more about JHK/LA's "Smart Corridor" Project and JHK's traffic control systems, June 14, 1993.

L. Head and P. Mirchandani (with Mike Ward of Loral AeroSys.) visited the Jet propulsion Laboratory to discuss mutual interests in the IVHS area, and data fusion and parallel processing for IVHS applications, June 15, 1993.

REFERENCES

TRAF-NETSIM User's Manual, U.S. Department of Transportation, Federal Highway Administration, November 1989.

Baras, J. S., William S. Levin and T. L. Lin, (1979), "Discrete-Time Point Processes in Urban Traffic Queue Estimation", IEEE Transactions on Automatic Control, AC-24, No. 1, 12-27.

Chang, E. C.-P., S. L. Cohen, C. Liu, N. A. Chaudary and Carroll, (1988), "MAXBAND-86 : A Program for Optimizing Left-Turn Phase Sequence in Multilateral Closed Networks", Transportation Research Record 1181, 61-67.

Chaudhary, N. A. and C. J. Messer, (1993), "Passer IV - A Program for Optimizing Signal Timing in Grid Networks," Texas Transportation Institute, The Texas A&M University System, College Station, Texas, prepared for the Transportation Research Board, 72nd Annual Meeting, Washington, DC.

DeGroot, M. H., (1970), Optimal Statistical Decisions, McGraw-Hill, Inc., New York, NY.

Gartner, N., H., (1983), "OPAC: A Demand-Responsive Strategy for Traffic Signal Control", Transportation Research Record No. 906, 75-81.

Gartner, N., H., (1981), "Discussion of Improved Estimation of Traffic Flow for Real-time Control", Transportation Research Record, No. 795, 38-39.

Gartner, N. H., P. J. Tarnoff and C. M. Andrews, (1991), "Evaluation of the Optimized Policies for Adaptive Control (OPAC) Strategy" Transportation Research Board.

Gill, P. E., W. Murray and M. H. Wright, (1981), Practical Optimization, Academic Press, London.

Hadi, M. A. and C. E. Wallace, (1992), "Improved Optimization Effeciency in TRANSYT-7F," Transportation Research Center, University of Florida, Gainesville, Florida, for presentation at the ORSA/TIMS Joint National Meeting, Orlando Florida.

Higle, J. and S. Nagarajan, (1992), "Real-time Prediction of Turning Flows: A Dynamic Bayes Procedure" Interim Report - RHODES Project: PHASE I.

Little, J. D. C., M. D. Kelson and N. H. Gartner, (1981), "MAXBAND; A Program for Setting Signals on Arteries and Triangular Networks", Transportation Research Record 795, 40-46.

Okatani, I. and Y. J. Stephanedes, (1984), "Dynamic Prediction of Traffic Volume Through Kalman Filtering Theory", Transportation Research Part B : Methodological, Vol. 18B, No. 1, 1-11.

Segall, A., (1976), "Recursive Estimation from Discrete-TIme point Processes", IEEE Transactions on Information Theory, IT-22, no. 4, July 1976, 422-431.

Sen, S., (1991), "Coordinated Optimization of Phases", Working Paper, Systems and Industrial Engineering Department, University of Arizona.

Stephanedes, Y. J., P. G. Michalopoulos and R. A. Plum, (1981), "Improved Estimation of Traffic Flow for Real-time Control", Transportation Research Record, No. 795, 28-38.

Tarnoff, P. J., "The results of FHWA urban traffic control research: An interim report," Traffic Engineering, Vol. 45, pp. 27-35, April 1976.

APPENDICES

APPENDIX A: DISPATCH 1.0 Program Listing

```
/*=================================================================*/
/*=                                                              =*/
/*=           Intersection Model and Simulation Version 3        =*/
/*=                                                              =*/
/*============================================================ ========*/
/*=                                                              =*/
/*=      Campbell Ave. & Sixth St. Intersection, Tucson, Arizona  =*/
/*=                                                              =*/
/*=   Developed By:              Dr. K. Larry Head                =*/
/*=   Revision:                  June 2, 1992                     =*/
/*=   Programmed By:             Greg Tomooka                     =*/
/*=   Started:                   June 9, 1992                     =*/
/*=   Completed:                 June 10, 1992                    =*/
/*=   Revised:                   June 29, 1992                    =*/
/*=   Programming Assisted By:   Dr. K. Larry Head                =*/
/*=                                                              =*/
/*=   Purpose of Program:                                        =*/
/*=                                                              =*/
/*=   This program simulates an eight-phase intersection.  Given that  =*/
/*=   the detector is DETECT time units from the stop bar, queues for  =*/
/*=   each phase (1, ..., 8) will build up or decrease according to a  =*/
/*=   controller programmed for determining the number of units leaving =*/
/*=   from the stop bar.  Also, the queues do not build up until the   =*/
/*=   units reach the stop bar (DETECT time units after crossing the   =*/
/*=   detector).                                                 =*/
/*=                                                              =*/
/*=   This updated version will decide when to change phase      =*/
/*=   combinations based upon a linear combination of delay and total  =*/
/*=   number of stops.  The program evaluates in a range equal   =*/
/*=   to the DETECT value about the original PHASETIME value.  This    =*/
/*=   is not necessarily a symmetrical range about PHASETIME.    =*/
/*=                                                              =*/
/*============================================================ ========*/

/*============================================================ ========*/
/*=                                                              =*/
/*= Program is set for the following MAXIMUM number of values    =*/
/*=    Number of Stages, J = 500                                 =*/
/*=    Number of Phases, P = 200                                 =*/
/*=    Number of Discrete Time Units, T = 1000                   =*/
/*=                                                              =*/
/*= The input file is read into a structure in the following order:  =*/
/*=    Intersection ID number                                    =*/
/*=    Number of phases                                          =*/
/*=    Minimum green time                                        =*/
/*=    Clearance time                                            =*/
/*=    Total number of stages                                    =*/
/*=    Stages                                                    =*/
/*=    Time horizon                                              =*/
/*=    Arrival data for phases: 1 2 3 4 5 6 7 8                  =*/
/*=                                                              =*/
/*=    Input File:        queues.dat                             =*/
/*=    OutPut File:       optimal.sol                            =*/
/*=                                                              =*/
/*============================================================ ========*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/*  ----------------  Constants  ---------------  */
```

```
#define             JMAX      500      /*  maximum number-of stages            */
#define             PMAX     200      /*  maximum number of phases            */
#define             TMAX    1000      /*  maximum time horizon        */

#define             DF_SFRATE      1   /*  saturation flow rate          */
#define             DF_DETECT      7   /*  travel time from detector to stop
                                  line                    */
#define             MAX_HORIZON   300 /*  maximum allowable time horizon       */

typedef struct {
   int id ;              /*  Identification number of intersection       */
   int P ;               /*  Number of possible phases            */
   int Gamma ;                  /*  Minimum green time              */
   int Delta ;                  /*  Clearance time; All phases red           */
   int J ;               /*  Total number of stages requested          */
   int stage[JMAX] ;     /*  Stages                                */
   int T ;               /*  Total time horizon                    */
   int a[TMAX+2][PMAX+2] ; /*  Arrival data                              */
   } intersection_t ;

intersection_t Tucson ;

double linearcombo();                          /*  function returning p.i.     */

main()
{
        int     p ;                     /*  current stage number             */
        int     t ;                     /*  current time               */
        int     s ;                     /*  stage index                */
        int     q[TMAX+2][PMAX+2] ;     /*  table of queue lengths           */
        int     tphase[TMAX+2] ;        /*  current phase during time t  */
        int     phase ;                        /*  current stage                     */
        int     sfrate ;                /*  input sat. flow rate       */
        int     detect ;                /*  input time from det. to stop */
        int     phasetime ;             /*  input change of phase time   */
        float   stop_wt ;               /*  input stops weight         */
        float   delay_wt ;              /*  input delay weight         */
        int     time ;                  /*  input total time horizon          */

        int     c ;                     /*  change of phase index             */
        double  pi[TMAX+2] ;                   /*  p.i.  for changing phase at c  */
        int     stops[TMAX+2][PMAX+2] ;    /*  total stops for each phase      */
        int     delay[TMAX+2][PMAX+2] ;    /*  total delay for each phase      */
        int     t_stops[TMAX+2] ;       /*  total stops for changing phase
                                            at c                  */
        int     t_delay[TMAX+2] ;       /*  total delay for changing phase
                                            at c                  */
        int     optq[TMAX+2][PMAX+2] ;         /*  table of optimal queue lengths */
        double  optpi ;                        /*  optimal p.i.
*/
        int     opttime ;               /*  optimal time to change phase   */
        int     opttphase[TMAX+2] ;     /*  optimal phase combination
                                            for corresponding time units   */

        FILE    *Data, *OutPut ;        /*  Input and Output Files            */

 /*  ===================  READ THE DATA FROM INPUT FILE  =================  */

        Data = fopen("queues.dat", "r") ;
        OutPut = fopen("optimal.sol", "w") ;

        fscanf(Data,"%d", &Tucson.id) ;
        fprintf(OutPut,"-----------------------------------------------------------------
-----\n") ;
        fprintf(OutPut,"   Information for Intersection Identification     # %12d\n",
Tucson.id) ;
        fprintf(OutPut,"-----------------------------------------------------------------
-----\n") ;

        fscanf(Data,"%d", &Tucson.P) ;
        fprintf(OutPut,"  Number of Phases =  %4d", Tucson.P) ;

        fscanf(Data,"%d", &Tucson.Gamma) ;
        fprintf(OutPut,"           Minimum Green Time =  %8d\n", Tucson.Gamma) ;

        fscanf(Data,"%d", &Tucson.Delta) ;
        fprintf(OutPut,"  Clearance Time =     %3d", Tucson.Delta) ;

        fscanf(Data,"%d", &Tucson.J) ;
        fprintf(OutPut,"           Total Number of Stages =  %4d\n", Tucson.J) ;

        fprintf(OutPut,"  Sequence of Stages:   ") ;
        for (p = 1; p <= Tucson.J; ++p) {
           fscanf(Data,"%d", &Tucson.stage[p]) ;
           fprintf(OutPut,"%3d ", Tucson.stage[p]) ;
        }
        fprintf(OutPut,"\n") ;

        fscanf(Data,"%d", &Tucson.T) ;
        fprintf(OutPut,"  Time Horizon =  %9d\n\n", Tucson.T) ;

        fprintf(OutPut,"Arrival Data\n") ;
        fprintf(OutPut,"------------------------------\n") ;
        fprintf(OutPut,"Time   1  2  3  4  5  6  7  8\n") ;
        fprintf(OutPut,"------------------------------\n") ;

        for (t = 0; t <= Tucson.T; ++t) {
           fprintf(OutPut,"%3d    ", t) ;
           for (p = 1; p <= Tucson.P; ++p) {
              fscanf(Data,"%d", &Tucson.a[t][p]) ;
              fprintf(OutPut,"%2d ", Tucson.a[t][p]) ;
           }
           fprintf(OutPut,"\n") ;
         if(t%10 == 0)
           fprintf(OutPut,"\n") ;
        }
        fprintf(OutPut,"\n") ;


/*  ==================  REQUEST FOR INPUT INFORMATION  ==================  */

        printf("\n") ;
        printf("\n=========================================================
========================\n") ;
        printf("\n           Intersection Model and Simulation:  An Optimization
Program\n") ;
        printf("\n=========================================================
========================") ;

/*  =======================  REQUEST FOR TIME HORIZON  ====================  */

        printf("\n\n          The following information is required.\n") ;
        printf(  "\n          To enter the default value in parentheses, enter 0 (a
zero).") ;
        printf(  "\n          Unless otherwise noted, all inputs should be of integer
form.\n\n\n") ;
```

```
        printf(  "Enter total time horizon (Default time horizon = %4d  time units):
",Tucson.T) ;
        scanf("%d", &time) ;

        if (time > 0) {
           if (time > MAX_HORIZON) {
              Tucson.T = MAX_HORIZON ;
              printf("\nWARNING!  --->   You exceeded the maximum time horizon.\n") ;
              printf(  "                 Time Horizon has been set to %5d\n\n",
MAX_HORIZON) ;
           }
           else Tucson.T = time ;
           for (t = Tucson.T + 1; t <= Tucson.T; ++t) {
              fprintf(OutPut,"%3d    ", t) ;
              for (p = 1; p <= Tucson.P; ++p) {
                 fscanf(Data,"%d", &Tucson.a[t][p]) ;
                 fprintf(OutPut,"%2d ", Tucson.a[t][p]) ;
              }
              fprintf(OutPut,"\n") ;
            if(t%10 == 0)
              fprintf(OutPut,"\n") ;
           }
           fprintf(OutPut,"\n") ;
        }

        printf("\nEnter saturation flow rate (1 = Default rate):  ") ;
        scanf("%d", &sfrate) ;
        printf("\nEnter travel time from detector to stop line (7 = Default time):  ")
;
        scanf("%d", &detect) ;
        printf("\nEnter initial time when phase combination is scheduled to change\n")
;
        printf(  "   (Default change time = %4d; this value must be between %3d and
%3d):  ", Tucson.T/2, 0, Tucson.T) ;
        scanf("%d", &phasetime) ;

        printf("\n\n\nCaution:  There are only two (2) measures of effectiveness for
this model.\n") ;
        printf("\nThe sum of the two weights must add up to exactly one (1.0).\n\n\n")
;
        printf("\nEnter decimal weight for stops (No default value,\n") ;
        printf(  "     but delay weight will be calculated for you.):  ") ;
        scanf("%f", &stop_wt) ;
        if (stop_wt >= 1.0000) {
           delay_wt = 0.0000 ;
           printf("\nNOTICE:     Since stop weight is greater than or equal to one,\n")
;
           printf(  "             it has been set to one and delay weight has been set
to zero.\n\n") ;
        }
        else if (stop_wt <= 0.0000) {
           stop_wt = 0.0000 ;
           delay_wt = 1.0000 ;
           printf("\nNOTICE:     Since stop weight is less than or equal to zero,\n") ;
           printf(  "             it has been set to zero and delay weight has been set
to one.\n\n") ;
        }
        else if (stop_wt != 1.0000) {
           delay_wt = (1.000000000 - stop_wt) ;
           printf("\nBased upon the value entered for stop weight, delay weight =
%8.5f", delay_wt) ;
        }
        printf("\n\n\nResults are being sent to an output file labeled 'optimal.sol'
...\n\n\n\n\n") ;

        if (phasetime == 0) phasetime = Tucson.T/2 ;
        if (sfrate == 0) sfrate = DF_SFRATE ;
        if (detect == 0) detect = DF_DETECT ;

        fprintf(OutPut,"Current Saturation Flow Rate = %7d \n", sfrate) ;
        fprintf(OutPut,"Delay from Detector to Stop Line = %3d ", detect) ;
        fprintf(OutPut,"\n\n\n") ;


/*  =======================  INITIALIZE VARIABLES  ======================  */

    optpi = 999999999999999.55555000 ;
    opttime = 0 ;

    for(t = 0; t <= Tucson.T; ++t) {
       t_stops[t] = 0 ;
       t_delay[t] = 0 ;
       pi[t] = 0.0 ;
       for(p = 1; p <= Tucson.J; ++p)
             q[t][p] = 0 ;
             optq[t][p] = 0 ;
            stops[t][p] = 0 ;
            delay[t][p] = 0 ;
    }

    for(p = 1; p <= Tucson.J; ++p)
          q[0][p] = Tucson.a[0][p] ;


/*  =====================  SIMULATE INTERSECTION  =====================  */

for (c = (phasetime - detect/3); c <= (phasetime + (2 * detect)/3); ++c) {
    pi[c] = 0.0 ;
    s = 1;
    for(t = 0; t <= Tucson.T; ++t) {
       tphase[t] = Tucson.stage[s] ;
       if(t == c) {
         phase = Tucson.stage[s] ;
         ++s ;
       }
       if(t >= detect) {
         for(p = 1; p <= Tucson.P; ++p) {
            q[t+1][p] = q[t][p] + Tucson.a[t-detect][p] -
                        depart(tphase[t], q[t][p],
                        Tucson.a[t-detect][p], t, p, sfrate) ;
            delay[c][p] = delay[c][p] + q[t+1][p] ;
            if (q[t+1][p] > q[t][p])
               stops[c][p] = stops [c][p] + (q[t+1][p] - q[t][p]) ;
         }
       }
       else
         for(p = 1; p <= Tucson.P; ++p) q[t][p] = 0 ;
    }                                  /*  End of t loop  */

    for (p = 1; p <= Tucson.P; ++p) {
       t_stops[c] = stops[c][p] + t_stops[c] ;
       t_delay[c] = delay[c][p] + t_delay[c] ;
    }
    pi[c] = linearcombo(t_stops[c], t_delay[c], stop_wt, delay_wt) ;
    if (pi[c] < optpi) {
       optpi = pi[c] ;
```

```
      opttime = c ;
      for(t = 0; t <= Tucson.T; ++t) {
        opttphase[t] = tphase[t] ;
        for(p = 1; p <= Tucson.P; ++p)
           optq[t][p] = q[t][p] ;
      }
   }                                 /*  End of if pi[c]   */
}                             /*  End of c loop  */


/*  ======================  PRINT QUEUE MATRIX ========================  */

   fprintf(OutPut,"\n---------------------------------------------------------------
------------") ;
   fprintf(OutPut,"\nOptimal Performance Index =  %7.3f  When Changing Phase Combo at
t =  %3d", optpi, opttime) ;
   fprintf(OutPut,"\n---------------------------------------------------------------
------------") ;

   fprintf(OutPut,"\n\n\n                              Optimal Queue Matrix\n\n") ;
   fprintf(OutPut,"-----------------------------------------------------------------
---------") ;
   fprintf(OutPut,"\nTime  Phase1  Phase2  Phase3  Phase4  Phase5  Phase6  Phase7
Phase8  Phases\n") ;
   fprintf(OutPut,"----  ------  ------  ------  ------  ------  ------  ------  -----
-  ------\n") ;
   for(t = 0; t <= Tucson.T; ++t) {
      fprintf(OutPut,"%3d   ", t) ;
      for(p = 1; p <= 8; ++p)
        fprintf(OutPut,"%4d    ", optq[t][p]) ;
      if((opttphase[t] == opttphase[t-1]) || (t == 0))
         fprintf(OutPut,"%4d\n", opttphase[t]) ;
      else
         fprintf(OutPut,"%4d ***\n", opttphase[t]) ;
      if(t%10 == 0)
        fprintf(OutPut,"\n") ;
   }

   fprintf(OutPut,"\n") ;
   fprintf(OutPut,"***  Note that phase combination changes here.\n") ;
   fprintf(OutPut,"\n\n\n\n\n") ;


/*  =============  PRINT TOTALS FOR EACH PHASE CHANGE INDEX =============  */

   fprintf(OutPut,"\nAnalysis for Phase Combination Change at time t   E   [ %4d,
%4d]", (phasetime - detect/3), (phasetime + (2 * detect)/3)) ;
   fprintf(OutPut,"\n---------------------------------------------------------------
------------") ;
   fprintf(OutPut,"\nPerformance Index = ( %8.5f * Total Stops  ) + ( %8.5f * Total
Delay  )\n\n", stop_wt, delay_wt) ;

for (c = (phasetime - detect/3); c <= (phasetime + (2 * detect)/3); ++c) {
      if ((c - phasetime + detect/3) == 6)
        fprintf(OutPut,"\n\n\n\n\n") ;
      if ( ((c - phasetime + detect/3 + 1)%7) == 0)
        fprintf(OutPut,"\n") ;
      fprintf(OutPut,"\nPhase Results for Changing Phase Combination at Time:
%4d\n", c) ;
      fprintf(OutPut,"------------------------------------------------------------")
;
      fprintf(OutPut,"\nPhase:    1     2     3     4     5     6     7     8
Total\n") ;
      fprintf(OutPut,"Stops:") ;
      for (p = 1; p <= Tucson.P; ++p)
         fprintf(OutPut,"%5d ",stops[c][p]) ;
      fprintf(OutPut," %6d",t_stops[c]) ;

      fprintf(OutPut,"\nDelay:") ;
      for (p = 1; p <= Tucson.P; ++p)
         fprintf(OutPut,"%5d ",delay[c][p]) ;
      fprintf(OutPut," %6d\n",t_delay[c]) ;

      fprintf(OutPut,"                                    ------------------------------------
\n") ;
      fprintf(OutPut,"                                    Performance Index = %16.4f\n\n", pi[c])
;
}              /*  End of for c loop        */


}   /*  ==================== END OF MAIN PROGRAM  =======================  */


/*  ======================  CALCULATE DEPARTURES  ======================  */

int depart(int phase, int q, int fin, int t, int p, int sfrate)
{
   int fout ;                         /*  number of departures  */

   if ((p == phase/10) || (p == phase%10)) {       /*  Green signal  */
      if ((q != 0) && (q >= sfrate))
         fout = sfrate ;
      else if ((q != 0) && (q < sfrate))
         fout = q ;
      else
         fout = fin ;                      /*  Green signal  */
   }
   else
      fout = 0 ;                           /*  Red signal          */
   return(fout) ;
}


/*  =====================  CALCULATE CONVEX COMBINATION  ================  */

double linearcombo(int brakes, int waits, double stop_wt, double delay_wt)
{
   double     coco ;

   coco = stop_wt * brakes + delay_wt * waits ;
   return (coco) ;
}
```

APPENDIX B: DISPATCH 1.0 Sample Program Output

```
--------------------------------------------------------------------
   Information for Intersection Identification     #    123456789
--------------------------------------------------------------------
  Number of Phases =      8          Minimum Green Time =          2
  Clearance Time =        1          Total Number of Stages =      2
  Sequence of Stages:     37  48
  Time Horizon =         314

Arrival Data
  ----------------------------
Time   1  2  3  4  5  6  7  8
----------------------------
  0    0  0  0  0  0  0  1  0

  1    0  0  0  1  0  0  0  0
  2    0  0  0  0  0  0  0  1
  3    0  0  0  0  0  0  0  0
  4    0  0  0  0  0  0  0  1
  5    0  0  0  1  0  0  0  0
  6    0  0  0  0  0  0  0  0
  7    0  0  0  0  0  0  0  0
  8    0  0  0  0  0  0  1  0
  9    0  0  0  0  0  0  0  0
 10    0  0  0  1  0  0  0  0

 11    0  0  0  0  0  0  0  1
 12    0  0  0  0  0  0  0  0
 13    0  0  1  0  0  0  0  0
 14    0  0  0  0  0  0  0  1
 15    0  0  1  0  0  0  0  2
 16    0  0  0  0  0  0  0  1
 17    0  0  1  0  0  0  0  0
 18    0  0  0  0  0  0  0  0
 19    0  0  0  0  0  0  0  0
 20    0  0  1  0  0  0  0  1

 21    0  0  0  0  0  0  0  0
 22    0  0  0  0  0  0  0  1
 23    0  1  0  0  0  0  0  1
 24    0  0  0  0  0  0  0  0
 25    0  0  0  0  0  0  0  0
 26    0  0  0  0  0  0  0  1
 27    0  0  1  0  0  1  0  0
 28    0  0  0  1  0  1  0  1
 29    0  0  0  0  0  0  0  0
 30    0  0  0  0  0  0  0  0

 31    0  0  0  0  0  0  0  0
 32    0  0  0  0  0  0  0  0
 33    0  0  0  0  0  0  0  1
 34    0  0  0  0  0  0  0  1
 35    0  0  0  0  0  0  0  1
 36    1  0  0  0  0  0  1  0
 37    0  0  0  0  0  0  0  0
 38    0  0  0  0  0  0  0  0
 39    1  0  0  0  0  0  0  0
 40    0  0  0  0  0  0  0  0

 41    0  0  0  0  0  0  0  0
 42    0  0  0  0  0  0  0  0
 43    0  0  0  0  0  0  0  0
```

44	0	0	0	0	0	0	0	0
45	0	0	0	1	0	0	0	0
46	0	0	0	1	0	0	0	0
47	1	0	0	0	0	0	1	0
48	0	0	0	0	0	0	0	0
49	0	0	0	0	0	0	0	0
50	0	0	0	0	0	0	0	0
51	0	0	0	0	0	0	0	1
52	0	0	0	0	0	0	0	0
53	0	0	0	0	0	0	0	0
54	0	0	0	0	0	0	0	1
55	0	0	0	0	0	0	0	2
56	0	0	0	0	0	0	0	1
57	0	0	0	0	0	0	0	0
58	0	0	1	0	0	0	0	1
59	0	0	0	0	0	0	0	1
60	0	0	0	0	0	0	1	0
61	0	0	0	0	0	0	0	0
62	0	0	0	1	0	0	0	0
63	0	0	0	0	0	0	0	1
64	0	0	0	0	0	0	0	1
65	0	0	0	0	0	0	1	1
66	0	0	0	0	0	0	0	1
67	1	0	0	0	0	1	0	0
68	1	0	1	0	0	0	0	1
69	0	0	1	0	0	1	0	1
70	0	0	0	0	0	0	0	2
71	0	0	0	0	0	0	0	0
72	0	0	0	0	0	0	0	0
73	0	0	0	0	0	0	0	0
74	0	0	0	0	0	0	0	0
75	0	0	0	0	0	0	0	1
76	0	0	0	0	0	0	0	0
77	0	0	0	0	0	0	0	1
78	0	0	0	0	0	0	0	2
79	0	1	0	0	0	1	0	0
80	0	0	0	1	0	0	1	0
81	0	1	0	1	0	0	0	0
82	0	0	0	0	0	0	0	0
83	0	0	0	0	0	0	0	0
84	0	0	0	0	0	0	0	0
85	0	1	0	0	0	1	0	2
86	0	0	0	1	0	1	0	1
87	2	1	1	0	0	0	0	0
88	1	0	0	0	0	0	0	0
89	0	0	0	1	0	1	0	0
90	1	0	0	0	0	1	0	0
91	0	0	0	0	0	1	0	0
92	0	0	0	0	0	0	1	0
93	0	0	0	0	0	0	0	1
94	0	0	0	0	0	0	0	1
95	0	0	0	0	0	0	0	0
96	0	0	1	0	0	0	0	0
97	0	0	0	0	0	0	0	0
98	0	0	0	0	0	0	0	0
99	0	0	0	0	0	0	0	0
100	0	0	0	0	0	0	1	0
101	0	0	0	0	0	0	1	2

102	1	0	1	0	0	0	0	0
103	0	0	0	0	0	0	0	2
104	0	0	0	1	0	0	0	0
105	0	0	0	1	0	0	0	0
106	0	0	0	1	0	0	0	0
107	0	0	1	0	0	0	0	0
108	0	0	0	0	0	0	0	0
109	0	1	0	0	0	0	0	0
110	0	0	0	1	0	0	0	0
111	1	0	0	1	0	0	0	1
112	0	0	0	0	0	0	0	0
113	0	0	0	0	0	0	0	0
114	0	0	0	0	0	0	0	0
115	2	0	0	0	0	0	0	0
116	0	0	0	0	0	0	0	0
117	0	0	0	0	0	0	1	1
118	0	0	0	0	0	0	0	1
119	0	0	0	0	0	0	0	0
120	0	0	0	0	0	0	0	0
121	0	0	0	0	0	0	0	0
122	1	0	0	0	0	0	1	0
123	0	0	0	0	0	0	0	1
124	0	0	0	0	0	0	0	0
125	0	0	0	0	0	0	0	0
126	0	0	0	0	0	0	0	0
127	0	0	0	0	0	0	0	0
128	0	0	0	0	0	0	0	1
129	0	0	0	0	0	0	0	0
130	0	0	0	0	0	0	0	0
131	0	0	0	0	0	0	0	0
132	0	1	0	0	0	0	0	0
133	0	0	0	0	0	0	0	0
134	0	0	0	1	0	0	0	0
135	0	0	0	0	0	0	0	0
136	0	0	0	0	0	1	0	0
137	0	0	0	0	0	0	0	0
138	0	0	0	0	0	0	1	0
139	0	0	0	0	0	1	0	1
140	0	0	0	0	0	0	0	0
141	0	0	0	0	0	0	0	0
142	0	0	0	0	0	0	0	0
143	0	0	0	0	0	0	0	1
144	0	0	0	0	0	0	0	0
145	0	0	0	0	0	0	0	0
146	0	0	0	0	0	0	0	0
147	0	0	0	0	0	0	0	0
148	0	0	0	0	0	0	0	1
149	0	0	2	0	0	0	0	2
150	0	0	1	0	0	0	0	1
151	0	0	0	0	0	0	0	2
152	0	0	0	0	0	0	0	0
153	0	0	0	1	0	0	0	0
154	0	0	0	0	0	0	1	0
155	0	0	2	1	0	0	0	0
156	0	0	1	0	0	0	0	0
157	1	0	0	0	0	0	0	0
158	0	0	1	0	0	0	0	1
159	0	0	0	0	0	0	0	1
160	0	0	0	0	0	0	0	C

161	0	0	0	0	0	0	0	0
162	0	0	0	1	0	1	0	0
163	0	0	0	0	0	0	0	0
164	0	0	0	0	0	0	0	0
165	0	0	0	0	0	0	0	0
166	0	1	0	0	0	0	0	0
167	0	0	0	1	0	0	0	0
168	0	2	0	0	0	0	0	0
169	0	0	0	0	1	0	0	0
170	0	0	0	0	0	0	0	0
171	0	0	0	0	0	0	0	1
172	0	0	0	0	0	0	0	0
173	0	0	0	0	0	0	0	0
174	0	0	0	2	0	0	0	0
175	0	0	0	0	0	0	0	0
176	0	0	0	0	0	0	0	0
177	0	0	2	0	0	1	0	0
178	0	1	0	0	0	1	0	1
179	1	0	0	0	0	0	0	0
180	0	0	0	0	0	0	0	0
181	0	1	0	0	0	0	0	0
182	0	1	0	0	0	0	0	1
183	0	0	1	0	0	0	0	1
184	0	0	0	0	0	2	0	0
185	1	1	0	0	0	0	0	0
186	0	0	0	0	0	0	0	2
187	0	0	0	0	0	0	0	0
188	0	0	0	0	0	0	0	0
189	0	0	0	1	0	0	0	1
190	0	0	0	0	0	0	0	0
191	0	0	0	0	0	0	0	0
192	0	0	0	0	0	0	0	1
193	0	1	0	0	0	0	0	0
194	0	0	0	0	0	0	0	0
195	0	0	0	0	0	0	0	0
196	0	0	0	0	0	0	0	0
197	0	0	0	0	0	0	0	1
198	0	0	0	1	0	0	0	2
199	0	0	0	0	0	0	0	0
200	0	0	0	0	0	0	0	0
201	0	0	0	0	0	0	0	0
202	0	1	0	0	0	0	0	1
203	0	1	0	0	0	0	0	0
204	0	1	0	0	0	0	0	1
205	0	0	0	0	0	1	0	1
206	0	0	0	0	0	0	0	0
207	0	0	1	1	0	0	0	0
208	0	0	0	0	0	0	0	0
209	0	0	0	0	0	0	0	0
210	0	0	0	0	0	1	0	0
211	0	0	0	0	0	0	0	0
212	0	0	0	0	0	0	0	0
213	1	0	0	0	0	0	0	0
214	0	0	0	0	0	0	0	0
215	0	0	0	1	0	0	0	0
216	0	0	0	0	0	0	0	0
217	0	0	0	0	0	1	0	1
218	0	0	0	0	0	0	0	0

219	0	0	0	0	0	0	0	0
220	0	0	0	0	0	0	0	0
221	0	0	0	0	0	0	0	0
222	0	0	0	0	0	0	0	0
223	0	0	0	0	1	0	0	0
224	0	0	0	0	0	0	0	0
225	0	0	0	1	0	0	0	0
226	0	0	0	1	0	0	0	1
227	0	0	0	1	0	0	0	1
228	0	0	0	0	0	0	0	0
229	0	0	0	0	0	0	0	0
230	0	0	0	1	0	0	0	0
231	0	0	0	0	0	0	0	0
232	0	0	0	0	0	0	0	0
233	0	0	0	0	0	0	0	0
234	0	0	0	0	0	0	0	0
235	0	0	0	0	0	0	0	0
236	0	0	0	1	0	0	0	1
237	0	0	0	1	0	0	0	0
238	0	0	0	1	0	0	0	0
239	0	0	0	0	0	0	1	0
240	0	0	0	0	0	0	0	1
241	0	0	1	0	0	0	0	1
242	0	0	0	0	0	0	0	0
243	0	0	0	0	0	0	0	0
244	0	0	0	0	0	0	0	1
245	0	0	2	0	0	0	0	2
246	0	0	1	0	0	0	0	0
247	0	0	1	0	0	0	0	0
248	0	0	0	0	0	0	0	0
249	0	0	0	0	0	0	0	0
250	0	0	1	2	0	0	0	0
251	0	0	0	0	0	1	0	1
252	0	0	2	0	0	2	0	0
253	0	0	0	0	0	0	0	0
254	0	1	0	0	0	0	0	0
255	0	0	0	0	0	0	0	1
256	0	0	0	0	0	0	0	0
257	0	0	0	1	0	0	0	0
258	0	0	0	0	0	1	0	0
259	0	0	0	0	0	0	0	0
260	0	0	0	0	0	0	0	0
261	0	0	0	0	0	0	0	0
262	0	0	0	0	0	0	0	0
263	0	0	0	0	0	0	0	0
264	0	0	0	0	0	0	1	0
265	0	1	0	0	0	0	1	1
266	0	1	0	0	0	0	0	0
267	0	1	0	1	0	0	0	1
268	0	1	0	1	0	0	1	0
269	0	0	0	0	0	0	0	0
270	0	0	0	0	0	0	0	0
271	0	0	0	0	0	0	0	0
272	0	0	0	1	0	1	0	2
273	1	0	1	0	0	0	0	0
274	0	1	1	0	0	2	0	0
275	1	0	0	1	0	0	0	0
276	0	0	0	0	0	0	0	1

```
277    1  0  1  1  0  0  0  1
278    0  0  0  1  0  1  0  1
279    0  1  0  0  0  0  0  0
280    0  0  0  0  0  0  0  0

281    0  0  0  0  0  0  0  1
282    0  0  1  0  0  0  0  0
283    0  0  1  0  0  0  0  0
284    0  0  0  1  0  0  0  1
285    0  0  0  0  0  0  1  0
286    0  0  0  0  0  0  0  0
287    0  0  0  0  0  0  0  0
288    0  0  0  0  0  0  0  0
289    0  0  0  0  0  0  0  0
290    0  1  0  0  0  0  0  1

291    0  1  1  0  0  0  0  0
292    1  0  0  0  0  0  0  1
293    0  0  1  0  0  0  0  1
294    0  0  0  0  0  0  0  1
295    0  0  0  0  0  0  0  0
296    0  1  0  0  0  0  0  0
297    0  0  0  0  0  0  0  0
298    0  0  0  0  0  0  0  0
299    0  0  0  1  0  0  0  0
300    0  0  0  0  0  0  0  0

301    0  0  0  0  0  0  0  0
302    0  0  0  0  0  0  0  0
303    0  0  0  0  0  0  0  0
304    0  0  0  0  0  1  0  0
305    0  0  0  0  0  1  0  0
306    0  0  0  0  0  0  0  1
307    0  0  0  0  0  0  0  0
308    0  0  0  0  0  0  0  0
309    0  0  0  0  0  0  0  0
310    0  0  0  0  0  0  0  0

311    0  0  0  0  0  0  0  0
312    0  0  0  0  0  0  0  0
313    0  0  0  0  0  0  0  0
314   22 28 38 45  2 30 19 97

Current Saturation Flow Rate =       1
Delay from Detector to Stop Line =  10


-----------------------------------------------------------------------
Optimal Performance Index =  15631.000  When Changing Phase Combo at t =  154
-----------------------------------------------------------------------


                          Optimal Queue Matrix

----------------------------------------------------------------------
Time   Phase1   Phase2   Phase3   Phase4   Phase5   Phase6   Phase7   Phase8   Phases
----   ------   ------   ------   ------   ------   ------   ------   ------   ------
  0       0        0        0        0        0        0        0        0       37

  1       0        0        0        0        0        0        0        0       37
  2       0        0        0        0        0        0        0        0       37
  3       0        0        0        0        0        0        0        0       37
  4       0        0        0        0        0        0        0        0       37
```

5	0	0	0	0	0	0	0	0	37
6	0	0	0	0	0	0	0	0	37
7	0	0	0	0	0	0	0	0	37
8	0	0	0	0	0	0	0	0	37
9	0	0	0	0	0	0	0	0	37
10	0	0	0	0	0	0	0	0	37
11	0	0	0	0	0	0	0	0	37
12	0	0	0	1	0	0	0	0	37
13	0	0	0	1	0	0	0	1	37
14	0	0	0	1	0	0	0	1	37
15	0	0	0	1	0	0	0	2	37
16	0	0	0	2	0	0	0	2	37
17	0	0	0	2	0	0	0	2	37
18	0	0	0	2	0	0	0	2	37
19	0	0	0	2	0	0	0	2	37
20	0	0	0	2	0	0	0	2	37
21	0	0	0	3	0	0	0	2	37
22	0	0	0	3	0	0	0	3	37
23	0	0	0	3	0	0	0	3	37
24	0	0	0	3	0	0	0	3	37
25	0	0	0	3	0	0	0	4	37
26	0	0	0	3	0	0	0	6	37
27	0	0	0	3	0	0	0	7	37
28	0	0	0	3	0	0	0	7	37
29	0	0	0	3	0	0	0	7	37
30	0	0	0	3	0	0	0	7	37
31	0	0	0	3	0	0	0	8	37
32	0	0	0	3	0	0	0	8	37
33	0	0	0	3	0	0	0	9	37
34	0	1	0	3	0	0	0	10	37
35	0	1	0	3	0	0	0	10	37
36	0	1	0	3	0	0	0	10	37
37	0	1	0	3	0	0	0	11	37
38	0	1	0	3	0	1	0	11	37
39	0	1	0	4	0	2	0	12	37
40	0	1	0	4	0	2	0	12	37
41	0	1	0	4	0	2	0	12	37
42	0	1	0	4	0	2	0	12	37
43	0	1	0	4	0	2	0	12	37
44	0	1	0	4	0	2	0	13	37
45	0	1	0	4	0	2	0	14	37
46	0	1	0	4	0	2	0	15	37
47	1	1	0	4	0	2	0	15	37
48	1	1	0	4	0	2	0	15	37
49	1	1	0	4	0	2	0	15	37
50	2	1	0	4	0	2	0	15	37
51	2	1	0	4	0	2	0	15	37
52	2	1	0	4	0	2	0	15	37
53	2	1	0	4	0	2	0	15	37
54	2	1	0	4	0	2	0	15	37
55	2	1	0	5	0	2	0	15	37
56	2	1	0	6	0	2	0	15	37
57	2	1	0	6	0	2	0	15	37
58	3	1	0	6	0	2	0	15	37
59	3	1	0	6	0	2	0	15	37
60	3	1	0	6	0	2	0	15	37
61	3	1	0	6	0	2	0	15	37
62	3	1	0	6	0	2	0	16	37

63	3	1	0	6	0	2	0	16	37
64	3	1	0	6	0	2	0	16	37
65	3	1	0	6	0	2	0	17	37
66	3	1	0	6	0	2	0	19	37
67	3	1	0	6	0	2	0	20	37
68	3	1	0	6	0	2	0	20	37
69	3	1	0	6	0	2	0	21	37
70	3	1	0	6	0	2	0	22	37
71	3	1	0	6	0	2	0	22	37
72	3	1	0	6	0	2	0	22	37
73	3	1	0	7	0	2	0	22	37
74	3	1	0	7	0	2	0	23	37
75	3	1	0	7	0	2	0	24	37
76	3	1	0	7	0	2	0	25	37
77	3	1	0	7	0	2	0	26	37
78	4	1	0	7	0	3	0	26	37
79	5	1	0	7	0	3	0	27	37
80	5	1	0	7	0	4	0	28	37
81	5	1	0	7	0	4	0	30	37
82	5	1	0	7	0	4	0	30	37
83	5	1	0	7	0	4	0	30	37
84	5	1	0	7	0	4	0	30	37
85	5	1	0	7	0	4	0	30	37
86	5	1	0	7	0	4	0	31	37
87	5	1	0	7	0	4	0	31	37
88	5	1	0	7	0	4	0	32	37
89	5	1	0	7	0	4	0	34	37
90	5	2	0	7	0	5	0	34	37
91	5	2	0	8	0	5	0	34	37
92	5	3	0	9	0	5	0	34	37
93	5	3	0	9	0	5	0	34	37
94	5	3	0	9	0	5	0	34	37
95	5	3	0	9	0	5	0	34	37
96	5	4	0	9	0	6	0	36	37
97	5	4	0	10	0	7	0	37	37
98	7	5	0	10	0	7	0	37	37
99	8	5	0	10	0	7	0	37	37
100	8	5	0	11	0	8	0	37	37
101	9	5	0	11	0	9	0	37	37
102	9	5	0	11	0	10	0	37	37
103	9	5	0	11	0	10	0	37	37
104	9	5	0	11	0	10	0	38	37
105	9	5	0	11	0	10	0	39	37
106	9	5	0	11	0	10	0	39	37
107	9	5	0	11	0	10	0	39	37
108	9	5	0	11	0	10	0	39	37
109	9	5	0	11	0	10	0	39	37
110	9	5	0	11	0	10	0	39	37
111	9	5	0	11	0	10	0	39	37
112	9	5	0	11	0	10	0	41	37
113	10	5	0	11	0	10	0	41	37
114	10	5	0	11	0	10	0	43	37
115	10	5	0	12	0	10	0	43	37
116	10	5	0	13	0	10	0	43	37
117	10	5	0	14	0	10	0	43	37
118	10	5	0	14	0	10	0	43	37
119	10	5	0	14	0	10	0	43	37
120	10	6	0	14	0	10	0	43	37

121	10	6	0	15	0	10	0	43	37	
122	11	6	0	16	0	10	0	44	37	
123	11	6	0	16	0	10	0	44	37	
124	11	6	0	16	0	10	0	44	37	
125	11	6	0	16	0	10	0	44	37	
126	13	6	0	16	0	10	0	44	37	
127	13	6	0	16	0	10	0	44	37	
128	13	6	0	16	0	10	0	45	37	
129	13	6	0	16	0	10	0	46	37	
130	13	6	0	16	0	10	0	46	37	
131	13	6	0	16	0	10	0	46	37	
132	13	6	0	16	0	10	0	46	37	
133	14	6	0	16	0	10	0	46	37	
134	14	6	0	16	0	10	0	47	37	
135	14	6	0	16	0	10	0	47	37	
136	14	6	0	16	0	10	0	47	37	
137	14	6	0	16	0	10	0	47	37	
138	14	6	0	16	0	10	0	47	37	
139	14	6	0	16	0	10	0	48	37	
140	14	6	0	16	0	10	0	48	37	
141	14	6	0	16	0	10	0	48	37	
142	14	6	0	16	0	10	0	48	37	
143	14	7	0	16	0	10	0	48	37	
144	14	7	0	16	0	10	0	48	37	
145	14	7	0	17	0	10	0	48	37	
146	14	7	0	17	0	10	0	48	37	
147	14	7	0	17	0	11	0	48	37	
148	14	7	0	17	0	11	0	48	37	
149	14	7	0	17	0	11	0	48	37	
150	14	7	0	17	0	12	0	49	37	
151	14	7	0	17	0	12	0	49	37	
152	14	7	0	17	0	12	0	49	37	
153	14	7	0	17	0	12	0	49	37	
154	14	7	0	17	0	12	0	50	37	
155	14	7	0	17	0	12	0	50	48	***
156	14	7	0	16	0	12	0	49	48	
157	14	7	0	15	0	12	0	48	48	
158	14	7	0	14	0	12	0	47	48	
159	14	7	0	13	0	12	0	47	48	
160	14	7	2	12	0	12	0	48	48	
161	14	7	3	11	0	12	0	48	48	
162	14	7	3	10	0	12	0	49	48	
163	14	7	3	9	0	12	0	48	48	
164	14	7	3	9	0	12	0	47	48	
165	14	7	3	8	0	12	1	46	48	
166	14	7	5	8	0	12	1	45	48	
167	14	7	6	7	0	12	1	44	48	
168	15	7	6	6	0	12	1	43	48	
169	15	7	7	5	0	12	1	43	48	
170	15	7	7	4	0	12	1	43	48	
171	15	7	7	3	0	12	1	42	48	
172	15	7	7	2	0	12	1	41	48	
173	15	7	7	2	0	13	1	40	48	
174	15	7	7	1	0	13	1	39	48	
175	15	7	7	0	0	13	1	38	48	
176	15	7	7	0	0	13	1	37	48	
177	15	8	7	0	0	13	1	36	48	
178	15	8	7	0	0	13	1	35	48	
179	15	10	7	0	0	13	1	34	48	

180	15	10	7	0	1	13	1	33	48
181	15	10	7	0	1	13	1	32	48
182	15	10	7	0	1	13	1	32	48
183	15	10	7	0	1	13	1	31	48
184	15	10	7	0	1	13	1	30	48
185	15	10	7	0	1	13	1	29	48
186	15	10	7	0	1	13	1	28	48
187	15	10	7	0	1	13	1	27	48
188	15	10	9	0	1	14	1	26	48
189	15	11	9	0	1	15	1	26	48
190	16	11	9	0	1	15	1	25	48
191	16	11	9	0	1	15	1	24	48
192	16	12	9	0	1	15	1	23	48
193	16	13	9	0	1	15	1	23	48
194	16	13	10	0	1	15	1	23	48
195	16	13	10	0	1	17	1	22	48
196	17	14	10	0	1	17	1	21	48
197	17	14	10	0	1	17	1	22	48
198	17	14	10	0	1	17	1	21	48
199	17	14	10	0	1	17	1	20	48
200	17	14	10	0	1	17	1	20	48
201	17	14	10	0	1	17	1	19	48
202	17	14	10	0	1	17	1	18	48
203	17	14	10	0	1	17	1	18	48
204	17	15	10	0	1	17	1	17	48
205	17	15	10	0	1	17	1	16	48
206	17	15	10	0	1	17	1	15	48
207	17	15	10	0	1	17	1	14	48
208	17	15	10	0	1	17	1	14	48
209	17	15	10	0	1	17	1	15	48
210	17	15	10	0	1	17	1	14	48
211	17	15	10	0	1	17	1	13	48
212	17	15	10	0	1	17	1	12	48
213	17	16	10	0	1	17	1	12	48
214	17	17	10	0	1	17	1	11	48
215	17	18	10	0	1	17	1	11	48
216	17	18	10	0	1	18	1	11	48
217	17	18	10	0	1	18	1	10	48
218	17	18	11	0	1	18	1	9	48
219	17	18	11	0	1	18	1	8	48
220	17	18	11	0	1	18	1	7	48
221	17	18	11	0	1	19	1	6	48
222	17	18	11	0	1	19	1	5	48
223	17	18	11	0	1	19	1	4	48
224	18	18	11	0	1	19	1	3	48
225	18	18	11	0	1	19	1	2	48
226	18	18	11	0	1	19	1	1	48
227	18	18	11	0	1	19	1	0	48
228	18	18	11	0	1	20	1	0	48
229	18	18	11	0	1	20	1	0	48
230	18	18	11	0	1	20	1	0	48
231	18	18	11	0	1	20	1	0	48
232	18	18	11	0	1	20	1	0	48
233	18	18	11	0	1	20	1	0	48
234	18	18	11	0	2	20	1	0	48
235	18	18	11	0	2	20	1	0	48
236	18	18	11	0	2	20	1	0	48
237	18	18	11	0	2	20	1	0	48

238	18	18	11	0	2	20	1	0	48
239	18	18	11	0	2	20	1	0	48
240	18	18	11	0	2	20	1	0	48
241	18	18	11	0	2	20	1	0	48
242	18	18	11	0	2	20	1	0	48
243	18	18	11	0	2	20	1	0	48
244	18	18	11	0	2	20	1	0	48
245	18	18	11	0	2	20	1	0	48
246	18	18	11	0	2	20	1	0	48
247	18	18	11	0	2	20	1	0	48
248	18	18	11	0	2	20	1	0	48
249	18	18	11	0	2	20	1	0	48
250	18	18	11	0	2	20	2	0	48
251	18	18	11	0	2	20	2	0	48
252	18	18	12	0	2	20	2	0	48
253	18	18	12	0	2	20	2	0	48
254	18	18	12	0	2	20	2	0	48
255	18	18	12	0	2	20	2	0	48
256	18	18	14	0	2	20	2	0	48
257	18	18	15	0	2	20	2	0	48
258	18	18	16	0	2	20	2	0	48
259	18	18	16	0	2	20	2	0	48
260	18	18	16	0	2	20	2	0	48
261	18	18	17	0	2	20	2	0	48
262	18	18	17	0	2	21	2	0	48
263	18	18	19	0	2	23	2	0	48
264	18	18	19	0	2	23	2	0	48
265	18	19	19	0	2	23	2	0	48
266	18	19	19	0	2	23	2	0	48
267	18	19	19	0	2	23	2	0	48
268	18	19	19	0	2	23	2	0	48
269	18	19	19	0	2	24	2	0	48
270	18	19	19	0	2	24	2	0	48
271	18	19	19	0	2	24	2	0	48
272	18	19	19	0	2	24	2	0	48
273	18	19	19	0	2	24	2	0	48
274	18	19	19	0	2	24	2	0	48
275	18	19	19	0	2	24	3	0	48
276	18	20	19	0	2	24	4	0	48
277	18	21	19	0	2	24	4	0	48
278	18	22	19	0	2	24	4	0	48
279	18	23	19	0	2	24	5	0	48
280	18	23	19	0	2	24	5	0	48
281	18	23	19	0	2	24	5	0	48
282	18	23	19	0	2	24	5	0	48
283	18	23	19	0	2	25	5	0	48
284	19	23	20	0	2	25	5	0	48
285	19	24	21	0	2	27	5	0	48
286	20	24	21	0	2	27	5	0	48
287	20	24	21	0	2	27	5	0	48
288	21	24	22	0	2	27	5	0	48
289	21	24	22	0	2	28	5	0	48
290	21	25	22	0	2	28	5	0	48
291	21	25	22	0	2	28	5	0	48
292	21	25	22	0	2	28	5	0	48
293	21	25	23	0	2	28	5	0	48
294	21	25	24	0	2	28	5	0	48
295	21	25	24	0	2	28	5	0	48

```
296      21      25      24      0      2      28      6      0      48
297      21      25      24      0      2      28      6      0      48
298      21      25      24      0      2      28      6      0      48
299      21      25      24      0      2      28      6      0      48
300      21      25      24      0      2      28      6      0      48

301      21      26      24      0      2      28      6      0      48
302      21      27      25      0      2      28      6      0      48
303      22      27      25      0      2      28      6      0      48
304      22      27      26      0      2      28      6      0      48
305      22      27      26      0      2      28      6      0      48
306      22      27      26      0      2      28      6      0      48
307      22      28      26      0      2      28      6      0      48
308      22      28      26      0      2      28      6      0      48
309      22      28      26      0      2      28      6      0      48
310      22      28      26      0      2      28      6      0      48

311      22      28      26      0      2      28      6      0      48
312      22      28      26      0      2      28      6      0      48
313      22      28      26      0      2      28      6      0      48
314      22      28      26      0      2      28      6      0      48

***  Note that phase combination changes here.




Analysis for Phase Combination Change at time t   E   [  154,   163]
-----------------------------------------------------------------------------
Performance Index = (  0.25000 * Total Stops  ) + (  0.75000 * Total Delay  )

Phase Results for Changing Phase Combination at Time:     154
-------------------------------------------------------------
Phase:     1     2     3     4     5     6     7     8   Total
Stops:    22    28    26    17     2    29     6    54     184
Delay:  3685  3164  2163  1417   218  3927   355  5851   20780
                                 ---------------------------------
                                 Performance Index =     15631.0000

Phase Results for Changing Phase Combination at Time:     155
-------------------------------------------------------------
Phase:     1     2     3     4     5     6     7     8   Total
Stops:    22    28    26    17     2    29     6    54     184
Delay:  3685  3164  2163  1437   218  3927   355  5924   20873
                                 ---------------------------------
                                 Performance Index =     15700.7500

Phase Results for Changing Phase Combination at Time:     156
-------------------------------------------------------------
Phase:     1     2     3     4     5     6     7     8   Total
Stops:    22    28    26    17     2    29     6    54     184
Delay:  3685  3164  2163  1457   218  3927   355  5997   20966
                                 ---------------------------------
                                 Performance Index =     15770.5000

Phase Results for Changing Phase Combination at Time:     157
-------------------------------------------------------------
Phase:     1     2     3     4     5     6     7     8   Total
```

```
Stops:   22    28    26    17     2    29     6    54     184
Delay: 3685  3164  2163  1478   218  3927   355  6070   21060
                                -----------------------------------
                                Performance Index =       15841.0000

Phase Results for Changing Phase Combination at Time:      158
-------------------------------------------------------------
Phase:    1     2     3     4     5     6     7     8   Total
Stops:   22    28    26    17     2    29     6    55     185
Delay: 3685  3164  2163  1499   218  3927   355  6143   21154
                                -----------------------------------
                                Performance Index =       15911.7500

Phase Results for Changing Phase Combination at Time:      159
-------------------------------------------------------------
Phase:    1     2     3     4     5     6     7     8   Total
Stops:   22    28    24    17     2    29     6    56     184
Delay: 3685  3164  1851  1520   218  3927   355  6216   20936
                                -----------------------------------
                                Performance Index =       15748.0000




Phase Results for Changing Phase Combination at Time:      160
-------------------------------------------------------------
Phase:    1     2     3     4     5     6     7     8   Total
Stops:   22    28    23    17     2    29     6    57     184
Delay: 3685  3164  1696  1541   218  3927   355  6289   20875
                                -----------------------------------
                                Performance Index =       15702.2500

Phase Results for Changing Phase Combination at Time:      161
-------------------------------------------------------------
Phase:    1     2     3     4     5     6     7     8   Total
Stops:   22    28    23    17     2    29     6    58     185
Delay: 3685  3164  1696  1562   218  3927   355  6362   20969
                                -----------------------------------
                                Performance Index =       15773.0000

Phase Results for Changing Phase Combination at Time:      162
-------------------------------------------------------------
Phase:    1     2     3     4     5     6     7     8   Total
Stops:   22    28    23    17     2    29     6    58     185
Delay: 3685  3164  1696  1583   218  3927   355  6435   21063
                                -----------------------------------
                                Performance Index =       15843.5000

Phase Results for Changing Phase Combination at Time:      163
-------------------------------------------------------------
Phase:    1     2     3     4     5     6     7     8   Total
Stops:   22    28    23    19     2    29     6    58     187
Delay: 3685  3164  1696  1607   218  3927   355  6510   21162
                                -----------------------------------
                                Performance Index =       15918.2500
```

APPENDIX C: Listing of COP Program

```
/*=======================================================================  ===*/
/*=                                                                      =*/
/*=                     Coordinated Optimization of Phases (COPs)            =*/
/*=                                                                      =*/
/*====================================================================  =============*/

/*  ========================  ACKKNOWLEDGEMENTS  ============================
                              -----------------

     THE COP MODEL WAS DESIGNED BY DR. SUVRAJEET SEN, PROFESSOR
     IN THE SYSTEMS AND INDUSTRIAL ENGINEERING DEPARTMENT
     AT THE UNIVERSITY OF ARIZONA.

     THE SUCCESSFUL DEVELOPMENT OF COP HAS BEEN THE RESULT
     OF THE COOPERATIVE EFFORTS OF THE FOLLOWING INDIVIDUALS
     WHO MADE THIS PROGRAM FULLY OPERATIONAL:

       DR. S. SEN           DR. K.L. HEAD         DR. P. SANCHEZ

     SYSTEMS AND INDUSTRIAL ENGINEERING DEPARTMENT HEAD:  DR. P. MIRCHANDANI

    ===================================================================  ==========

   --- CODED   08-04-92       BY G. TOMOOKA

   --- TITLE - COP

   --- FUNCTION - THIS IS THE PRIMARY EXECUTIVE ROUTINE OF THE COP MODEL

   --- ARGUMENTS - NONE

    -------------------------  DESCRIPTION  -----------------------------------
                               -----------

     THIS PROGRAM BEGINS BY READING AN INPUT FILE CONTAINING THE FOLLOWING
     INFORMATION:

                     INTERSECTION IDENTIFICATION NUMBER
                     NUMBER OF PHASES
                     MINIMUM GREEN TIME
                     CLEARANCE TIME
                     TOTAL NUMBER OF STAGES
                     STAGES
                     TIME HORIZON
                     ARRIVAL DATA (A COLUMN FOR EACH MOVEMENT WITH VALUES
                                   REPRESENTING "PHASE" REQUESTS )

     NEXT, COP DETERMINES THE AMOUNT OF TIME TO ASSIGN TO EACH STAGE
     (PHASE COMBINATION, E.G. 26 REPRESENTS A STAGE WHICH ALLOWS MOVEMENTS
     2 AND 6 TO PROCEED UNDER GREEN TIME).  FURTHER, CCP ASSIGNS A STAGE
     AT LEAST THE MINIMUM GREEN TIME OR ASSIGNS NO TIME (I.E., SKIPS THE
     STAGE).  IN ORDER TO ASSIGN THE MINIMUM GREEN TIME, THERE MUST BE
     AT LEAST (GAMMA + DELTA) TIME UNITS REMAINING TO ALLOW VEHICLES TO TRAVEL
     THROUGH THE INTERSECTION.  NOTE THAT THE MINIMUM GREEN TIME MUST BE
     AT LEAST EQUAL TO:  (TOTAL TIME HORIZON/NUMBER OF STAGES).

     THE COP ALGORITHM INCORPORATES A DYNAMIC PROGRAMMING METHOD.  SPECIFICALLY,
     IT FINDS THE OPTIMAL SOLUTION BY UTILIZING THE BACKWARD AND FORWARD
     RECURSION TECHNIQUES.  THE OPTIMAL SOLUTION IS SENT TO AN OUTPUT FILE
     CALLED "COP.OUT".
   ALSO, THERE IS A TIMING MECHANISM INCLUDED IN COP TO DETERMINE THE TIME
   ELAPSED WHILE EXECUTING THE BACKWARD AND FORWARD RECURSIONS.  THESE TIMINGS
   ARE STORED IN AN OUTPUT FILE CALLED "COP.TIME".

  ----------------------- THIS ROUTINE CALLED BY ---------------------------
                          ---------------------

                                NONE

  ------------------------ THIS ROUTINE CALLS -----------------------------
                            -----------------

              F()            - ADDS UP STOPS ON OPPOSING MOVEMENTS
                               DURING GREEN AND STOPS ON ALL
                               MOVEMENTS DURING CLEARANCE TIME
              LAST_F()       - IS FOR THE LAST STAGE, SIMILAR TO F();
                               ADDS UP STOPS ON OPPOSING MOVEMENTS
                               DURING GREEN AND STOPS ON ALL
                               MOVEMENTS DURING CLEARANCE TIME
              G()            - CALCULATES THE AMOUNT OF GREEN TIME
                               TO ASSIGN TO A STAGE
              COP_MIN()      - DETERMINES THE MINIMUM OF TWO INTEGERS
              GETRUSAGE()    - OBTAINS START AND FINISH TIMES
                               TO DETERMINE CPU TIME OF ALGORITHM

  ------------------------ GLOSSARY OF CONSTANTS --------------------------
                           ---------------------

 JMAX                 MAXIMUM NUMBER OF STAGES
 PMAX                 MAXIMUM NUMBER OF PHASES
 TMAX                 MAXIMUM TIME HORIZON (TIME UNITS OR TIME INTERVALS)

  ------------------------ GLOSSARY OF STRUCTURES -------------------------
                           ---------------------

 INTERSECTION_T       STRUCTURE TYPE DEFINITION FOR EACH INTERSECTION
    ID        INTERSECTION IDENTIFICATION NUMBER
    P                 NUMBER OF PHASES POSSIBLE
    GAMMA     MINIMUM ALLOWABLE GREEN TIME (TIME UNITS)
    DELTA     CLEARANCE TIME FOR INTERSECTION DURING ALL RED
    J                 TOTAL NUMBER OF STAGES
    STAGE[]   ARRAY OF STAGES
    T                 TOTAL TIME HORIZON (TIME UNITS OR TIME INTERVALS)
    A[][]     MATRIX OF ARRIVAL DATA (PHASE REQUEST DURING TIME INTERVAL)

  ---------------------- GLOSSARY OF VARIABLE NAMES -----------------------
                         -------------------------

 M, I, J      INDEX FOR THE CURRENT STAGE
 V[][]        MATRIX OF VALUES FROM VALUE FUNCTION
 X[][]        MATRIX OF DECISION
 S            STATE VARIABLE REPRESENTING AMOUNT OF TIME REMAINING
 XJ           DECISION VARIABLE
 SJ_STAR      OPTIMAL STATE
 SJ_STAR_PLUS         OPTIMAL NEXT STATE
 STOPS        NUMBER OF STOPS PER STAGE
 TOTAL_STOPS          TOTAL NUMBER OF STOPS FOR THE SOLUTION
 VALUEJ       VALUE FUNCTION EVALUATION
 TEMP                 TEMPORARY VARIABLE

  ----------------------------------------------------------------------  */
```

```
#include <stdio.h>
#include <math.h>
#include <sys/types.h>
#include <sys/times.h>
#include <sys/time.h>
#include <sys/resource.h>

#define              JMAX    500                  /*  Maximum number of stages  */
#define              PMAX    200                 /*  Maximum number of phases  */
#define              TMAX    1000                /*  Maximum time horizon      */
#define              BIG_INT        999999999          /*  Large integer value to
                                                   initialize value function  */

typedef struct {
   int id ;                   /*  Identification number of intersection  */
   int P ;                    /*  Number of possible phases              */
   int Gamma ;                    /*  Minimum green time                     */
   int Delta ;                    /*  Clearance time; All phases red         */
   int J ;                    /*  Total number of stages requested       */
   int stage[JMAX] ;          /*  Stages                                 */
   int T ;                    /*  Total time horizon                     */
   int a[TMAX+2][PMAX+2] ;    /*  Arrival data                           */
   } intersection_t ;

intersection_t Tucson ;

int f() ;
int last_f() ;
int g() ;
int cop_min() ;
int getrusage() ;

int main()
{
        int     m, i, j ;                         /*  Current stage number  */
        int     v[PMAX+2][TMAX+2] ;              /*  Table of values       */
        int     X[PMAX+2][TMAX+2] ;              /*  Table of decisions    */
        int     s ;                              /*  State variable        */
         int      xj ;                                /*  Decision variable     */
         int      sj_star ;                      /*  Optimal state         */
         int      sj_star_plus ;                      /*  Optimal next state    */
        int     stops ;                               /*  Stops per stage
*/
        int     total_stops ;             /*  Total stops for all stages   */
         int      valuej ;                /*  Value function evaluation    */
         int      temp ;                       /*  Temporary storage variable   */
        FILE    *Data ;                        /*  Input data file           */
        FILE    *OutPut ;                 /*  Output file               */
        FILE    *Timings ;                /*  File containing CPU timings
                                              for dynamic program          */

        struct rusage   first_time;       /*  Get starting time            */
        struct rusage   second_time;         /*  Get finishing time            */

 /*  ===================  READ THE DATA FROM INPUT FILE  ==================  */

        getrusage(RUSAGE_SELF, &first_time) ;        /*  Get routine start time  */

        Data = fopen("cop.dat", "r") ;                       /*  Open data file  */
        OutPut = fopen("cop.out", "w") ;            /*  Open output file   */
        Timings = fopen("cop.time", "a") ;    /*  Append to timing file   */

        fprintf(OutPut,"Intersection Information:\n") ;
        fprintf(OutPut,"----------------------------------------------\n" ) ;

        fscanf(Data,"%d", &Tucson.id) ;
        fprintf(OutPut,"Intersection Identification Number:  %d\n", Tucson.id) ;

        fscanf(Data,"%d", &Tucson.P) ;
        fprintf(OutPut,"Number of Phases =  %d\n", Tucson.P) ;

        fscanf(Data,"%d", &Tucson.Gamma) ;
        fprintf(OutPut,"Minimum Green Time =  %d\n", Tucson.Gamma) ;

        fscanf(Data,"%d", &Tucson.Delta) ;
        fprintf(OutPut,"Clearance Time =  %d\n", Tucson.Delta) ;

        fscanf(Data,"%d", &Tucson.J) ;
        fprintf(OutPut,"Total Number of Stages =  %d\n", Tucson.J) ;

        fprintf(OutPut,"Sequence of Stages:   ") ;
        for (i = 1; i <= Tucson.J; ++i) {
           fscanf(Data,"%d", &Tucson.stage[i]) ;
           fprintf(OutPut,"%d ", Tucson.stage[i]) ;
           if ((i%10) == 0) fprintf(OutPut,"\n                          ") ;
        }  /*  End for i loop  */
        fprintf(OutPut,"\n") ;

        fscanf(Data,"%d", &Tucson.T) ;
        fprintf(OutPut,"Time Horizon =  %d\n\n", Tucson.T) ;

        fprintf(OutPut,"Arrival Data\n") ;
        fprintf(OutPut,"------------------------\n") ;
        fprintf(OutPut," 1  2  3  4  5  6  7  8\n") ;
        fprintf(OutPut,"------------------------\n") ;

        for (i = 0; i <= Tucson.T; ++i) {
           for (j = 1; j <= Tucson.P; ++j) {
              fscanf(Data,"%d", &Tucson.a[i][j]) ;
              fprintf(OutPut,"%2d ", Tucson.a[i][j]) ;
           }  /*  End for j loop  */
           if ((i % 10) == 0) fprintf(OutPut,"\n") ;
           fprintf(OutPut,"\n") ;
        }  /*  End for i loop  */
        fprintf(OutPut,"\n\n") ;

/*  Get routine end time  */
        getrusage(RUSAGE_SELF, &second_time) ;

        fprintf(Timings,"  %5d              %3d", Tucson.T, Tucson.Gamma) ;
        fprintf(Timings, "                    %.3f   ",
                second_time.ru_utime.tv_sec - first_time.ru_utime.tv_sec +
                 1.0e-6 * (second_time.ru_utime.tv_usec -
                first_time.ru_utime.tv_usec)) ;



/*  ====================  BEGIN THE BACKWARD RECURSION  =================  */

        getrusage(RUSAGE_SELF, &first_time);         /*  Get routine start time  */

        for(s = 0; s <= Tucson.T; ++s) {                /*  Initializing  */
           v[Tucson.J+1][s] = 0 ;
```

```
         X[Tucson.J+1][s] = 0 ;
      }  /*  End for s loop  */

      m = Tucson.J+1 ;                        /*  Backward recursion begins  */
      for(j = Tucson.J; j > 0; --j) {

         --m ;                                     /*  Change the phase  */
         for(s = 0; s <= Tucson.T; ++s) {
            v[j][s] = BIG_INT ;
            if(s < Tucson.Gamma + Tucson.Delta) {
               X[j][s] = 0 ;
                if(j==Tucson.J) v[j][s] = last_f(s,s,Tucson.stage[m],Tucson.P,
                                                Tucson.T) ;
                else v[j][s] = v[j+1][s] ;
            }
            else {
                xj = 0 ;
                X[j][s] = xj ;
                if(j==Tucson.J) valuej = last_f(s,s,Tucson.stage[m],Tucson.P,
                                               Tucson.T) ;
                else valuej = f(xj,s,Tucson.stage[m],Tucson.P,Tucson.T)  ;
               temp = valuej + v[j+1][s - g(xj,Tucson.Delta)] ;
               v[j][s] = temp ;
               for (xj = Tucson.Gamma; xj <= s - Tucson.Delta; ++xj) {
                   if(j==Tucson.J) valuej = last_f(s,s,Tucson.stage[m],
                                                  Tucson.P, Tucson.T) ;
                   else valuej = f(xj,s,Tucson.stage[m],Tucson.P,Tucson.T)  ;
                  temp = valuej + v[j+1][s - g(xj,Tucson.Delta)] ;
                   if(temp < v[j][s]) {
                      v[j][s] = temp ;
                      X[j][s] = xj ;
                   }
               }  /*  End for xj loop  */
              }  /*  End else loop  */
        }  /*  End for s loop  */
      }  /*  End for j loop  */


/*  ====================  BEGIN THE FORWARD RECURSION  ===================  */

       fprintf(OutPut,"        Coordinated Optimization of Phases (COP) Solution\n") ;
       fprintf(OutPut,"-------------------------------------------------------------
---\n") ;

        sj_star = Tucson.T ;
       stops = f(X[1][sj_star],sj_star,Tucson.stage[1],Tucson.P,Tucson.T) ;
       fprintf(OutPut," j = %3d  State =  %3d  Decision = %3d  Phases = %3d  Stops =
%3d\n", 1, sj_star, X[1][sj_star], Tucson.stage[1], stops) ;
       total_stops = stops ;

        for(j = 1; j < Tucson.J; ++j) {
           sj_star_plus = sj_star - g(X[j][sj_star],Tucson.Delta) ;
          stops = f(X[j+1][sj_star_plus],sj_star_plus,Tucson.stage[j+1],
                   Tucson.P,Tucson.T) ;
           fprintf(OutPut," j = %3d  State =  %3d  Decision = %3d  Phases = %3d  Stops
= %3d\n", j+1, sj_star_plus, X[j+1][sj_star_plus], Tucson.stage[j+1],
 stops) ;
           sj_star = sj_star_plus ;
          total_stops += stops ;
        }  /*  End for j loop  */

       fprintf(OutPut,"\n                                            Total Stops =
%3d\n\n", total_stops) ;
```

```
        getrusage(RUSAGE_SELF, &second_time) ;          /*  Get routine end time  */

        fprintf(Timings, "                %.3f\n\n",
           second_time.ru_utime.tv_sec - first_time.ru_utime.tv_sec +
            1.0e-6 * (second_time.ru_utime.tv_usec -
           first_time.ru_utime.tv_usec)) ;

        return 0;

}  /*  End of main function  */



/*  ====================  ADD UP TOTAL NUMBER OF STOPS  ==================  */

int f(int x, int s, int stage, int P, int T)
{
        int k ;
        int sum ;
         int k_max ;
        int j ;

        sum = 0 ;

         k_max = cop_min(T+1, T-s+g(x, Tucson.Delta)) ;

/*  Add all stops on the opposing movements  */

        for(j = 1; j <= P; ++j) {
           for(k = T-s; k < k_max; ++k)
               if ((j != stage/10) && (j != stage%10))
                sum += Tucson.a[k][j] ;
        }   /*  End for j loop  */

/*  Add all stops on all movements during delta  */

        for(j = 1; j <= P; ++j) {
           for(k = T-s+x+1; k < k_max; ++k)
               if ((j == stage/10) || (j == stage%10))
                        sum += Tucson.a[k][j] ;
        }   /*  End for j loop  */
        return(sum) ;
}   /*  End of int f function  */


/*  ========================  ADD UP ALL STOPS  ========================  */

int last_f(int x, int s, int stage, int P, int T)
{
        int k ;
        int sum ;
         int k_max ;
        int j ;

        sum = 0 ;

        k_max = cop_min(T+1, T-s+x) ;


        for(j = 1; j <= P; ++j) {
           for(k = T-s; k < k_max; ++k)
/*  Add stops on all opposing movements  */

              if ((j != stage/10) && (j != stage%10))
                       sum += Tucson.a[k][j] ;

/*  Add stops on all movements  */

          for(k = T-s+x+1; k < k_max; ++k)
                       sum += Tucson.a[k][j] ;

       }   /*  End for j loop  */
       return(sum) ;
}   /*  End of int last_f function  */


/*  ==================  CALCULATE AMOUNT OF GREEN TIME  ==================  */

int g(int z, int Delta)
{
        if(z == 0 ) return(0) ;
        else return(z+Delta) ;
}   /*  End of int g function  */


/*  ======================  DETERMINE MINIMUM VALUE  =====================  */

int cop_min(int x, int y)
{
       if(x < y) return(x) ;
       else return(y) ;
}   /*  End of int cop_min function  */
```

APPENDIX D: Listing of the Output of the COP Program

```
Intersection Information:
----------------------------------------------
Intersection Identification Number:  9375782
Number of Phases =  8
Minimum Green Time =  2
Clearance Time =  1
Total Number of Stages =  8
Sequence of Stages:   37 48 26 38 15 38 37 16
Time Horizon =  40

Arrival Data
-----------------------
 1  2  3  4  5  6  7  8
-----------------------
 0  0  1  0  1  0  0  0

 0  0  1  0  0  0  1  0
 0  0  0  0  0  0  1  1
 0  0  1  0  0  0  0  0
 0  0  1  0  0  0  1  0
 0  0  0  1  0  0  0  1
 0  0  0  1  0  0  0  0
 0  0  0  2  0  0  1  0
 0  0  0  1  0  0  0  0
 0  0  0  1  0  0  0  0
 0  0  0  2  0  0  0  1

 0  0  0  0  0  0  1  1
 0  0  0  0  0  0  0  1
 1  0  0  1  0  0  0  1
 0  0  0  1  0  0  0  1
 0  0  0  0  0  0  0  1
 0  0  0  0  0  0  0  1
 0  0  0  0  0  0  1  1
 0  0  0  0  0  0  0  1
 1  0  0  0  1  0  0  1
 0  0  1  0  0  0  0  1

 0  0  1  0  0  0  0  1
 1  0  0  0  0  0  0  1
 0  0  1  0  0  0  0  0
 0  0  1  0  0  0  0  0
 0  0  1  0  2  0  0  1
 1  0  1  0  1  0  0  1
 1  0  0  0  0  0  0  1
 0  0  1  0  0  0  0  0
 1  0  0  0  2  0  0  1
 1  0  0  0  0  0  0  1

 0  0  0  0  1  0  0  0
 1  0  0  0  1  0  0  0
 1  0  0  0  0  0  0  0
 0  0  0  0  1  0  0  0
 1  0  0  0  0  0  1  0
 0  0  1  0  0  0  1  0
 0  0  0  0  0  0  1  0
 0  0  0  0  0  0  1  0
 3  0  0  0  0  3  0  0
 0  0  0  0  0  0  0  0
```

```
        Coordinated Optimization of Phases (COP) Solution
-----------------------------------------------------------------
 j =   1  State =   40  Decision =   4  Phases =  37  Stops =   2
 j =   2  State =   35  Decision =   9  Phases =  48  Stops =   3
 j =   3  State =   25  Decision =   0  Phases =  26  Stops =   0
 j =   4  State =   25  Decision =  13  Phases =  38  Stops =   9
 j =   5  State =   11  Decision =   5  Phases =  15  Stops =   2
 j =   6  State =    5  Decision =   0  Phases =  38  Stops =   0
 j =   7  State =    5  Decision =   3  Phases =  37  Stops =   1
 j =   8  State =    1  Decision =   0  Phases =  16  Stops =   0

                                           Total Stops =  17
```

APPENDIX E: Subroutine CONTROL

```
        SUBROUTINE CONTROL (NODE)
C
C
C       SUBROUTINE CONTROL
C
C       PURPOSE : THIS IS AN INTERFACE ROUTINE FOR EXTERNAL CONTROL LOGIC.
C                 IT CALLS A CONTROL ROUTINE, AND SETS NETSIM DETECTOR
C                 DATA TO CALL THE PHASES INDICATED BY THE CONTROL LOGIC.
C
C       ARGUMENTS :  NODE - THE CURRENT NODE BEING UPDATED BY NETSIM
C
C       INPUT : INTEGER RETURN CODE FROM CONTROL LOGIC.
C               THE PHASE BEING CALLED IS DETERMINED BY THE BIT PATTERN:
C               EACH BIT WHICH IS ON WILL INDICATE A PHASE TO BE CALLED,
C               87654321.  IF NO BITS ARE ON (RETURN CODE = 0), THE DETECTOR
C               OUTPUT WILL NOT BE CHANGED, I.E. NETSIM'S BUILT IN CONTROL
C               LOGIC WILL BE USED.
C
C       OUTPUT : NONE
C
C       SUBROUTINES CALLED : SIGNAL LOGIC ROUTINES
C
C       CALLED BY : DETQ5
C
C       VARIABLES USED :
C             DPPOUT     ARRAY OF DETECTOR DATA BY PHASE AND PORT
C             DTEXST     ARRAY OF DETECTORS WHICH EXIST DEFINED BY PHASE AND
C                        PORT.
C                        INTERSECTION SPECIFIC.  (0,1) = (DOES NOT, DOES) EXIST
C             ITEMP      TEMPORARY VARIABLE
C             IRC        RETURN CODE FROM EXTERNAL SIGNAL CONTROL LOGIC
C             IPHASE     DO LOOP VARIABLE FOR LOOPING OVER PHASES
C             IPORT      DO LOOP VARIABLE FOR LOOPING OVER PORTS
C             INPP       PHASE AND PORT ID NUMBER (POSITION IN DPPOUT ARRAY)
C
C
C PUT VARIABLE DECLARATIONS HERE
C
        IMPLICIT INTEGER (A-Q, T, S)
C
        COMMON /SIN345/DPPOUT  (8 * 6)
        COMMON /SIN354/DTEXST  (8 * 6)
        COMMON /SIN355/LOWRAM (751)
        COMMON /SIN104/CLOCK
C
C CALL THE EXTERNAL CONTROL LOGIC (SELECT ONE, COMMENT OUT THE REST)
C
C                semi-actuated, multiple intersection algorithm
C         SYNC_TIME = LOWRAM(38) + 1
C         IRC = TRANSLATE(SYNC_TIME, NODE)
C
C                multiple intersection, fixed time algorithm
         SYNC_TIME = LOWRAM(38) + 1
         IRC = FIXED(SYNC_TIME, NODE)
C
C                simple, single intersection, fixed time algorithm
C        CALL TEST(IRC)
C
C
C WRITE PHASE CALL INFORMATION TO FILE 'phasecalls'
C
```

```
        N = NODE
        IF ((N.EQ.335).OR.(N.EQ.369).OR.(N.EQ.401).OR.(N.EQ.483)) THEN
          WRITE(35,9000) SYNC_TIME, CLOCK, NODE, IRC
 9000   FORMAT('SYNC: ',I2,'   CLOCK: ',I3,'   NODE: ',I3,'   IRC: ',I2)
        ENDIF
C
C IF THE EXTERNAL LOGIC RETURNED 0, DO NOTHING.
C
      IF (IRC .EQ. 0)  THEN
                                                        GO TO 100
      ENDIF
C
C CLEAR ALL OF THE CURRENT DETECTOR DATA
C
      DO 10 IPHASE = 1, 8
        DO 20 IPORT = 1, 6
          INPP = (IPHASE - 1) * 6 + IPORT
          DPPOUT(INPP) = 0
  20    CONTINUE
  10  CONTINUE
C
C SET DETECTOR DATA TO CALL PHASES SPECIFIED BY EXTERNAL CONTROLLER
C
      DO 30 IPHASE = 1, 8
        ITEMP = IAND(IRC, 2**(IPHASE - 1))
        IF (ITEMP .EQ. 0)                               GO TO 30
        DO 40 IPORT = 1, 6
          INPP = (IPHASE - 1) * 6 + IPORT
          DPPOUT(INPP) = 1
  40    CONTINUE
  30  CONTINUE
C
 100  CONTINUE
      RETURN
C
      END
```

APPENDIX F: **Programmers Notes**

Versions of TRAF-NETSIM used

This research was all conducted using and enhanced version of TRAF-NETSIM which includes surveillance detector capabilities. It is probably an enhancement of version 3.0, but the exact version is not known. The source code and the documentation for card type 42 were provided by FHWA. The version of GTRAF used was that provided with TRAF-NETSIM version 3.0 for MS-DOS. The latest version of TRAF-NETSIM, version 3.1, may have some differences to that used here.

Defining phase movements in the input deck

When defining the phases at the intersections to be controlled externally, is best to define each phase to represent a complete set of allowed movements, so that only one phase need be active at a time. Although the external control interface allows calls to be placed on multiple phases, like 2 and 6 in a dual ring configuration, the phases may not switch at the same time. For example, during testing, a simulation was run which placed calls to phases on both rings of the controller, 2 and 6, 4 and 8, 1 and 5, and 3 and 7. When phases 2 and 6 were terminating, sometimes phase 2 would switch 1 second before phase 6 would switch. The simple solution was to group the phase pairs together, and only call one phase at a time.

Interfacing C and FORTRAN code

External signal logic does not have to be developed in FORTRAN. Other languages can be linked to the FORTRAN simulation. The sample fixed and semi-actuated controllers were developed entirely in C. See those files fixed.c and translate.c for examples of how to write C functions that can be used by a FORTRAN program. Following are descriptions of a few of the techniques.

- C functions which are called directly by a FORTRAN program should have an underscore appended to the function name, i.e. void foo_(). This underscore should only appear in the C code, not in the FORTRAN code.

- FORTRAN common blocks can be accessed in C as structures. See translate.c for an example, where common block SIN368 is accessed.
- Variables passed from FORTRAN to C as parameters are passed by variable. In the C program they should be treated as pointers to whatever variable type they represented in the FORTRAN code.
- Variables returned by a C function are returned by value, not as pointers. For example, to return an integer from a C function, simply return an integer value, not a pointer.
- Don't forget that FORTRAN arrays indices start with 1 and C arrays begin with 0. There is no data lost in translation, just be sure to index the array properly for each given language.

Compiling, linking and running under UNIX

The following describes how to build and run the TRAF-NETSIM executable under UNIX. Assuming that all files are contained in the same subdirectory, compile all files to object form using 'f77 -Nx600 -c *.o' for the FORTRAN files and 'cc -c *.c' for any C files. Then, to link everything into an executable file called netsim, use 'f77 -o netsim *.o'. To run a simulation, then, use IO redirection, such as 'netsim < input.trf > outfile'.

Viewing graphics files generated under UNIX

If the simulation was run under UNIX, it is possible to transfer the graphics files to a MS-DOS computer for display using GTRAF. The files can be transferred using Kermit, but it is much faster to transfer them using a floppy diskette. Note that some of the following discussion will apply specifically to the computers in the University of Arizona RHODES research lab.

On catalina, a collection of utilities called Mtools has been installed which allows MS-DOS format floppy diskettes to be read and written directly in the SPARCstation floppy drive. There are manpages which describe the utilities. To process the graphics, the files that are needed are the .trf input file and the graphics files produced by the simulator. For example, if you run a simulation titled 'tucson' then the command 'mcopy tucson* a:' will copy all the necessary files to a floppy disk in the drive. Type 'eject' to eject the disk. Note that the files can get very large, so it may be necessary to limit the simulation run time and/or compress the files before transferring them to disk. Once the files are copied, take the disk to the PC. The files need to be copied into the directory c:\tsis\traf\io and renamed as described in the GTRAF manual. An easy way to do this is to set up a batch file to do the copying and renaming. See TUCSON.BAT in that directory for an example. You can simply replace the word TUCSON is that batch file with whatever name your simulation used. After the data is copied and renamed, then you need to start TSIS and run the CONVERT utility to convert the data, and the CASELOG utility to ad the name of your data to the list of available cases. The graphics files can then be viewed as normal with ANETG or SNETG.

A few points to remember when using the graphics programs are:

1) They will not display the surveillance detector information.
2) They have limits on the number of detectors and vehicles allowed on the network. If those limits are exceeded, then the graphics programs will not work.
3) There may be some discrepancy between the signal switching times displayed under ANETG and SNETG. SNETG appears to be more accurate. The data in the output file 'signals' can be used to verify switching times. Remember that signals are updated at the end of the time step. If a call is placed on a new phase at time N, it can not take effect until time N+1. Remember also that there is usually a 1 or 2 second delay from the time that the new call is received until the old phase begins to terminate.

Obtaining presence information from surveillance detectors

Surveillance detectors can be defined to be of the presence type, as well as the passage type. The subroutine POLL in code block surv.f is responsible for producing the presence counts at surveillance detectors. It is called at the end of each time step by subroutine CLNUP, found in code block nets.f. According to the documentation received from FHWA (the memos written by H. Chen), the presence counts are supposed to be packed into variable DTMOD. It appears that this feature may not be implemented correctly, since the count values printed to the file 'data' are always zero for presence detectors. This is not certain, however, and the presence detection capabilities were not investigated very thoroughly. This information is provided just to give a starting point to work from, so that in the future, presence information can be provided to external signal logic.

APPENDIX G: Subroutines SENSOR and OUTDATA

```
      SUBROUTINE SENSOR (IDIST, IDT, VLENTH, IFPOSB, ILEAD, ISPD, IV)
C
C
C --- CODED   05-05-86 BY A. HALATI
C --- RECODED 11-29-88 BY H. CHEN TO FIX THE DETECTION LOGIC
C --- RECODED  8-12-89 BY H. CHEN TO REMOVE REDUNDENT LOGIC TESTS
C --- REVISED 11-20-90 BY H. CHEN TO ADD SURVEILLANCE DETECTOR LOGIC
C
C --- TITLE - REGISTER DETECTOR ACTUATIONS MODULE - 3232.4433.1
C
C --- FUNCTION - THIS ROUTINE REGISTERS A DETECTOR ACTUATION AND
C                DETERMINES THE DETECTOR PULSE WIDTH DEPENDING ON THE
C                DETECTOR TYPE
C
C --- ARGUMENTS - IDIST  : DISTANCE TRAVELLED IN CURRENT TIME STEP
C                 IDT    : DETECTOR ID NUMBER
C                 VLENTH : VEHICLE LENGTH
C                 IFPOSB : DISTANCE BETWEEN FRONT BUMPER AND UPSTREAM
C                          NODE AT THE BEGINNING OF THE TIME STEP
C                 ILEAD  : DISTANCE BETWEEN DETECTORS LEADING EDGE AND
C                          UPSTREAM NODE
C                 ISPD  : VEHICLE SPEED AT END OF TIME STEP, INPUT
CDT   ADDED THE FOLLOWING ARGUMENT
C                 IV    : VEHICLE ID NUMBER
C--
C
C -------------------  DESCRIPTION  ---------------------------------
C                      -----------
C
C     THIS ROUTINE GENERATES VEHICLE ACTUATIONS. THE DETECTOR PULSE
C     WIDTH IS GENERATED IN ACCORDANCE WITH THE DETECTOR TYPE. FOR
C     A PULSE DETECTOR A PULSE WIDTH OF 3 TENTH-OF-A-SECOND IS
C     GENERATED.
C
C -----------------  THIS ROUTINE CALLED BY  ------------------------
C                    ----------------------
C
C                 DETECT - MODULE - 3232.4433
C
C -------------------  THIS ROUTINE CALLS  --------------------------
C                      ------------------
C
C                            NONE
C
C -----------------  GLOSSARY OF VARIABLE NAMES  --------------------
C                    --------------------------
C
C     DTLEN    DETECTOR LENGTH
C     DTMOD    DETECTOR TYPE. FLAG (1/0) IF (PASSAGE, PRESENCE)
C     IBTDET   DETECTION BEGIN TIME IN 0.1 SECOND INCREMENTS
C     IDTIME   DETECTOR ARRAY. BEGINNING TIME OF ACTUATION
C     INC      DO LOOP INDEX FOR TENTH-OF-A-SECOND INCREMENTS
C     LASTD    DETECTOR ARRAY. END TIME OF ACTUATION
C     LASTDT   DETECTION END TIME IN 0.1 SECOND INCREMENTS
C     IDELTA   DISTANCE TRAVELLED IN ONE TENTH-OF-A-SECOND
C     IL       DISTANCE TO BE TRAVELLED BEFORE DETECTION BEGINS OR ENDS
C     IPOS     VEHICLE POSITION AT THE END OF EACH TENTH-OF-A-SECOND
C     WACT     FLAG (.T., .F.) IF VEHICLE (WAS, WAS NOT) DETECTED
C     WDET     FLAG (.T., .F.) IF AN ACTUATION IS REGISTERED
C
C -------------------------------------------------------------------
C
```

```
      IMPLICIT INTEGER (A-B, D-Q, S-V, X), REAL (R,Z),LOGICAL (W,Y)
C
CDT            THESE VARIABLES WERE ADDED TO THE COMMON LIST
      COMMON /SIN104/ CLOCK
      COMMON /SIN368/ ICOUNTS (50)
C--
      COMMON /SIN313/ DTLEN (1)
      COMMON /SIN341/ IDTIME(1)
      COMMON /SIN342/ LASTD (1)
      COMMON /SIN340/ WDET  (1)
      COMMON /SIN314/ DTMOD (1)
C
C              RETURN IF VEHICLE HAS NOT YET CROSSED THE LEADING
C              EDGE AND IS NOT IN MOTION.
C
      IF (IFPOSB.LT.ILEAD .AND. IDIST.LE.0)             RETURN
C
C              SET TIME OF ACTUATION TO 1 AND COMPUTE THE TIME
C              WHEN THE VEHICLE LEAVES THE SENSING AREA.
C
      IDELTA = IDIST
      IPOS = 0
      LASTDT  = 0
      IBTDET = 0
C
CDT            NOTE: THE LOGIC WHICH DETERMINES BEGIN AND END ACTUATION
C              TIMES HAS BEEN CONVERTED FROM REAL TO INTEGER VARIABLES.
C              THIS ELIMINATES MACHINE ERROR IN THE REAL ARITHMATIC AND WILL
C              ALLOW 1/10 FT RESOLUTION.  VARIABLES RPOS, RDELTA, AND RL ARE
C              NOW IPOS, IDELTA, AND IL.  THEY NOW REPRESENT FT*10 INSTEAD
C--            OF WHOLE FT.
CDT            SET A DETECTION "MODE" TO REPRESENT WHETHER A DETECTION
C              WAS A BEGIN, END, OR CONTINUATION OF AN ACTUATION.  THIS
C              IS USEFUL FOR MONITORING PRESENCE DETECTORS.
C              INITIALIZE MODE TO BE 0 = CONTINUING DETECTION THROUGHOUT
C              THIS TIME STEP
      MODE = 0
C--
C              CHECK IF VEHICLE IS ALREADY ON DETECTOR
C
      IF (ILEAD - IFPOSB .LE. 0) THEN
          IBTDET = 1
          IL = ((ILEAD - IFPOSB + VLENTH) * 10) + DTLEN(IDT)
C
      ELSE
C
C              VEHICLE HAS NOT YET REACHED THE DETECTOR. COMPUTE
C              THE INCREMENT OF TIME IN TENTH-OF-A-SECOND FROM THE
C              BEGINNING OF THE CURRENT TIME STEP WHEN THE VEHICLE
C              FRONT BUMPER CROSSES THE DETECTOR LEADING EDGE.
C
          IL = (ILEAD - IFPOSB) * 10
          DO 30 INC = 1, 10
          IPOS = IPOS + IDELTA
   30     IF (IPOS .GE. IL)                               GOTO 40
   40     CONTINUE
C
CDT            SET ACTUATION MODE = 1, INDICATING THAT THIS IS A NEW
C              DETECTION STARTING DURING THIS TIME STEP
          MODE = 1
C--
          IBTDET = MIN0 (10, INC)
CDT            DECREMENT IPOS TO PREVENT IT FROM BEING INCREMENTED TWICE
C              FOR THE SAME TIME INCREMENT.
```

```
          IPOS = IPOS - IDELTA
C--
          IL = IL + DTLEN(IDT) + (VLENTH*10)
      ENDIF
C
C               COMPUTE THE INCREMENT OF TIME IN TENTH-OF-A-SECOND
C               FROM THE BEGINNING OF TIME STEP THAT THE REAR BUMPER
C               CROSSES THE TRAILING EDGE OF THE DETECTOR.
C
      DO 50 INC = IBTDET, 10
      IPOS = IPOS + IDELTA
   50 IF (IPOS .GT. IL)                               GOTO 60
   60 LASTDT = MINO (10, INC)

C
CDT             SET MODE TO INDICATE THAT THIS IS A DETECTION WHICH
C               ENDS DURING THIS TIME STEP.  SET MODE = 2 IF THE DETECTION
C               BEGAN IN AN EARLIER TIME STEP AND ENDS HERE.  SET MODE = 3
C               IF THE DETECTION BOTH BEGAN AND ENDED WITHIN THIS SAME TIME
C               STEP.
C
       IF (IPOS .GT. IL) THEN
           IF (MODE.EQ.0) THEN
                MODE = 2
           ELSE
                MODE = 3
           ENDIF
       ENDIF
C--
C
C               CHECK DETECTOR MODE OF OPERATION. IF A PASSAGE
C               DETECTOR THEN REGISTER AN ACTUATION ONLY IF THE FRONT
C               BUMPER IS UPSTREAM OF THE DETECTOR LEADING EDGE AT
C               THE BEGINNING OF THE TIME STEP, AND GENERATE A PULSE
C               WIDTH OF 3 TENTH-OF-A-SECOND.
CHB             STORE VEHICLE SPEED, AND INCREMENT ACTUATION COUNTER.
C
      WACT = .FALSE.
      IF (MOD(DTMOD(IDT), 2**3).EQ.1) THEN
         IF (ILEAD .GT. IFPOSB) THEN
            WACT = .TRUE.
            LASTDT = IBTDET + 2
C
CDT                 SET THE ACTUATION MODE = 3 FOR PASSAGE DETECTORS.  NOTE THAT
C                   THE .3 SEC PULSE WIDTH MAY ACTUATLLY CARRY OVER INTO THE
C                   NEXT TIME STEP.  THE ACTUATION, HOWEVER, IS ONLY REGISTERED
C                   DURING THE FIRST TIME STEP IN WHICH IT OCCURS.
            MODE = 3
C--
            DTMOD(IDT) = DTMOD(IDT) + (ISPD - MOD(DTMOD(IDT)/2**3,2**7))
     1                   * 2**3 + 2**10
         ENDIF
      ELSE
         WACT = .TRUE.
      ENDIF
C
CDT             NOTE: VEHICLES COUNTS FOR PRESENCE DETECTORS ARE PROCESSED
C--             IN ROUTINE SURV.F.  (COUNTS MAY NOT BE FUNCTIONAL)
C
C
CDT ---------------------------------------------------------------
      IF (WACT) THEN
C               WRITE DETECTOR DATA TO AN OUTPUT FILE 'data'.  THE FILE
C               IS OPENED IN MODULE NETXEC.F
```

```
C
      START = IBTDET
      END = LASTDT
C             STATION NUMBER IS ONLY APPLICABLE TO SURVEILLANCE
C             DETECTORS
      STATION = MOD(DTMOD(IDT)/2**23, 2**8)
C             DETECTOR TYPE 1 = PASSAGE, 0 = PRESSENCE
      TYPE = MOD(DTMOD(IDT), 2**3)
C             NUM IS THE VEHICLE COUNT FOR PASSAGE DETECTORS
      NUM = MOD(DTMOD(IDT)/2**10, 2**13)
      WRITE(20,1000) IDT,STATION,MODE,TYPE,IV,CLOCK,START,END,NUM
 1000  FORMAT (10I7)
C
C             INCREMENT DETECTION COUNTER FOR SURVEILANCE
C             DETECTORS WITH STATION NUMBERS
      IF (STATION .NE. 0) THEN
         ICOUNTS(STATION) = ICOUNTS(STATION) + 1
      ENDIF
C
C             WRITE OUT EXTRA DATA FOR DEBUGGING PURPOSES
C       WRITE(20,2000) DTLEN(IDT),VLENTH,IDIST,IFPOSB,ILEAD,IL
C 2000  FORMAT (6I6)
    ENDIF
C------------------------------------------------------------
C
C
C            CHECK IF OTHER ACTUATION WAS REGISTERED BY THIS
C            DETECTOR IN THIS TIME STEP. IF IT WAS THEN ADJUST
C            THE BEGINNING DETECTION TIME AND END DETECTION TIME
C
C
      IF (WACT) THEN
         WDET(IDT) = .TRUE.
         IF (IDTIME(IDT) .EQ. 0) THEN
            IDTIME(IDT) = IBTDET
         ELSE
            IDTIME(IDT) = MIN0 (IDTIME(IDT), IBTDET)
         ENDIF
         LASTD(IDT) = MAX0 (LASTD(IDT), LASTDT)
      ENDIF
C
      RETURN
      END
```

```
       SUBROUTINE OUTDATA
C
C      PURPOSE: THIS ROUTINE IS CALLED ONCE PER TIME STEP.  IT OUTPUTS
C               NETSIM DATA TO FILES, AND RESETS SURVEILLANCE DETECTOR
C               COUNTS.
C               NOTE THAT THE FILES ARE OPENED IN MODULE NETXEC.
C
C      CALLED BY: NETSIM
C
C      SUBROUTINES CALLED:  NONE
C
      IMPLICIT INTEGER (A-Z)
C
      COMMON /SIN368/ ICOUNTS (50)
      COMMON /SIN074/ SIGT (1)
      COMMON /SIN390/ XSIGT (1)
      COMMON /SIN073/ NACT  (1)
      COMMON /SIN115/ TTLND
      COMMON /SIN075/ NMAP (1)
      COMMON /SIN355/LOWRAM (751)
      COMMON /SIN104/ CLOCK
C
C                   THIS CODE WRITES THE SURVEILANCE DETECTOR COUNTS TO
C                   THE OUTPUT FILE "counts"
C
        WRITE(30,2000) CLOCK, ICOUNTS(1),ICOUNTS(2),ICOUNTS(3),
     1  ICOUNTS(4),ICOUNTS(5),ICOUNTS(6),ICOUNTS(7),ICOUNTS(8),
     2  ICOUNTS(9),ICOUNTS(10),ICOUNTS(11),ICOUNTS(12),ICOUNTS(13),
     3  ICOUNTS(14),ICOUNTS(15),ICOUNTS(16),ICOUNTS(17),ICOUNTS(18),
     4  ICOUNTS(19),ICOUNTS(20),ICOUNTS(21),ICOUNTS(22),ICOUNTS(23),
     5  ICOUNTS(24),ICOUNTS(25),ICOUNTS(26),ICOUNTS(27),ICOUNTS(28)
C     6  ICOUNTS(29),ICOUNTS(30),ICOUNTS(31),ICOUNTS(32),ICOUNTS(33),
C     7  ICOUNTS(34)
 2000    FORMAT(I5, 28I2)
C
C                   RESET THE COUNTERS TO ZERO
       DO DET = 1, 50
          ICOUNTS(DET) = 0
       ENDDO
C
C                   WRITE OUT THE SIGNAL STATES TO FILE "signals"
C
       TEMPCLK = CLOCK + 1
       SYNC = MOD( (LOWRAM(38) + 1), 90) + 1
       WRITE(40,2100) TEMPCLK, SYNC
 2100  FORMAT('TIME: ',I4,'   SYNC: ',I4 )
       DO NOD = 1, TTLND
C          IF (NMAP(NOD) .EQ. 401) THEN
             IF (NACT(NOD) .GT. 0) THEN
                WRITE(40,2200) NMAP(NOD),MOD(XSIGT(NACT(NOD)),2**4),
     1          MOD(XSIGT(NACT(NOD)) / 2**12, 2**4)
             ELSE
                WRITE(40,2300) NMAP(NOD),MOD(SIGT(NOD),2**4)
             ENDIF
C          ENDIF
       ENDDO
 2200  FORMAT(I10,I6,I4)
 2300  FORMAT(I10,I6)
C
       RETURN
       END
```

APPENDIX H: Complete Listing of Changes to TRAF-NETSIM

The following descriptions list all changes that have been made to the original TRAF-NETSIM source code. Note that in the source code, all changes are denoted by comments with CDT being the begin comment, and C-- being the ending comment. Searching for CDT in the code will quickly locate the modifications.

Code Block: detect.f
Routine: SUBROUTINE DETECT
Description of changes:
1) A modification was made to allow a vehicle to actuate multiple detectors during a single time step. The change fixed problems which occurred with overlapping and adjacent detectors. See (Tarico, 1992) for further details.
2) The argument IV, which is the vehicle ID number, was added to the parameters passed to subroutine SENSOR.

Code Block: detect.f
Routine: SUBROUTINE SENSOR
Description of changes:
This is the only subroutine which received extensive modifications.
1) The argument IV was added to the parameter list.
2) Common blocks SIN104 and SIN368 were added to the common list.
3) Vehicle position parameters were converted from real number to integer variables. See (Tarico, 1992) for further details on this change.
4) Code was added to determine an actuation 'mode' which helps in interpretation of collected detector data. The mode is interpreted as follows:

mode 1 =>The actuation was first registered (began) during the current time step.

mode 2 => The actuation ended during the current time step.

mode 3 => The actuation both began and ended during the current time step.

mode 0 => The actuation continues throughout this time step. It began is some previous time step, and will end in some later time step.

5) A block of code was added which outputs detection information to the file 'data'. This block is also where the surveillance detection counters are incremented when appropriate.

Code Block: nets.f
Routine: SUBROUTINE NETSIM
Description of changes:
A call was added to a new subroutine OUTDATA.

Code Block: nets.f
Routine: SUBROUTINE UPACT
Description of changes:
The argument IN, which is the current node number, was added to the parameter list in the call to subroutine DETQ5.

Code Block: nets.f
Routine: SUBROUTINE DETQ5
Description of changes:
1) The argument IN was added to the parameter list.
2) Common block SIN075 was included.
3) A call was added to the new subroutine CONTROL, which is the external signal logic interface routine.

Code Block: netxec.f
Routine: PROGRAM TRAF
Description of changes:
1) Common block SIN368 was included. This is the block which contains the detector counts.
2) A block of code was added which open files for output data. Within this block of code, the surveillance detector counts are initialized to zero also.

Code Block: netxec.f
Routine: BLOCK DATA SNET3
Description of changes:
The common block SIN368 is defined here.

Code Block: outdata.f
Routine: SUBROUTINE OUTDATA
Description of new subroutine:

The main purposes of this subroutine are to output and reset the surveillance detector counts, and to output the signal states. It is called once per time step.

Code Block: control.f

Routine: SUBROUTINE CONTROL

Description of new subroutine:

This subroutine serves as the interface between TRAF-NETSIM and any external signal logic. To completely disable external signal logic, remove the call to this routine in subroutine DETQ5. The functions performed by this subroutine to call any desired external signal and to set the TRAF-NETSIM detector data to call the phase(s) indicated by the external logic.

APPENDIX I: Descriptions of New TRAF-NETSIM Output Data

Some of the changes made to TRAF-NETSIM allow descriptive data from a simulation to be stored in output files. Four such files are created. They may be useful for debugging, or for providing data for other experiments. Each of the files is described below.

data

This file contains detector data describing every actuation that was registered during the simulation. Data is output every time an actuation is registered, regardless of the detector type. Modify subroutine DETECT to change the included data. Following is a sample:

```
ID    STATION   MODE   TYPE     VID    CLOCK   BEGIN    END     COUNT
27       0        3      1       6       7       6       8        1
32       0        3      1       5       7       5       7        1
36       0        3      1       4       7       8      10        1
11       0        3      1       2       7       5       7        1
97       0        3      1      10       7       1       3        1
90       0        3      1       9       7       6       8        1
38       0        3      1       7       9      10      12        1
95       0        3      1      30       9      10      12        1
34       0        3      1      29      10       2       4        1
46       0        3      1       8      10      10      12        1
```

ID is the TRAF-NETSIM internal detector number.
STATION is the station number assigned to surveillance detectors on card type 42, or zero for other types of detectors.
MODE represents the detection mode, and is provided to assist in interpreting this data. It is interpreted as follows:

mode 1 =>The actuation was first registered (began) during the current time step.
mode 2 => The actuation ended during the current time step.
mode 3 => The actuation both began and ended during the current time step.
mode 0 => The actuation continues throughout this time step. It began is some previous time step, and will end in some later time step.

TYPE indicates the detector type. 0 indicates a presence detector, and 1 indicates a passage detector.

VID is the TRAF-NETSIM internal vehicle ID number of the vehicle which caused the actuation.
CLOCK is the time step during which the actuation was registered. Note that this value is reset to zero at the beginning of the simulation period.
BEGIN is the 1/10 second interval at which the actuation began.
END is the 1/10 second interval at which the actuation ended.-
COUNT is the cumulative count of actuations registered at that particular detector if it is a passage detector, or zero if it is a presence detector.

counts

This file contains the counts collected at the surveillance detectors. The total counts are output at the end of each time step. These counts are the same as those provided to the external signal logic, if that is used. Modify subroutine OUTDATA to change the included data. Following is a sample:

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
2 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 1 0 0 0 0
3 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 0 0
5 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0
```

The first column is the time step during which the counts were collected. Each of the remaining contains the counts at one of the surveillance detectors. Column 2 corresponds to the surveillance detector with station number 1, etc. Note that only detectors with station numbers are counted.

signals

This file contains the signal states at each time step. Any intersections can be included. Modify routine OUTDATA to change which intersections are included. Following is a sample:

```
1
    275     4    8
```

```
        335     4   8
        337     4   8
        369     4   8
        334     4   8
        403     0   0
        483     2   6
        399     2   6
        484     2   6
        538     2   6
        481     2   6
    2
        275     4   8
        335     4   8
        337     4   8
        369     4   8
        334     4   8
        403     0   0
        483     2   6
        399     2   6
        484     2   6
        538     2   6
        481     2   6
```

The 1 and 2 located to the left represent the time steps during which the signal states were active. In the other data, the first number is a node number as specified in the .trf input file. The next two numbers represent the phases active on ring 1 and ring 2 of the controller at that intersection. Intersections which do not use a dual ring controller will only show one phase value. A phase value of 0 indicates that the signal is in a all-red state. Note that phases are still considered to be active during the amber period.

phasecalls

This file contains the phases which were called by the external control logic. More precisely, is shows the integer return code which is returned from the external signal logic to subroutine CONTROL. Modify subroutine CONTROL to change the data in this file. Following is a sample:

```
SYNC:  1    CLOCK:    0    NODE: 335    IRC:   1
SYNC:  1    CLOCK:    0    NODE: 369    IRC:   1
SYNC:  1    CLOCK:    0    NODE: 401    IRC:   1
SYNC:  1    CLOCK:    0    NODE: 483    IRC:   1
SYNC:  2    CLOCK:    1    NODE: 335    IRC:   1
SYNC:  2    CLOCK:    1    NODE: 369    IRC:   1
SYNC:  2    CLOCK:    1    NODE: 401    IRC:   1
SYNC:  2    CLOCK:    1    NODE: 483    IRC:   1
```

SYNC is the value of the network-wide synchronization clock. This value will not be available unless some intersection in the network is defined to have a semi-actuated signal controller in the .trf input file.
CLOCK is the elapsed time in seconds in either the initialization or simulation periods. Note that is is reset to zero at the beginning of the simulation period.
NODE is the node at which the signal state was being updated.
IRC is the integer return code that was returned by the external signal logic.

APPENDIX J: External Fixed-Time Signal Control Logic

```
/*=========================================================================*/
/*=                                                                       =*/
/*=                  Fixed Timing Plan Implementation                     =*/
/*=                                                                       =*/
/*=========================================================================*/
/*=                                                                       =*/
/*=  Signal Switching Using Fixed Timing Plans                            =*/
/*=                                                                       =*/
/*=    City:              Tucson, Arizona                                 =*/
/*=    Intersections:     Speedway Blvd. & Campbell Ave.                  =*/
/*=                       Campbell Ave.  & University Blvd.               =*/
/*=                       Sixth St.      & Campbell Ave.                  =*/
/*=                       Broadway Blvd. & Campbell Ave.                  =*/
/*=                                                                       =*/
/*=  Concept Developed By:      Dr. K. Larry Head                         =*/
/*=  Programmed By:             Greg Tomooka                              =*/
/*=  Completed:                 September 1,  1992                        =*/
/*=  Revised:                   September 14, 1992                        =*/
/*=  revised:                       9-17-92 by Doug Tarico                =*/
/*=  Programming Assisted By:   Dr. K. Larry Head                         =*/
/*=                             Doug Tarico                               =*/
/*=                                                                       =*/
/*=  Purpose of Program:                                                  =*/
/*=                                                                       =*/
/*=  This program will determine if the signal at an intersection         =*/
/*=  should be changed to the next phase according to a synchronized      =*/
/*=  clock.  If a phase is skipped (i.e., no left turn phase), then       =*/
/*=  the duration for that phase is set to zero in the timing plan.       =*/
/*=                                                                       =*/
/*=========================================================================*/

/*=========================================================================*/
/*=  Program Specifics:                                                   =*/
/*=                                                                       =*/
/*=  INPUTS:                                                              =*/
/*=                                                                       =*/
/*=  The timing plans are read from file 'fixed.plans'                    =*/
/*=  in the following format:                                             =*/
/*=                                                                       =*/
/*=      Total sets of timing plans (i.e., number of intersections)       =*/
/*=      Cycle length                                                     =*/
/*=     Intersection #i ID number, offset time, phase                     =*/
/*=        durations of each phase (in proper sequence)                   =*/
/*=                                                                       =*/
/*=  where i = 1 to total intersections and a proper sequence             =*/
/*=  of phases is:                                                        =*/
/*=                 MST  MSL  SST  SSL                                    =*/
/*=  where                                                                =*/
/*=      MST = Main Street Thru traffic with permissive left turns        =*/
/*=      MSL = Main Street Left turns                                     =*/
/*=      SST = Side Street Thru traffic with permissive left turn         =*/
/*=      SSL = Side Street Left turns                                     =*/
/*=                                                                       =*/
/*=  Also, the intersection requested and the current time will           =*/
/*=  be read from the keyboard in the following format:                   =*/
/*=                                                                       =*/
/*=      Intersection ID number                                           =*/
/*=      Current time                                                     =*/
/*=========================================================================*/
/*=  OUTPUTS:                                                             =*/
/*=                                                                       =*/
/*=     Phase which signal remain in or be set to.                        =*/
/*=                                                                  =*/
/*=   Assumptions:                                                   =*/
/*=      1)  All phases allow permissive right turns                       =*/
/*=     2)  All times and phase durations have units of SECONDS          =*/
/*=      3)  Phase duration is zero for all non-existent phases (i.e.,   =*/
/*=          phases that are essentially skipped)                    =*/
/*=                                                                  =*/
/*============================================================== ========*/

#include <stdio.h>

#define          MAXNODES    100  /*       maximum number of nodes                */
#define          MINLEFT     2    /*       minimum left turn phase length      */
#define          MAXLEFT     35   /*       maximum left turn phase length      */
#define          MINTHRU     15   /*       minimum thru phase length           */
#define          MAXTHRU     60   /*       maximum thru phase length           */


int fixed_(t, r)
     int *t, *r;
{
        int i ;                                /*  current node index            */
        int no_nodes ;                       /*  total number of nodes
                                          in timing plans file          */
    static int cycle ;                         /*  network-wide cycle length       */
        int node_id[MAXNODES] ;               /*  current node number          */
    static int offset[MAXNODES] ;              /*  node offset value                   */
        int mst[MAXNODES] ;            /*  Main Street Thru duration       */
        int msl[MAXNODES] ;            /*  Main Street Left duration       */
        int sst[MAXNODES] ;            /*  Side Street Thru duration       */
        int ssl[MAXNODES] ;            /*  Side Street Left duration       */

        int mstb[MAXNODES] ;           /*  Main Street Thru start time     */
        int mste[MAXNODES] ;                /*  Main Street Thru end time      */
        int mslb[MAXNODES] ;           /*  Main Street Left start time     */
        int msle[MAXNODES] ;           /*  Main Street Left end time       */
        int sstb[MAXNODES] ;           /*  Side Street Thru start time     */
        int sste[MAXNODES] ;           /*  Side Street Thru end time       */
        int sslb[MAXNODES] ;           /*  Side Street Left start time     */
        int ssle[MAXNODES] ;           /*  Side Street Left end time       */

   static int  tr_mstb[MAXNODES] ;  /*  Main Street Thru start time     */
   static int  tr_mste[MAXNODES] ;  /*  Main Street Thru end time       */
   static int  tr_mslb[MAXNODES] ;  /*  Main Street Left start time     */
   static int  tr_msle[MAXNODES] ;  /*  Main Street Left end time       */
   static int  tr_sstb[MAXNODES] ;  /*  Side Street Thru start time     */
   static int  tr_sste[MAXNODES] ;      /*  Side Street Thru end time      */
   static int  tr_sslb[MAXNODES] ;  /*  Side Street Left start time     */
   static int  tr_ssle[MAXNODES] ;  /*  Side Street Left end time       */

   static int set_phase[MAXNODES] ;          /*  Switch/Keep current phase      */

        int req ;                      /*  Index for requested node        */
        int req_node ;                      /*  Requested node I.D. number      */
        int time ;                     /*  Current time                 */
        int tr_time ;                  /*  Current time transformed, main
                                        street thru now at zero         */
        int sum[MAXNODES] ;            /*  Sum of all phases @ each node  */

        FILE *Plans ;

   static int get_parameters;    /* flag to indicate if parameters have been
                                    read in from the file fixed.plans      */
```

```
/* only read the data file the first time this routine is called */

  if (!get_parameters) {

/*  ==================  READ THE DATA FROM INPUT FILE  ================= */

   Plans = fopen("fixed.plans", "r") ;

   fscanf(Plans,"%d", &no_nodes) ;
   fscanf(Plans,"%d", &cycle) ;

   for(i = 1; i <= no_nodes; ++i) {
      fscanf(Plans,"%d %d %d %d %d %d",
            &node_id[i], &offset[i], &mst[i], &msl[i], &sst[i], &ssl[i]) ;
      sum[i] = mst[i] + msl[i] + sst[i] + ssl[i] ;
      if(sum[i] != cycle) fprintf(stderr,"\nWARNING!!!!!  --  Phase durations for node
%3d do not sum to the cycle length of %3d", i, cycle) ;
   }


/*  ==================  SET PHASE CHANGE TIMES ON CLOCK  =================  */


   for(i = 1; i <= no_nodes; ++i) {
      mstb[i] = offset[i] ;                         /*  Main Street Thru  */
      if((offset[i] + mst[i]) > cycle)
        mste[i] = offset[i] + mst[i] - cycle - 1 ;
      else mste[i] = offset[i] + mst[i] ;

      if(msl[i] == 0)
        mslb[i] = msle[i] = mste[i] ;
      else {
         mslb[i] = mste[i] + 1;                             /*  Main Street Left  */
         if((mslb[i] + msl[i]) > cycle )
           msle[i] = mste[i] + msl[i] - cycle - 1 ;
         else msle[i] = mste[i] + msl[i] ;
      }

      sstb[i] = msle[i] + 1 ;                               /*  Side Street Thru  */
      if((sstb[i] + sst[i]) > cycle)
        sste[i] = msle[i] + sst[i] - cycle - 1 ;
      else sste[i] = msle[i] + sst[i] ;

      if(ssl[i] == 0)
        sslb[i] = ssle[i] = sste[i] ;
      else {
         sslb[i] = sste[i] + 1 ;                     /*  Side Street Left  */
         if((sslb[i] + ssl[i]) > cycle)
           ssle[i] = sste[i] + ssl[i] - cycle - 1 ;
         else ssle[i] = sste[i] + ssl[i] ;
      }

/*  ==============  TRANSFORM PHASE RANGES TO START AT ZERO  ==============  */

      tr_mstb[i] = 0 ;
      tr_mste[i] = mst[i] - 1 ;
      tr_mslb[i] = mst[i] ;
      tr_msle[i] = mst[i] + msl[i] - 1 ;
      tr_sstb[i] = mst[i] + msl[i] ;
      tr_sste[i] = mst[i] + msl[i] + sst[i] - 1 ;
      tr_sslb[i] = mst[i] + msl[i] + sst[i] ;
      tr_ssle[i] = mst[i] + msl[i] + sst[i] + ssl[i] - 1 ;
   } /* end for */

   fclose(Plans);
   get_parameters = 1;

 }  /* end  "if (!get_parameters)" */


/*  =============== Assign parameters from NETSIM variables =============== */

/* assign time and req_node from parameters passed from netsim */

   time = *t;
   req_node = *r;

   if(req_node == 335) req = 1 ;
   else if(req_node == 369) req = 2 ;
   else if(req_node == 401) req = 3 ;
   else if(req_node == 483) req = 4 ;
               /* return a zero if the intersection is not one of the
                    four under the control of this plan                 */
   else return(0);


/*  ====================  SET REQUESTED NODE SIGNALS  ====================  */

   if(time >= offset[req])
      tr_time = time - offset[req] ;
   else
      tr_time = time - offset[req] + cycle ;

/* the bits of the phase number indicate the phase to call */

   if((tr_time >= tr_mstb[req]) && (tr_time <= tr_mste[req])) {
      set_phase[req] = 1 ;  /* phase 1 */
   }
   else if((tr_time >= tr_mslb[req]) && (tr_time <= tr_msle[req])) {
      set_phase[req] = 2 ;  /* phase 2 */
   }
   else if((tr_time >= tr_sstb[req]) && (tr_time <= tr_sste[req])) {
      set_phase[req] = 4 ;  /* phase 3 */
   }
   else if((tr_time >= tr_sslb[req]) && (tr_time <= tr_ssle[req])) {
      set_phase[req] = 8 ;  /* phase 4 */
   }

/*
fprintf(stderr, "node: %d  time: %d  phase: %d\n",req_node,time,set_phase[req]);
*/

return (set_phase[req]) ;

} /*=========================== END OF FIXED ======================== */
```

Input File:

```
  4  90
335  68   50  15  20   5
369  79   50   0  40   0
401  26   40  15  25  10
483  78   40  10  30  10
```

APPENDIX K: External Semi-Actuated Signal Control Logic

```
/*=================================================================== ========*/
/*=                                                                  =*/
/*=                Semi-Actuated Control Implementation                    =*/
/*=                                                                  =*/
/*=================================================================== ========*/
/*=                                                                  =*/
/*=   Signal Switching Using Semi-Actuated Control                         =*/
/*=                                                                 =*/
/*=     City:             Tucson, Arizona                           =*/
/*=     Intersections:    Speedway Blvd. & Campbell Ave.             =*/
/*=                       Campbell Ave.  & University Blvd.                =*/
/*=                      Sixth St.      & Campbell Ave.              =*/
/*=                       Broadway Blvd. & Campbell Ave.             =*/
/*=                                                                  =*/
/*=   Programmed By:              Greg Tomooka and Doug Tarico             =*/
/*=   Completed:                  October 28,  1992                       =*/
/*=   Revised:                    October 28, 1992                        =*/
/*=                                                                        =*/
/*=================================================================== ========*/

/*=================================================================== ========*/
/*=   Program Specifics:                                            =*/
/*=                                                                 =*/
/*=   Called By:          translate                                 =*/
/*=                                                                 =*/
/*=   Subroutines Called: start_gap, update_gap, switch_phases            =*/
/*=                                                                        =*/
/*=================================================================== ========*/
/*=   Assumptions:                                                  =*/
/*=      1)  All phases allow permissive right turns                       =*/
/*=     2)  Time resolution is 1 second.                            =*/
/*=      3)  Phase duration of zero indicates a non-existent phase.     =*/
/*=      4)  Current network has a maximum of four phases at each node. =*/
/*=      5)  Parameter information must have nodes read in the same     =*/
/*=          order in which they occur in the detector data                =*/
/*=                                                                 =*/
/*=================================================================== ========*/
/*=   INPUTS:                                                             =*/
/*=                                                                 =*/
/*=   The timing plans are read from file 'semi_act.dat'              =*/
/*=   in the following format:                                            =*/
/*=                                                                 =*/
/*=       Number of intersections                                   =*/
/*=       Cycle length                                                    =*/
/*=      Intersection #i ID number, yield_point, P min. green, P max.    =*/
/*=          green, P unit extension, P added time per actuation,        =*/
/*=          P max initial green, P min. gap, P max. gap,                =*/
/*=         P time to reduce to min. gap, P force-off time,                =*/
/*=                                                                 =*/
/*=   where i = 1 to total intersections,                                 =*/
/*=       MST = Main Street Thru traffic with permissive left turns       =*/
/*=       MSL = Main Street Left turns                                    =*/
/*=       SST = Side Street Thru traffic with permissive left turn        =*/
/*=       SSL = Side Street Left turns                                    =*/
/*=   and P = phase denoted by:                                           =*/
/*=       P = MST, MSL, SST, or SSL.                                      =*/
/*=                                                                 =*/
/*=  Parameters passed in are:                                            =*/
/*=     sync     : NETSIM sync clock time                           =*/
/*=     node_id : intersection ID number                            =*/
/*=     phase1, phase2, phase3, phase4 :                            =*/
/*=     Counts of the number of calls on each phase.                      =*/
/*=                                                                    =*/
/*==================================================================  ========*/
/*=   OUTPUTS:                                                              =*/
/*=                                                                    =*/
/*=      Returns an integer code which specifies which phase should be  =*/
/*=     the currently active phase for the specified intersection.     =*/
/*=                                                                    =*/
/*==================================================================  ========*/
/*=   Program Description:                                             =*/
/*=                                                                    =*/
/*=    This program will simulate a semi-actuated traffic signal       =*/
/*=    controller.  Essentially, the controller simulated here   =*/
/*=    operates under the following conditions:                        =*/
/*=                                                                    =*/
/*=       1.  Main street through may terminate only at a specified     =*/
/*=           yield point if there is demand on any other phase,   =*/
/*=       2.  All other phases (except MST) may terminate due to   =*/
/*=           force-off, maximum green reached, or allotted green time   =*/
/*=           expended,                                                 =*/
/*=       3.  All other phases (except MST) may be active after the     =*/
/*=           yield point (given there is demand for that phase) and    =*/
/*=           can only change to a phase in the following sequence up   =*/
/*=           to and including phase 1:  2, 3, 4, 1,                    =*/
/*=       4.  To assure progression, MST must be active between the     =*/
/*=           the following two points:  The last phase's force-off     =*/
/*=           point in the cycle and the yield point,                   =*/
/*=       5.  If there are no calls on any phases, the controller will  =*/
/*=           rest in MST (i.e., phase 1), and                          =*/
/*=       6.  Phase 1 shall always refer to MST.                 =*/
/*=                                                                    =*/
/*==================================================================  ========*/

#include <stdio.h>

#define          MAXN                5        /*  maximum number of nodes        */
#define    MAXP               5        /*  maximum number of phases        */


/*===  function declarations  ===*/

int   switch_phases() ;
float start_gap() ;
float update_gap() ;


/*   =====================   GLOBAL VARIABLES   =====================    */

int    n ;                              /*  current node index           */
int    p ;                              /*  current phase index          */

static int gap_activated[MAXN] ;      /*  flag denoting if gap at this
                                          node in the current phase
                                          has been activated         */
static int    gap_out[MAXN] ;             /* flag: gap-out has occurred  */
static float  max_gap[MAXN][MAXP] ;         /*  maximum gap between calls  */
static float  gap[MAXN] ;               /*  current gap "size"       */
static float  min_gap[MAXN][MAXP] ;         /*  minimum gap between calls  */
static float  time_to_reduce[MAXN][MAXP] ; /*  time to reduce to min. gap */
static float  reduce_time[MAXN][MAXP] ;    /*  current time_to_reduce
                                          time elapsed             */

static float  force_off[MAXN][MAXP] ;      /*  phase force-off time (max) */
static int    current_phase[MAXN] ;        /*  current phase on node      */
static float  current_phase_start[MAXN] ;  /*  current phase start time   */
static int    init_done[MAXN] ;            /*  flag denoting whether
                                           initial green time on this
                                           phase has elapsed         */

static float  min_green[MAXN][MAXP] ;      /*  minimum green time per
                                           phase                      */
static float  init_green[MAXN][MAXP] ;     /*  actual initial green      */
static float  total_green_used[MAXN] ;        /*  amount of green time used
                                           in current phase           */
static float  total_green[MAXN][MAXP] ;    /*  actual total green         */
static float  current_time ;                  /*  current, "internal"
                                           clock time           */
static float  time_since_last_call[MAXN] ; /*  time since last call in
                                           current phase            */
static int    opposing_calls[MAXN] ;          /*  flag denoting whether or
                                           not an opposing call has
                                           been read for the activated
                                           phase on this node E [0,1] */
static int    total_calls[MAXN][MAXP] ;    /*  total calls on each phase
                                           since last signal change   */
static float  beg_per[MAXN] ;              /*  begin time of permissive
                                           period */
static float  end_per[MAXN] ;              /*  end time of permissive
                                           period   */
static   FILE  *out;                       /*  file for any status
                                           messages    */


/*    ================   START OF SEMI-ACTUATED PROGRAM   ===============    */

int semi_act(int sync, int node_id, int phase1, int phase2, int phase3,
             int phase4)
{
   static int nodes_processed ;              /*  number of nodes processed  */

   int          par_node_id ;                          /*  parameter node number          */
   static int no_nodes ;             /*  total number of nodes
                                          in timing plans file       */
   static int no_phases ;            /*  maximum phases per node         */
   static int yield_point[MAXN] ;    /*  sync phase yield point      */
   static float add_before_green[MAXN][MAXP] ;  /*  added green per call
                                              before phase activated */
   static float max_init_green[MAXN][MAXP] ;    /*  maximum initial green   */
   static float  add_during_green[MAXN][MAXP] ; /*  added green per call
                                              during activated phase */
   static float  max_green[MAXN][MAXP] ;    /*  maximum green time, but
                                          none for MST (phase 1)      */

   float  new_green ;                         /*  temporary new green         */
   int    sync_time ;                         /*  NETSIM sync time                */
   int    calls[MAXN][MAXP] ;                 /*  calls made on each phase
                                          for current time unit       */

   char header[80] ;                      /*  parameter data title info  */
   float allowable_gap ;                  /* gap size used when checking
                                             for gap-out                */
   static float time_since_yp[MAXN] ;     /* time elapsed since yield
                                             point was reached          */

   FILE        *Parameters ;                 /* parameter input file         */
   static int initialized=0;                 /* flag to indicate if
                                         initialization has been done */
```

```
/*              ****************************************************            */
/*   --------->    EXECUTE THIS INITIALIZATION ONLY ONCE!     <---------        */
/*              ****************************************************            */

   if (!initialized) {

/*   ==========   READ PARAMETERS IN FOR EACH NODE IN GRID  ============       */

   Parameters = fopen("semi_act.dat","r") ;

   fgets(header, 80, Parameters) ;
   fscanf(Parameters,"%d %d", &no_nodes, &no_phases) ;
   fgets(header, 80, Parameters) ;
   fgets(header, 80, Parameters) ;

   for (n = 1; n <= no_nodes; ++n) {
      fscanf(Parameters,"%d %d %f %f", &par_node_id, &yield_point[n],
                                       &beg_per[n], &end_per[n] ) ;
      fgets(header, 80, Parameters) ;
      fgets(header, 80, Parameters) ;
      for (p = 1; p <= no_phases; ++p) {
         fscanf(Parameters,"%f %f %f %f %f %f %f %f %f",
                &min_green[n][p],          &max_green[n][p],
                &add_before_green[n][p], &add_during_green[n][p],
                &max_init_green[n][p],   &min_gap[n][p],    &max_gap[n][p],
                &time_to_reduce[n][p],   &force_off[n][p]) ;
      }   /*  End for (p = 1; p <= no_phases; ++p)   */
   }   /*  End for (n = 1; n <= no_nodes; ++n)   */

   fclose(Parameters);

/*   ====================   INITIALIZE VARIABLES   ====================     */

   out = fopen("semi_act.out","w") ;
   current_time = 0.0 ;
   nodes_processed = 0 ;

   for (n = 1; n <= no_nodes; ++n) {
      time_since_last_call[n] = 0.0 ;              /*  Initialize time since
                                                last call              */
      gap_activated[n]        = 0  ;          /*  Set gap activation off  */
      gap_out[n]              = 0 ;            /*  set gap-out false        */
      current_phase[n]        = 1 ;           /*  Set all nodes to MST     */
      init_done[n]            = 0 ;           /*  Set initial green flag  */
      current_phase_start[n]  = current_time ;  /*  Set current phase start */
      total_green_used[n]     = 0.0 ;           /*  Set total green used    */
      for (p = 1; p <= no_phases; p++) {
         init_green[n][p]  = min_green[n][p] ;
          total_green[n][p] = min_green[n][p];
      }   /*  End for (p = 1; p <= no_phases; p++)   */
   }   /*  End for (n = 1; n <= no_nodes; ++n)   */

   initialized = 1;

}

/*                   ***********************************                       */
/*   ----------------  END OF ONE-TIME INITIALIZATION  ----------------        */
/*                   ***********************************                       */


/*=========================================================== ============*/
/*****                        BEGIN CONTROL LOGIC                  *****/
/*=========================================================== ============*/
```

```
   fprintf(out,"\nCurrent Time:  %5.2f", current_time) ;

/*   ========  CONVERT PARAMETERS PASSED FROM CALLING ROUTINE  =========     */

   if (node_id == 335)
      n = 1 ;
   else if (node_id == 369)
      n = 2 ;
   else if (node_id == 401)
      n = 3 ;
   else
      n = 4 ;

   sync_time = sync ;

   calls[n][1] = phase1 ;
   calls[n][2] = phase2 ;
   calls[n][3] = phase3 ;
   calls[n][4] = phase4 ;


/*  =====================  UPDATE TIME SINCE LAST CALL  ================  */

   if (calls[n][current_phase[n]] > 0 )
       time_since_last_call[n] = 0.0 ;
   else
       time_since_last_call[n] += 1.0 ;


/*   ===================  BEGIN LOOP OVER ALL PHASES  ==================   */

   for (p = 1; p <= no_phases; ++p) {


/*   ===================   UPDATE TOTAL CALL COUNTER   =================   */

   total_calls[n][p] += calls[n][p] ;


/*   ==================   Set "opposing calls" flag   =================   */

   if ((calls[n][p] > 0) && (current_phase[n] != p))
      opposing_calls[n] = 1 ;


/*   ==================   Check if initial green done   ================   */

   if ( ((current_time - current_phase_start[n]) > init_green[n][p]) &&
        (current_phase[n] == p))
      init_done[n] = 1 ;


/*  ====================  REDUCE GAP IF APPROPRIATE  =================  */

   if ( (current_phase[n] == p) && (p != 1) && (opposing_calls[n])
        && (!gap_out[n]) ) {
      if (gap_activated[n])
         gap[n] = update_gap() ;
      else   /*   !gap_activated[n]   */
         gap[n] = start_gap() ;
   }   /*  End if ( (current_phase[n] == p) && (p != 1) && ...)   */
```

```
/*  =======================  CHECK FOR GAP-OUT  ======================   */

   if ( (current_phase[n] == p) && gap_activated[n] && !gap_out[n] ) {
      if (gap[n] <= add_during_green[n][p]) {
         if (gap[n] > min_gap[n][p])
            allowable_gap = gap[n];
         else
            allowable_gap = min_gap[n][p];
      }
      else
         if (add_during_green[n][p] > min_gap[n][p])
            allowable_gap = add_during_green[n][p];
         else
            allowable_gap = min_gap[n][p];

      if (time_since_last_call[n] > allowable_gap) {
         gap_out[n] = 1;

      fprintf(out,"\n\n >>>> gap-out has occurred <<<<");
      fprintf(out,"\nNode: %3d  Gap = %5.2f  Time Bet. Calls = %5.2f  Last Phase:
%d\n", node_id, gap[n], time_since_last_call[n], p) ;
fprintf(out,"Extension time: %5.2f  Allowable gap: %5.2f  Min Gap: %5.2f \n",
add_during_green[n][p], allowable_gap, min_gap[n][p]);
      }  /*  End if time_since...    */

   }   /*   End if current_phase...    */

/*   ==================   ADD EXTENSION TIME IF NOT MST   =================   */

   if ((current_phase[n] == p) && (p != 1) && (!gap_out[n])
       && (calls[n][p] > 0)) {
      new_green = add_during_green[n][p] * calls[n][p] + total_green[n][p] ;
      if(new_green < max_green[n][p])
         total_green[n][p] = new_green ;
      else
         total_green[n][p] = max_green[n][p] ;
   }   /*  End of if ( (current_phase[n] == p) && ...   */


/*  ================  ADD INITIAL TIME TO NON_ACTIVE PHASES  ===========   */

   if ((calls[n][p] > 0) && (current_phase[n] != p) && (p != 1) ) {
      new_green = min_green[n][p] + add_before_green[n][p] * total_calls[n][p] ;
         if ((init_green[n][p] < max_init_green[n][p]) &&
             (new_green < max_init_green[n][p]))
            init_green[n][p] = new_green ;
         else
            init_green[n][p] = max_init_green[n][p] ;
   }   /*  End of if ((calls[n][p] > 0) && (current_phase[n] == p)) */


}    /* =====  END OF LOOP OVER ALL PHASES (p loop) ===== */


/*   ============   CHECK IF PHASE CHANGE SHOULD OCCUR    ============   */

    p = current_phase[n];

/* ======  FIRST CHECK FOR SWITCH OUT OF MST  =========  */

   if (p == 1) {
      if ( (opposing_calls[n]) && (sync_time == yield_point[n]) ) {
         time_since_yp[n] = 0.0 ;
         fprintf(out,"\n\nCalling switch_phases -- yielding to side street\n");
         fprintf(out,"\nNode: %3d  sync time = %3d   yield point = %3d  Last Phase:
%d\n", node_id, sync_time, yield_point[n], p) ;
        current_phase[n] = switch_phases() ;
        fprintf(out,"New Phase: %d\n", current_phase[n]) ;
      }  /* end of if (opposing calls... */
   }   /* end of if (p == 1) */


/*  ====== CHECK FOR SWITCH OUT OF OTHER PHASES  ========  */

   else {

/*   =====  CHECK FOR END OF ACCUMULATED GREEN TIME  ===== */

      if ( init_done[n] && opposing_calls[n] &&
           (total_green_used[n] >= total_green[n][p]) ) {
         fprintf(out,"\n\nCalling switch_phases -- total_green used up (out of
green)\n");
         fprintf(out,"\nNode: %3d  Tot Grn = %5.2f   Tot Grn Usd = %5.2f  Last Phase:
%d\n",node_id,total_green[n][p],total_green_used[n],p);
        current_phase[n] = switch_phases() ;
        fprintf(out,"New Phase: %d\n", current_phase[n]) ;
      }   /*  End of if ( (init_done[n] && opposing_calls[n] ...   */

/*  ================  CHECK FOR FORCE-OFF  ==============  */

      if ( ( time_since_yp[n] >= force_off[n][p]) ) {
         fprintf(out,"\n\nCalling switch_phases -- force-off\n");
        fprintf(out,"\nNode: %3d  Force Off = %5.2f  Yeild = %3d  Sync = %3d  Last
Phase: %d\n", node_id, force_off[n][p], yield_point[n], sync_time, p);
        current_phase[n] = switch_phases() ;
        fprintf(out,"New Phase: %d\n", current_phase[n]) ;

      }  /* End of if (total_green_used[n] >= ... */


   }  /* end of else */


/*   =========  CHECK FOR MAX GREEN EXCEEDED  ===========   */

   p = current_phase[n];

   if ( (p != 1) && (total_green_used[n] >= max_green[n][p]) ) {
      fprintf(out,"\n\nCalling switch_phases -- max-green (terminate phase)\n");
      fprintf(out,"\nNode: %3d  Max green = %5.2f   Grn Used = %5.2f  Last Phase:
%d\n", node_id, max_green[n][p], total_green_used[n], p) ;
      if (max_green[n][p] == 0.0)
         fprintf(out,"\n\nNote that phase %d was SKIPPED here because it does not
exist !\n\n\n", p) ;
        current_phase[n] = switch_phases() ;
        fprintf(out,"New Phase: %d\n", current_phase[n]) ;
      }  /*  End of if ( (p!=1) && total_green_used[n] >= max_green ...  */


/*   ===================  INCREMENT TIME COUNTERS  ====================  */

   total_green_used[n] += 1.0 ;

   time_since_yp[n] += 1.0 ;

   nodes_processed += 1 ;
   if (nodes_processed == no_nodes) {
      nodes_processed = 0 ;
```

```
     current_time += 1.0 ;
   )   /*  End if (nodes_processed == no_nodes)  */


/******************************************************************* ************/
/******                                      FUNCTIONS                        *****/
/******************************************************************* ************/


/*   ===========  RETURN PHASE IDENTIFIER TO CALLING ROUTINE  ===========  */

   switch (current_phase[n]) {
      case 1 : {
        return(1);
        break;
      }
      case 2 : {
        return(2);
        break;
      }
      case 3 : {
        return(4);
        break;
      }
      case 4 : {
        return(8);
        break;
      }
      default : return(0);
   }


   fprintf(out,"\n") ;

}   /*   End of function semi_act   */


/*   =================   FUNCTION:  switch_phases()    ================    */

int switch_phases()
{
   int  phase, next_phase, i ;
   int phase_set ;

   phase_set = 0 ;


   if (p == 1) {
      for (i=1; i<4; i++) {
         if (!phase_set) {
            phase = ( ((p-1+i) % 4) + 1) ;
            if (total_calls[n][phase] > 0) {
               next_phase = phase ;
              phase_set = 1 ;
            }
         }
      }
   }
   else if (p == 2) {
      for (i=1; i<3; i++) {
         if (!phase_set) {
            phase = ( ((p-1+i) % 4) + 1) ;
            if (total_calls[n][phase] > 0) {
               next_phase = phase ;
```

```
                phase_set = 1 ;
             }
          }
       }
    }
    else if (p == 3) {
       for (i=1; i<2; i++) {
          if (!phase_set) {
             phase = ( ((p-1+i) % 4) + 1) ;
             if (total_calls[n][phase] > 0) {
                next_phase = phase ;
                phase_set = 1 ;
             }
          }
       }
    }
    else if (p == 4) {
       next_phase = 1 ;
    }


    if (!phase_set)
       next_phase = 1 ;

    init_done[n]            = 0 ;              /*   Reset initial green done flag   */
    opposing_calls[n]       = 0 ;          /*   Reset opposing call flag        */
    total_green_used[n]     = 0.0 ;        /*   Reset amount of green used      */
    total_calls[n][next_phase] = 0 ;       /*   Reset total calls on new phase  */
    total_calls[n][p]          = 0 ;       /*   Reset total calls on prev phase */
    current_phase_start[n] = current_time ;   /*   Set current phase start      */
    time_since_last_call[n] = 0.0 ;
    gap_activated[n]        = 0 ;                 /*   Reset gap activation      */
    gap_out[n]              = 0 ;             /*   reset gap-out status        */
    init_green[n][p]            = min_green[n][p] ;
    total_green[n][next_phase] = init_green[n][next_phase];

    return(next_phase) ;

 }   /*   End of FUNCTION:  switch_phases()   */


 /*   ====================   FUNCTION:  start_gap()   ==================    */

 float start_gap()
 {
    float new_gap ;

    gap_activated[n] = 1 ;
    gap_out[n] = 0 ;
    new_gap = max_gap[n][current_phase[n]] ;

    time_since_last_call[n] = 0.0 ;

    reduce_time[n][current_phase[n]] = 0.0 ;
    return(new_gap) ;

 }   /*   End of FUNCTION:  start_gap()   */


 /*   ===================   FUNCTION:   update_gap()   =================    */

 float update_gap()
```

```
{
   float newer_gap ;
   if (time_to_reduce[n][current_phase[n]] != 0.0)
      newer_gap = max_gap[n][current_phase[n]]-((max_gap[n][current_phase[n]] -
              min_gap[n][current_phase[n]])/time_to_reduce[n][current_phase[n]] )
              * reduce_time[n][current_phase[n]] ;
   else
      newer_gap = 0.0 ;
   reduce_time[n][current_phase[n]] += 1 ;

   if (newer_gap > min_gap[n][current_phase[n]])
      return(newer_gap) ;
   else
      return(min_gap[n][current_phase[n]]) ;

}   /*   End of FUNCTION:  update_gap()    */
```

Input File:

```
no_nodes no_phases
4        4
node  yield  beg_per end_per
335    68     0.0    57.0
    min    max   ad_b4 ad_du maxinit mingap maxgap reduce force-off
    22.0   67.0   0.5   5.0   27.0     1.1    7.5   20.0   100.0
     9.0   42.0   0.0   2.0    9.0     1.1    2.0    5.0    15.0
    22.0   67.0   0.5   5.0   27.0     1.1    7.5   20.0    45.0
     9.0   42.0   0.0   2.0    9.0     1.1    2.0    5.0    57.0

369    79     0.0    35.0
    min    max   ad_b4 ad_du maxinit mingap maxgap reduce force-off
    21.0   66.0   0.5   5.0   26.0     1.1    7.5   20.0   100.0
     0.0    0.0   0.0   0.0    0.0     0.0    0.0    0.0     0.0
    16.0   66.0   0.5   4.0   21.0     1.1    7.0   15.0    35.0
     0.0    0.0   0.0   0.0    0.0     0.0    0.0    0.0     0.0

401    26     0.0    56.0
    min    max   ad_b4 ad_du maxinit mingap maxgap reduce force-off
    22.0   67.0   0.5   5.0   27.0     1.1    7.5   20.0   100.0
     8.0   41.0   0.0   2.0    8.0     1.1    2.0    5.0    12.0
    21.0   66.0   0.5   5.0   26.0     1.1    7.5   20.0    44.0
     9.0   42.0   0.0   2.0    9.0     1.1    2.0    5.0    56.0

483    78     0.0    50.0
    min    max   ad_b4 ad_du maxinit mingap maxgap reduce force-off
    22.0   67.0   0.5   5.0   27.0     1.1    7.5   20.0   100.0
     9.0   42.0   0.0   2.0    9.0     1.1    2.0    5.0    12.0
    16.0   66.0   0.5   5.0   21.0     1.1    6.0   15.0    38.0
     9.0   42.0   0.0   2.0    9.0     1.1    2.0    5.0    50.0
```

APPENDIX L: TRAF-NETSIM Input File for Simulated Network

```
Campbell Arterial.  External control of nodes 355,369,401,483                 00
Doug Gettman, Doug Tarico             10  21  92University of Arizona      0  01
   0   1           7               0         0 0   31100        2569    3233  02
 900                                                                          03
                  60                                                          04
                                               0   0tucnl2                    05
 275 3352440 300 150 3 2 1           337 369 334     369          35 0        11
 335 2752440 180 180 3 1 1           274 175 276     175          35 0        11
 335 3691565 140     3 1             370 401 368     401          35 0        11
 369 3351565 300 150 3 2 1           334 275 337     275          35 0        11
 369 4011445         4          1    403 483 399     483          35 0        11
 401 3691445 200     3 1             368 335 370     335          35 0        11
 401 4832350 120 120 3 2 1           484 538 481     538          35 0        11
 483 4012350         4          1    399 369 403     369          35 0        11
 483 5382540 180     3 1             539 638 537     638          40 0        11
 538 4832540 120 120 3 2 1           481 401 484     401          40 0        11
 333 3341315 150     3 1             234 335 434     335          35 0        11
 334 3331315 180     3 1             433 332 233     332          35 0        11
 334 3351305     150 4   1      1    275 337 369     337          35 0        11
 335 3341305 150     3 1             434 333 234     333          35 0        11
 335 3372685 250 150 3 1 1           237 338 437     338          35 0        11
 337 3352685     150 4   1      1    369 334 275     334          35 0        11
 399 4012205     100 3   1      1    369 403 483     403          30 0        11
 401 3992205     150 3   1      1    499 398 299     398          30 0        11
 401 4032690         3          1    303 404 485     404          35 0        11
 403 4012690         3          1    483 399 369     399          35 0        11
 481 4832210      75 3   1      1    401 484 538     484          30 0        11
 483 4812210         3          1    581 480 381     480          30 0        11
 483 4841380         3          1    384 485 584     485          35 0        11
 484 4831380         3          1    538 481 401     481          35 0        11
 484 4851345         3          1    403 486 585     486          35 0        11
 485 4841345         3          1    584 483 384     483          35 0        11
 403 4852360  80  80 1 1 1           486 585 484     585          30 0        11
 485 4032360  70     1 1             401 303 404     303          30 0        11
 175 275 500 150     3 1             276 335 274     335          35 0        11
 275 175 500         3                  8175                      35          11
 274 275 500 100     1 1             175 276 335     276          35 0        11
 275 274 500         1                  8274                      35          11
 275 276 500         1                  8276                      35          11
 276 275 500 100     1 1             335 274 175     274          35          11
8274 274             1                   275                                  11
8175 175             1                   275                                  11
8276 276             1                   275                                  11
 233 333 500 150 150 1 1 1           334 433 332     433          35          11
 333 233 500         1                  8233                      35          11
 332 333 500 180     3 1             233 334 433     334          35          11
 333 332 500         3                  8332                      35          11
 433 333 500 150 150 1 1 1           332 233 334     233          35          11
 333 433 500         1                  8433                      35          11
8233 233             3                   333                                  11
8332 332             3                   333                                  11
8433 433             1                   333                                  11
 234 334 500 180 100 1 1 1           335 434 333     434          35          11
 334 234 500         1                  8234                      35          11
 434 334 500 180 100 1 1 1           333 234 335     234          35          11
 334 434 500         1                  8434                      35          11
8234 234             1                   334                                  11
8434 434             1                   334                                  11
 237 337 500     150 3   1      1    338 437 335     437          35          11
 337 237 500         2                  8237                      35          11
 338 337 500 250 150 3 1 1           437 335 237     335          35          11
 337 338 500         2                  8338                      35          11
 437 337 500 150 150 2 1 1        335 237 338    237        35        11
 337 437 500         2               8437                   35        11
8237 237             2                337                             11
8338 338             3                337                             11
8437 437             2                337                             11
 368 369 500     100 2   1   1    335 370 401    370        25        11
 369 368 500         1               8368                   25        11
 370 369 500  75     1 1          401 368 335    368        35        11
 369 370 500         1               8370                   35        11
8368 368             1                369                             11
8370 370             1                369                             11
 299 399 500     200 1   1        401 499 398    499        25        11
 399 299 500         1               8299                   25        11
 398 399 500         3       1    299 401 499    401        30        11
 399 398 500         2               8398                   30        11
 499 399 500 100     1 1          398 299 401    299        25        11
 399 499 500         1               8499                   25        11
8299 299             1                399                             11
8398 398             3                399                             11
8499 499             1                399                             11
 303 403 500 100     1 1          404 485 401    485        30        11
 403 303 500         1               8303                   30        11
 404 403 500         3       1    485 401 303    401        35        11
 403 404 500         2               8404                   35        11
8303 303             1                403                             11
8404 404             3                403                             11
 381 481 500 120     1 1          483 581 480    581        25        11
 481 381 500         1               8381                   25        11
 480 481 500         3       1    381 483 581    483        30        11
 481 480 500         2               8480                   30        11
 581 481 500 120     1 1          480 381 483    381        25        11
 481 581 500         1               8581                   25        11
8381 381             1                481                             11
8480 480             3                481                             11
8581 581             1                481                             11
 384 484 500 100     1 1          485 584 483    584        25        11
 484 384 500         1               8384                   25        11
 584 484 500 100     1 1          483 384 485    384        25        11
 484 584 500         1               8584                   25        11
8384 384             1                484                             11
8584 584             1                484                             11
 486 485 500         3       1    585 484 403    484        35        11
 485 486 500         2               8486                   35        11
 585 485 500  80  80 1 1 1        484 403 486    403        30        11
 485 585 500         1               8585                   30        11
8486 486             3                485                             11
8585 585             1                485                             11
 537 538 500 100     2 1          483 539 638    539        35        11
 538 537 500         2               8537                   35        11
 539 538 500 100     2 1          638 537 483    537        35        11
 538 539 500         2               8539                   35        11
 638 538 500 180     3 1          537 483 539    483        40        11
 538 638 500         3               8638                   40        11
8537 537             2                538                             11
8539 539             2                538                             11
8638 638             3                538                             11
 275 175     100                      275 274    100                  21
 275 276     100                      333 233    100                  21
 333 332     100                      333 433    100                  21
 334 234     100                      334 434    100                  21
 337 237     100                      337 338    100                  21
 337 437     100                      369 368    100                  21
 369 370     100                      399 299    100                  21
 399 398     100                      399 499    100                  21
 403  303        100                   403  404       100             21
 481  381        100                   481  480       100             21
 481  581        100                   484  384       100             21
 484  584        100                   485  486       100             21
 485  585        100                   538  537       100             21
 538  539        100                   538  638       100             21
8175  175        100                  8274  274       100             21
8276  276        100                  8233  233       100             21
8433  433        100                  8234  234       100             21
8434  434        100                  8237  237       100             21
8338  338        100                  8437  437       100             21
8368  368        100                  8370  370       100             21
8299  299        100                  8398  398       100             21
8499  499        100                  8303  303       100             21
8404  404        100                  8381  381       100             21
8480  480        100                  8581  581       100             21
8384  384        100                  8584  584       100             21
8486  486        100                  8585  585       100             21
8537  537        100                  8539  539       100             21
8638  638        100                  8332  332       100             21
 481  483  2331989  230                401  483  6001150   30         21
 484  483  2501770  460                538  483  350  460  340        21
 484  485  1302440  210                403  485  250  370  280        21
 486  485  1302270  200                585  485  270  190  470        21
 233  333  150  270   30               332  333   602600  160         21
 334  333   502350  100                433  333  160  260   50        21
 334  335  4801740  450                275  335  5701830  310         21
 337  335  3701680  270                369  335  3102310  450         21
 335  337  1402550  130                237  337  220  610  160        21
 338  337  1502570  200                437  337  140  590  210        21
 399  401  4101210  290                369  401  4301820  350         21
 403  401  1401200  260                483  401  3301750  100         21
 401  403  1401390  120                303  403  130  600  100        21
 404  403   901330  150                485  403  110  590   70        21
 274  275   55   20   25               276  275   60   15   25       21
 175  275    5   90    5               335  275    5   90    5       21
 234  334   66   14   22               434  334   39    8   53       21
 333  334    5   90    5               335  334    5   90    5       21
 368  369   78    0   22               370  369   66   14   22       21
 335  369    5   90    5               401  369    5   90    5       21
 398  399    5   90    5               401  399    5   90    5       21
 299  399   53   29   18               499  399   27   19   54       21
 480  481    5   90    5               483  481    5   90    5       21
 381  481   73    8   19               581  481   32   27   31       21
 384  484   43   17   40               584  484   22    8   70       21
 483  484    5   90    5               485  484    5   90    5       21
 537  538   18    3   14               539  538   23    1   47       21
 638  538    2   95    3               483  538    3   95    2       21
 175       2758175                                                    35
 274       2758274                                                    35
 276       2758276                                                    35
 233       3338233                                                    35
 332       3338332                                                    35
 433       3338433                                                    35
 234       3348234                                                    35
 434       3348434                                                    35
 237       3378237                                                    35
 338       3378338                                                    35
 437       3378437                                                    35
 368       3698368                                                    35
 370       3698370                                                    35
 299       3998299                                                    35
 398       3998398                                                    35
 499       3998499                                                    35
```

```
303      4038303                                                                         35
404      4038404                                                                         35
381      4818381                                                                         35
480      4818480                                                                         35
581      4818581                                                                         35
384      4848384                                                                         35
584      4848584                                                                         35
486      4858486                                                                         35
585      4858585                                                                         35
537      5388537                                                                         35
539      5388539                                                                         35
638      5388638                                                                         35
175 11                                                                     311 1133     36
274 11                                                                     261 1083     36
276 11                                                                     361 1083     36
233 11                                                                      50  889     36
332 11                                                                       0  839     36
433 11                                                                      50  789     36
234 11                                                                     181  889     36
434 11                                                                     181  789     36
237 11                                                                     583  889     36
338 11                                                                     633  839     36
437 11                                                                     583  789     36
368 11                                                                     261  683     36
370 11                                                                     361  683     36
299 11                                                                      90  589     36
398 11                                                                      40  539     36
499 11                                                                      90  489     36
303 11                                                                     583  589     36
404 11                                                                     633  539     36
381 11                                                                      90  354     36
480 11                                                                      40  304     36
581 11                                                                      90  254     36
384 11                                                                     449  354     36
584 11                                                                     449  254     36
486 11                                                                     633  304     36
585 11                                                                     583  254     36
537 11                                                                     261   50     36
539 11                                                                     361   50     36
638 11                                                                     311    0     36
=========+=========+=========+=========+=========+=========+=====  ====+=====
337 335   1 3250  1    60  1  2 3250  2    60  1  3 3250  3    60  1            42
334 335   1 3250  4    60  1  2 3250  5    60  1  3 3250  6    60  1            42
334 335   4    0  7   700  0                                                    42
337 335   4    0  8   700  0                                                    42
275 335   1 3250  9    60  1  2 3250 10    60  1  3 3250 11    60  1            42
369 335   1 3250 12    60  1  2 3250 13    60  1  3 3250 14    60  1            42
369 335   6    0 15   700  0                                                    42
275 335   6    0 16   700  0                                                    42
335 369   9 2000 17    60  1                                                    42
401 369   9 2000 18    60  1                                                    42
368 369   2   80 19   250  0                                                    42
370 369   1 1500 20    60  1                                                    42
403 401   1 3200 21    60  1  2 3200 22    60  1                                42
399 401   1 3200 23    60  1  2 3200 24    60  1                                42
399 401   3    0 25   700  0                                                    42
403 401   3    0 26   700  0                                                    42
369 401   1 1200 27    60  1  2 1200 28    60  1  3 1200 29    60  1            42
483 401   1 1200 30    60  1  2 1200 31    60  1  3 1200 32    60  1            42
369 401   4    0 33   500  0                                                    42
483 401   4    0 34   500  0                                                    42
=========+=========+=========+=========+=========+=========+=====  ====+=====
481 483   1 2900 35    60  1  2 2900 36    60  1                                42
484 483   1 2900 37    60  1  2 2900 38    60  1                                42
```

```
481 483   3     0 39    700  0                                                         42
484 483   3     0 40    700  0                                                         42
538 483   1  3000 41     60  1   2 3000 42     60  1   3 3000 43     60  1             42
401 483   1  3000 44     60  1   2 3000 45     60  1   3 3000 46     60  1             42
538 483   6     0 47    700  0                                                         42
401 483   6     0 48    700  0                                                         42
275 175 275 276 275 335 275 274 275                                      311 1083      43
333 233 333 334 333 433 333 332 333                                       50  839      43
334 234 334 335 334 434 334 333 334                                      181  839      43
335 275 335 337 335 369 335 334 335                                      311  839      43
337 237 337 338 337 437 337 335 337                                      583  839      43
369 335 369 370 369 401 369 368 369                                      311  683      43
399 299 399 401 399 499 399 398 399                                       90  539      43
401 369 401 403 401 483 401 399 401                                      311  539      43
403 303 403 404 403 485 403 401 403                                      583  539      43
481 381 481 483 481 581 481 480 481                                       90  304      43
483 401 483 484 483 538 483 481 483                                      311  304      43
484 384 484 485 484 584 484 483 484                                      449  304      43
485 403 485 486 485 585 485 484 485                                      583  304      43
538 483 538 539 538 638 538 537 538                                      311   50      43
333  90 24   0 35                        35          35          100          100      44
334  90 28   0 35                        35          35          100          100      44
337  90 36   0 40                        40          40          100          100      44
399  90  1   0 28                        28          28          100          100      44
403  90 89   0 40                        40          40          100          100      44
481  90 42   0 30                        30          30          100          100      44
484  90  9   0 27                        27          27          100          100      44
485  90 54   0 30                        30          30          100          100      44
275  90 73   0 35                        35          35          100          100      44
538  90  8   0 30                        30          30          100          100      44
275     2 11122      22122      11122      22122                                       45
275     6 11122      22122      11122      22122                                       45
275     4 22122      11122      22122      11122                                       45
275     8 22122      11122      22122      11122                                       45
333     2 22122      11122      22122      11122                                       45
333     6 22122      11122      22122      11122                                       45
333     4 11122      22122      11122      22122                                       45
333     8 11122      22122      11122      22122                                       45
334     2 22122      11122      22122      11122                                       45
334     6 22122      11122      22122      11122                                       45
334     4 11122      22122      11122      22122                                       45
334     8 11122      22122      11122      22122                                       45
335     1 22122      11122      22122      11122                                       45
335     2 22122      12122      22122      12122                                       45
335     3 11122      22122      11122      22122                                       45
335     4 12122      22122      12122      22122                                       45
337     2 22122      11122      22122      11122                                       45
337     6 22122      11122      22122      11122                                       45
337     4 11122      22122      11122      22122                                       45
337     8 11122      22122      11122      22122                                       45
369     1 11122      22122      11122      22122                                       45
369     3 22122      11122      22122      11122                                       45
399     6 22122      11122      22122      11122                                       45
399     2 22122      11122      22122      11122                                       45
399     4 11122      22122      11122      22122                                       45
399     8 11122      22122      11122      22122                                       45
401     1 22122      11122      22122      11122                                       45
401     2 22122      12122      22122      12122                                       45
401     3 11122      22122      11122      22122                                       45
401     4 12122      22122      12122      22122                                       45
403     2 22122      11122      22122      11122                                       45
403     4 11122      22122      11122      22122                                       45
481     2 22122      11122      22122      11122                                       45
481     6 22122      11122      22122      11122                                       45
```

```
481   4 11122     22122     11122     22122                                        45
481   8 11122     22122     11122     22122                                        45
483   1 22122     11122     22122     11122                                        45
483   2 22122     12122     22122     12122                                        45
483   3 11122     22122     11122     22122                                        45
483   4 12122     22122     12122     22122                                        45
484   2 22122     11122     22122     11122                                        45
484   6 22122     11122     22122     11122                                        45
484   4 11122     22122     11122     22122                                        45
484   8 11122     22122     11122     22122                                        45
485   2 22122     11122     22122     11122                                        45
485   6 22122     11122     22122     11122                                        45
485   4 11122     22122     11122     22122                                        45
485   8 11122     22122     11122     22122                                        45
538   2 11122     22122     11122     22122                                        45
538   6 11122     22122     11122     22122                                        45
538   4 22122     11122     22122     11122                                        45
538   8 22122     11122     22122     11122                                        45
275 6 1  11 3250       60  1  2 3250       60  1  3 3250       60  1               46
275 4 1  21    0      700     1 1500       60  1                                   46
275 2 1  31 3250       60  1  2 3250       60  1  3 3250       60  1               46
275 8 1  41    0      700     1 1500       60  1                                   46
333 2 1  21 1500       60  1  2 1500       60  1  3 1500       60  1               46
333 2 1  21 3250       60  1  2 3250       60  1  3 3250       60  1               46
333 4 1  31 3000       60  1  1    0      700                                      46
333 6 1  41 1500       60  1  2 1500       60  1  3 1500       60  1               46
333 6 1  41 3250       60  1  2 3250       60  1  3 3250       60  1               46
333 8 1  11    0      700     1 3000       60  1                                   46
334 8 1  11    0      700     1 2000       60  1                                   46
334 2 1  21 1500       60  1  2 1500       60  1  3 1500       60  1               46
334 2 1  21 3250       60  1  2 3250       60  1  3 3250       60  1               46
334 4 1  31    0      700     1 2000       60  1                                   46
334 6 1  41 1500       60  1  2 1500       60  1  3 1500       60  1               46
334 6 1  41 3250       60  1  2 3250       60  1  3 3250       60  1               46
335 3 1  11 3250       60  1  2 3250       60  1  3 3250       60  1               46
335 3 1  11 1500       60  1  2 1500       60  1  3 1500       60  1               46
335 3 1  11    0       60     2    0       60     3    0       60                  46
335 4 1  16    0    20700                                                          46
335 1 1  21 1500       60  1  2 1500       60  1  3 1500       60  1               46
335 1 1  21 3250       50  1  2 3250       60  1  3 3250       60  1               46
335 2 1  24    0    20700                                                          46
335 3 1  31    0       60     2    0       60     3    0       60                  46
335 3 1  31 1500       60  1  2 1500       60  1  3 1500       60  1               46
335 3 1  31 3250       60  1  2 3250       60  1  3 3250       60  1               46
335 4 1  36    0    20700                                                          46
335 1 1  41 1500       60  1  2 1500       60  1  3 1500       60  1               46
335 1 1  41 3250       60  1  2 3250       60  1  3 3250       60  1               46
335 2 1  44    0    20700                                                          46
337 8 1  11    0      700     2    0      700  1  3    0      700  1               46
337 8 1  11 3000       60  1  2 3000       60  1                                   46
337 2 1  21 1500       60  1  2 1500       60  1  3 1500       60  1               46
337 2 1  21 3250       60  1  2 3250       60  1  3 3250       60  1               46
337 4 1  32 3000       60  1  2    0      700     1    0      700                  46
337 6 1  41 1500       60  1  2 1500       60  1  3 1500       60  1               46
337 6 1  41 3250       60  1  2 3250       60  1  3 3250       60  1               46
369 3 1  42   80      250                                                          46
369 3 1  21 1500       60  1                                                       46
369 1 1  11  100       60  1                                                       46
369 1 1  31  100       60  1                                                       46
399 8 1  11    0       60     1  160       60     1  320       60                  46
399 8 1  11  480       60                                                          46
399 4 1  31    0       60     1  160       60     1  320       60                  46
399 4 1  31  480       60     1 1500       60  1                                   46
401 3 1  11    0       60     2    0       60     3    0       60                  46
```

```
401 3 1  11 1200      60  1  2 1200       60  1  3 1200      60  1          46
401 4 1  14    0   20500                                                    46
401 2 1  23    0   20700                                                    46
401 3 1  31    0      60     2    0       60     3    0      60             46
401 3 1  31 1200      60  1  2 1200       60  1  3 1200      60  1          46
401 4 1  34    0   20500                                                    46
401 1 1  41 3200      60  1  2 3200       60  1                             46
401 2 1  43    0   20700                                                    46
403 4 1  11    0     700     1 1500       60  1                             46
403 2 1  23    0     700                                                    46
403 4 1  31    0     700     1 1500       60  1                             46
403 2 1  43    0     700                                                    46
481 8 1  11    0     700     1 1500       60  1                             46
481 4 1  31    0     700     1 1500       60  1                             46
483 3 1  11    0      60     2    0       60     3    0      60             46
483 3 1  11 3000      60  1  2 3000       60  1  3 3000      60  1          46
483 4 1  16    0   20700                                                    46
483 1 1  21 2900      60  1  2 2900       60  1                             46
483 2 1  23    0   20700                                                    46
483 3 1  31    0      60     2    0       60     3    0      60             46
483 3 1  31 3000      60  1  2 3000       60  1  3 3000      60  1          46
483 4 1  36    0   20700                                                    46
483 1 1  41 2900      60  1  2 2900       60  1                             46
483 2 1  43    0   20700                                                    46
484 8 1  11  600     160  1  1    0      150     1 1500      60  1          46
484 4 1  31  600     160  1  1    0      300     1 1500      60  1          46
485 8 1  11    0      60     1 1500       60  1                             46
485 4 1  31    0      60     1 1500       60  1                             46
538 6 1  11 3000      60  1  2 3000       60  1  3 3000      60  1          46
538 4 1  22 1500      60  1                                                 46
538 8 1  42 1500      60  1                                                 46
275 2   60 1550     5     202      20     0 750 40 15000010000000  15       47
275 6   60 1550     5     202      20     0 750 40 15000010000000  15       47
275 4   60 1050     5     102      10     0 600 30 30000000000000  30       47
275 8   60 1050     5     102      10     0 600 30 30000000000000  30       47
333 2   60 1550     5     202      20     0 750 40 20000010000000  20       47
333 6   60 1550     5     202      20     0 750 40 20000010000000  20       47
333 4   60 1020     5     102      10     0 600 35 30000000000000  30       47
333 8   60 1020     5     102      10     0 600 35 30000000000000  30       47
334 2   60 1550     5     202      20     0 750 40 20000010000000  20       47
334 6   60 1550     5     202      20     0 750 40 20000010000000  20       47
334 4   60 1020     5     102      10     0 600 30 30000000000000  30       47
334 8   60 1020     5     102      10     0 600 30 30000000000000  30       47
335 1   60  111     0       1 9999 01 110 110 40 20000000010000  20       47
335 2   60  111     0       1 9999 01 110 110 40 20000000000000  20       47
=========+=========+=========+=========+=========+=========+===== ====+
335 3   60  111     0       1 9999 01 110 110 40 20000000000000  20       47
335 4   60  111     0       1 9999 01 110 110 40 20000000010000  20       47
337 2   60 1550     5     202      20     0 750 40 25000010000000  25       47
337 6   60 1550     5     202      20     0 750 40 25000010000000  25       47
337 4   60 1020     5     102      10     0 600 35 30000000000000  30       47
337 8   60 1020     5     102      10     0 600 35 30000000000000  30       47
369 1   60  111     0       1 9999 01 110 110 40 20000000000000  20       47
369 3   60  111     0       1 9999 01 110 110 40 20000000000000  20       47
399 2   60 1550     5     202      20     0 750 30 20000010000000  20       47
399 6   60 1550     5     202      20     0 750 30 20000010000000  20       47
399 4   60 1020     5     102      10     0 600 30 30000000000000  30       47
399 8   60 1020     5     102      10     0 600 30 30000000000000  30       47
401 1   60  111     0       1 9999 01 110 110 40 20000000010000  20       47
401 2   60  111     0       1 9999 01 110 110 40 20000000000000  20       47
401 3   60  111     0       1 9999 01 110 110 40 20000000010000  20       47
=========+=========+=========+=========+=========+=========+===== ====+
401 4   60  111     0       1      01 110 110 40 20000000000000  20       47
403 4   60 1550     5     202      20     0 750 40 15000010000000  15       47
```

```
 403 2  45  520      5    102      10    0 600 35 2000000000000  20                 47
 481 2  60 1550      5    202      20    0 750 35 1500001000000  15                 47
 481 6  60 1550      5    202      20    0 750 35 1500001000000  15                 47
 481 4  60  550 -    5    102      10    0 600 30 2500000000000  25                 47
 481 8  60  550      5    102      10    0 600 30 2500000000000  25                 47
 483 1  60  111      0      1 9999 01 110 110 40 2000000010000  20                 47
 483 2  60  111      0      1 9999 01 110 110 40 2000000000000  20                 47
 483 3  60  111      0      1 9999 01 110 110 40 2000000010000  20                 47
 483 4  60  111      C      1 9999 01 110 110 40 2000000000000  20                 47
 484 4  60 1030      5    152      15    0 600 30 2500000000000  25                 47
 484 8  60 1030      5    152      15    0 600 30 2500000000000  25                 47
 484 2  60 40 0      0    402       0    0   0 40 1500001000000  15                 47
 484 6  60 40 0      0    402       0    0   0 40 1500001000000  15                 47
 485 2  60 1550      5    202      20    0 750 40 2000001000000  20                 47
 485 6  60 1550      5    202      20    0 750 40 2000001000000  20                 47
 485 4  60 1050      5    152      15    0 600 40 2500000000000  25                 47
 485 8  60 1050      5    152      15    0 600 40 2500000000000  25                 47
 538 2  60 1550      5    202      20    0 750 40 2000001000000  20                 47
 538 6  60 1550      5    202      20    0 750 40 2000001000000  20                 47
 538 4  60 1050      5    152      15    0 600 30 3000000000000  30                 47
 538 8  60 1050      5    152      15    0 600 30 3000000000000  30                 47
8274 274 175        8276 276 125        8332 3321650                                50
8433 433 130        8233 233 255        8434 434 360                                50
8234 234 210        8437 437 420        8338 3381550                                50
8237 237 540        8368 368 130        8370 370  55                                50
8303 303 330        8404 404 720        8381 381  15                                50
8581 581  25        8384 384 145        8584 584  75                                50
8486 4861550        8585 585 290        8539 539  75                                50
8175 175 980        8638 638 980        8537 537  35                                50
8480 4801040        8398 398 850        8299 299 150                                50
8499 499  60                                                                        50
*========+=========+=========+=========+=========+=========+=========+=====  ====+=====
 275 335 369 401 483 538          0                                                 90
   1   0   2   0   3   0   4   0   5   0   6   0   7   0                           140
  75  85  95  95 100 105 105 110 110 120                                           147
   0                                                                               170
   1                                                                               210
```