ADAPTIVE CONTROL STRATEGY FOR ISOLATED INTERSECTION AND TRAFFIC NETWORK

A Dissertation

Presented to

The Graduate Faculty of The University of Akron

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

Chun Shao

May, 2009

ADAPTIVE CONTROL STRATEGY FOR ISOLATED INTERSECTION AND TRAFFIC NETWORK

Chun Shao

Dissertation

Approved:	Accepted:
_____	_____
Advisor Dr. Ping Yi	Department Chair Dr. Wieslaw K. Binienda
_____	_____
Committee Member Dr. William H Schneider IV	Dean of the College Dr. George K. Haritos
_____	_____
Committee Member Dr. Ernian Pan	Dean of the Graduate School Dr. George R. Newkome
_____	_____
Committee Member Dr. Kevin L Kreider	Date
_____	
Committee Member Dr Zhong-Hui Duan	

ABSTRACT

Traffic congestion due to increased travel demands is a common problem in urban areas across the United States. Among the many solutions to traffic congestion, operational treatment providing more efficient traffic operation is attractive due to its relatively low cost. Besides traditional operation treatments such as actuated control and signal coordination, adaptive signal control strategies have becoming increasingly popular since 1980s. The advantages of the existing adaptive control strategies, such as SCATS, SCOOT, OPAC and RHODES, over the actuated control scheme are reviewed and their limitations are used as the basis for improvement in this research. Two adaptive control logics, PODE and GABNOR, are proposed for isolated intersection and traffic network optimization. By using real-time traffic as input, PODE dynamically searches movement combinations for phasing and timing decisions that minimize piecewise system delay. In GABNOR, the obstacles preventing the application of Genetic Algorithm in real-time have been addressed and possible solutions have been presented. Implemented as computer programs, PODE and GABNOR are compared with other control logics and show competitive optimization ability. The results have been analyzed with statistic tools and the system sensitivity to traffic arrival pattern and system parameters are also analyzed. Future works are introduced to further examine and improve the performance of PODE and GABNOR.

DEDICATION

To my wife and our baby Daniel

To my parents

You are the most important part of my present life!

ACKNOWLEDGEMENTS

First of all, I would like to praise the almighty God for all the guidance in my life and giving me wisdom and strength to finish this dissertation.

I would like to express my deepest sense of gratitude to my advisor Professor Ping Yi, who is a wise advisor, aggressive researcher and patient teacher. Without his guidance and persistent help this dissertation would not have been possible.

I would like to thank my committee member Dr. Zhong-Hui Duan for her valuable help on Genetic Algorithm. My thanks also go to Mr. Darren Moore and Mr. Steven Oberlin for their help on reviewing my dissertation. I also want to thank my colleagues in the transportation laboratory for their supports and helps.

I would take this opportunity to express my profound gratitude to my beloved parents. This thesis is dedicated to them in appreciation.

My final, and most heartfelt, acknowledgment must go to my wife Feiran Yu for supporting me with her unconditional love. Those supports are not only in the toughest time of my experiments, but also in every day's life. Honey, you are the best gift for me from God!

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Traffic congestion due to increasing travel demands is a common problem in urban areas across the United States. According to the Texas Transportation Institute's report (1), the total delay that U.S. drivers experienced has reached 4.2 billion hours in 2005, which was 0.8 billion hours in 1982 correspondingly. Along with 2.9 billion gallons of fuel wasted by delay, nation-wide vehicle delay costs $78.2 billion dollars for drivers. Among the many solutions to traffic congestion, operational treatment providing more efficient traffic operation is becoming increasingly attractive due to its relatively low cost. In 2005, 292 million hours of delay and $5.4 billion of congestion cost were saved by signal modifications.

Besides traditional operation treatments such as actuated control and signal coordination, adaptive signal control strategies have been increasingly popular since 1980s. Among the adaptive control strategies, Sydney Coordinated Adaptive Traffic System (SCATS) and Split Cycle Offset Optimisation Technique (SCOOT) are two outstanding strategies for traffic network optimization. In early 1980's, SCATS was introduced in Sydney, Australia (2) and became known for its implementation of Divorce-Marriage method which select intersections for coordination dynamically. Since its introduction, SCATS has been deployed in over 16,000 intersections in 93 cities in 21

counties by June, 2007 (3). Almost at the same time, SCOOT was presented as a third generation adaptive traffic signal control system in United Kingdom (4). The SCOOT uses gradient search to incrementally adjust the green split, cycle length and offset of each intersection in the traffic network. In nearly three decades, SCOOT has been deployed in over 200 cities and towns world widely and its ability in improving traffic condition has been widely evaluated. Other famous adaptive control systems such as Optimization Policies for Adaptive Control (OPAC) (5) with rolling horizon optimization and Real-Time Hierarchical Optimized Distributed and Effective System (RHODES) (6) with hierarchical optimization have also been developed and tested in recent years in the United States sponsored by the Federal Highway Administration. OPAC was first presented by N. H. Gartner in 1983, and uses a simplified Dynamic Programming (DP) method and Rolling Horizon approach to optimize signal control for isolated intersection. OPAC was later expanded to include a coordination/synchronization strategy suitable for arterials and networks control (7). Similarly, RHODES is a hierarchical strategy introduced by Head and Mirchandani in 1992 (6). RHODES has different algorithms for intersection control and network control. At the intersection control level, RHODES applies a procedure called Controlled Optimization of Phases (COP) (8) which optimizes phasing and timing by using a DP method. For network control, RHODES uses the REALBAND model (9) based on platoon flow prediction. According to previous studies (10), the decline of vehicle delay in OPAC varies from 3.9% to 15.94% compared with the actuated control logic in three field tests. The software simulation performed for RHODES (11) also showed significant vehicle delay reduction for both low and high levels of traffic demand against actuated control.

However, current adaptive control logic has several limitations including uncertain traffic flow prediction, difficulty in estimating the arrival time, and lack of a self-adjusting mechanism; all are discussed in the next chapter. Rapid developments of recent years in other fields, such as artificial intelligence, information and computer science, have revealed cutting-edge approaches in adaptive traffic signal control. Motivated by the need for improvement in adaptive control logic, this research has developed and tested a novel adaptive control strategy and tested its effectiveness in comparison with other existing signal control algorithms.

CHAPTER II

LITERATURE REVIEW

Adaptive control logic for isolated intersection is different from the one for traffic arterial and network. It is necessary to discuss the control logic in two different levels (intersection and network) individually. The following section reviews the current practice of adaptive control logic in each level and discusses the possibility of improvement.

2.1 Adaptive Control Logic for Isolated Intersection

Adaptive traffic signal control for isolated intersection is advantageous over the conventional type of control because no preset plans are specified in advance. The algorithms dynamically compute the signal timing plan based on real-time data obtained from upstream detectors. These timing plans continuously adjust cycle length, green split and phase sequence to provide better progression and minimize delay. From previous studies, adaptive control logic provides comparable or better performance than actuated control (12). In spite of those advantages, application of adaptive control is still characterized by some limitations which warrant improvement.

2.1.1 Uncertainty in Predicting Future Traffic Flows

Adaptive control strategies rely mainly on the prediction of the arriving flow (13). There are two types of prediction. The first is based on the real-time data measured in the field to estimate the movement of vehicles detected. The other is based upon historical data to predict the future arriving flow. For convenience of referencing, we call the former *estimation* and the latter *prediction*. While long-term optimization is ideal for reaching the global optimums, the real-time data based control relies on a number of short-term optimizations to reduce uncertainty in traffic demand and improve accuracy in computation. The short-term optimization, usually in the order of 30 to 60 seconds, makes it possible for all the optimizing processes to be based on estimation rather than prediction. For example, the COP algorithm used in RHODES optimizes the phase sequence every 30 to 40 seconds depending on the upstream detector's location. The OPAC strategy relies on data from the past 50 to 100 seconds (14), therefore its effectiveness is largely dependent on the accuracy of flow prediction. No matter how the traffic information is obtained, there will always be some difference between the predicted and the field condition. Hence, a desirable adaptive control strategy should reduce reliance on prediction as much as possible.

2.1.2 Difficulty in Arrival Time Estimation

A reliable estimation model must be developed to provide real-time traffic information for adaptive control. Vehicle arrival information is typically obtained from detectors placed upstream of the intersection, and the objective of estimation is to obtain the vehicle travel time between the upstream detector and the intersection stop line

(5, 14). For system optimization, the ability to estimate traffic conditions for a longer duration is desirable, but because of geometric constraints and uncertainties in vehicle arrivals, there is most always a tradeoff between the estimation duration and data accuracy. The geometric layout of the travel time estimation is shown in Figure 2.1.

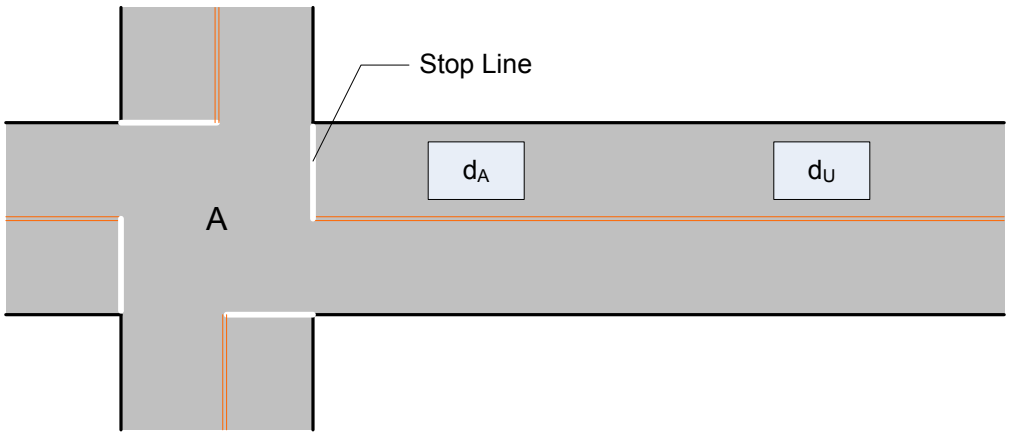
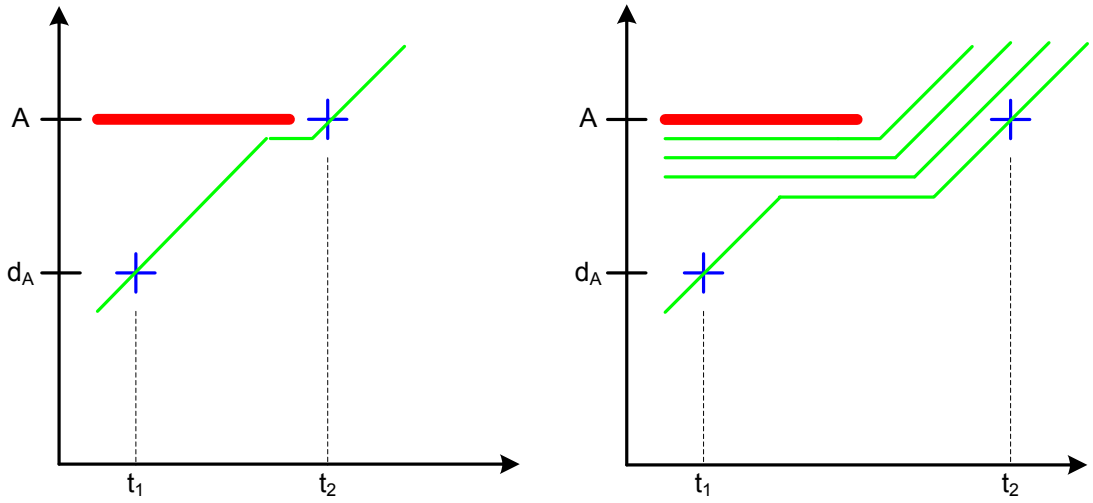


Figure 2.1 Geometric Layout for Vehicle Arrival Time Estimation

RHODES (15) splits travel time into two parts, the time between upstream detector d_U and arrival detector d_A, and the time from d_A to intersection A. Detector d_A is located several hundred feet upstream of intersection A in order to provide long enough "reaction time" for the system to adjust the signals. In heavy traffic conditions, however, the travel time from d_A to intersection A cannot be easily determined because it is largely affected by the existing queue and the signal status at intersection A. Let us consider two cases shown in Figure 2.2:



(a) Vehicle arrival without existing queue (b) Vehicle arrival with existing queue

Figure 2.2 Travel Time between d_A and A with Different Queue Sizes

$$\mu_i = \delta_i + \tau_i \quad (2.1)$$

Where,

I: Index for vehicles arriving at intersection A,

μ_i: Estimated arrival time of vehicle i to intersection A,

δ_i: Estimated arrival time of vehicle i at detector d_A,

τ_i: Estimated travel time between d_A and intersection A.

In Equation 2.1, δ_i is affected by where the vehicle is detected (d_U in Figure 2.1) and the distance to reach d_A. On the one hand, there are errors in predicting δ_i (15), on the other hand, the travel time τ_i is not merely to divide the distance between d_A and intersection A by the approaching speed of the vehicle. Different queue lengths remaining in the approach will affect τ_i, as depicted in Figure 2.2. In addition, the approaching speed is likely to change as the arriving vehicle gets close to the queue, which also affects the estimated vehicle arrival time to the intersection.

Similarly, OPAC defines travel time as the time for the vehicle to travel between the upstream and the downstream signals (14) and the difficulty in travel time estimation must also be dealt with in congested traffic. This problem exists in the optimization process of every adaptive control system today. However, very limited up to date work is reported in the literature as to how the varying queue length will affect the arrival time estimation.

2.1.3 Lack of Self-adjusting Mechanism

The effectiveness of adaptive control strategies also relies on reasonable estimation of system parameters governing queue formation/dissipation, start-up delay, and vehicle clearance. The start-up delay and the vehicle releasing rate may be different from time to time due to the influence of construction, incident, and even the weather condition. The differences cannot be accounted for if static parameters are used in the model, and the cumulated error can become large enough to offset any systems advantage over other types of control. However, most of the existing adaptive control strategies do not contain a self-adjusting mechanism.

In summary, improvements over adaptive control logic for isolated intersection should include the following:

1. Data used in optimization come from real-time detection and estimation. Use of long-term based prediction from historical data is not desired;
2. The arrival time estimation model is reliable and adaptive to a variety of traffic conditions; and

3. The control logic must contain a self-adjusting mechanism to monitor system operation and make corrections.

2.2 Adaptive Control Logic for Traffic Network

Compared with isolated intersections, traffic networks prefer coordinated timing plans usually determined by four parameters: cycle length, green split, offset and phase sequence. Since the number of possible signal timing plans increased exponentially against the incremental number of intersections involved in the network, the algorithm finding the global best solution, such as Dynamic Programming, cannot yield an optimal result in a short time. Most adaptive control strategies employed near-optimal searching algorithms provide the approximate best solution in real-time.

Among the near-optimal searching algorithms, Genetic Algorithm (GA) has shown its compatible ability in many areas. Although GA became popular through the work of Holland in 1975 (16), practical application of it started in the late 1980s with the dramatic increase in desktop computational power. The application of GA in traffic signal timing optimization started from the early 1990s. Foy et al. (17) introduced a method using GA to optimize a four-intersection network by minimizing vehicle delay. Hadi and Wallace presented a similar GA problem in 1993 (18) and suggested to combine with TRANSYT-7F to optimize all four parameters which are cycle time, phase sequence, green split and offset. Park et al. proposed in 1999 a GA approach to optimize traffic network, especially in oversaturated conditions (19). In Park's model, the fractional values are first used to represent all the four parameters of signal timing. His model was later combined with CORSIM simulation to optimize a 31 nodes traffic

network (20). Though the ability of GA to optimize signal timing has been demonstrated, the above efforts are all focused on offline optimization, in which the computation burden generally is too large to prevent applications in real-time. For example, in Park's model, the optimization time in each step (update interval) varies from several minutes to several hours depending on the complexity of the traffic network and the calculation speed of the computer processor. Another obstacle prevents the offline system to be applied in real-time is data collection. The traffic information used in optimization, such as the traffic demand, should be obtained in real-time from field instead of using prerecorded data. Other problems such as how to shift from one timing plan to another should also be addressed. In 2004, Lee et al. (21) introduced a real-time application to provide optimized acyclic signal operation through rolling horizon method based on the Genetic Algorithm. While this work represents an improvement, the model is relatively simple in details to collect and process data for only a 3-intersection linear system.

In short, GA has shown its ability in off-line traffic network optimization but its application in real-time traffic control is very limited. To implement adaptive control logic with GA, the following obstacles should be overcome:

1. Optimization should be finished in relatively short time to fulfill the requirement of real-time operation;
2. Data collection, such as volume, headway and etc., should be accurate enough; and
3. Switching from one timing plan to another should be smoothly and quickly.

CHAPTER III

METHODOLOGY

The adaptive control logic described in this proposal is divided into two parts: one for the isolated intersection, and the other for the entire traffic network. In an isolated intersection optimization, the logic called Piecewise Optimum Delay Estimation (PODE) (22) is used while in traffic network optimization, the logic named Genetic Algorithm Based Network Optimization in Real-time (GABNOR). The following section will discuss the logic in detail.

3.1 PODE for Isolated Intersection

PODE is a piecewise decision strategy that optimizes signal operations in short-term intervals based on field detection. Similar to other adaptive control schemes, PODE utilizes data input on vehicle arriving, system queue, and at the stop line (Figure 3.1). The special features of PODE include flexible interval length and self-adjustment. The length of piecewise optimization interval can vary from several seconds to a few dozen, and the self-adjusting program is performed at the beginning of each interval to correct estimation error and reset system parameters if necessary. All the possible movement combinations are treated as candidate phases and will be assigned to each interval. A lane group is associated with each phase which is the collection of lanes receiving the green

signal. For each interval, PODE will exam all possible interval length and phase combinations (for instance, there are 120 combinations if the range of interval length is 15 seconds and in an eight-phase operation). The performance index of each combination is compared and optimized phase and length is selected. Interval by interval, the sequence of the phases and the length of each interval are dynamically determined by the algorithm to achieve system optimization according to real-time traffic data.

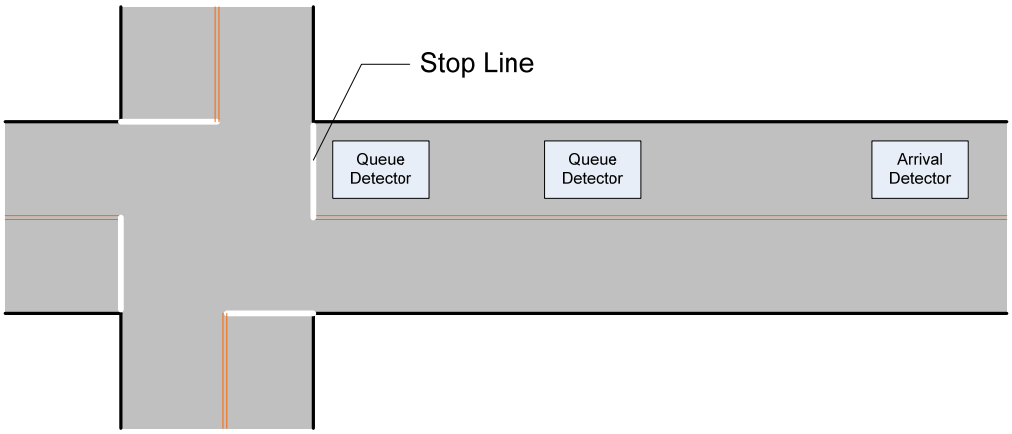


Figure 3.1 PODE Detectors

The objective function of adaptive control strategies is commonly used to minimize system wide vehicle delay or to maximize the intersection utilization, such as the throughput of the whole intersection. In PODE, either objective function can be selected and we chose the former for the model discussion in this paper. The selected objective function serves as the basis for comparison among different optimization phase sequence and interval lengths.

The total system delay during the optimization is the summation of vehicle delay in the existing queue at each second in each lane at each approach, as shown in Equation 3.1.

$$D_j = \sum_{approach} \sum_{lane} \sum_{t=1}^{c_j} \lambda_{j,t} \tag{3.1}$$

Where,

j: Optimization interval index,

D_j: Delay of intersection during interval j,

c_j: Length of interval j (sec),

t: Time index at one second incremental inside interval, t = 1 to c_j,

$\lambda_{j,t}$: Queue size at time t in interval j (veh).

For simplicity of presentation, we use only one lane in an approach to explain the modeling details. For any optimization interval j, we can get the following equation:

$$\lambda_{j,t} = \lambda_{j,t-1} + \alpha_{j,t} - \gamma_{j,t} \tag{3.2}$$

Where

$\alpha_{j,t}$: Estimated arriving vehicles at the t second in interval j (veh),

$\gamma_{j,t}$: Estimated vehicle releases at the t second in interval j (veh).

Through iteration, we can rewrite the delay calculation equation as the following:

$$\sum_{t=1}^{c_j} \lambda_{j,t} = c_j \cdot \lambda_{j,0} + \sum_{t=1}^{c_j} \left(c_j - t + 1\right) \cdot \left(\alpha_{j,t} - \gamma_{j,t}\right) \tag{3.3}$$

Clearly, the system wide delay is dependent on the estimation of $\lambda_{j,0}$, $\alpha_{j,t}$ and $\gamma_{j,t}$. The research explains the estimation process of these parameters one by one.

3.1.1 Initial Queue Size

$\lambda_{j,0}$ is the initial queue size at the beginning of each optimization interval. It is also the final queue size at the end of last optimization interval, that is

$$\lambda_{j,0} = \lambda_{j-1,c_{j-1}} . \tag{3.4}$$

When the system is first initialized, the queue size is set to zero, that is $\lambda_{1,0} = 0$. As the optimization progresses, the queue size is tracked interval by interval.

Since the number of arriving and releasing vehicles is each estimated, there is a likely difference between the estimate and the actual data. This will result in error in queue size estimation. To avoid accumulation of errors, PODE has built in a procedure to calibrate the queue size, which is part of the self-adjusting program.

The queue adjusting procedure is performed with the help of a detector at the stop line and a queue detector which is placed in a short distance upstream (usually 200 ft) to track the input and output within the section. Our experiments showed that a queue tail exceeding the queue detector caused little difference in PODE optimization. The adjustment will not be made until the lane group has lost the right of way (from green to red) for a time period long enough to allow the queue to be stable. To reduce system input and hardware reliance, PODE does not require precise queue length measurement and uses only approximate queue size information in the self-adjusting algorithm.

3.1.2 Vehicle Arrivals

Because of the difficulty in determining the arrival time at the stop line, we estimate the arrival time at the queue tail instead. If there is no waiting vehicle, the stop line naturally becomes the queue tail. With the estimated queue size $\lambda_{j,t}$ and vehicle's arrival information, the arrival time at the queue tail is estimated with the help of a car-following model.

Once a vehicle is detected by the upstream detector, its speed and location are tracked every second by the system according to the following equation:

$$l_{j,t} = l_{j,t-1} - v_{j,t-1} \tag{3.5}$$

Where:

$l_{j,t}$, Distance between the stop line and the arriving vehicle at time t in interval j,

$v_{j,t}$, Speed of arriving vehicle at time t in interval j.

The speed of each vehicle at time t is estimated by the car-following model (which is not described here as it is well known to researchers). The time lag of response and sensitivity in this model (23) are PODE's system parameters and are adjusted periodically by comparing the estimated versus the actual arrivals obtained from the self-adjusting program discussed in the previous section. For example, when the estimation is consistently larger than the actual for five intervals, the response time lag is reduced and the sensitivity increased. By means of this model, the location of the arriving vehicle at any time t is determined.

At the same time, as the queue is being released the length of queue is tracked by using a pre-established queue releasing wave speed obtained from field observations. A function is established to determine queue length with the input of queue size and time. At any time t of optimization interval j, we will compare the queue length $q_{j,t}$ with each arriving vehicle's location $l_{j,t}$ and update the arrivals. For example, if the vehicle's location has not reached the queue tail, no additional vehicle will be counted to join the queue.

3.1.3 Vehicle Releases

Vehicle releases $\gamma_{j,t}$ are determined at each second depending on the queue size $\lambda_{j,t}$, the arrivals $\alpha_{j,t}$, and the selected phase p_j for the optimization interval j. The number of vehicles that can be released into the intersection is constrained by the following:

1. If no green time is assigned, the releases $\gamma_{j,t}$ for the corresponding movements will be zero;
2. If green time is assigned and there is an existing queue that cannot be cleared in this second by estimation, the release $\gamma_{j,t}$ for the corresponding lane will be determined by the maximum queue release rate r; and
3. If the queue size $\lambda_{j,t}$ is small (including zero) and can be cleared in this second by estimation, the number of released vehicles will be the summation of queue size $\lambda_{j,t-1}$ and vehicle arrivals $\alpha_{j,t}$ during the same second.

Similarly, the basic vehicle releasing rate r is measured from the field considering the start-up delay. Collectively, the calculation of vehicle releases can be written as

$$\gamma_{j,t} = \begin{cases} 0 & \text{NOT in the lane group of phase } p_j \\ \min\left(r, \lambda_{j,t-1} + \alpha_{j,t}\right) & \text{in the lane group of phase } p_j \end{cases} \quad (3.6)$$

Where,

r: Maximum queue release rate measured from field,

p_j: Selected phase for optimization interval j.

3.1.4 Optimization Process

For any given interval length c_j and selected phase p_j, Equation 3.3 can be rewritten as

$$\sum_{t=1}^{c_j} \lambda_{j,t} = \text{Delay}\left(c_j, p_j\right) \quad (3.7)$$

Since the selected objective of PODE algorithm is to find the minimum system wide delay among different possible interval lengths, the measure of performance, M_j, is

the system delay D_j divided by the optimization interval length c_j. The objective function for M_j can be written as follows:

$$\min(M_j) = \min\left(\frac{D_j}{c_j}\right) = \min\left(\sum_{approach}\sum_{lane}\frac{\text{Delay}(c_j, p_j)}{c_j}\right). \quad (3.8)$$

By varying c_j and p_j in Equation 3.8, M_j is calculated sequentially for all possible combinations of c_j and p_j. In each iteration, M_j is recorded as it relates to c_j and p_j (exemplified in Table 3.1). At the end of the iterations, a comparison is made in the last row (maximum c_j) of the table to select the best phase p_j corresponding to the minimum performance index. Next, the M_j values within the selected phase column are compared and the c_j from the cell having the minimum M_j value will selected as the next interval to use.

Table 3.1 Sample Performance Index Table

pj / cj (sec)	1	2	3	4	5**	6	7	8
5	N/A	N/A	N/A	N/A	110.26	N/A	N/A	N/A
6	N/A	N/A	N/A	N/A	110.30	N/A	N/A	N/A
7	N/A	N/A	N/A	N/A	110.31	N/A	N/A	N/A
8	N/A	N/A	N/A	N/A	110.33	N/A	N/A	N/A
9	N/A	N/A	N/A	N/A	110.36	N/A	N/A	N/A
10	N/A	N/A	N/A	N/A	110.38	N/A	N/A	N/A
11	123.67	91.99	129.67	86.63	110.43	116.58	104.59	99.72
12	123.80	92.07	129.83	86.32	110.55	116.73	104.67	99.80
13	124.01	92.32	129.99	86.00	110.62	116.89	104.73	99.85
14	124.27	92.88	130.34	85.85	110.86	117.22	104.97	100.10
15	124.44	93.03	130.62	85.59	111.04	117.47	105.11	100.23
16	124.65	93.26	130.91	85.37	111.31	117.70	105.35	100.29
17	124.73	93.51	131.03	85.23*	111.67	117.91	105.52	100.31
18	124.96	93.77	131.27	85.34	111.92	118.06	105.74	100.24
19	125.37	94.01	131.49	85.48	112.20	118.28	105.98	100.18
20	125.51	94.34	131.77	85.60	112.48	118.43	106.12	100.09

* Selected C_j and P_j
** Current phase when optimizing

Second Comparison

First Comparison

The basic steps in PODE are as follows:

1. Collect and record vehicle arrival information from the upstream detector at each second;
2. Adjust the initial queue size $\lambda_{j,0}$ at the beginning of each optimization interval, according to detection by the queue detector and stop line detector;
3. Adjust system parameters, such as maximum queue release rate, etc., if necessary;
4. Design possible combinations of phase p_j and interval length c_j;
5. Estimate the vehicle arrivals $\alpha_{j,t}$ by means of tracking vehicle location $l_{j,t}$ and queue length $q_{j,t}$;
6. Estimate the vehicle releases $\gamma_{j,t}$ with selected phase p_j, queue size $\lambda_{j,t}$ and vehicle arrivals $\alpha_{j,t;}$
7. Repeat steps 5 and 6 till the vehicle arrivals and releases at any second in interval j have been determined;
8. Calculate the performance index M_j based on estimated $\lambda_{j,0}$, $\alpha_{j,t}$ and $\gamma_{j,t;}$
9. Repeat step 5-9 until all the feasible combinations of p_j and c_j have been checked;
10. Compare performance index M_j among different p_j at the maximum c_j, choose the p_j with minimum M_j;
11. Compare performance index M_j among different c_j within the chosen p_j. Select c_j with minimum M_j; and
12. Return selected c_j and p_j as optimized result.

A flow chart is shown in Figure 3.2 to help understand the PODE optimization process. A sample optimization result is shown in Figure 3.3. There are eight potential phases to choose from in each optimization interval. It can be seen from the chart that interval j and interval j+1 are both adopting phase 2 which gives the lane group in north and south bound the right of the way. After that, phase 7 is assigned to the interval where the north bound left turn and through movements are given the green indication. Notice that a switch between phase 2 and phase 7 occurred when optimization interval shifted from j+1 to j+2. This means a clearance time (yellow plus red) is given at the beginning of interval j+2 to end the green for the south bound traffic while the green is continued for the north bound through movement. Notice that the interval length is flexible and the length of a phase is not limited by the interval length. For instance, the length of phase 2 is 20 seconds since it is selected by both intervals j and j+1.

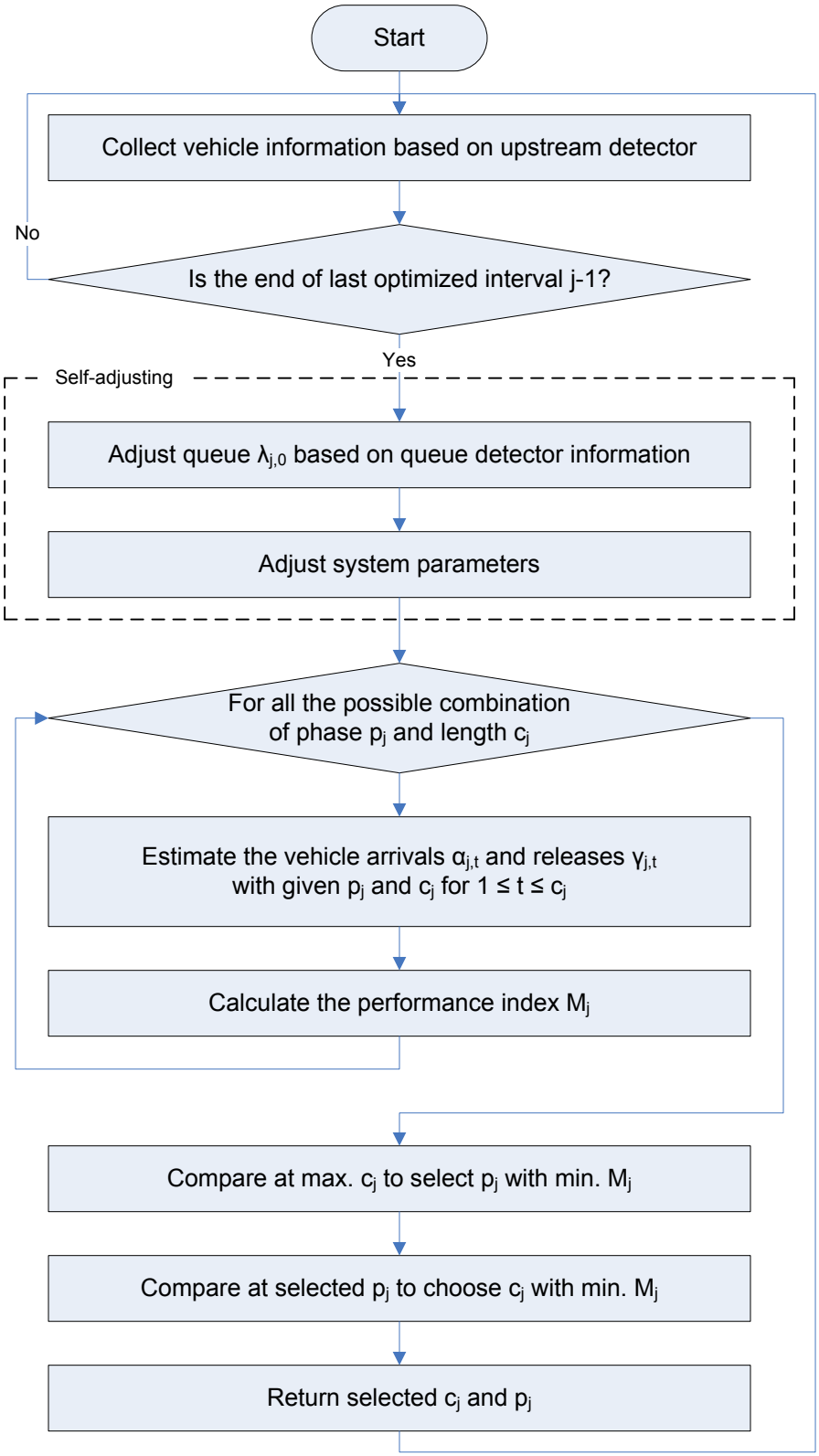


Figure 3.2 Flow Chart of PODE Algorithm

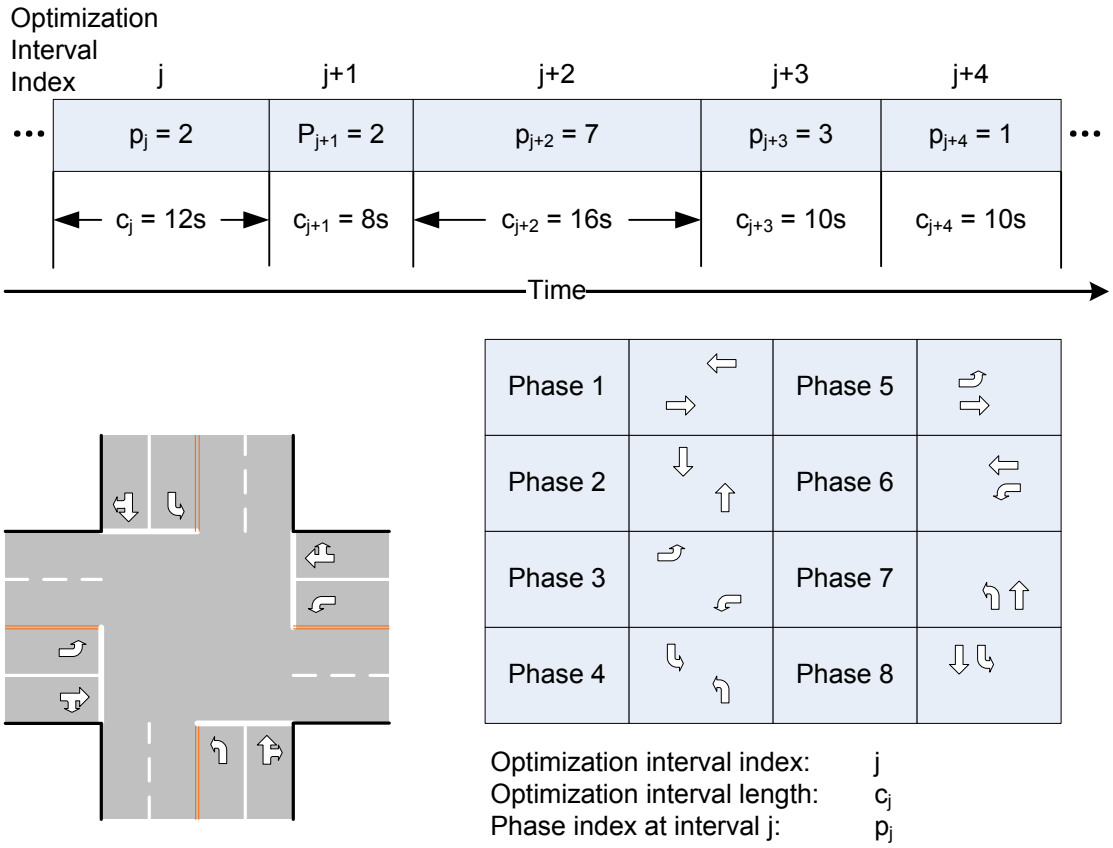


Figure 3.3 Sample Optimization Result

3.2 GABNOR for Traffic Network

Similar to other traffic optimization systems, the objective function of GABNOR is to minimize the total vehicle delay of the traffic network in the evaluation time horizon. Briefly, GABNOR optimize traffic network by searching among possible timing plans and select the optimal one. Since different timing plans differ in cycle length and offset, the length of evaluation time horizon will be different also. In order to compare different timing plans, we adopt the average vehicle delay per second as the objective to be minimized as shown in equation 3.9

$$\forall p \in P, \quad Minimize \quad \frac{n.Delay(p)}{p.length} = \sum_{t=0}^{p.length} \sum_{i}^{n.intersections} \sum_{m}^{i.movements} m.delay[t] \tag{3.9}$$

Where,

P: collection of all possible timing plans,

p: timing plan object,

n: network object,

n.Delay(p): method of network *n* to calculate delay with given *p,*

p.length: the length of evaluation time horizon for timing plan *p,*

t: time index,

n.intersections: the collection of all intersections in network *n,*

i: intersection object,

i.movements: the collection of all movements in intersection *i*,

m: movement object,

m.delay[t]: delay of movement *m* at time *t.*

As the growth of possible number of timing plans is exponential to the increase of number of intersections in the network, exhausting search is impossible to solve this problem with current computing power. GA, in this case, can perform better searching result in limited time.

Similar to other GA based algorithms, GABNOR includes a GA Engine to optimize signal timing and evaluate each timing plan by using a Mesoscopic Internal Simulator. A special process to help GABNOR shift from one timing plan to another is also included. We will introduce these key components in the following section.

3.2.1 Data Collection and Analysis

To perform on-line optimization, vital information including volume for each movement, vehicle headway, startup delay and etc. should be collected in real-time continuously. Among these data, the volume for each movement of an intersection is the most difficult one to obtain. In GABNOR, we will calculate vehicle turning movements by tracking detectors' status and traffic signal code second-by-second. To illustrate the very basic idea of this method, we will take a four-leg intersection as an example. As shown in Figure 3.4, there is only one lane in each leg for each direction, which means each lane is shared by left turn, right turn and through vehicles. Detectors are placed at the stop bar in all approaches of this intersection. There are two types of detectors: input detectors (white) which provide the detection of vehicles approaching the intersection and output detectors (gray) which provide the detection of vehicles leaving the intersection. For each turning movement, a fixed detector pair is activated. For example, any northbound left turn vehicle will first activate detector 1 then detector 8. Thus, we can get a table of detection sequence for each turning movement as shown in Table 3.2. The basic idea of this method is to find out vehicle turning movements based on detection sequence table according to the recorded detections and signal status.

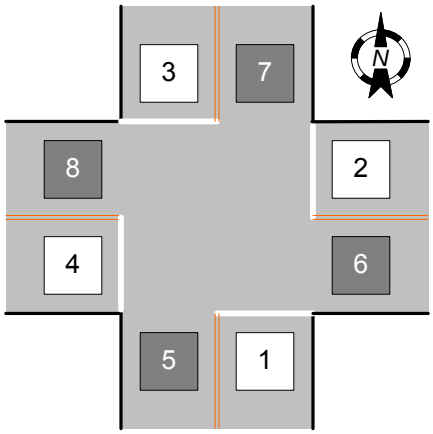


Figure 3.4 Detector Configuration for Typical Four-leg Intersection

Table 3.2 Turning Movement Detection Sequence Table

Input Detector	Output Detector	Movement	Abbreviation
1	7	Northbound Through	NT
1	6	Northbound Right Turn	NR
1	8	Northbound Left Turn	NL
2	8	Westbound Through	WT
2	7	Westbound Right Turn	WR
2	5	Westbound Left Turn	WL
3	5	Southbound Through	ST
3	8	Southbound Right Turn	SR
3	6	Southbound Left Turn	SL
4	6	Eastbound Through	ET
4	5	Eastbound Right Turn	ER
4	7	Eastbound Left Turn	EL

At signalized intersection, only part of the vehicles approaching the intersection can get the right of way at any moment. The signal information can help us to filter the candidate movements. However, the process to identify turning movements by searching detection pairs can still be very complicated when there are multiple combinations available caused by shared lane or other factors. To identify the turning movements under any circumstance, there are three modules in the process, Input Detection Recording

Module, Output Detection Matching Module and Input Detection Cleanup Module. As shown in, the detail processes of these three modules are introduced below in Figure 3.5.

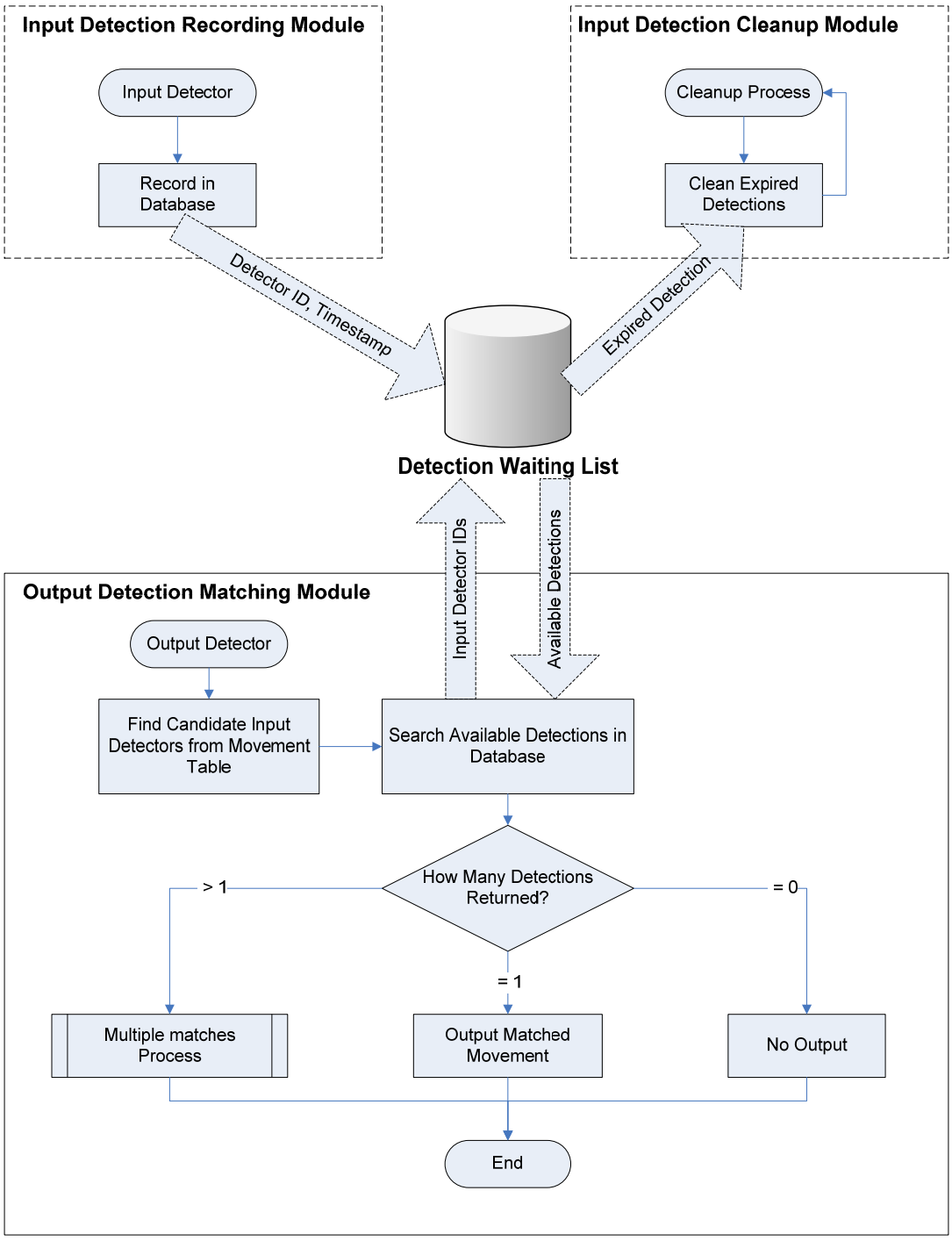


Figure 3.5 Flowchart of Turning Movement Identification Process

3.2.1.1 Input Detection Recording Module

This module is trigged by the detection from input detectors. Once the status of an input detector changes from activated to deactivated, which means that a vehicle has left the detector. The detector's ID and the deactivation timestamp will be recorded and saved in the waiting list. The saved records will be used in the other two modules.

3.2.1.2 Output Detection Matching Module

This module is trigged by the detection from output detectors. An output detection means a vehicle left the intersection and there should be one and only one input detection matched with it. However, in practice, it will not always just return one input detection from the waiting list. As shown in Figure 3.5, there are three possible cases need to be taken care of.

Case I: There is no matched detection in the waiting list.

This situation can be caused by either a miss detection on input detectors or false detection from output detectors. Since any detection in the future will not be helpful to solve this problem, we will not output anything except mark the output detection as an error.

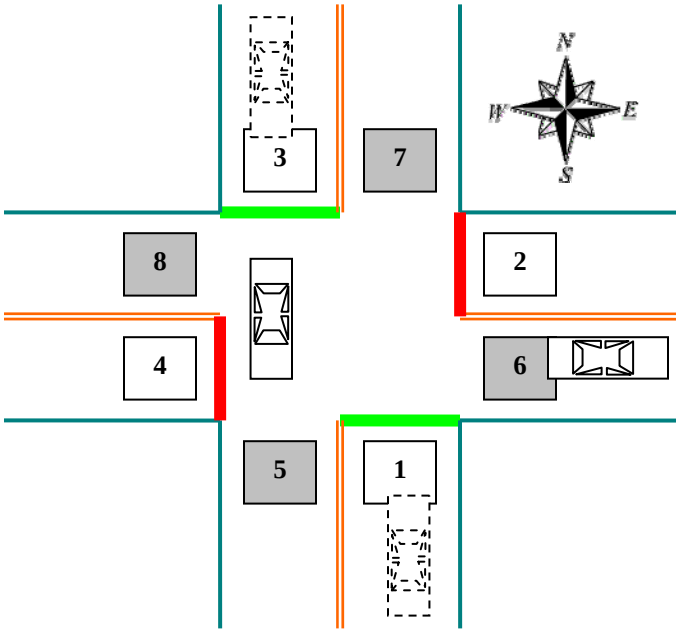
Case II: There is only one matched detection in the waiting list.

This is the best condition for detection match. We will simply search the turning

movement table and find the corresponding movement based on the detection pair. System will output this movement as the result and mark the input and output detection as identified.

Case III: There is more than one matched detection in the waiting list

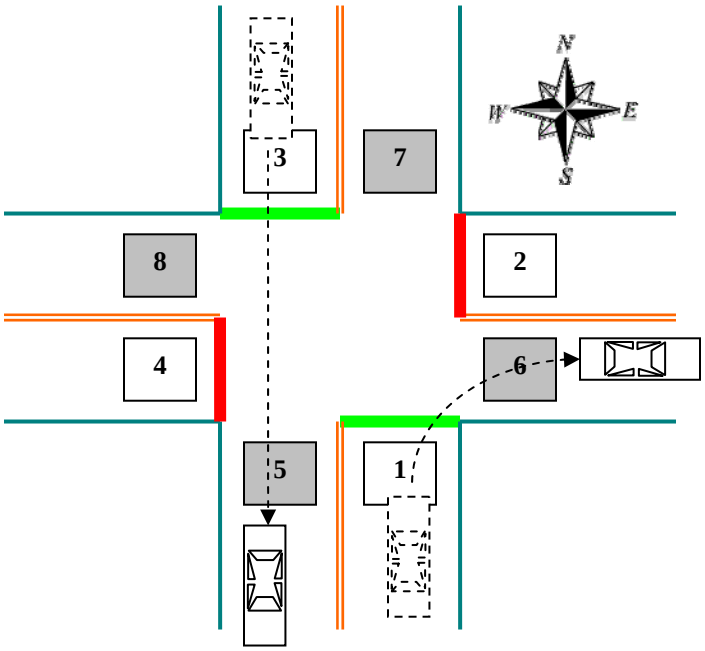
This is the most complicated situation we need to handle in the algorithm. It can be caused by many reasons, such as false detections of input detector, vehicles not cleared in the intersection and etc. To illustrate the process when we have more than one candidate, a simple example will be discussed. Consider a four-leg intersection with one lane on each direction as shown in Figure 3.6. There is a vehicle moving from south to east (northbound right turn) and another vehicle moving from north to south (southbound through). At any moment, the vehicles' position is shown in Figure 3.6 (a). There are two candidate input detections both available for the output detection. The system will then hold all detections and not make a decision until the southbound vehicle leaves the intersection. As shown in Figure 3.6 (b), there are now two output detections, one from detector 5 and the other from detector 6. Though there are still two possible matches for output detection 6, there is only one input detection to match with output detection 5. The algorithm will output the movement as southbound through. Consequently, we only have one input detection to match with output detector 6. The algorithm will then output northbound right turn and all the output detections have been matched and the turning movements are determined.



Input Detections: 1, 3
Output Detection:6

Possible Matches:
1->6 Northbound Right Turn
3->6 Southbound Left Turn

(a) Unsolvable Situation Caused by Vehicle not Cleared the Intersection



Input Detections: 1, 3
Output Detection:5, 6

Possible Matches for 6:
1->6 Northbound Right Turn
3->6 Southbound Left Turn

Possible Matches for 5:
3->5 Southbound Through

(b) Solvable Situation when Vehicle Cleared the Intersection

Figure 3.6 Example of Multiple Matches

3.2.1.3 Input Detection Cleanup Module

This module is an independent process which cleanups the expired input detections in the detection waiting list. Input detections which have no matched output detection for a given time should be removed from the waiting list to keep the system working properly. These non-matched detections may be caused by false detection from input detectors or miss detection on output detectors. The expiration time is preset by the system (usually the length of phase in experience) and the cleanup process is performed every second. This module is very crucial in helping solve the problem of multiple matches in the algorithm.

3.2.2 Genetic Algorithm Engine

A GA engine is built in GABNOR to help optimizing traffic networks. The encoding method of this engine and its parameter selection are introduced in the following.

3.2.2.1 Signal Timing Plan Encoding

Before introducing the GA used in GABNOR, we will introduce the encoding mechanism to help understand the modeling process. To avoid generating an unsuitable timing plan and reduce the optimization time, a green split is calculated from the traffic volume information instead of being encoded. The other parameters, such as cycle length, offset and phase sequence, are encoded in a binary string using fractional value. In addition, half cycle is supported in GABNOR and is encoded also. An example of encoding for a timing plan of a two-intersection traffic network is shown in Figure 3.7. In this example, six bits are used to represent the common cycle length of the intersections

in the network. For each intersection, 12 bits are used to encode half cycle, offset and phase sequence. Among these 12 bits, one bit is used to represent whether this intersection is half cycle or not, seven bits are used to represent the offset, and the last four bits are used to represent the phase sequence (lead or lag). With the encoded binary string shown in Figure 3.7, we can interpret to a signal timing plan for the whole network. The cycle length of the first intersection is 128 seconds and the offset is 74 seconds, while the cycle length of second intersection is 64 seconds due to half cycle and the offset is 37 seconds. The phase sequences are as shown in the figure. With this encoding mechanism, we can easily convert between a signal timing plan and its corresponding binary string.

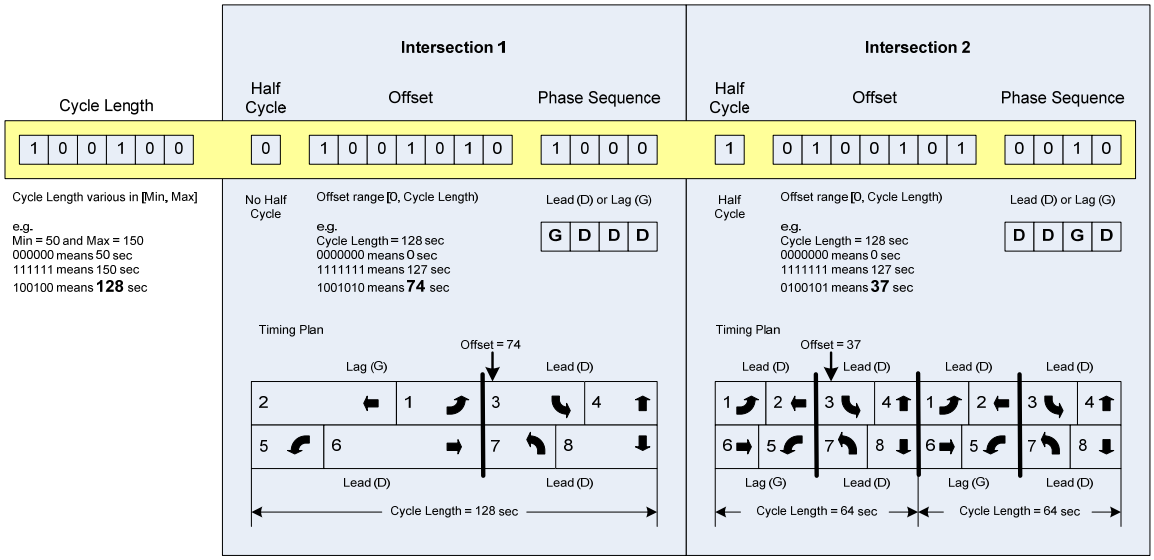


Figure 3.7 Sample Timing Plan Encoding

3.2.2.2 Genetic Algorithm and its Parameters

Generally, Genetic Algorithm includes the following processes:

1. *Initialization*

2. *Evaluation*

3. *Reproduction*

4. *Termination*

Initialization process randomly establishes the first generation of chromosomes, which is a collection of different traffic timing plans in GABNOR. After that, GA will perform the *Evaluation* process which will evaluate the fitness of each chromosome. In GABNOR, the fitness value is the total delay of traffic network estimated by a mesoscopic internal simulator. After all the chromosomes have been evaluated, the *Reproduction* process will produce a new generation based on the fitness value of each chromosome. There are three operations in the reproduction process: *Selection*, *Crossover* and *Mutation*. *Selection* will pick up two chromosomes according to a given scheme. In GABNOR, we adopted the commonly used Rank Selection plus Elitism approach to perform this operation. *Crossover* will generate child chromosome(s) with parent chromosomes selected in *Selection* operation. In GABNOR, Uniform Crossover is used to generate new timing plans with given crossover probability. The *Mutation* operation will randomly change the bits in chromosome by given mutation probability to keep variations in the chromosomes. Once a new generation has been built, the *Evaluation* operation will be repeated until the termination criteria are fulfilled. In GABNOR, the major termination criteria is whether the calculation time has exceed a given upper limit or not, which will ensure the optimal result can be applied in real-time.

Three major parameters are used in the optimization process of GA, population size, crossover probability and mutation probability. Population size is the number of

individual chromosomes in each generation. Since we are doing real-time optimization, a trade-off must be made between larger populations that may not converge in time and smaller populations that may converge to local optimum. The population size in GABNOR is selected according to a general rule of thumb that the population size is close to the length of binary string of chromosome. We also set the crossover probability as 0.7 and mutation probability as 0.1 according to Kovvali and Messer's study (24). We noticed that in Park's work (19) the optimized results are found insensitive to the parameters of GA for traffic network optimization and there is no exception in GABNOR.

3.2.3 Parallel Fitness Evaluation

The most time-complex process in GA is the fitness evaluation. For a real-time optimization system, GABNOR must optimize signal timing in a very short time span. With the help of the potential parallel feature of GA, we designed and implemented a Parallel Fitness Evaluation process to calculate the fitness value of multiple chromosomes simultaneously.

Parallel Fitness Evaluation is a client-server based program. The server will maintain a pool of unevaluated chromosomes (timing plans) and the client will send the request to the server asking for evaluating. If there is any chromosome available in the pool, the server will select one and send it to the client. The result will be sent back to the server after the client finishes the evaluation. With the help of multi-threading (25) and asynchronous socket communication (26), the server can handle multiple clients' requests at the same time. Theoretically, the evaluation speed will increase linearly to the number of clients.

3.2.4 Mesoscopic Internal Evaluator

To evaluate different timing plans, an internal mesoscopic evaluator is used in GABNOR. With given phase codes of evaluation time horizon, the evaluator estimates vehicle delay based on input-output modeling. From Equation 3.9, the total network delay is the summary of delay at each movement of each intersection. To calculate the delay, we track the arrival flow, departure flow and delayed vehicles of each movement second by second. The arrival flow is determined by the upstream movements' departure flow, the travel condition on the feed approach and it will not be affected by the signal timing or queue in the approach. The departure flow means the vehicle release at each second of the movement. It is determined by signal timing, delayed vehicles, capacity of lanes and etc. As shown in Figure 3.8, with given arrival flows and signal timing (phase codes), the departure flow can be estimated second by second. The delayed vehicles at each second mean the vehicles slowed down by signal timing, queue or other factors. The summary of delay at each second is the total delay of that movement as shown in Figure 3.8. The estimation of each variable will be discussed in the next part.

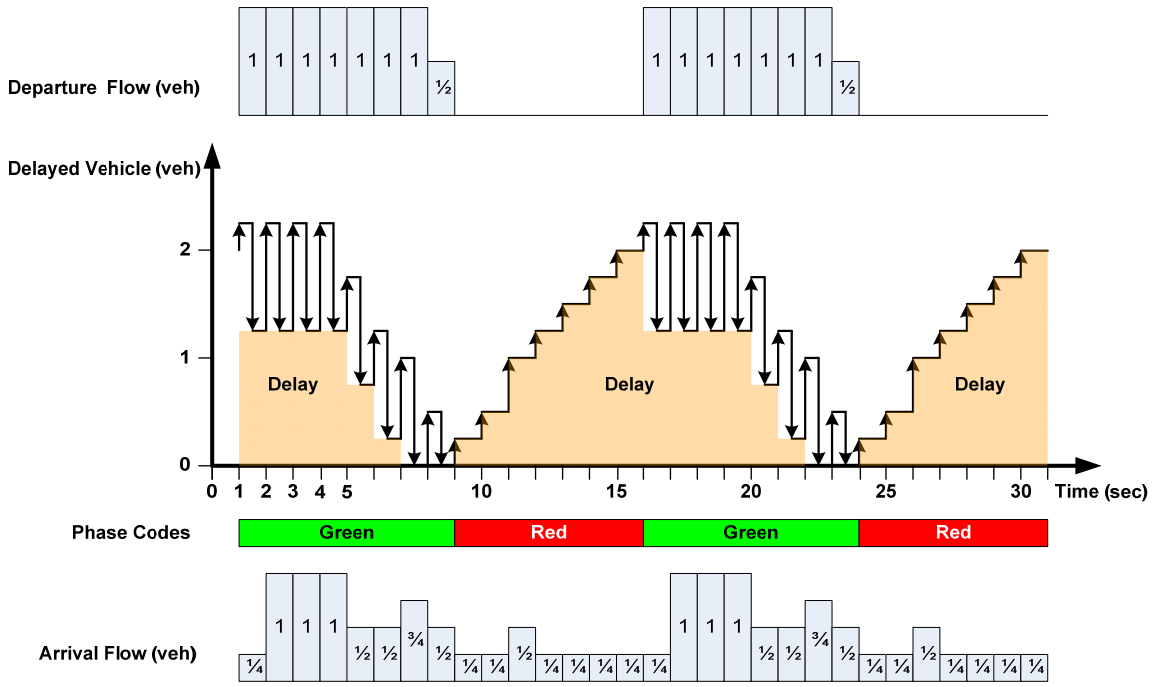


Figure 3.8 Delay Calculation in GABNOR

3.2.4.1 Arrival Flow

The arrival flow is determined by the departure flow of upstream movements and the travel condition of its feed approach and upstream movements. If there is no upstream movement exists, the arrival flow will follow the Poisson distribution. Platoon dispersion used in equation 3.10 will not be introduced here in order to keep the simplicity of the model. The calculation of arrival flow is as follows

$$Suppose\ \ app = m.feedapproach$$

$$m.arrival[t] = \begin{cases} \sum_{u}^{app.feedmovements} u.departure[t - app.traveltime - u.traveltime] & if\ \ app.feedmovements\ exist \\ \min\ \kappa,\ \dfrac{\Gamma(\kappa+1,\lambda)}{\kappa!} \geq rand(t), where\ \ \lambda = \dfrac{m.volume}{3600}\ \ and\ \kappa = 0,1,2,\ldots & if\ \ app.feedmovements\ not\ exist \end{cases}$$

(3.10)

where,

m: movement object,

m.feedapporach: the feed approach of movement *m,*

t: time index,

m.arrival[t]: the arrival flow of movement *m* at time *t,*

app.feedmovements: the collection of feed movements of feed approach *app* of movement *m,*

u: movement object for upstream movement,

u.departure[t]: the departure flow of *u* at time *t,*

app.traveltime: the average travel time on approach *app* from field data,

u.traveltime: the average travel time of movement *u* from field data,

rand(t): random value between 0 and 1 generated by random seed *t,*

m.volume: the volume of movement *m* from field data,

Γ(κ,λ): Incomplete gamma function which is $\int_{\lambda}^{\infty} x^{\kappa-1}e^{-x}dx$.

3.2.4.2 Departure Flow

The departure flow is determined by the phase code, delayed vehicles, the capacity of lanes and other factors. The calculation of departure flow can be described by a series of criteria. Suppose we need to determine the departure flow of movement *m* at time *t*, or *m.departure[t]*, the criteria are as follows:

$$\begin{aligned} &m.departure[0] \le m.arrival[0] \\ &m.departure[t] \le m.delay[t-1] + m.arrival[t] \qquad for\ t > 0 \end{aligned} \tag{3.11}$$

where,

m.delay[t]: the delayed vehicles of movement *m* at time *t,*

m.arrival[t]: the arrival flow of movement *m* at time *t.*

Besides the above criteria, the departure flow is also limited by the feed approach capacity, which can be presented by

$$m.departure[t] \le \sum_{l}^{m.feedlanes} l.saturationflowrate \tag{3.12}$$

Where,

l: lane object,

m.feedlanes: the collection of feed lanes of movement *m,*

l.saturationflowrate: the saturation flow rate of lane *l* from field data.

If the phase code of the intersection at time *t* doesn't allow movement *m* to release vehicles, the following criteria needs to be added:

$$m.departure[t] \le 0 \tag{3.13}$$

The departure flow is the maximum value that accomplishes all the above criteria.

3.2.4.3 Delayed Vehicles

Delayed vehicles means the number of vehicle being delayed at each movement. It is determined by the arrival flow and the departure flow as follows:

$$\begin{aligned} &m.delay[0] = m.arrival[0] - m.departure[0] \\ &m.delay[t] = m.delay[t-1] + m.arrival[t] - m.departure[t] \quad for \;\; t > 0 \end{aligned} \tag{3.14}$$

Where,

m: movement object,

t: time index,

m.delay[t]: the delayed vehicles of movement *m* at time *t,*

m.arrival[t]: the arrival flow of movement *m* at time *t,*

m.departure[t]: the departure flow of movement *m* at time *t.*

3.2.5 Signal Transition between Timing Plans

Whenever a new timing plan is generated, switching from the current timing plan to the new one is necessary. There are two major methods used in practice, one called Dwell and the other called Shortway. Dwell will hold the current signal status until synchronized. It is faster than Shortway method that most transition can be finished in one cycle. However, since the signal is held, it will block other directions traffic until it is synchronized. According to the study performed by Shelby, Bullock and Gettman (27), Dwell transition is the most disruptive transition method over a range of conditions. Shortway, on the contrast, provides superior performance compared with Dwell, but usually needs longer transition time. With advanced computing technology, we can choose a transition method intelligently to reduce the effect of transition period to traffic flow.

Since GABNOR is a real-time optimizer, moving smoothly and quickly from one timing plan to another is essential for traffic operation. On the other hand, the transition will cost time and affect the performance of the whole traffic network. To evaluate the timing plan, the transition time should also be included, thus the phase codes for the transition time needs to be calculated. To illustrate the determination of phase codes for transition, let us consider the transition of one intersection as shown in Figure 3.9 with the following notations:

C_1: Cycle length of current timing plan,

F_1: Offset of current timing plan,

C_2: Cycle length of new timing plan,

F_2: Offset of new timing plan,

X: Length of time interval where phase codes needs to be decided.

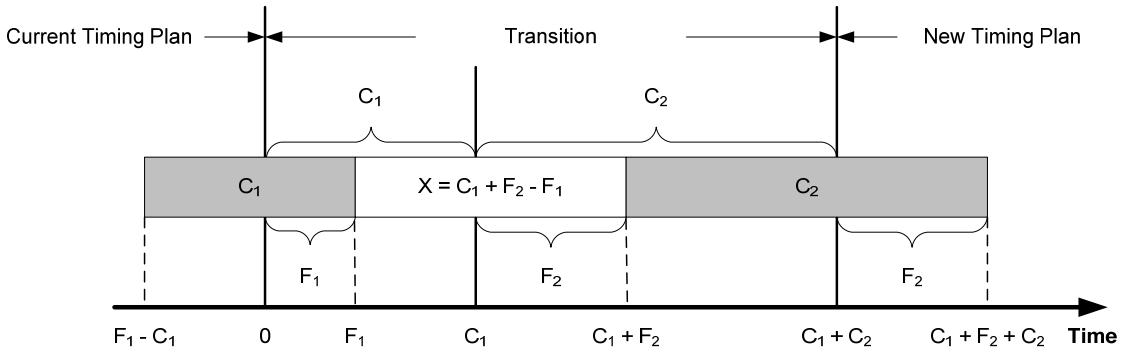


Figure 3.9 Transition between Two Timing Plans

As shown in the figure, the horizontal axis is time. The cycle length of current timing plan for this intersection is C_1 and the offset is F_1. Suppose the timing plan needs to be switched to a new one with C_2 as the cycle length and F_2 as the offset at time 0. The gray area in the figure represents the region in which phase codes are already determined. For instance, time F_1-C_1 to time F_1 represents a complete cycle of current timing plan without offset, while time C_1+F_2 to time C_1+F_2+C_2 represents the complete cycle of new timing plan without offset. The phase codes in the blank area shown in Figure 3.9 needs to be filled in. Now the problem of generating phase codes for entire transition period has been narrowed down to the blank area X.

Since the start time of X is at the end of one complete cycle of current timing plan, the phase codes in X are independent to the current timing plan. Similarly the phase codes in X are independent to the new timing plan. Thus, we can apply the same green split and phase sequence of new timing plan in X with different cycle length and no offset. According to different lengths of X, we have different schemes as follows:

Scheme I: $C_{Min} \leq X \leq C_{Max}$

In this case, X fulfills the criteria of cycle length and the phase codes can be as long as one complete cycle with length X.

Scheme II: $2C_{Min} < X \leq 2C_{Max}$ and $C_{Max} < X$

When X is larger than maximum cycle length but located between two times the minimum cycle length and two times the maximum cycle length, we can divide X into two complete cycles. When X is an even number, the cycle length is $\frac{X}{2}$; if X is an odd number, the lengths of two new cycles will be $\frac{X-1}{2}$ and $\frac{X+1}{2}$, respectively.

Scheme III: $0< X < C_{Max} - C_2$ and $X < C_{Min}$

When X is smaller than the minimum cycle length, we can extend the blank area with C_2 and fill it with one cycle. In this case, $X+C_2$ must be less than the maximum cycle length.

Since $X=C_1+F_2-F_1$, the possible value of X is in the range of $[0, 2C_{Max}]$. Besides the above three situations, the value of X could also be located in $(C_{Max} - C_2, C_{Min})$ or $(C_{Max}, 2C_{Min})$. In practice, the maximum cycle length is usually larger than two times the minimum cycle length, thus the only exception of X will be $(C_{Max} - C_2, C_{Min})$. Since this situation will be very rare in the optimization, we can just discard it by considering the new timing plan as invalid.

Considering the above schemes, we get the phase codes for transition time. Combined with the phase codes of new timing plan, the evaluation time horizon is prepared for evaluation.

CHAPTER IV

IMPLEMENTATION

To evaluate the performance of the optimization system, a microscopic simulation program is adopted as evaluation platform in our experiment. Its developed programming interface enables the user to interact with the simulation model. Our algorithms are then implemented as computer program to be tested with the microscopic evaluation platform. In this chapter, we will discuss the detail of the implementation.

4.1 Microscopic Evaluation Platform

Microscopic simulation software is widely used in transportation engineering. It not only assists the engineer to evaluate the timing plan before applying it in the field, but also be helpful in travel time study, traffic safety and many other areas. Besides simulation function, some software also provides programming interface for users to interact with it during the simulation to perform complex studies. Among these interactive simulation programs, VISSIM (28) is an outstanding one. VISSIM is a leading microscopic simulation program developed by PTV AG in Germany for multi-modal traffic flow modeling. It is used world widely because of its efficient network editing, sophisticated vehicle behavior modeling and detailed analysis options. Besides the above features, VISSIM also provides a well developed Application Programming Interface

(API) for users to apply their own logic in traffic signal control. The structure of VISSIM API is shown in Figure 4.1. In our research, we use VISSIM 4.30-05 as our microscopic evaluation platform.

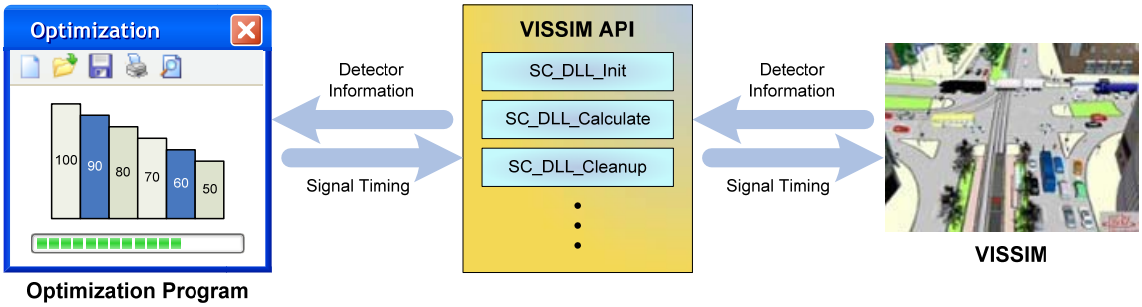


Figure 4.1 Structure of VISSIM API

4.2 PODE

The algorithm of PODE is integrated and implemented with VISSIM API. In order to keep the algorithm independent to specific intersection, we use a configuration file to save the geometric and signal timing information of the intersection. The initial values of parameters used in the algorithm are also saved in the configuration file.

4.2.1 Configuration File

An Extensible Markup Language (XML) formatted configuration file is used to describe the target intersection and the parameters. For the target intersection, signal timing information (candidate phases) and the geometric information such as approaches, lanes and detectors are saved in the configuration file. The algorithm will first read the document and parse it with an XML engine, such as Microsoft XML DOM (Document Object Model). The parsed value will then be used during the optimization calculation. The full text of a sample configuration file for PODE is attached in Appendix A and the comments are shown in Table 4.1.

Table 4.1 XML Configuration File Comments

XML Configuration File Comments			
Element	approach		
	Attributes	**Type**	**Meaning**
	index	integer	Sequence index
	laneNum	integer	Number of lanes in this approach
	type	string	Type of this approach, such as "veh" or "ped"
	Sub Elements		
	lane		
Element	lane		
	Attributes	**Type**	**Meaning**
	index	integer	Sequence index
	queueDetNum	integer	Number of queue detectors in it
	lane	integer	ID of lane in the approach
	queueLeaveCars	float	Queue release rate in this lane
	weight	float	Weight of this lane in optimization
	Sub Elements		
	countDet		
	queueDet		
Element	countDet		
	Attributes	**Type**	**Meaning**
	id	integer	ID of count detector
	distance	float	Position of this detector
Element	queueDet		
	Attributes	**Type**	**Meaning**
	id	integer	ID of queue detector
	index	integer	Sequence index
	distance	float	Position of this detector
	queuemin	integer	Min. number of queue when detector is occupied
	queuemax	integer	Max. number of queue when detector is occupied
Element	sg		
	Attributes	**Type**	**Meaning**
	index	integer	Sequence index
	minGreen	integer	Minimum green of signal group
	approachNum	integer	Number of approaches with right of the way when the signal group is in green
Element	Phase		
	Attributes	**Type**	**Meaning**
	index	integer	Sequence index
	Sub Elements		
	sg		

4.2.2 Algorithm

The algorithm of PODE is implemented with Microsoft Visual Studio 2003 in C++ language. In order to simplify the internal data sharing, the data collection module and optimization algorithm are integrated together. The algorithm first read the configuration file to initialize the system and then optimizes the signal timing in real time. Part of the source code is available in Appendix B.

4.3 GABNOR

For network optimization, GABNOR is much more complicated than PODE. Since multiple intersections are optimized simultaneously, the data collection and algorithm module should be separated rather than integrated as in PODE. As shown in Figure 4.2, GABNOR is mainly composed with a traffic information database and three cooperated modules, which are Traffic Data Collection and Analysis (TDCA) module, Traffic Network Optimization (TNO) module, and Evaluation Platform Interface (EPI) module. Evaluation platform is a microscopic simulation program with the function of interactive signal control usually provided as an API. EPI module will utilize the API function to collect information from the detectors and traffic signals in the simulation program and forward to TDCA module. Another major task of EPI is applying optimized timing plan received from TNO in the evaluation platform. The independence of EPI to evaluation platform makes it possible to perform seamless transition among different simulation programs or even hardware equipment. Once the raw data sent from EPI has been received by TDCA module, it will analyze the data and extract useful information to be saved in the database. One example of such useful information is vehicle turning

movements. Traffic information database in Figure 4.2 saves all vital traffic information and gets ready for querying requests by TNO module or any other traffic data analysis program. TNO module will optimize the traffic network periodically based on the traffic information obtained from the database and the optimal timing plan generated will then be sent to EPI module for application. All three modules and database are connected with each other in an Ethernet network. The detailed implementation of these modules is introduced in the following chapter.

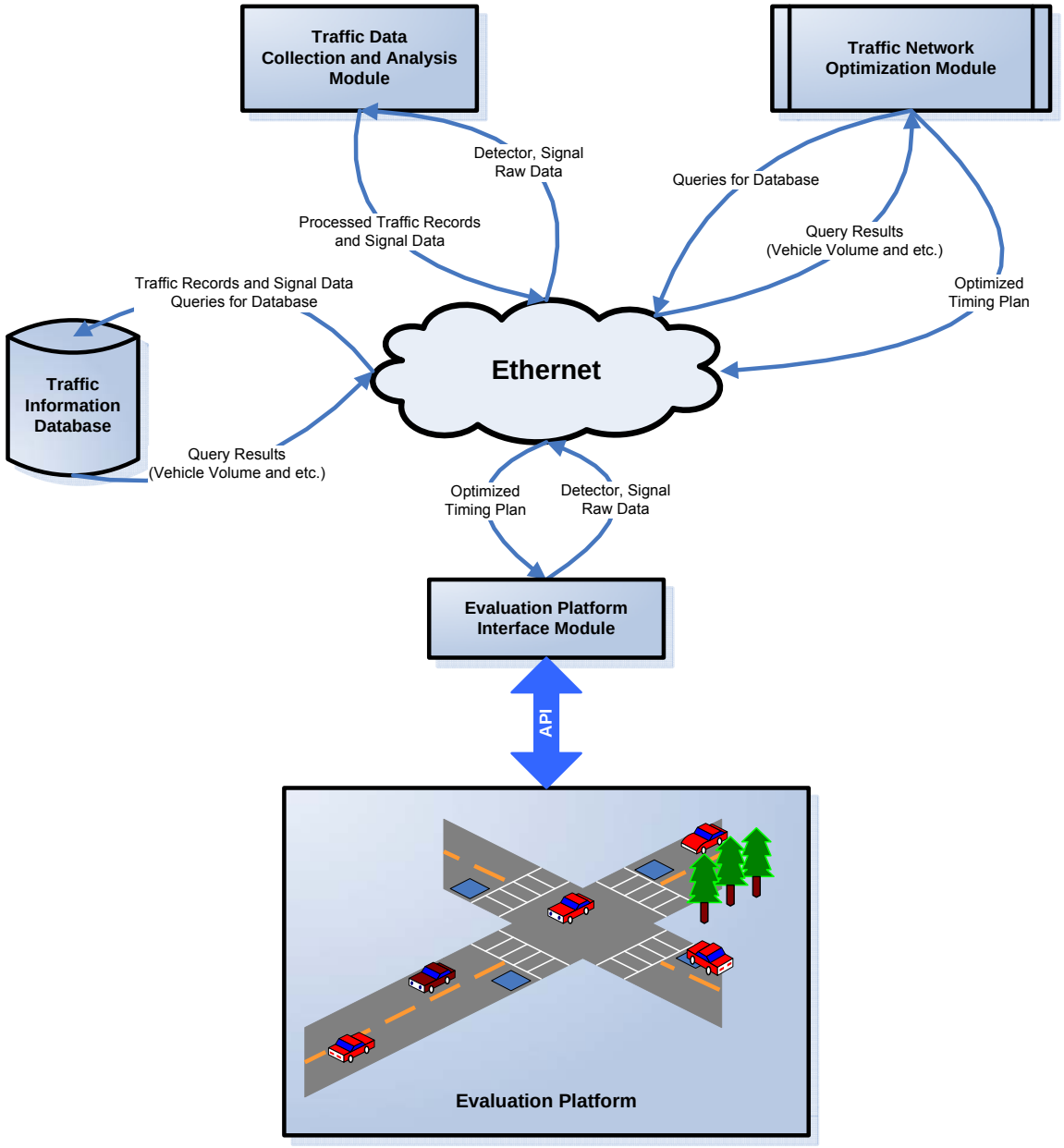


Figure 4.2 GABNOR System Architecture

4.3.1 Configuration File and Visual Tool

Similar to PODE, there is a configuration file in XML format saving the initial parameter values and geometric and signal information of the whole traffic network. Since the complexity of the configuration file will increase along with the increment of

the traffic network scale, it is inefficient to configure this file in text format manually. A visual tool is then needed to assist the configuration of the whole network. In GABNOR, a visual configuration assistant program is developed with Microsoft Visual Basic .NET 2008 as shown in Figure 4.3. The traffic network editing method is similar to VISSIM and a well trained user can build up a nine-intersection traffic network in couple hours.

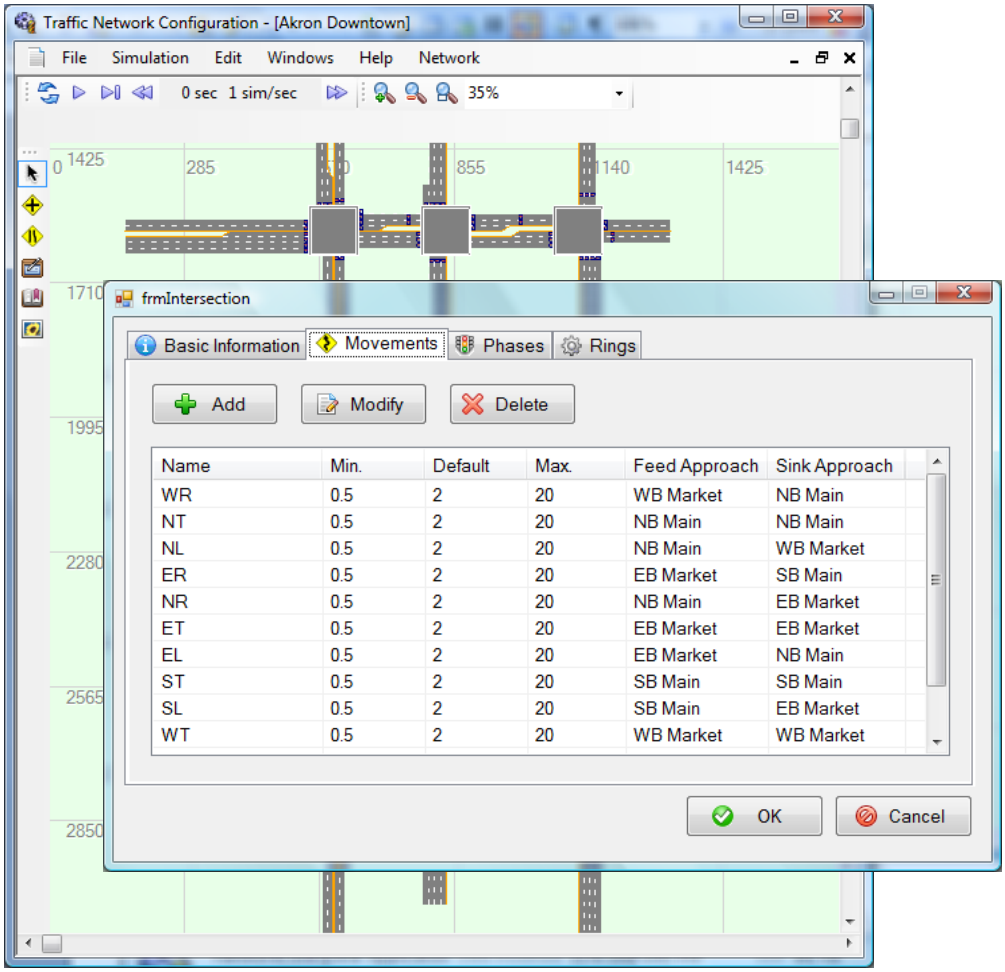


Figure 4.3 Visual Configuration Tool

4.3.2 Traffic Information Database

Traffic information collected and extracted in real-time is essential to optimization. In order to provide simplistic method to save and load the data, a database

is created to store the useful traffic information. Microsoft SQL Server 2005 is used as the existing computational drivers in our programming language. The diagram of the database model is shown in Figure 4.4. There are three tables in the database, VehicleMovements, Detections and SignalLogs. The VehicleMovements table saves vehicle turning movements identified from the detection data. Each turning movement record consists of the intersection where it belongs to, the name of this movement, the detectors' activation and deactivation time, and the type of movement. User can later query statistical information such as volume, average travel time and etc. The queried data will then be used by TNO to perform optimization. Detections table saves detections records for all the detectors in the system. IntersectionID and DetectorID are used to help identify the detector while each detection's active time and its duration are also saved in the database. Type and memo are used to mark the detection either an error or part of an identified movement. The data saved in Detections table can be used to update system parameters such as average headway, start-off delay and etc. SignalLogs table records the signal status of each intersection at each second. These signal data will be helpful in both network optimization and system parameters updating. The SQL file for creation of the three tables is listed in APPENDIX C.

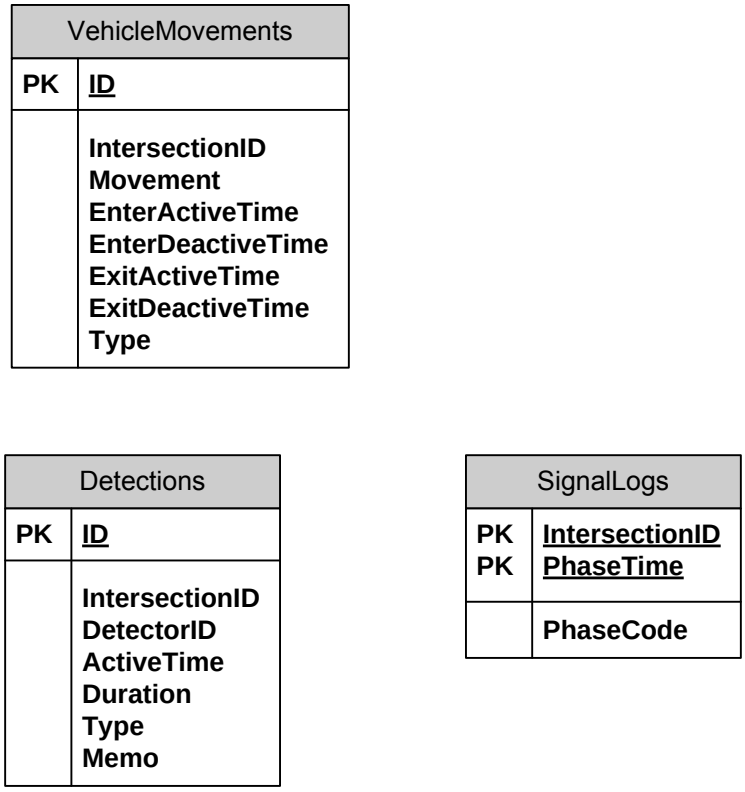


Figure 4.4 Diagram of Traffic Information Database

4.3.3 Evaluation Platform Interface Module

EPI is implemented with Microsoft Visual Studio 2003 in C++ language according to the API provided by VISSIM. The main function of EPI is linking GABNOR with microscopic simulation software to perform the evaluation. A socket client is built in EPI to communicate with TDCA module to transfer detection and signal data. Every second, EPI will send a message to TDCA containing the data collected from the simulation software. This message is a string with the following format:

Time:Type:Signal Update:Cycle Index:Phase Code[:Detector ID:

Activation Num[:ms|speed|length]:Deactivation Num[:ms|speed|length]]

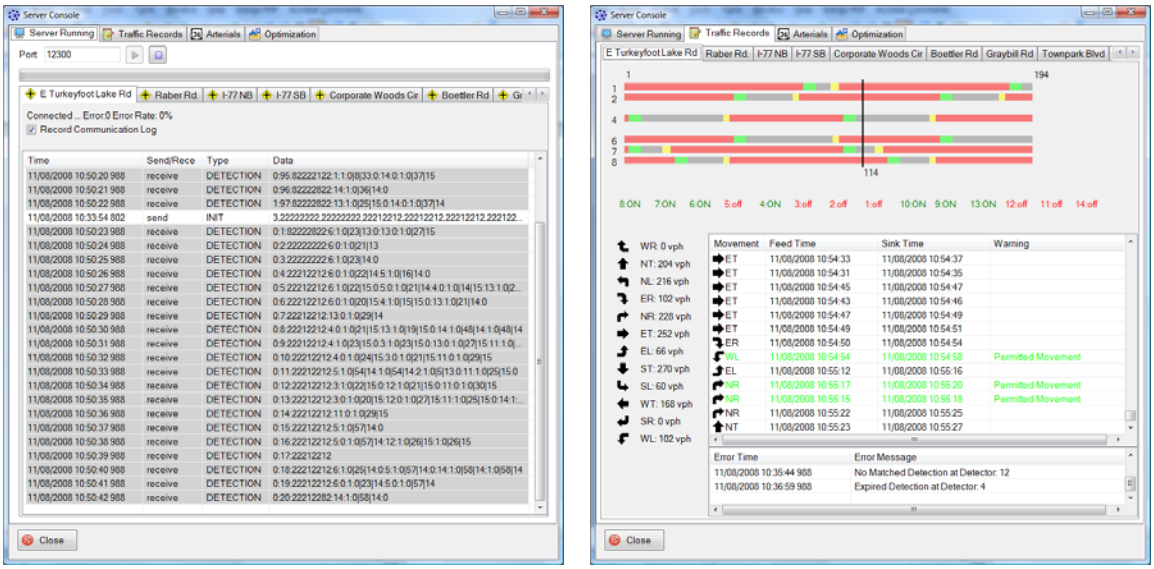
Where:

Time: Time on the client,

Type:	Type of message, such as "INIT" or "DETECTION",
Signal Update:	Indicator of whether need update timing plan or not,
Cycle Index:	Current time index in the cycle,
Phase Code:	An encoded string to save the signal status information,
Detector ID:	Identification number of detector with detection reported,
Activation Num:	Number of vehicles arrived at (activate) the detector,
Deactivation Num:	Number of vehicles left (deactivate) the detector,
ms:	Million second of the detection,
speed:	Speed of the vehicle (optional),
length:	Length of the vehicle (optional).

4.3.4 Traffic Data Collection and Analysis Module

After the message sent from EPI received by TDCA module, an analysis procedure will be executed to filter useful information. Some information, such as signal status and detections, can be stored in the database without much process, while the others need further computation such as vehicle turning movement. TDCA module implements the vehicle identification process and other data analysis functions. Interfaces shown in Figure 4.5 are also provided for users to monitor the status of TDCA module.

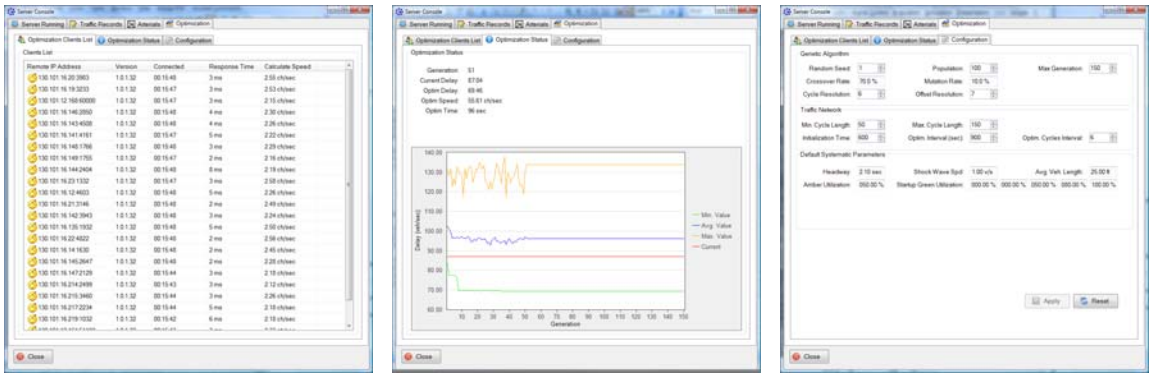


(a) Message Sending/Receiving Interface (b) Signal and Detector Status Monitor

Figure 4.5 TDCA Module Interface

4.3.5 Traffic Network Optimization Module

TNO is a client-server structured module with parallel computation capability. The server maintains a pool of connections with all the clients and another pool of timing plans needs to be evaluated. The evaluation will be performed on the client side and the server will dynamically assign timing plans to the clients. Through the given interface shown in Figure 4.6, user can check the status of the clients, track the optimization status and setup the optimization parameters, such as population, crossover rate and mutation rate.



(a) Clients List (b) Status Monitor (c) Parameters Configuration

Figure 4.6 Optimization Server Interface

Client program shown in Figure 4.7 will receive the timing plan sent from the server and send back the evaluation result. Besides evaluation function, auto-updating is another feature of the client program. Any update package released in the server will be downloaded by the clients automatically. This feature will not only ensure that all the clients using the same function to evaluate the timing plan, but also save substantial time during the system developing where the evaluation function usually updated frequently.

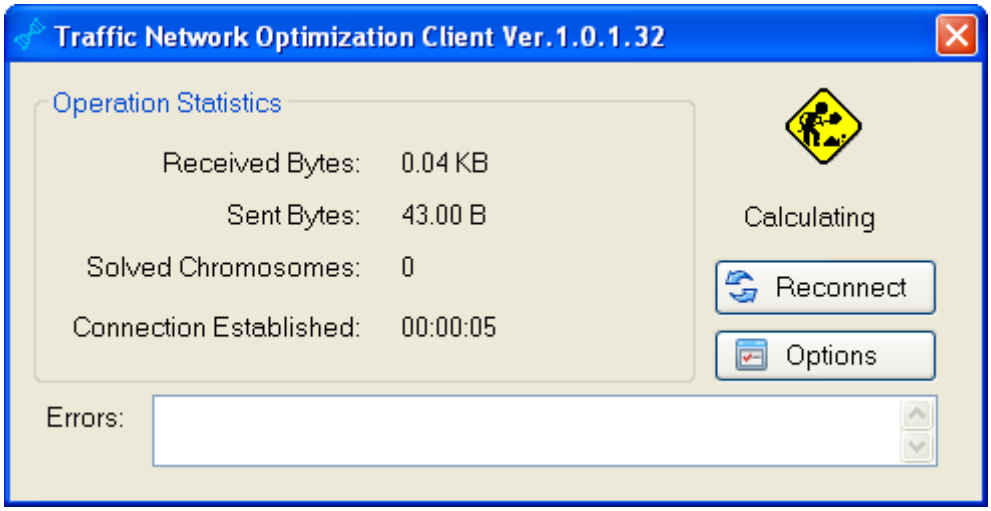


Figure 4.7 Optimization Client Program

CHAPTER V

SIMULATION EVALUATION

After the algorithms have been implemented, we investigate their performance on the simulation evaluation platform.

5.1 PODE

As adaptive control logic for isolated intersection, PODE was evaluated with different traffic load compared with well-adjusted actuated control at an isolated intersection. The detail of the evaluation is discussed in the following part.

5.1.1 Evaluation Scenario

PODE system testing and evaluation is performed on a four-legged intersection. Each approach of the intersection has two lanes for the through movements with a left turn pocket while the curb lane is shared with right turn vehicles. A screen snapshot of the operation is shown in Figure 5.1. During the evaluation, we changed the traffic demand of the whole intersection from 1600 veh/hr to near 7000 veh/hr at the intervals of 400 veh/hr. The high volume represents an oversaturated situation because the theoretical and practical capacity for this intersection is reported in the range of 6000 to 6400 veh/hr considering start-up delay and clearance time (29). Many testing runs were made at each

volume level with different traffic load distributions on each approach, as exemplified in Table 5.1.

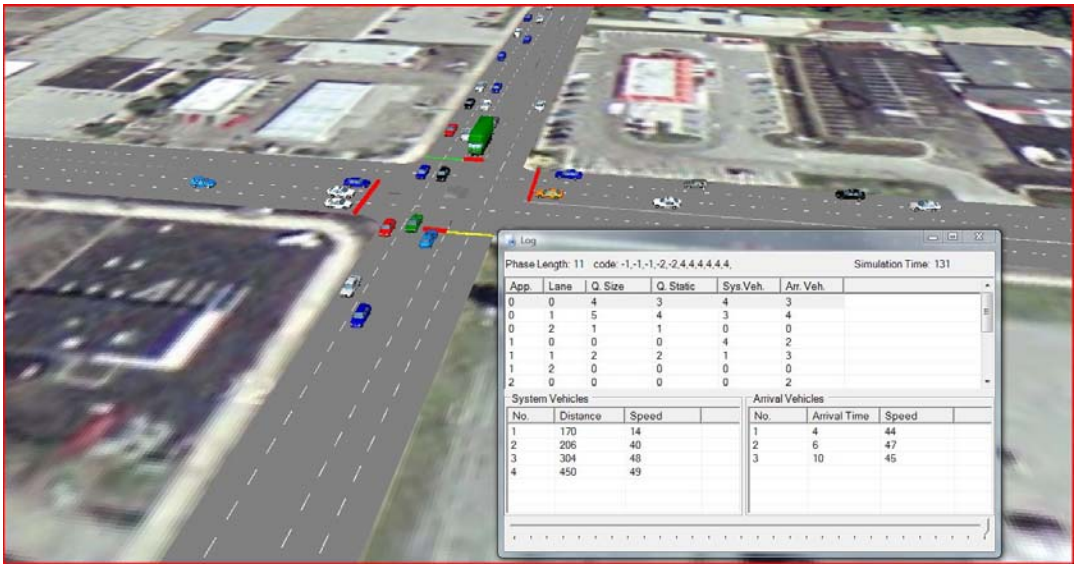


Figure 5.1 Screen Snapshot of PODE Evaluation

Table 5.1 Sample Traffic Demand Distribution on Each Approach

Traffic Demand Load Distribution (veh/hr)				
Approach 1	**Approach 2**	**Approach 3**	**Approach 4**	**Total Demand**
…				
800	800	800	800	3200
700	900	700	900	3200
600	1000	600	1000	3200
600	600	1000	1000	3200
700	700	900	900	3200
900	900	900	900	3600
800	1000	800	1000	3600
700	1100	700	1100	3600
700	700	1100	1100	3600
800	800	1000	1000	3600
1000	1000	1000	1000	4000
900	1100	900	1100	4000
800	1200	800	1200	4000
800	800	1200	1200	4000
900	900	1100	1100	4000
…				

5.1.2 Benchmark

We used the actuated control logic as the benchmark for comparison. The setup for phasing and timing in actuated control follows the standard of National Electrical Manufacturers Association (NEMA) for a fully actuated eight-phase dual-ring controller. Different locations for placing the actuating detector are tested in each run in order to give a fair consideration of the actuated control, and the best result is included as the benchmark performance. The parameters of actuated control are also adjusted at different traffic demand levels and load distributions to get the best performance. One (and the most important) such parameters is the maximum green for each phase. As shown in Table 5.2, each row represents different traffic demand and distribution. The maximum green for each phase is the one which produced minimum vehicle delay. From the table we can see that the best maximum green varies for different traffic demands and distributions.

Table 5.2 Best Maximum Green Affected by Traffic Demand and Distribution in Actuated Control

Traffic Demand (veh/hr)				Maximum Green for Phase (sec)							
App. 1	App. 2	App. 3	App. 4	1	2	3	4	5	6	7	8
…											
800	800	800	800	10	16	10	16	10	16	10	16
700	900	700	900	10	16	10	18	10	16	10	18
600	1000	600	1000	10	14	10	22	10	14	10	22
600	600	1000	1000	10	20	10	20	10	20	10	20
700	700	900	900	10	20	10	20	10	20	10	20
900	900	900	900	10	18	10	18	10	18	10	18
800	1000	800	1000	10	18	10	22	10	18	10	22
700	1100	700	1100	10	16	10	22	10	16	10	22
700	700	1100	1100	10	24	10	24	10	24	10	24
800	800	1000	1000	10	22	10	22	10	22	10	22
1000	1000	1000	1000	10	22	10	22	10	22	10	22
900	1100	900	1100	10	22	10	24	10	22	10	24
800	1200	800	1200	10	18	10	26	10	18	10	26
800	800	1200	1200	10	28	10	28	10	28	10	28
900	900	1100	1100	10	28	10	28	10	28	10	28
…											

5.1.3 Evaluation Results

PODE is tested next at this intersection under the same traffic and geometric conditions. With input from the upstream detector (placed 1000 ft upstream), queue detector (placed 200 ft upstream) and stop line detector, PODE optimized the intersection operation at the interval range of five to 20 seconds. The clearance time for each phase includes three seconds yellow plus two seconds red which is the same with the one in actuated control. There are eight candidate phases for selection and 128 length and phase combinations for each interval to compare and choose. The simulation results with PODE are shown in Figure 5.2 based on seventy runs which include fourteen levels of traffic

demands with five different loading distributions at each level. We can see from this figure that PODE consistently outperforms the actuated control logic, and the vehicle delay from PODE is notably reduced in high volume situations. Specifically, in the low to medium demand range, from 1600 veh/hr to 4800 veh/hr, the saving in vehicle delay by PODE against the best performance of actuated control is within ten seconds/veh. In high traffic demand, from 5200 veh/hr to 6400 veh/hr (actuated control cannot handle higher demand), actuated control begins to increasingly show large "jumps" in delay, which means its performance is unstable with different traffic load distributions. In comparison, although vehicle delay in PODE also goes up with the increase in traffic demand, the change is much smoother and the increments are much smaller compared with actuated control. In addition, PODE can handle a larger volume (exceeding 6800 veh/hr) whereas actuated control breaks down (rapid and continuous queue growth) when volume reaches 6400 veh/hr. It should be pointed out that there is no need to manually adjust the PODE system parameters during the seventy runs, whereas in the actuated control logic the location of the detectors and the maximum green have to be changed in order to obtain its best performance as discussed earlier.

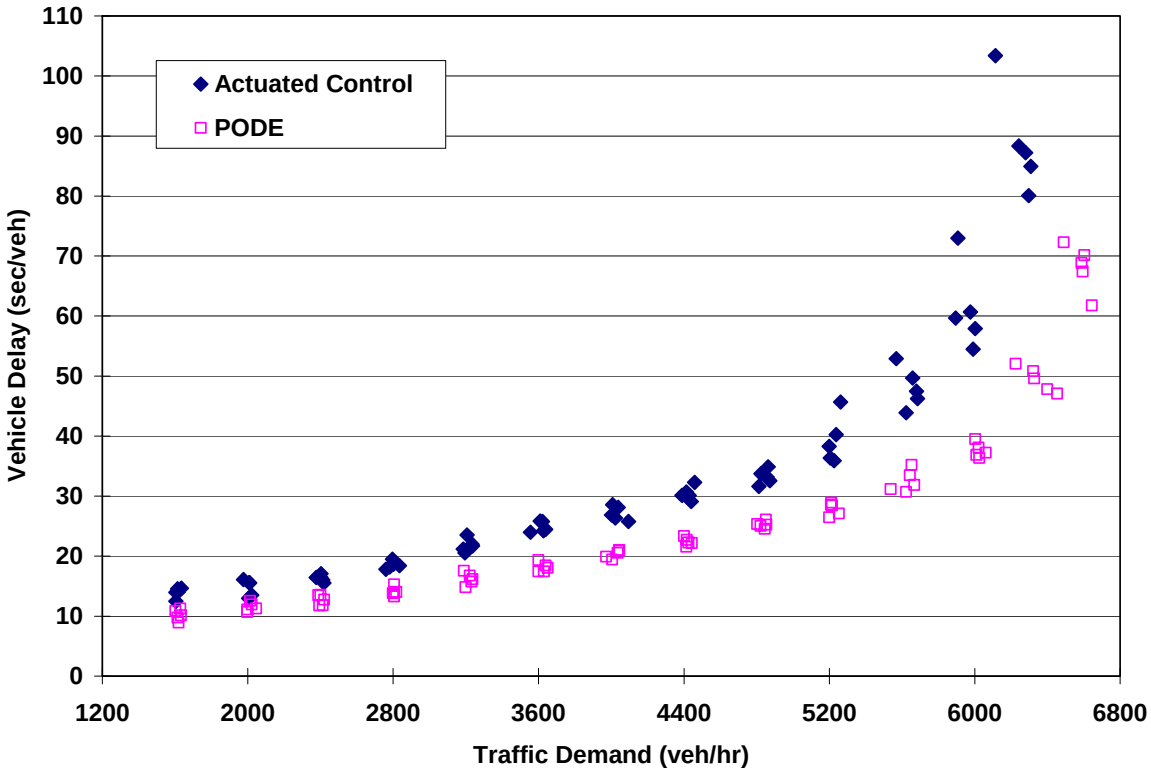


Figure 5.2 Vehicle Delay Comparison between Actuated Control and PODE

Table 5.3 shows the percentage of reduction in vehicle delay. The range of reduction varies from 20% to 45%. With the increase of traffic demand, the percentage of reduction also increases indicating that PODE works very well in heavy to oversaturated traffic situations. The reduction of average vehicle delay for high traffic demand (larger than 5200 veh/hr) starts at 11 seconds and goes up to 40 seconds or 45%. The phase duration in PODE changes with traffic demand. We used a minimum green to handle extremely low volumes, and the system extended the green to as long as 50 seconds to accommodate very high volumes. In comparison, the phase duration in actuated control went up to 80 seconds in order to obtain best operation results. Most cycle lengths in PODE are between 45 and 80 seconds while in actuated control are from 60 to 200 seconds.

Table 5.3 Delay Reduction in PODE Against Actuated Control

Traffic Demand (veh/hr)	Average Delay (sec/veh)		Reduction (seconds)	Reduction Percentage
	Actuated Control	PODE		
1600	13.94	10.28	3.66	26.26%
2000	14.73	11.86	2.87	19.48%
2400	16.30	13.02	3.28	20.12%
2800	18.51	14.84	3.67	19.83%
3200	21.78	16.78	5.00	22.96%
3600	24.85	18.54	6.31	25.39%
4000	27.14	20.76	6.38	23.51%
4400	30.45	23.00	7.45	24.47%
4800	33.19	24.44	8.75	26.36%
5200	39.29	28.10	11.19	28.48%
5600	48.03	33.50	14.53	30.25%
6000	61.14	39.04	22.10	36.15%
6400	88.78	48.80	39.98	45.03%
6800	N/A	62.10	N/A	N/A
Overall Average Reduction Percentage				26.79%

Compared with other control logic, PODE has also showed that it is able to further improve system optimization. According to OPAC field tests (10) and simulation results (30), the reduction percentage is from 10% to 20% against actuated control. There is very little reported work of RHODES at a single intersection. Sen and Head (8) presented a chart to show the difference of delay between COP (algorithm used at intersection control level in RHODES) and actuated control for an isolated intersection with three through lanes plus one right turning bay and one (some approaches have two) left turning bay. From this chart, the average delay from COP increased from 15 sec/veh to 35 sec/veh as the traffic demand increased from 4500 veh/hr to 4800 veh/hr. In the case of PODE, however, the delay is around 20 sec/veh to 25 sec/veh at the same level of traffic demand according to Figure 5.2. Nevertheless, we believe the model implementation and testing has shown the competitive strength of PODE to perform well

consistently, especially in heavy and oversaturated traffic. Since we did not directly implement the aforementioned models, one should be cautions when citing the above statistics in comparison.

5.1.4 Sensitivity of Vehicle Arrival Pattern

Reliable adaptive control logic should continuously performed well independent to external factors, such as vehicle arrival pattern. To verify whether PODE is sensitive to vehicle arrival pattern or not, we run the evaluation model five times with the same traffic demand in Table 5.1 but different seeds for vehicle generation. The test results shown in Figure 5.3 indicate that the PODE performs well consistently when different vehicle generating seeds are used.

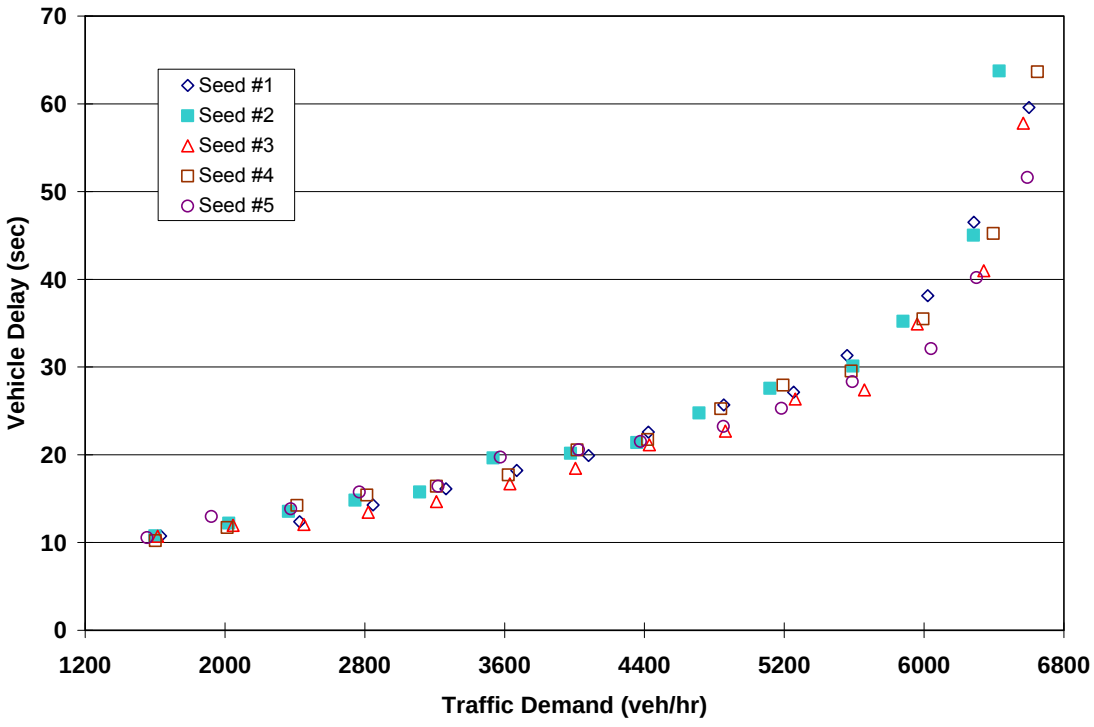


Figure 5.3 PODE Sensitivity Test of Vehicle

5.2 GABNOR

As adaptive control logic for traffic network, GABNOR was evaluated with different scenarios compared with the optimized timing plan generated from widely used signal timing optimization tool. The detail of the evaluation is discussed in the following part.

5.2.1 Parallel Computation Evaluation

First, the capability of parallel computation of GABNOR is tested. There are total 37 desktop computers involved in this experiment. One of these computers acts as the server while the others installed with client program to evaluate chromosomes. The computation speed of GABNOR increased up to 54 chromosomes per second (ch/sec) while the number of clients reaches to 36. However, as shown in Figure 5.4, the parallel computation has met a threshold of 54 ch/sec after the number of computer reaches 24. With further investigation of the computation capability of the clients, we find out that 24 computers are Intel Core 2 CPU with 3.00 GB RAM while the other 12 computers are Pentium 4 with RAM in the range from 512 MB to 2.00 GB. Computer with Core 2 CPU can calculate up to 3.5 ch/sec while the slowest Pentium 4 computer can only evaluate 1.5 ch/sec or 700 million seconds per chromosome. Similar to “Wooden Barrel Theory”, the performance of parallel computation is determined by the slowest, not the fastest computer in the system. Thus we check the number of chromosomes evaluated by the slowest computer in each generation. In our experiment, we have 75 chromosomes to be evaluated in each generation and the slowest computer usually evaluates two chromosomes, in another word, 1400 ms, in each generation on average. The

computation speed can be calculated by 75 chromosomes / 1400 ms = 53.5 ch/sec, which is very close to the speed we observed.

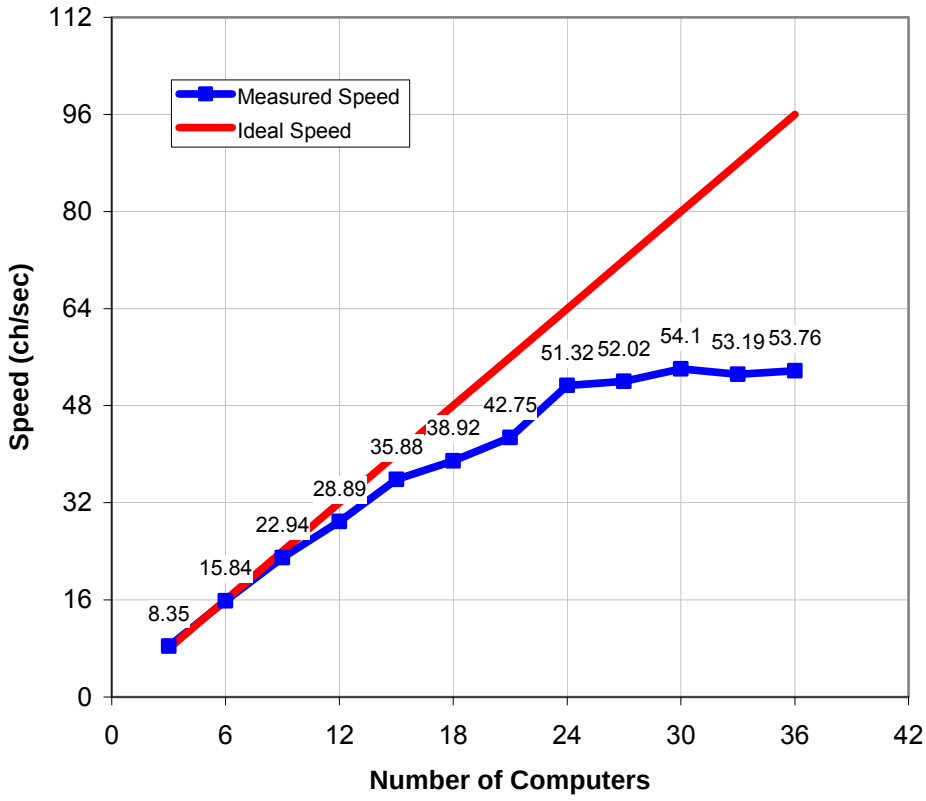


Figure 5.4 Parallel Computation Speed Experiment with 36 Clients

To increase the speed of evaluation, we can either limit the number of chromosomes evaluate by the slowest computer or increase the computation power of the slowest client in the system. Since we have 24 computers which have similar computation power in range from 3.0 ch/sec to 3.5 ch/sec, we did another test which only includes these 24 computers. As shown in Figure 5.5, the test results show the calculation speed continuously increased along with the increment of the computer number. When the computer number increases, the difference between the ideal speed and measured speed also increases. This may be caused by the non-parallelizable part in the

optimization program, such as reproduction process, and the time lost in communication between server and clients. With the help of parallel computation, we can evaluate the performance of GABNOR in real-time.

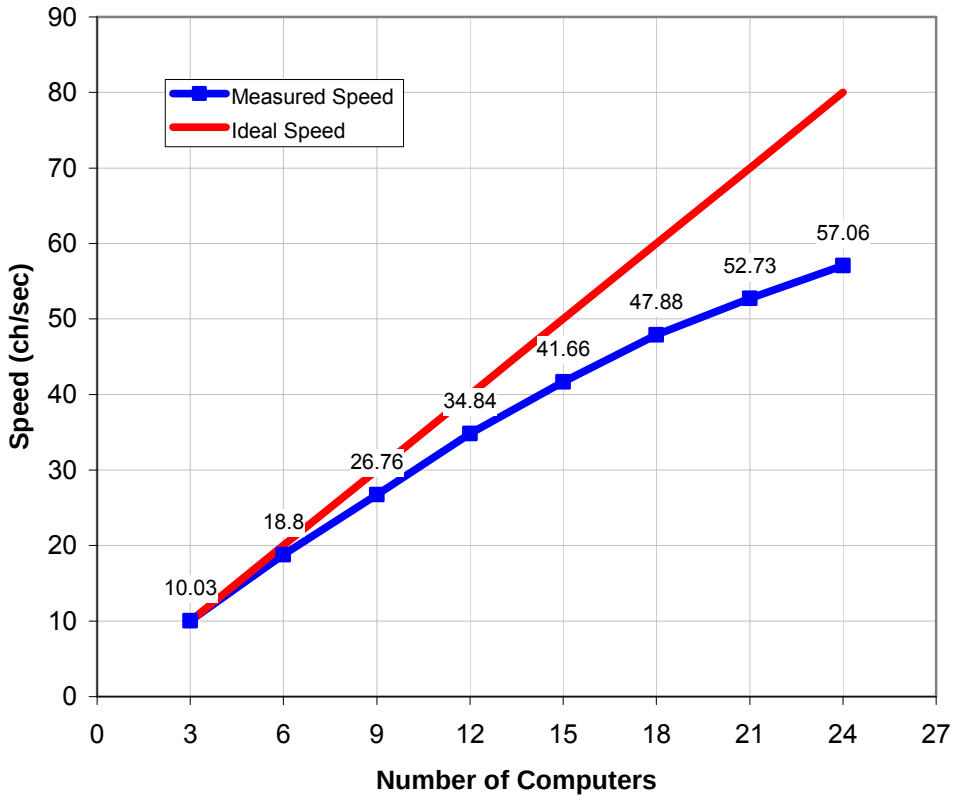


Figure 5.5 Parallel Computation Speed Experiment with 24 Clients

5.2.2 Evaluation Scenarios

There are two scenarios tested during the evaluation of GABNOR. Each of them composed with nine intersections while one is an arterial highway and the other is a grid urban traffic network. The two different scenarios are selected to evaluate the performance of GABNOR under different network configurations.

5.2.2.1 Arterial

The first scenario is a nine-intersection arterial located at Green, Ohio. The geometric layout of this arterial is shown in Figure 5.6. Besides regular at-grade intersections, this corridor also includes a diamond interchange. The main street, Massillon Rd, is a 1.76 mile two-way arterial and the speed limit varies from 35 mph to 45 mph. Afternoon peak hour volume is modeled in the evaluation program.

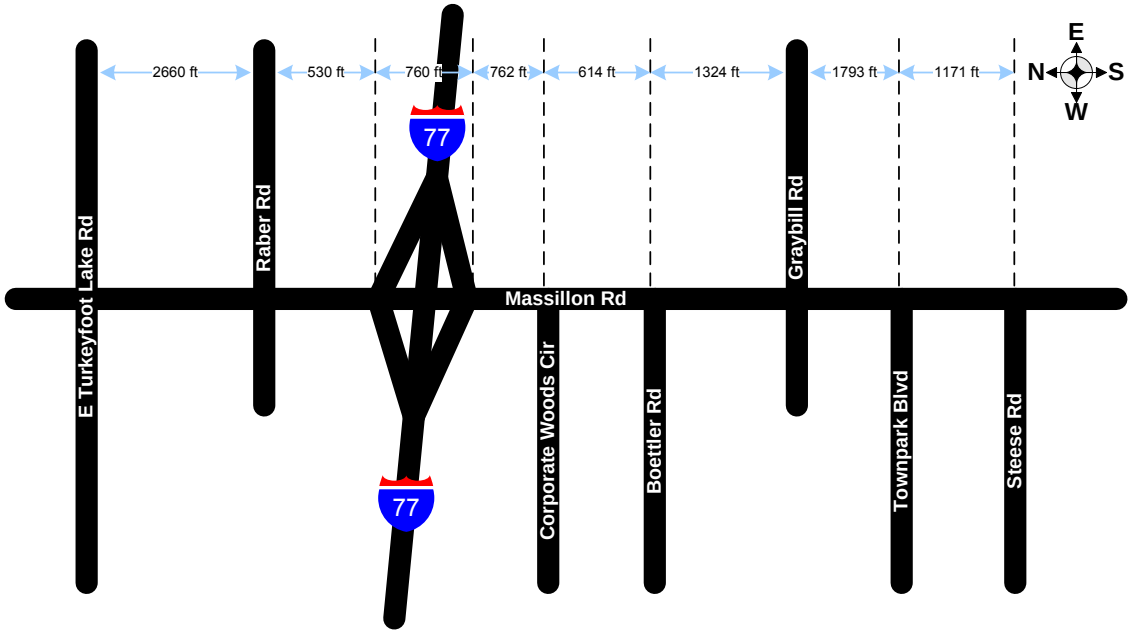


Figure 5.6 Arterial of Massillon Rd in Green, Ohio

5.2.2.2 Grid Network

To verify the capability of optimizing grid traffic network of GABNOR, we also tested it in a 3x3 grid network in downtown Akron, Ohio. As shown in Figure 5.7, the grid traffic network consists of two one-way streets and four two-way streets. The length of each block is shown in the figure and the traffic volume information is randomly assigned.

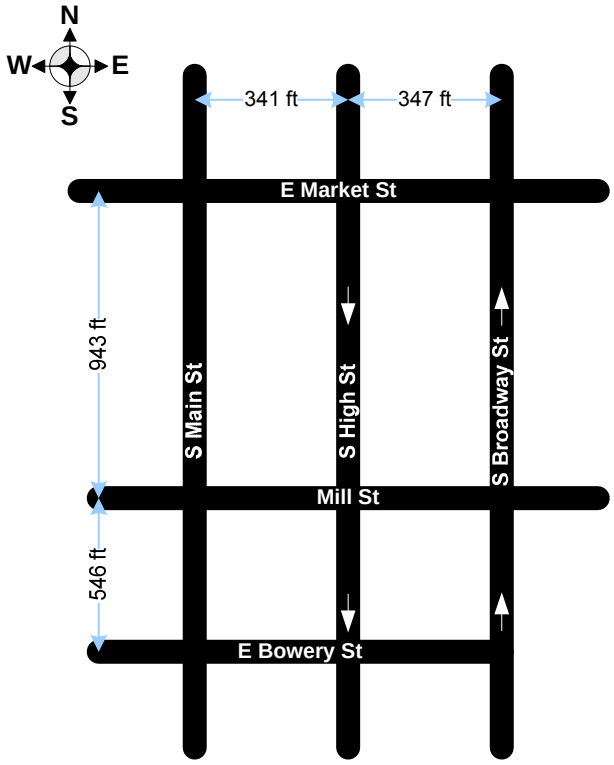


Figure 5.7 Grid Traffic Network in Downtown Akron, Ohio

5.2.3 Benchmark

To evaluate the performance of GABNOR, SYNCHRO is chosen to generate benchmark timing plan of the comparison. Both the nine-intersection arterial and grid network with their volume information was modeled in SYNCHRO. After that, recommended optimization procedure from SYNCHRO has been applied. The signal timing plan was then optimized with the network cycle length varying from 50 to 150 seconds. The increment was two seconds due to half cycle option. The best timing plan was exported to and evaluated by VISSIM.

5.2.4 Evaluation Results

Meanwhile, the arterial and grid network was also modeled in GABNOR. Because of GABNOR's adaptive capability, it doesn't require the volume information. The population size of GA was set to 50 and the maximum generation was 150. Crossover and mutation probability were 0.7 and 0.1 typically. The minimum cycle length and the maximum cycle length were the same with SYNCHRO which were 50 and 150 seconds. The optimization interval was set to 15 minutes and the maximum GA calculation time was limited to 200 seconds. Evaluations of GABNOR and benchmark timing plan from SYNCHRO were finished in VISSIM through ten randomly seeded simulation runs. Each simulation lasted 70 minutes including ten minutes for warm up time. The performance of the arterial was sampled in the last 45 minutes of the simulation which was after the first optimization of GABNOR (at 25 minutes).

5.2.4.1 Arterial

Generally, the average delay of whole arterial is 15% lower in GABNOR among those ten simulation runs. From the average delay of each intersection shown in Figure 5.8, we can find that the timing plan generated in GABNOR tends to balance the vehicle delay among those intersections. As shown in Table 5.4, delay, stops and queue length of each simulation run have been listed and the data are tested with statistic tools. The Shapiro-Wilk test (31) shows that all six groups of data are normally distributed with 95% confidence level. The improvements of the delay, stops and queue length in GABNOR are all statistical significant according to the T-Test.

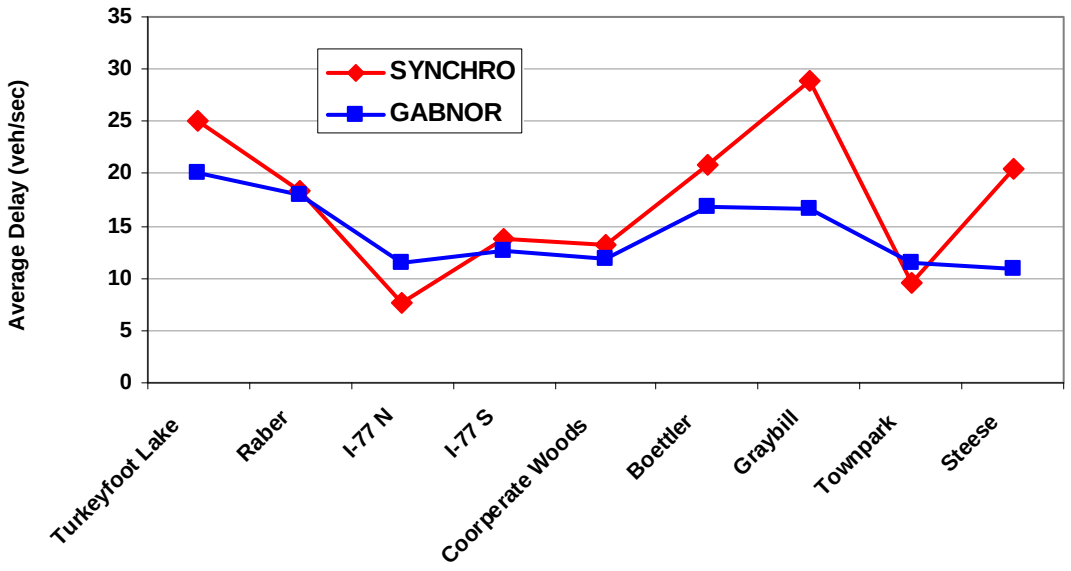


Figure 5.8 Average Delay of Intersections in the Arterial

Table 5.4 Evaluation Results of Arterial in City of Green, Ohio

Random Seed		Delay (sec)		Stops		Queue Length (ft)	
		Synchro	GABNOR	Synchro	GABNOR	Synchro	GABNOR
1		17.10	14.10	0.51	0.40	25.40	18.70
2		15.60	13.40	0.46	0.37	21.40	18.40
3		16.20	13.60	0.48	0.39	22.90	17.60
4		19.40	15.90	0.59	0.42	33.10	23.40
5		17.70	15.10	0.53	0.42	26.90	21.20
6		16.80	14.50	0.51	0.42	25.40	19.70
7		17.30	14.90	0.51	0.42	26.50	20.40
8		16.90	14.00	0.51	0.42	25.60	20.90
9		16.50	13.80	0.49	0.39	24.60	20.10
10		17.20	15.90	0.51	0.45	26.90	22.90
Average		17.07	14.52	0.51	0.41	25.87	20.33
Standard Deviation		1.0133	0.9041	0.0343	0.0226	3.0862	1.8679
Difference		14.94%		19.61%		21.41%	
Shapiro-Wilk Test	p-value*	0.2594	0.3266	0.1410	0.2721	0.1707	0.9645
	Normal Dist	YES	YES	YES	YES	YES	YES
F-Test	p-value*	0.3698		0.1148		0.0754	
	Homo. Vari.	YES		YES		YES	
T-Test	p-value*	0.0000		0.0000		0.0000	
	Diff. Sig.	YES		YES		YES	

* Reject the null hypothesis when p-value < 0.05

5.2.4.2 Grid Network

For grid network optimization, as shown in Figure 5.7, GABNOR yields very close result to the benchmark timing plan and the difference in delay is less than 1%. When observing the simulation in detail, we found that the timing plan generated by GABNOR is similar to the one from SYNCHRO except for shorter cycle length. This may be caused by relatively small dimension (3 intersections at each direction) of the network allows both SYNCHRO and GABNOR approach to the global best timing plan.

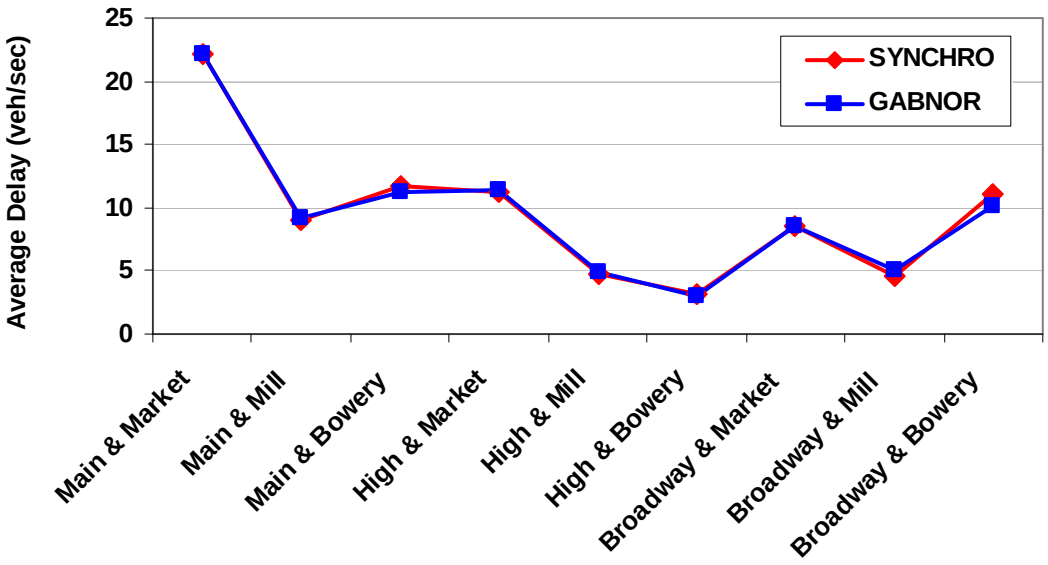


Figure 5.9 Average Delay of Intersections in Grid Network

Comparison of detailed performance between SYNCHRO and GABNOR is shown in Table 5.5. From the results, GABNOR has almost the same result of SYNCHRO. T-Test results also show that the difference of delay and queue length between two methods is not significant. Mann-Whitney U test is used to examine the difference in stops since the F-Test doesn't support the homogeneity of variance in stops. As shown in Figure 5.10, the result indicates that the difference is not significant.

Another interesting finding is that GABNOR prefer to reduce the queue length instead of the stops. This is supported by the observation that GABNOR usually generates smaller cycle length than SYNCHRO.

Table 5.5 Evaluation Results of Grid Network in City of Akron, Ohio

Random Seed		Delay (sec)		Stops		Queue Length (ft)	
		Synchro	GABNOR	Synchro	GABNOR	Synchro	GABNOR
1		10.20	10.20	0.39	0.39	9.60	9.60
2		10.20	10.40	0.39	0.41	9.80	9.80
3		10.20	10.20	0.39	0.43	9.80	9.30
4		10.30	10.10	0.40	0.41	9.70	9.40
5		10.60	10.60	0.41	0.41	10.10	10.10
6		10.20	10.20	0.39	0.39	9.60	9.60
7		10.00	10.00	0.39	0.39	9.40	9.40
8		10.10	10.10	0.39	0.39	9.90	9.90
9		9.90	9.90	0.38	0.38	9.30	9.30
10		10.60	10.40	0.40	0.41	10.30	10.00
Average		10.23	10.21	0.39	0.40	9.75	9.64
Standard Deviation		0.2263	0.2079	0.0082	0.0152	0.3028	0.2951
Difference		0.20%		-2.04%		1.13%	
Shapiro-Wilk Test	p-value*	0.1864	0.7376	0.1275	0.1201	0.9329	0.2944
	Normal Dist	YES	YES	YES	YES	YES	YES
F-Test	p-value*	0.4022		0.0404		0.4704	
	Homo. Vari.	YES		NO		YES	
T-Test	p-value*	0.5911				0.0932	
	Diff. Sig.	NO				NO	

* Reject the null hypothesis when p-value < 0.05

Ranks

	Group	N	Mean Rank	Sum of Ranks
Stops	1.00	10	9.05	90.50
	2.00	10	11.95	119.50
	Total	20		

Test Statistics[b]

	Stops
Mann-Whitney U	35.500
Wilcoxon W	90.500
Z	-1.182
Asymp. Sig. (2-tailed)	.237
Exact Sig. [2*(1-tailed Sig.)]	.280[a]

a. Not corrected for ties.

b. Grouping Variable: Group

Figure 5.10 Mann-Whitney U Test for Stops in Grid Network Optimization

5.2.5 Detector Error Impact Study

In practice, it is difficult to reach 100% detector accuracy. In the field experiment of visual detection, the detector's accuracy is affected by many factors, including the height and angle of the camera, weather and traffic volume. In some cases, the detection error can reach as high as 20%. There are primarily two types of detections error, miss count and double count. Miss count means a vehicle passed over a detector without detection reported. On the contrast, double count means detection reported without a vehicle passed over the detector or two detections reported when there is only one vehicle. The turning movement identification process is designed to reduce the impact from the detector error. To evaluate the performance of this process, we intensively generate detection errors in the EPI module to simulate the field situation. The turning movement

count identified is then compared with the one without detection error. The performance of GABNOR with detection error is also examined to test its fault tolerance of input data.

In this experiment, EPI randomly generate 10% detection errors which include 5% double count and 5% miss count. The same vehicle arrival pattern is used in this test and the result is listed in Table 5.6. From this table, we can find that the error percentage of turning movement count has been reduced to 3.68% and the difference of system performance compared with no error situation is not statistical significant.

Table 5.6 Results of Detection Error Tolerance Experiment with 10% Error

Random Seed		Turning Movement Error	Delay (sec)	Stops	Queue Length (ft)
1		3.99%	15.30	0.41	23.40
2		2.39%	14.70	0.40	21.40
3		4.34%	14.30	0.40	20.10
4		3.87%	14.60	0.43	21.50
5		2.96%	13.90	0.39	17.90
6		3.34%	15.50	0.43	22.00
7		4.29%	13.80	0.38	19.90
8		3.60%	14.40	0.40	21.80
9		2.43%	13.50	0.37	18.70
10		5.62%	14.10	0.41	19.50
Average		3.68%	14.41	0.40	20.62
Standard Deviation		0.9760%	0.6385	0.0193	1.6844
Difference with 0% Error			0.76%	1.95%	-1.43%
Shapiro-Wilk Test	p-value*	0.7053	0.7975	0.6244	0.9106
	Normal Dist.	YES	YES	YES	YES
F-Test		p-value*	0.1574	0.3237	0.3816
		Homo. Variance	YES	YES	YES
T-Test		p-value*	0.7731	0.3353	0.7513
		Diff. Sig.	NO	NO	NO

* Reject the null hypothesis when p-value < 0.05

Next, the error percentage is raised to 20% including 10% double count and 10% miss count. As shown in Table 5.7, the error percentage of turning movement count has been controlled around 6% and the system can still yield good performance. T-Test

shows there is not significant difference between 20% error and no error. That means GABNOR can keep almost the same performance even under high detection data error.

Table 5.7 Results of Detection Error Tolerance Experiment with 20% Error

Random Seed		Turning Movement Error	Delay (sec)	Stops	Queue Length (ft)
1		7.36%	15.20	0.41	22.20
2		5.45%	15.60	0.44	21.60
3		6.01%	14.60	0.42	19.70
4		7.82%	14.60	0.42	19.50
5		7.44%	13.50	0.37	18.40
6		5.62%	13.70	0.39	18.60
7		4.99%	15.10	0.44	19.80
8		5.48%	16.40	0.49	23.40
9		3.98%	15.00	0.42	21.50
10		6.00%	14.30	0.41	18.90
Average		6.01%	14.80	0.42	20.36
Standard Deviation		1.2043%	0.8641	0.0321	1.7005
Difference with 0% Error			-1.93%	-2.68%	-0.15%
Shapiro-Wilk Test	p-value*	0.5373	0.9299	0.3295	0.3213
	Normal Dist.	YES	YES	YES	YES
F-Test		p-value*	0.4475	0.1549	0.3921
		Homo. Variance	YES	YES	YES
T-Test		p-value*	0.5760	0.4292	0.9746
		Diff. Sig.	NO	NO	NO

* Reject the null hypothesis when p-value < 0.05

5.2.6 Sensitivity of System Parameters

There are three key parameters for GA, population size, crossover rate and mutation rate. The default testing value is 50 for population, 0.7 for crossover rate and 0.1 for mutation rate. To examine the sensitivity of GABNOR to these system parameters, we have tried different parameter values in the optimization of arterial network respectively.

5.2.6.1 Population Size

The larger population size will expand the searching space while the limited calculation power restricts the selection of the size. According to the calculation power in our experiment, we have nearly 60 ch/sec computation speed while the optimization time should be limited in one or two cycles at most, which is around 100 seconds. That means we can evaluate 6,000 chromosomes in total. Finding a balance between the population size and the number of generations is the primary focus.

Besides the default 50 population size, we evaluate the performance of GABNOR with the population size 75 and 100. According to the results of T-Test shown in Table 5.8, there is no significant difference in delay and queue length between 75 and 50 as the population size. Mann-Whitney U test is used to examine the difference in stops between two different population sizes since the F-Test yield failed result on homogeneity of variance. The result also indicates no significant difference in stops as shown in Figure 5.11.

Table 5.8 Evaluation Results with 75 Population Size

Random Seed		Delay (sec)		Stops		Queue Length (ft)	
		75	Default	75	Default	75	Default
1		14.80	14.10	0.41	0.40	19.70	18.70
2		14.00	13.40	0.39	0.37	19.40	18.40
3		13.80	13.60	0.39	0.39	19.60	17.60
4		15.80	15.90	0.41	0.42	24.60	23.40
5		15.90	15.10	0.41	0.42	25.20	21.20
6		15.30	14.50	0.42	0.42	22.10	19.70
7		14.10	14.90	0.41	0.42	21.00	20.40
8		15.50	14.00	0.41	0.42	22.70	20.90
9		14.90	13.80	0.41	0.39	20.50	20.10
10		14.80	15.90	0.42	0.45	20.30	22.90
Average		14.89	14.52	0.41	0.41	21.51	20.33
Standard Deviation		0.7460	0.9041	0.0103	0.0226	2.0830	1.8679
Difference		-2.55%		0.49%		-5.80%	
Shapiro-Wilk Test	p-value*	0.4837	0.3266	0.0053	0.2721	0.1249	0.9645
	Normal Dist	YES	YES	NO	YES	YES	YES
F-Test	p-value*	0.2881		0.0144		0.3753	
	Homo. Vari.	YES		NO		YES	
T-Test	p-value*	0.1897				0.0550	
	Diff. Sig.	NO				NO	

* Reject the null hypothesis when p-value < 0.05

Ranks

	Groups	N	Mean Rank	Sum of Ranks
Stops	1.00	10	9.60	96.00
	2.00	10	11.40	114.00
	Total	20		

Test Statistics[b]

	Stops
Mann-Whitney U	41.000
Wilcoxon W	96.000
Z	-.708
Asymp. Sig. (2-tailed)	.479
Exact Sig. [2*(1-tailed Sig.)]	.529[a]

a. Not corrected for ties.

b. Grouping Variable: VAR00003

Figure 5.11 Mann-Whitney U Test for Stops with Different Population Size

We then increase the population size to 100, the evaluation results are shown in Table 5.9. The delay and stops have no obvious difference from the one with 50 as the population size. However, the queue length increased about 10% and the difference is statistically significant. This may caused by insufficient generations to obtain well optimized results when the population size is too large. From these results, we can find this program prefers 50 or 75 in population size and no obvious difference in 100 with the current computation capability.

Table 5.9 Evaluation Results with 100 Population Size

Random Seed		Delay (sec)		Stops		Queue Length (ft)	
		100	Default	100	Default	100	Default
1		15.70	14.10	0.44	0.40	23.60	18.70
2		13.90	13.40	0.39	0.37	19.10	18.40
3		14.70	13.60	0.40	0.39	23.40	17.60
4		14.80	15.90	0.38	0.42	24.50	23.40
5		16.00	15.10	0.40	0.42	23.10	21.20
6		15.00	14.50	0.43	0.42	22.40	19.70
7		15.30	14.90	0.42	0.42	21.10	20.40
8		16.10	14.00	0.46	0.42	22.80	20.90
9		15.00	13.80	0.41	0.39	21.30	20.10
10		13.70	15.90	0.36	0.45	22.20	22.90
Average		15.02	14.52	0.41	0.41	22.15	20.33
Standard Deviation		0.8039	0.9041	0.0296	0.0226	1.5400	1.8679
Difference		-3.44%		-0.24 %		-8.95%	
Shapiro-Wilk Test	p-value*	0.6016	0.3266	0.9995	0.2721	0.7788	0.9645
	Normal Dist	YES	YES	YES	YES	YES	YES
F-Test	p-value*	0.3660		0.2169		0.3809	
	Homo. Vari.	YES		YES		YES	
T-Test	p-value*	0.2464		0.9385		0.0372	
	Diff. Sig.	NO		NO		YES	

* Reject the null hypothesis when p-value < 0.05

5.2.6.2 Crossover Rate

According to Kovvali and Messer's study (24), crossover rate is recommended from 0.5 to 0.8 in traffic signal optimization. A higher crossover factor may yield a faster

convergence. However, if it is too high, premature convergence becomes a problem. Besides the default value, we also tried 0.6 and 0.8 for crossover rate in GABNOR to assess its sensitivity to this parameter.

The results of GABNOR with crossover rate at 0.6 are shown in Table 5.10. From the result, we can find that GABNOR has no statistical significant difference in delay and stops, while the average queue length is 10% longer statistically significant compared with the one in default.

Table 5.10 Evaluation Results with 0.6 Crossover Rate

Random Seed		Delay (sec)		Stops		Queue Length (ft)	
		0.6	Default	0.6	Default	0.6	Default
1		13.70	14.10	0.38	0.40	17.80	18.70
2		15.30	13.40	0.43	0.37	21.50	18.40
3		14.90	13.60	0.41	0.39	20.40	17.60
4		16.00	15.90	0.38	0.42	24.30	23.40
5		16.30	15.10	0.41	0.42	27.10	21.20
6		14.80	14.50	0.42	0.42	22.00	19.70
7		15.30	14.90	0.42	0.42	23.10	20.40
8		14.40	14.00	0.38	0.42	22.50	20.90
9		14.80	13.80	0.37	0.39	20.90	20.10
10		15.10	15.90	0.42	0.45	24.30	22.90
Average		15.06	14.52	0.40	0.41	22.39	20.33
Standard Deviation		0.7442	0.9041	0.0220	0.0226	2.5427	1.8679
Difference		-3.72%		1.95%		-10.13%	
Shapiro-Wilk Test	p-value*	0.8550	0.3266	0.0672	0.2721	0.9800	0.9645
	Normal Dist	YES	YES	YES	YES	YES	YES
F-Test	p-value*	0.2856		0.4688		0.1859	
	Homo. Vari.	YES		YES		YES	
T-Test	p-value*	0.0669		0.4280		0.0056	
	Diff. Sig.	NO		NO		YES	

* Reject the null hypothesis when p-value < 0.05

The results of evaluation with crossover rate at 0.8 are listed in Table 5.11. All the three performance measurements, delay, stops and queue length, are similar to the one in default crossover rate. According to T-Test, the difference is not statistically significant.

From these results, the crossover rate can be chosen among 0.6, 0.7 and 0.8 without obvious difference on system performance.

Table 5.11 Evaluation Results with 0.8 Crossover Rate

Random Seed		Delay (sec)		Stops		Queue Length (ft)	
		0.8	Default	0.8	Default	0.8	Default
1		14.90	14.10	0.41	0.40	20.30	18.70
2		13.70	13.40	0.39	0.37	18.90	18.40
3		14.10	13.60	0.42	0.39	18.50	17.60
4		15.20	15.90	0.41	0.42	22.50	23.40
5		16.20	15.10	0.44	0.42	23.30	21.20
6		15.50	14.50	0.43	0.42	23.80	19.70
7		16.80	14.90	0.41	0.42	22.60	20.40
8		14.90	14.00	0.40	0.42	23.80	20.90
9		14.70	13.80	0.40	0.39	19.90	20.10
10		14.00	15.90	0.40	0.45	20.10	22.90
Average		15.00	14.52	0.41	0.41	21.37	20.33
Standard Deviation		0.9764	0.9041	0.0152	0.0226	2.0434	1.8679
Difference		-3.31%		-0.24%		-5.12%	
Shapiro-Wilk Test	p-value*	0.7374	0.3266	0.4406	0.2721	0.1607	0.9645
	Normal Dist	YES	YES	YES	YES	YES	YES
F-Test	p-value*	0.4112		0.1278		0.3967	
	Homo. Vari.	YES		YES		YES	
T-Test	p-value*	0.1880		0.8971		0.1343	
	Diff. Sig.	NO		NO		NO	

* Reject the null hypothesis when p-value < 0.05

5.2.6.3 Mutation Rate

Mutation assists in preventing the Genetic Algorithm from local convergence. However, if the mutation rate is too high, it will prevent convergence and destroy successful genotypes. Besides 0.1 mutation rate used in the evaluation, other three values, 0.05, 0.15 and 0.20, are also examined in our experiment. As shown in Table 5.13, there is no significant difference in delay by using 0.05 instead of 0.1 as the mutation rate though the queue length is longer in 0.05 mutation rate. Since the variances in stops are not homogeneous, we use Mann-Whitney U test instead of T-Test to check the difference.

As shown in Figure 5.12, the difference is significant. This means using 0.05 as the mutation rate can further reduce vehicle stops without affect the performance of delay.

Table 5.12 Evaluation Results with 0.05 Mutation Rate

Random Seed		Delay (sec)		Stops		Queue Length (ft)	
		0.05	Default	0.05	Default	0.05	Default
1		14.20	14.10	0.37	0.40	21.70	18.70
2		13.50	13.40	0.37	0.37	19.10	18.40
3		13.70	13.60	0.39	0.39	20.30	17.60
4		15.50	15.90	0.38	0.42	23.50	23.40
5		15.00	15.10	0.38	0.42	22.10	21.20
6		15.00	14.50	0.40	0.42	21.60	19.70
7		15.50	14.90	0.40	0.42	23.90	20.40
8		14.00	14.00	0.38	0.42	21.30	20.90
9		14.60	13.80	0.39	0.39	22.10	20.10
10		15.00	15.90	0.40	0.45	23.50	22.90
Average		14.60	14.52	0.39	0.41	21.91	20.33
Standard Deviation		0.7180	0.9041	0.0117	0.0226	1.4940	1.8679
Difference		-0.55%		5.85%		-7.77%	
Shapiro-Wilk Test	p-value*	0.3727	0.3266	0.1239	0.2721	0.6021	0.9645
	Normal Dist	YES	YES	YES	YES	YES	YES
F-Test	p-value*	0.2516		0.0320		0.2581	
	Homo. Vari.	YES		NO		YES	
T-Test	p-value*	0.6209				0.0025	
	Diff. Sig.	NO				YES	

* Reject the null hypothesis when p-value < 0.05

Ranks

	Groups	N	Mean Rank	Sum of Ranks
Stops	1.00	10	7.35	73.50
	2.00	10	13.65	136.50
	Total	20		

Test Statistics[b]

	Stops
Mann-Whitney U	18.500
Wilcoxon W	73.500
Z	-2.425
Asymp. Sig. (2-tailed)	.015
Exact Sig. [2*(1-tailed Sig.)]	.015[a]

a. Not corrected for ties.

b. Grouping Variable: VAR00008

Figure 5.12 Mann-Whitney U Test for Stops with 0.05 and 0.10 Mutation Rate

As mutation rate raise to 0.15, there is no significant difference in delay and stops, though the queue length is longer as shown in Table 5.13. Since the objective function of GABNOR is minimizing the vehicle delay, 0.15 can still be considered as acceptable mutation rate.

Table 5.13 Evaluation Results with 0.15 Mutation Rate

Random Seed		Delay (sec)		Stops		Queue Length (ft)	
		0.15	Default	0.15	Default	0.15	Default
1		14.40	14.10	0.38	0.40	19.70	18.70
2		13.20	13.40	0.37	0.37	18.70	18.40
3		15.00	13.60	0.41	0.39	19.70	17.60
4		15.80	15.90	0.41	0.42	24.30	23.40
5		15.00	15.10	0.42	0.42	20.90	21.20
6		15.20	14.50	0.44	0.42	20.70	19.70
7		15.90	14.90	0.43	0.42	23.40	20.40
8		14.70	14.00	0.39	0.42	20.90	20.90
9		14.20	13.80	0.36	0.39	20.30	20.10
10		15.50	15.90	0.41	0.45	23.40	22.90
Average		14.89	14.52	0.40	0.41	21.20	20.33
Standard Deviation		0.8130	0.9041	0.0262	0.0226	1.8643	1.8679
Difference		-2.55%		1.95%		-4.28%	
Shapiro-Wilk Test	p-value*	0.6240	0.3266	0.7176	0.2721	0.2171	0.9645
	Normal Dist	YES	YES	YES	YES	YES	YES
F-Test	p-value*	0.3785		0.3353		0.4978	
	Homo. Vari.	YES		YES		YES	
T-Test	p-value*	0.0750		0.2695		0.0231	
	Diff. Sig.	NO		NO		YES	

* Reject the null hypothesis when p-value < 0.05

For 0.20 mutation rate, the system performance dropped down significantly as shown in Table 5.14. Though the difference in stops is not significant according to Mann-Whitney U test, 0.20 mutation rate is not yield the same performance compared to other mutation rates. This is also supported by Kovvali and Messer's study (24) in which the maximum recommended mutation rate is 0.19 for GA in traffic signal optimization.

Table 5.14 Evaluation Results with 0.20 Mutation Rate

Random Seed		Delay (sec)		Stops		Queue Length (ft)	
		0.20	Default	0.20	Default	0.20	Default
1		16.30	14.10	0.42	0.40	27.70	18.70
2		13.40	13.40	0.39	0.37	18.00	18.40
3		15.00	13.60	0.40	0.39	21.10	17.60
4		15.70	15.90	0.41	0.42	22.00	23.40
5		15.70	15.10	0.42	0.42	26.10	21.20
6		15.50	14.50	0.41	0.42	24.10	19.70
7		15.90	14.90	0.41	0.42	23.30	20.40
8		15.50	14.00	0.43	0.42	24.70	20.90
9		15.70	13.80	0.42	0.39	22.30	20.10
10		15.30	15.90	0.41	0.45	23.50	22.90
Average		15.40	14.52	0.41	0.41	23.28	20.33
Standard Deviation		0.7832	0.9041	0.0114	0.0226	2.6968	1.8679
Difference		-6.06%		-0.49%		-14.51%	
Shapiro-Wilk Test	p-value*	0.0743	0.3266	0.4788	0.2721	0.9753	0.9645
	Normal Dist	YES	YES	YES	YES	YES	YES
F-Test	p-value*	0.3379		0.0262		0.1445	
	Homo. Vari.	YES		NO		YES	
T-Test	p-value*	0.0146				0.0121	
	Diff. Sig.	YES				YES	

* Reject the null hypothesis when p-value < 0.05

Ranks

	Group	N	Mean Rank	Sum of Ranks
Stops	1.00	10	10.50	105.00
	2.00	10	10.50	105.00
	Total	20		

Test Statistics[b]

	Stops
Mann-Whitney U	50.000
Wilcoxon W	105.000
Z	.000
Asymp. Sig. (2-tailed)	1.000
Exact Sig. [2*(1-tailed Sig.)]	1.000[a]

a. Not corrected for ties.

b. Grouping Variable: Group

Figure 5.13 Mann-Whitney U Test for Stops with 0.20 and 0.10 Mutation Rate

According to the above tests, the recommended values for the parameters of GA in GABNOR are summarized in Table 5.15. It could be the guide for future experiments of GABNOR.

Table 5.15 Suggested Values for Parameters of GABNOR

Parameters	Suggested Values
Population Size	50, 75 or 100
Crossover Rate	0.6, 0.7 or 0.8
Mutation Rate	0.05, 0.10 or 0.15

CHAPTER VI

CONCLUSIONS AND FUTURE WORKS

Adaptive signal control strategy for isolated intersection and traffic network has been introduced in this dissertation. The limitation and problems in existing traffic optimization strategies are addressed and used as the basis for developing PODE for isolated intersection and GABNOR for traffic network. The methodology of PODE and GABNOR has been discussed in detail and the implementation of both algorithms is presented. Simulation based evaluation has shown the competitive optimization capability of both algorithms. For PODE, the results demonstrate the encouraging potential in solving traffic congestion problem at isolated intersections, and it is supported by the system sensitivity test results. As for GABNOR, it has shown its competitive capability to optimize not only arterial, but also grid traffic network. Parallel computation feature helps GABNOR optimize the whole traffic network in real-time. System sensitivity tests demonstrate its independence to the parameters and its fault tolerance feature allows the system works stably under different situations.

However, both PODE and GABNOR are just a simple prototype of a mature real-time optimization system. The simulation results are very limited and more scenarios should be tested for each system. Some extreme situations, such as traffic accidents or oversaturated conditions, should be included in the evaluation. Moreover, additional field

tests should be considered to examine the system. Besides the above tasks, using microscopic internal evaluator instead of the current mesoscopic could further improve the performance of GABNOR as more accurate and detailed information can be obtained. To keep advancing PODE and GABNOR, it is important to incorporate the leading edge technologies, such as Vehicle Infrastructure Integration (VII) (32), into the system.

REFERENCES

1. 2007 Urban Mobility Report, Texas Transportation Institute, September 2007, http://tti.tamu.edu/documents/mobility_report_2007_wappx.pdf

2. Sims, A. G., and K. W. Dobinson. 1980. "The Sydney Coordinated Adaptive Traffic (SCAT) System: Philosophy and Benefits". *IEEE Transactions on Vehicular Technology.* VT-29, no. 2.

3. Australia. RTA Annual Report 2007. New South Wales Government, 2007.

4. Hunt, P.B, D.I Robertson, R.D Bretherton, and R.I Winton. SCOOT: A Traffic Responsive Method of Coordinating Signals. 1981.

5. Gartner, Nathan H. *OPAC: A Demand-Responsive Strategy for Traffic Signal Control*. [Washington, D.C.]: Transportation Research Board, National Research Council, 1983.

6. Head, K. Larry, Pitu B. Mirchandani, and Dennis Sheppard. 1992. "Hierarchical Framework for Real-Time Traffic Control". *Transportation Research Record.* no. 1360.

7. Gartner, N.H., et al, Implementation of the OPAC Adaptive Control Strategy in a Traffic Signal Network, *2001 IEEE Intelligent Transportation Systems Conference Proceedings*, 2001

8. Sen, S. and Head, K.L., Controlled Optimization of Phases at an Intersection, *Transportation Science Vol. 31 No. 1*, 1997

9. Dell'Olmo, P. and Mirchandani, P.B., REALBAND: An Approach for Real-Time Coordination of Traffic Flows on Networks, *Transportation Research Record 1494*, 1995

10. Gartner, N.H., Tarnoff, P.J., and Andrews, C.M., Evaluation of Optimized Policies for Adaptive Control Strategy, *Transportation Research Record 1324*, 1991

11. Mirchandani, P. and Head, L., RHODES: A Real-Time Traffic Signal Control System: Architecture, Algorithms, and Analysis, *Transportation Research Record, Part B*, 2001

12. Saiyed, S. and Stewart, J. A., An Assessment of Pre-timed, Actuated and Adaptive Signal Control Strategies for Unsaturated and Saturated Arterial Network, *Transportation Research Board*, 2004

13. Lawrence A Klein, *Sensor Technologies and Data Requirements for ITS*, ISBN 158053077X, Artech House Publisher, Boston, London, Jan 1, 2001

14. U.S. Department of Transportation/Office of the Secretary of Transportation, *Demand-Responsive Decentralized Urban Traffic Control Part II: Network Extensions*, 1983

15. Head, K.L., Event-Based Short-Term Traffic Flow Prediction Model, *Transportation Research Record 1510*, 1995

16. Holland, John H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Ann Arbor: University of Michigan Press, 1975.

17. Foy, M. D., R. F. Benekohal, and D. E. Goldberg. 1992. "Signal Timing Determination Using Genetic Algorithms". *TRANSPORTATION RESEARCH RECORD.* no. 1365: 108.

18. Hadi, M. A., and C. E. Wallace. 1993. "Hybrid Genetic Algorithm To Optimize Signal Phasing and Timing". *TRANSPORTATION RESEARCH RECORD.* no. 1421: 104.

19. Park, Byungkyu, Carroll J Messer, and Thomas Urbanik II. 1999. "Traffic Signal Optimization Program for Oversaturated Conditions: Genetic Algorithm Approach". *Transportation Research Record.* no. 1683: 133.

20. Park, Byungkyu, Nagui Rouphail, and Jerome Sacks. 2001. "Assessment of Stochastsic Signal Optimization Method Using Microsimulation". *Transportation Research Record.* no. 1748: 40-45.

21. Lee, J., B. Abdulhai, A. Shalaby, and E.-H. Chung. 2005. "Real-Time Optimization for Adaptive Traffic Signal Control Using Genetic Algorithms". *JOURNAL OF INTELLIGENT TRANSPORTATION SYSTEMS.* 9, no. 3: 111-122.

22. P. Yi, C. Shao, and Y. Wang. 2008 " Piecewise Optimum Delay Estimation for Improved Signal Control". *Will be published in Transportation Research Record 2008.*

23. Gartner, N.H., C.J. Messer and A.K. Rathi, *Chapter 4 Car Following Models, Revised Monograph on Traffic Flow Theory*, Transportation Research Board, 1997

24. Kovvali, V. G., and C. J. Messer. 2002. "SENSITIVITY ANALYSIS OF GENETIC ALGORITHM PARAMETERS IN TRAFFIC SIGNAL OPTIMIZATION". *81st Annual Meeting of the Transportation Research Board*

25. Microsoft, "Using Threading (C# Programming Guide)", http://msdn.microsoft.com/en-us/library/5xt1dysy(VS.80).aspx, July, 2008

26. Microsoft, "Using an Asynchronous Server Socket", http://msdn.microsoft.com/en-us/library/5w7b7x5f(VS.71).aspx, July, 2008

27. Shelby, S. G., D. M. Bullock, and D. Gettman. 2006. "Transition Methods in Traffic Signal Control". *TRANSPORTATION RESEARCH RECORD.* no. 1978: 130-140.

28. PTV AG: VISSIM, November, 2009, http://www.ptvag.com/traffic/software/vissim/

29. TRB, *Highway Capacity Manual*, National Research Council, Washington, D. C., 2000

30. Li, H.L., Zhang, L., and Gartner, N.H., Comparative Evaluation Of Three Adaptive Control Strategies: Opac, Tacos, And FLC, *Transportation Research Board*, 2006

31. Shapiro, S. S. and Wilk, M. B. (1965). "An analysis of variance test for normality (complete samples)", Biometrika, 52, 3 and 4, pages 591-611.

32. Vehicle Infrastructure Integration (VII), November, 2009, http://www.vehicle-infrastructure.org/

APPENDICES

APPENDIX A

SAMPLE CONFIGURATION FILE FOR PODE

```
<IOMConfig>
    <minTimeSteps>5</minTimeSteps>
    <maxTimeSteps>20</maxTimeSteps>
    <timeSlice>1</timeSlice>
    <yellow>3</yellow>
    <allred>2</allred>
    <startOffDelay>1</startOffDelay>
    <queueWaveCars>0.8</queueWaveCars>
    <leaveCars>1</leaveCars>
    <systemDistance>450</systemDistance>
    <avgVehQueueSpacing>6</avgVehQueueSpacing>
    <avgVehLength>16</avgVehLength>
    <limitSpeed>50</limitSpeed>
    <maxExceed>0</maxExceed>
    <maxDifference>100</maxDifference>
    <maxGreen>100</maxGreen>
    <changeDifference>0</changeDifference>
    <reducePara>4</reducePara>
    <reduceTime>7</reduceTime>
    <queueMaxSpeedPer>0.8</queueMaxSpeedPer>
    <queueSpeedupTime>8</queueSpeedupTime>
    <volume>800 800 800 800</volume>
    <pedestrian>
        <highVolume>30</highVolume>
        <mediumVolume>6</mediumVolume>
        <highDiff>14</highDiff>
        <mediumDiff>7</mediumDiff>
        <lowDiff>0</lowDiff>
        <maximumOut>30</maximumOut>
    </pedestrian>
    <detConfig node=1 sgNum=8 approachNum=4 phaseNum=8 >
        <approach index=0 laneNum=3 type="veh">
            <lane index=0 queueDetNum=3 lane=1 queueLeaveCars=0.59
weight=1 >
                <countDet id=33 distance=1000 />
```

```
                <queueDet id=1 index=0 distance=12 queuemin=1
queuemax=5 />
                <queueDet id=2 index=1 distance=94 queuemin=6
queuemax=9 />
                <queueDet id=3 index=2 distance=174 queuemin=10
queuemax=1000 />
            </lane>
            <lane index=1 queueDetNum=3 lane=2 queueLeaveCars=0.59
weight=1 >
                <countDet id=34 distance=1000 />
                <queueDet id=4 index=0 distance=12 queuemin=1
queuemax=5 />
                <queueDet id=5 index=1 distance=94 queuemin=6
queuemax=9 />
                <queueDet id=6 index=2 distance=174 queuemin=10
queuemax=1000 />
            </lane>
            <lane index=2 queueDetNum=3 lane=7 queueLeaveCars=0.59
weight=1 >
                    <countDet id=-1 distance=0 />
                    <queueDet id=7 index=0 distance=12 queuemin=1
queuemax=5 />
                    <queueDet id=8 index=1 distance=94 queuemin=6
queuemax=9 />
                    <queueDet id=41 index=2 distance=174 queuemin=10
queuemax=1000 />
                </lane>
        </approach>
        <approach index=1 laneNum=3 type="veh" >
            <lane index=0 queueDetNum=3 lane=1 queueLeaveCars=0.59
weight=1 >
                <countDet id=35 distance=1000 />
                <queueDet id=9 index=0 distance=12 queuemin=1
queuemax=5 />
                <queueDet id=10 index=1 distance=94 queuemin=6
queuemax=9 />
                <queueDet id=11 index=2 distance=174 queuemin=10
queuemax=1000 />
            </lane>
            <lane index=1 queueDetNum=3 lane=2 queueLeaveCars=0.59
weight=1 >
                <countDet id=36 distance=1000 />
                <queueDet id=12 index=0 distance=12 queuemin=1
queuemax=5 />
                <queueDet id=13 index=1 distance=94 queuemin=6
queuemax=9 />
                <queueDet id=14 index=2 distance=174 queuemin=10
queuemax=1000 />
            </lane>
            <lane index=2 queueDetNum=3 lane=7 queueLeaveCars=0.59
weight=1 >
                <countDet id=-1 distance=0 />
                <queueDet id=15 index=0 distance=12 queuemin=1
queuemax=5 />
                <queueDet id=16 index=1 distance=94 queuemin=6
queuemax=9 />
```

```
                <queueDet id=42 index=2 distance=174 queuemin=10
queuemax=1000 />
            </lane>
        </approach>
        <approach index=2 laneNum=3 type="veh" >
            <lane index=0 queueDetNum=3 lane=1 queueLeaveCars=0.59
weight=1 >
                <countDet id=37 distance=1000 />
                <queueDet id=17 index=0 distance=12 queuemin=1
queuemax=5 />
                <queueDet id=18 index=1 distance=94 queuemin=6
queuemax=9 />
                <queueDet id=19 index=2 distance=174 queuemin=10
queuemax=1000 />
            </lane>
            <lane index=1 queueDetNum=3 lane=2 queueLeaveCars=0.59
weight=1 >
                <countDet id=38 distance=1000 />
                <queueDet id=20 index=0 distance=12 queuemin=1
queuemax=5 />
                <queueDet id=21 index=1 distance=94 queuemin=6
queuemax=9 />
                <queueDet id=22 index=2 distance=174 queuemin=10
queuemax=1000 />
            </lane>
            <lane index=2 queueDetNum=3 lane=7 queueLeaveCars=0.59
weight=1 >
                <countDet id=-1 distance=0 />
                <queueDet id=23 index=0 distance=12 queuemin=1
queuemax=5 />
                <queueDet id=24 index=1 distance=94 queuemin=6
queuemax=9 />
                <queueDet id=43 index=2 distance=174 queuemin=10
queuemax=1000 />
            </lane>
        </approach>
        <approach index=3 laneNum=3 type="veh" >
            <lane index=0 queueDetNum=3 lane=1 queueLeaveCars=0.59
weight=1 >
                <countDet id=39 distance=1000 />
                <queueDet id=25 index=0 distance=12 queuemin=1
queuemax=5 />
                <queueDet id=26 index=1 distance=94 queuemin=6
queuemax=9 />
                <queueDet id=27 index=2 distance=174 queuemin=10
queuemax=1000 />
            </lane>
            <lane index=1 queueDetNum=3 lane=2 queueLeaveCars=0.59
weight=1 >
                <countDet id=40 distance=1000 />
                <queueDet id=28 index=0 distance=12 queuemin=1
queuemax=5 />
                <queueDet id=29 index=1 distance=94 queuemin=6
queuemax=9 />
                <queueDet id=30 index=2 distance=174 queuemin=10
queuemax=1000 />
```

```
            </lane>
            <lane index=2 queueDetNum=3 lane=7 queueLeaveCars=0.59
weight=1 >
                <countDet id=-1 distance=0 />
                <queueDet id=31 index=0 distance=12 queuemin=1
queuemax=5 />
                <queueDet id=32 index=1 distance=94 queuemin=6
queuemax=9 />
                <queueDet id=44 index=2 distance=174 queuemin=10
queuemax=1000 />
            </lane>
        </approach>
        <sg index=0 minGreen=6 approachNum=1 clearance=0 >
            <approach index=0 laneNum=2 >
                <lane index=0 />
                <lane index=1 />
            </approach>
        </sg>
        <sg index=1 minGreen=6 approachNum=1 clearance=0 >
            <approach index=0 laneNum=1 >
                <lane index=2 />
            </approach>
        </sg>
        <sg index=2 minGreen=6 approachNum=1 clearance=0 >
            <approach index=1 laneNum=2 >
                <lane index=0 />
                <lane index=1 />
            </approach>
        </sg>
        <sg index=3 minGreen=6 approachNum=1 clearance=0 >
            <approach index=1 laneNum=1 >
                <lane index=2 />
            </approach>
        </sg>
        <sg index=4 minGreen=6 approachNum=1 clearance=0 >
            <approach index=2 laneNum=2 >
                <lane index=0 />
                <lane index=1 />
            </approach>
        </sg>
        <sg index=5 minGreen=6 approachNum=1 clearance=0 >
            <approach index=2 laneNum=1 >
                <lane index=2 />
            </approach>
        </sg>
        <sg index=6 minGreen=6 approachNum=1 clearance=0 >
            <approach index=3 laneNum=2 >
                <lane index=0 />
                <lane index=1 />
            </approach>
        </sg>
        <sg index=7 minGreen=6 approachNum=1 clearance=0 >
            <approach index=3 laneNum=1 >
                <lane index=2 />
            </approach>
        </sg>
```

```
<phase index=0 >
    <sg index=0 code=2 />
    <sg index=1 code=1 />
    <sg index=2 code=2 />
    <sg index=3 code=2 />
    <sg index=4 code=2 />
    <sg index=5 code=1 />
    <sg index=6 code=2 />
    <sg index=7 code=2 />
</phase>
<phase index=1 >
    <sg index=0 code=1 />
    <sg index=1 code=2 />
    <sg index=2 code=2 />
    <sg index=3 code=2 />
    <sg index=4 code=1 />
    <sg index=5 code=2 />
    <sg index=6 code=2 />
    <sg index=7 code=2 />
</phase>
<phase index=2 >
    <sg index=0 code=2 />
    <sg index=1 code=2 />
    <sg index=2 code=2 />
    <sg index=3 code=1 />
    <sg index=4 code=2 />
    <sg index=5 code=2 />
    <sg index=6 code=2 />
    <sg index=7 code=1 />
</phase>
<phase index=3 >
    <sg index=0 code=2 />
    <sg index=1 code=2 />
    <sg index=2 code=1 />
    <sg index=3 code=2 />
    <sg index=4 code=2 />
    <sg index=5 code=2 />
    <sg index=6 code=1 />
    <sg index=7 code=2 />
</phase>
<phase index=4 >
    <sg index=0 code=1 />
    <sg index=1 code=1 />
    <sg index=2 code=2 />
    <sg index=3 code=2 />
    <sg index=4 code=2 />
    <sg index=5 code=2 />
    <sg index=6 code=2 />
    <sg index=7 code=2 />
</phase>
<phase index=5 >
    <sg index=0 code=2 />
    <sg index=1 code=2 />
    <sg index=2 code=1 />
    <sg index=3 code=1 />
    <sg index=4 code=2 />
```

```
            <sg index=5 code=2 />
            <sg index=6 code=2 />
            <sg index=7 code=2 />
        </phase>
        <phase index=6 >
            <sg index=0 code=2 />
            <sg index=1 code=2 />
            <sg index=2 code=2 />
            <sg index=3 code=2 />
            <sg index=4 code=1 />
            <sg index=5 code=1 />
            <sg index=6 code=2 />
            <sg index=7 code=2 />
        </phase>
        <phase index=7 >
            <sg index=0 code=2 />
            <sg index=1 code=2 />
            <sg index=2 code=2 />
            <sg index=3 code=2 />
            <sg index=4 code=2 />
            <sg index=5 code=2 />
            <sg index=6 code=1 />
            <sg index=7 code=1 />
        </phase>
    </detConfig>
</IOMConfig>
```

APPENDIX B

SAMPLE SOURCE CODE

sc_dll_main.cpp

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

void  SC_DLL_Calculate (unsigned long sc_no)
{
    /* Executes one pass through the controller logic of SC no. <sc_no>.
*/
    /* This function is called from VISSIM once per SC at the end of
each */
    /* signal control interval, after the (detector) data for all SC's
*/
    /* has been passed to the controller DLL.
*/

    /* ### */


/*******************************************************************
**/
    /* Init the parameter
*/

/*******************************************************************
**/
    int l = 0;    // Loop for Link
    int p = 0;    // Loop for Phase
    int ap = 0; // Loop for Approach
    int la = 0; // Loop for Lane
    int d = 0;    // Loop for Detector

    double initQueueMin[MAX_APPROACH_NUM][MAX_LANE_NUM];
```

```
    double initQueueMax[MAX_APPROACH_NUM][MAX_LANE_NUM];

    for ( ap = 0; ap < MAX_APPROACH_NUM; ap++ ) {
        for ( la = 0; la < MAX_LANE_NUM; la++ ) {
            for ( d = 0; d < MAX_QUEUE_DET; d++ ) {
                detState[ap][la][d] = false;
                detAct[ap][la][d] = false;
            }
            initQueueMin[ap][la] = 0;
            initQueueMax[ap][la] = FLT_MAX;
        }
    }


/*********************************************************************
**/
    /* Process the node and adjust the queue
*/
    /* Use the PODE algorithm to get the result
*/

/*********************************************************************
**/


/*********************************************************************
**/
    /* Get the detectors' information and calculate the limitation of
queue */

/*********************************************************************
**/
    for ( ap = 0; ap < para.detConfig->approachNum; ap++ ) {
        for ( la = 0; la < objAlgPODE-
>GetIntersectionApproach(para.detConfig, ap)->laneNum; la++ ) {
            // Check the count detector, get the speed
            Lane* lane = &objAlgPODE-
>GetIntersectionApproach(para.detConfig, ap)->lanes[la];
            if ( lane->countDet.id > 0 )
            {
                double speed = Det_VehSpeed(sc_no, lane->countDet.id) *
FT_SPEED;
                if ( speed > 0 && Det_FrontEnds(sc_no, lane-
>countDet.id) > 0 )
                {

/*********************************************************************
**/
                    /* Deal with the pedestrian queue
*/

/*********************************************************************
**/
                    if ( objAlgPODE-
>GetIntersectionApproach(para.detConfig, ap)->type == T_APP_PED )
```

```
                    {
                        queue[ap][la].queueSize += 1;
                    }
                    else
                    {

/*********************************************************************
**/
                        /* Calculate the arriving vehicle to estimate
the queue length          */

/*********************************************************************
**/
                        int intArrivingVeh = arrivalNumber[ap][la];
                        for ( int i = (int)(objResult->timeSteps -
(Sim_Time() - sg_start_time)); i < MAX_ARRIVAL_BUFFER_STEP; i++ )
                        {
                            if ( input[i][ap][la] > 0 )
                            {
                                intArrivingVeh++;
                            }
                        }


/*********************************************************************
**/
                        /* Calculate the estimated arrival time
*/

/*********************************************************************
**/
                        double queueLength = queue[ap][la].queueLength
+ intArrivingVeh * ( para.avgVehLength + para.avgVehQueueSpacing );
                        int time = 0;
                        if ( lane->countDet.distance > queueLength )
                        {
                            if ( speed > para.limitSpeed * 0.2 ||
queueLength < para.systemDistance )
                            {
                                if ( queueLength < para.systemDistance )
                                {
                                    time = (int)(( lane-
>countDet.distance - para.systemDistance ) / ( speed  *
para.timeSlice ));
                                }
                                else
                                {
                                    time = (int)(( lane-
>countDet.distance - queueLength ) / ( speed  * para.timeSlice ) +
para.reduceTime / 2);
                                }

                            }
                        }
```

```
/*********************************************************************
**/
                        /* No two vehicles arrive at the same second
*/

/*********************************************************************
**/
                        while ( input[time][ap][la] > 0 ) //||
Sim_Time() - sg_start_time + time <= objResult->timeSteps+5 )
                        {
                            time++;
                        }
                        if ( time >= MAX_ARRIVAL_BUFFER_STEP ) {
                            time = MAX_ARRIVAL_BUFFER_STEP - 1;
                        }

                        input[time][ap][la] = speed;
                        totalInput[(int)(time+Sim_Time())][ap][la] =
speed;
                    }
                }
            }

            // Check the Queue Detector
            for ( d = 0; d < lane->queueDetNum; d++ ) {
                int detPres = Det_Presence(sc_no, lane-
>queueDets[d].id);
                if( detPres > 0 )
                {
                    detAct[ap][la][d] = true;
                    if( Det_FrontEnds(sc_no, lane->queueDets[d].id) ==
0 ) detState[ap][la][d] = true;
                }
            }
        }
    }


/*********************************************************************
**/
    /* Adjust the queue size
*/

/*********************************************************************
**/
    for ( ap = 0; ap < para.detConfig->approachNum; ap++ ) {
        for ( la = 0; la < objAlgPODE-
>GetIntersectionApproach(para.detConfig, ap)->laneNum; la++ ) {
            Lane* lane = &objAlgPODE-
>GetIntersectionApproach(para.detConfig, ap)->lanes[la];
            for ( d = lane->queueDetNum - 1; d >= 0; d-- ) {
                if ( detState[ap][la][d] ) {
                    for ( int tempD = 0; tempD < d; tempD++ ) {
                        if ( detAct[ap][la][tempD] ) {
                            break;
```

```
                        }
                    }
                    if ( tempD < d || d == 0 ) {
                        initQueueMin[ap][la] = lane-
>queueDets[d].queuemin;
                        initQueueMax[ap][la] = lane-
>queueDets[d].queuemax;
                        break;
                    }
                }
            }
        }
    }



/*********************************************************************
**/
    /* Roll up the input array
*/

/*********************************************************************
**/

    for ( int i = 0; i < MAX_ARRIVAL_BUFFER_STEP - 1; i++ ) {
        for ( ap = 0; ap < MAX_APPROACH_NUM; ap++ ) {
            for ( la = 0; la < MAX_LANE_NUM; la++ ) {
                input[i][ap][la] = input[i+1][ap][la];
            }
        }
    }

    for ( ap = 0; ap < MAX_APPROACH_NUM; ap++ ) {
        for ( la = 0; la < MAX_LANE_NUM; la++ ) {
            input[MAX_ARRIVAL_BUFFER_STEP-1][ap][la] = 0;
        }
    }




/*********************************************************************
**/
    /* Adjust the queue information
*/
    /* and calculate the new result
*/

/*********************************************************************
**/

    if ( Sim_Time() - sg_start_time >= objResult->timeSteps )
    {
        sg_start_time = Sim_Time();
        //initialize input file as 0 in each horizon for next
prediction
```

```
        for ( ap = 0; ap < para.detConfig->approachNum; ap++ ) {
            for ( la = 0; la < objAlgPODE-
>GetIntersectionApproach(para.detConfig, ap)->laneNum; la++ ) {
                if ( objAlgPODE->GetIntersectionApproach(para.detConfig,
ap)->type == T_APP_PED || ( !objAlgPODE->isLaneInPhase(ap, la,
objResult->firstPhase, para) && waitingTime[ap][la] >
para.maxTimeSteps ) ) {
                    if( queue[ap][la].queueSize < initQueueMin[ap][la] )
queue[ap][la].queueSize = initQueueMin[ap][la];
                    if( queue[ap][la].queueSize > initQueueMax[ap][la] )
queue[ap][la].queueSize = initQueueMax[ap][la];
                    queue[ap][la].queueLength = queue[ap][la].queueSize
* (para.avgVehQueueSpacing + para.avgVehLength);
                    queue[ap][la].queueStatic = queue[ap][la].queueSize;
                    queue[ap][la].queueSpeed = 0;
                }
            }
        }


/*****************************************************************
**/
        /* Output the current system situation
*/

/*****************************************************************
**/
        TiXmlDocument log( "log.xml" );

        TiXmlElement* x_Time = new TiXmlElement("Time");
        x_Time->SetAttribute("index", (int)Sim_Time());

        for ( int i = 0; i < para.detConfig->approachNum; i++ )
        {
            TiXmlElement* x_Approach = new TiXmlElement("Approach");
            x_Approach->SetAttribute("index", i);
            x_Approach->SetAttribute("laneNum", objAlgPODE-
>GetIntersectionApproach(para.detConfig, i)->laneNum);

            for ( int j = 0; j < objAlgPODE-
>GetIntersectionApproach(para.detConfig, i)->laneNum; j++ )
            {
                TiXmlElement* x_Lane = new TiXmlElement("Lane");
                x_Lane->SetAttribute("index", j);
                x_Lane->SetAttribute("sysNum", arrivalNumber[i][j]);

                TiXmlElement* x_Queue = new TiXmlElement("Queue");
                x_Queue->SetAttribute("length",
(int)queue[i][j].queueLength);
                x_Queue->SetAttribute("size",
(int)queue[i][j].queueSize);
                x_Queue->SetAttribute("static",
(int)queue[i][j].queueStatic);
                x_Queue->SetAttribute("speed",
(int)queue[i][j].queueSpeed);
```

```
                x_Lane->LinkEndChild(x_Queue);

                for ( int k = 0; k < arrivalNumber[i][j]; k++ )
                {
                    TiXmlElement* x_SysVeh = new TiXmlElement("SysVeh");
                    x_SysVeh->SetAttribute("speed",
(int)queueArrival[i][j][k].speed);
                    x_SysVeh->SetAttribute("distance",
(int)queueArrival[i][j][k].distance);
                    x_Lane->LinkEndChild(x_SysVeh);
                }

                for ( int k = 0; k < MAX_ARRIVAL_BUFFER_STEP; k++ )
                {
                    if ( input[k][i][j] > 0 )
                    {
                        TiXmlElement* x_ArrVeh = new
TiXmlElement("ArrVeh");
                        x_ArrVeh->SetAttribute("arrTime", k);
                        x_ArrVeh->SetAttribute("speed",
(int)input[k][i][j]);
                        x_Lane->LinkEndChild(x_ArrVeh);
                    }
                }

                x_Approach->LinkEndChild(x_Lane);
            }

            x_Time->LinkEndChild(x_Approach);
        }



/*********************************************************************
**/
        /* Optimize Next Time Slice
*/

/*********************************************************************
**/
        if ( Sim_Time() > 30 )
        {
            int a = 0;
        }
        objLastResult = objResult;
        objResult = objAlgPODE->PODE(input, queue, queueArrival,
arrivalNumber, initDelay, para, greenTime, sgGreenTime, greenExtension,
pedWaitingTime, pingpongSwitch, objResult, &fDiff, (int)Sim_Time());


        totalDelay += objResult->delay;

        for ( int i = 0; i < objResult->timeSteps; i++ )
        {
            for ( ap = 0; ap < para.detConfig->approachNum; ap++ )
            {
```

```
                for ( la = 0; la < objAlgPODE-
>GetIntersectionApproach(para.detConfig, ap)->laneNum; la++ )
                {
                    if ( input[i][ap][la] > 0 )
                    {
                        totalVehicle += 1;
                        totalDelay -= para.systemDistance /
input[i][ap][la];
                    }
                }
            }
        }


/*****************************************************************
**/
        /* Find out ping pong switch SGs
*/

/*****************************************************************
**/
        bool bolRemain = false;
        int* lastResultCode = objAlgPODE->GetPhaseCode(objLastResult,
objLastResult->timeSteps-1)->phaseCodes;
        int* resultCode = objAlgPODE->GetPhaseCode(objResult,
objResult->timeSteps-1)->phaseCodes;
        if ( lastResultCode != NULL && resultCode != NULL )
        {
            for ( int i = 0; i < para.detConfig->sgNum; i++ )
            {
                if ( lastResultCode[i] == SG_STATE_GREEN &&
resultCode[i] == SG_STATE_GREEN )
                {
                    // We have same signal group remain green in the
new phase
                    bolRemain = true;
                    break;
                }
            }

        }

        for ( int i = 0; i < para.detConfig->sgNum; i++ )
        {
            if ( bolRemain )
            {
                if ( lastResultCode[i] == SG_STATE_GREEN &&
resultCode[i] != SG_STATE_GREEN )
                {
                    pingpongSwitch[i] = true;
                }
            }
            else
            {
                pingpongSwitch[i] = false;
            }
```

```
        }


/*********************************************************************
**/
        /* Record the phase and length
*/

/*********************************************************************
**/
        if ( objLastResult->firstPhase == objResult->firstPhase )
        {
            // Continue phase
            phaseLength[phaseNumber[objResult->firstPhase]-
1][objResult->firstPhase] += objResult->timeSteps;
        }
        else
        {
            // New phase
            phaseLength[phaseNumber[objResult->firstPhase]][objResult-
>firstPhase] = objResult->timeSteps - para.yellow - para.allred;
            phaseNumber[objResult->firstPhase] += 1;
        }


        x_Time->SetAttribute("length", objResult->timeSteps);
        x_Time->SetAttribute("code", objAlgPODE-
>PhaseCodeToString(objResult->phaseCode, ",").data());

        log.LinkEndChild(x_Time);

        log.AppendFile();
        log.Clear();

    }



/*********************************************************************
**/
    /* Set the traffic lights according to the result
*/

/*********************************************************************
**/
    for ( int i = 0; i < para.detConfig->sgNum; i++ )
    {
        int resultCode = objAlgPODE->GetPhaseCode(objResult,
(int)(Sim_Time()-sg_start_time))->phaseCodes[i];
        SG_SetState(sc_no, i+1, resultCode, 0);

        // Adjust green extension
        if ( resultCode == SG_STATE_GREEN || resultCode ==
SG_STATE_AMBER )
        {
```

```
            greenExtension[i] > 0 ? greenExtension[i]-- :
greenExtension[i]=0;
        }
    }


/*********************************************************************
**/
    /* Calculate the green time
*/

/*********************************************************************
**/
    int phase = objAlgPODE->GetPhaseCode(objResult, (int)(Sim_Time()-
sg_start_time))->phase;

    for ( ap = 0; ap < para.detConfig->approachNum; ap++ )
    {
        for ( la = 0; la < objAlgPODE-
>GetIntersectionApproach(para.detConfig, ap)->laneNum; la++ )
        {
            waitingTime[ap][la]++;
            if ( queue[ap][la].queueSize > 0 )
            {
                pedWaitingTime[ap][la]++;
            }
            else
            {
                pedWaitingTime[ap][la] = 0;
            }
            greenTime[ap][la]++;

            if ( phase >= 0 )
            {
                if ( objAlgPODE->isLaneInGreen(ap, la, objResult,
para) )
                {
                    waitingTime[ap][la] = 0;
                    pedWaitingTime[ap][la] = 0;
                }
                else
                {
                    greenTime[ap][la] = 0;
                }
            }
        }
    }
    //End of PODE
} /* SC_DLL_Calculate */
```

APPENDIX C

TRAFFIC DATABASE TABLE CREATION SQL FILE

```
/******************************************************/
/**                  Detections                      **/
/******************************************************/

USE [TrafficNetwork]
GO
/****** Object:  Table [dbo].[Detections]    Script Date: 11/10/2008
11:00:47 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[Detections](
      [ID] [bigint] IDENTITY(1,1) NOT NULL,
      [IntersectionID] [int] NOT NULL,
      [DetectorID] [int] NOT NULL,
      [ActiveTime] [datetime] NOT NULL,
      [Duration] [int] NOT NULL,
      [Type] [varchar](50) NOT NULL,
      [Memo] [varchar](max) NOT NULL,
 CONSTRAINT [PK_Detections] PRIMARY KEY CLUSTERED
(
      [ID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
SET ANSI_PADDING OFF
```

```
/*****************************************************/
/**                  SignalLogs                     **/
/*****************************************************/

USE [TrafficNetwork]
GO
/****** Object:  Table [dbo].[SignalLogs]    Script Date: 11/10/2008
11:01:44 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[SignalLogs](
      [IntersectionID] [int] NOT NULL,
      [PhaseTime] [datetime] NOT NULL,
      [PhaseCode] [varchar](64) NOT NULL,
 CONSTRAINT [PK_SignalLogs_1] PRIMARY KEY CLUSTERED
(
      [IntersectionID] ASC,
      [PhaseTime] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
SET ANSI_PADDING OFF




/*****************************************************/
/**                VehicleMovements                 **/
/*****************************************************/

USE [TrafficNetwork]
GO
/****** Object:  Table [dbo].[VehicleMovements]    Script Date:
11/10/2008 11:02:18 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[VehicleMovements](
      [ID] [bigint] IDENTITY(1,1) NOT NULL,
      [IntersectionID] [int] NOT NULL,
      [Movement] [varchar](255) NOT NULL,
      [EnterActiveTime] [datetime] NOT NULL,
      [EnterDeactiveTime] [datetime] NOT NULL,
      [ExitActiveTime] [datetime] NOT NULL,
      [ExitDeactiveTime] [datetime] NOT NULL,
      [Type] [tinyint] NOT NULL,
```

```
 CONSTRAINT [PK_VehicleMovements_IntersectionID] PRIMARY KEY CLUSTERED
(
      [ID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
SET ANSI_PADDING OFF
```