A Planning Model for Optimizing Locations of Changeable Message Signs

by

Jeffrey M. Henderson

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Civil Engineering

Waterloo, Ontario, Canada, 2004

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Changeable Message Signs (CMS) are commonly utilized by transportation agencies to inform motorists of traffic, roadway, and environmental conditions. They may be used to provide information, such as delay and alternate route guidance, in the event of an incident, construction or a roadway closure. The effectiveness of CMS in managing freeway traffic, however, is a function of many factors including the number of CMS installations, the location of CMS, the messages displayed, varied traffic network characteristics, and drivers' response to incident conditions and CMS information. The objective of this thesis is to develop a CMS location planning model that can be used by transportation agencies to develop a CMS location plan that could achieve the largest long-term benefit to the system.

This research is mainly motivated by the lack of systematic, robust and practical methods for locating CMS. State-of-practice methods rely mostly on the practitioner's experience and judgement. Other methods fail to incorporate reasonable driver behaviour models, consider time-varying demand, allow for computational efficiency on large networks, or consider the spatial variation of incidents on a traffic network.

A new CMS location optimization model has been developed that is unique in both model realism and computational efficiency. The model incorporates several components to estimate incident delay, predict driver response, estimate network-wide benefit, and choose those CMS locations that would provide the most benefit. Deterministic queuing methods are used in conjunction with historic incident characteristics to approximate the delay impact of an incident with and without CMS. A discrete choice model is used to predict the rate at which drivers would switch from the incident route to a less congested alternative under CMS information. A network traffic assignment model is then incorporated in an attempt to estimate the resulting traffic induced by incidents. Genetic algorithms are

utilized as an optimization technique to choose a set of CMS that would provide the most benefit.

An extensive computational analysis was performed on both a hypothetical network and a segment of Highway 401 through Toronto. A sensitivity analysis was performed to test the model's response to parameter and data estimation errors. The model was found to be most sensitive to the diversion model parameters. The model produced reasonable results with locations selected upstream of major freeway interchange diversion points. Considering the additional components included in the proposed model, and its ability to consider more location schemes, the proposed model may be considered superior to previous CMS location models.

Acknowledgements

The author would like to thank his supervisor, Professor L. Fu, whose advice, support, assistance, and encouragement have greatly enhanced the quality of this work. Professor S. Li also assisted on this research and his work was greatly appreciated.

The author would also like to thank Dr. Ali Mekky and the Ministry of Transportation of Ontario for providing the Toronto network data used in the case study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 Background

Freeway congestion in urban areas has increased steadily over the past 20 years. For example, from 1982 to 2000 the average annual delay per commuter in major U.S. urban areas increased from 16 hours to 62 hours (Schrank and Lomax, 2002). The total monetary cost for this delay increase was estimated to be $67.5 billion US$ based on 3.6 billion hours of delay and 5.7 billion gallons of excess fuel consumed.

Among the transportation community, it is generally believed that the strategy of meeting demand through new road construction is too cost-prohibitive to be effective and the focus should be shifted to better management of existing facilities. An increasingly popular management method is the use of intelligent transportation systems (ITS), the application of information technologies to transportation to save lives, time, money, energy and the environment. Advanced Traveler Information Systems (ATIS), an integral component of ITS, seek to provide accurate and timely information on traffic, weather and other travel-related conditions to travelers. One of the primary ATIS methods for transportation agencies to disseminate information to motorists is through the use of Changeable Message Signs (CMS), also known as Variable Message Signs (VMS).

CMS can generally be defined to be any electronically controlled message sign displayed either above or beside a roadway for the purpose of providing travel information to motorists. The message a CMS can display is usually programmed from a central traffic control centre and is updated, either manually or through a computer algorithm, to show current roadway conditions and events to drivers. CMS can be divided into one of two types, transportable or permanently mounted. As the name implies, transportable CMS are portable and may be moved based on current requirements. They are usually

set-up temporarily at some locations to inform motorists of scheduled events such as roadway construction, road closures or other types of special events. The use of permanently mounted CMS differs from the transportable type in that their primary use is to respond to unscheduled incidents, such as freeway accidents, with the additional capability of displaying scheduled event information. Additionally, CMS depend on other traffic management technologies such as loop detectors and closed circuit television (CCTV) for monitoring traffic conditions and incident detection algorithms for detecting incident occurrences in real time (Figure 1.1).

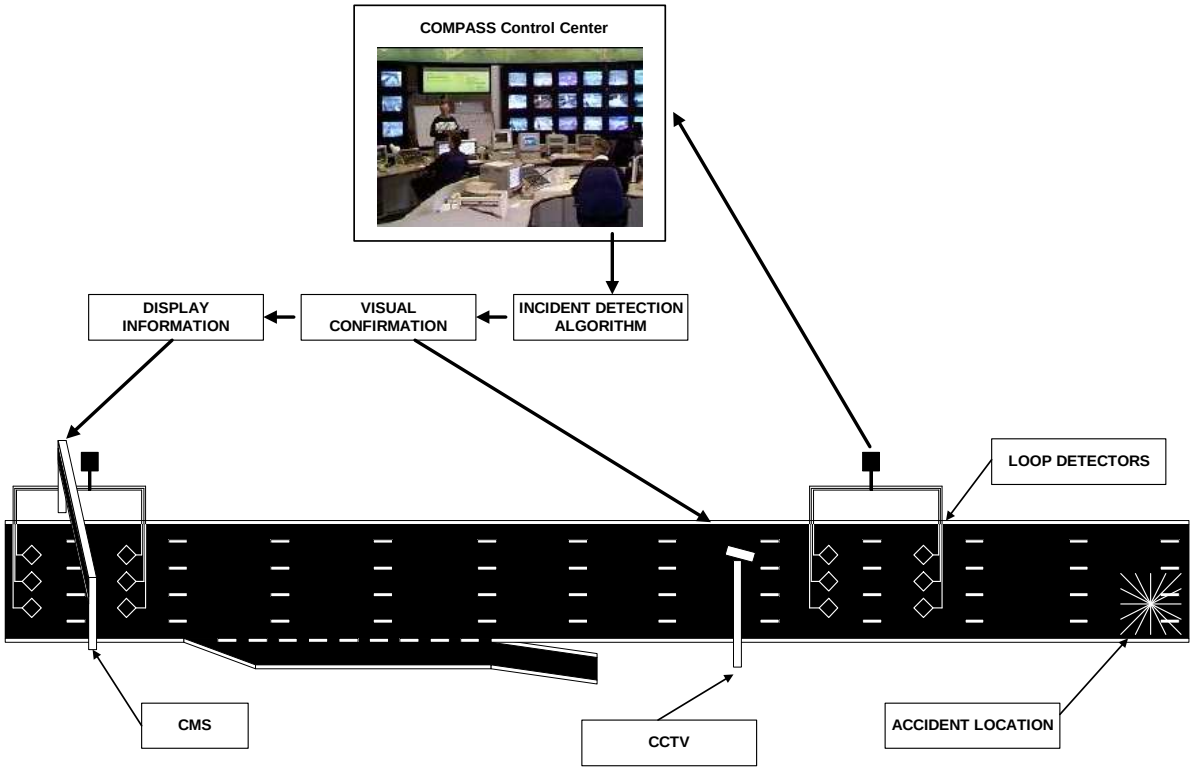


Figure 1.1: CMS Data and Communication for Toronto COMPASS System

The deployment levels in both Canada and the US support the belief that CMS are an

effective tool for information provision. In the US, 38 of the nation's 75 largest metropolitan areas utilize 1,799 CMS as part of their traffic management program (Gordon and Trombley, 2001). This number is expected to increase as many additional metropolitan areas are planning to implement CMS technologies, a trend represented in Figure 1.2. Compared to the US, the deployment level of CMS in Canada has been modest, with only 63 CMS in the Greater Toronto Area, Ottawa, and Vancouver combined (Fu et al., 2003).

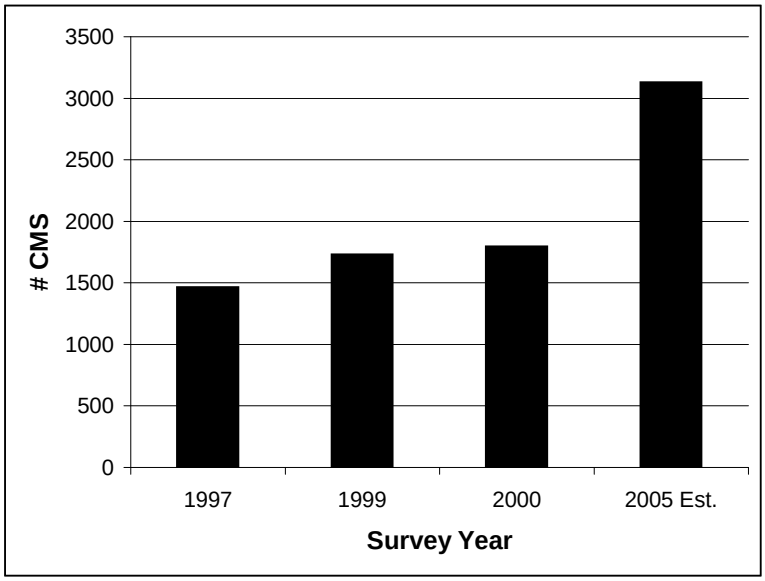


Figure 1.2: US CMS Deployment (Gordon and Trombley, 2001)

The installation of new ITS infrastructure, however, requires significant economic commitments. Furthermore, while transportation planners acknowledge CMS as a potentially effective tool, the cost component associated with location and development would render poor decisions difficult to correct. For instance, a recent survey of ITS technologies revealed the cost of installing a CMS to be in the range of $80,000 to $120,000 US$ exclusive of mounting and communications (Lockheed Martin and Odetics, 1997). An additional cost

of mounting ($25,000 to $125,000 US$) is also a consideration since attachment to existing highway structures is preferred, but not usually possible.

1.2 CMS Benefits in the Context of Freeway Traffic Management

Dudek (1997) performed a state-of-practice survey of CMS applications in 29 US states, one US turnpike authority and one Canadian province (Ontario). The survey indicated that CMS are used primarily for the following nine freeway management applications (percentage of agencies using):

- Incident/Traffic Management (72%),
- General Traffic Information/Warning (62%),
- Diversion Information (59%),
- Construction/Maintenance Support (55%),
- Warnings of Adverse Weather/Road Conditions (51%),
- Special Event Traffic Control (34%),
- Fog Warnings (31%),
- High Occupancy Vehicle/Contraflow Lane information (24%), and
- Reversible Lane Control (17%).

Incident and traffic management show the highest use of CMS (72%), indicative of the widespread belief that incident management has the most potential for system benefit.

This is supported by many studies that conclude incidents to be the cause of a significant portion of congestion-induced delay. As an example, the Schrank and Lomax study (2002) estimates that incidents cause anywhere between 52 and 58 percent of freeway delay. Additionally, it is known that during these incidents traffic equilibrium is disrupted to a point where the travel time through a freeway incident location is much higher than that through an alternative route. So the potential does exist to provide delay reduction for a large group of freeway motorists by diverting them through less congested areas.

The provision of general traffic information, the second most popular usage of CMS at 62%, has not been shown in previous studies to produce any significant reduction in system travel time. For example, a case study by Tarry and Graham (1995) suggested that diversion rates are usually very low when no incident information is presented to motorists. However, despite the common belief of its effect, advising motorists of demand-induced congestion and travel time information on their predetermined routes can reduce drivers' anxiety (Chatterjee et al., 2002).

Diversion information, usually given concurrently with incident information, highly depends on the availability of travel time information on alternative routes. These times, when not available, lead to the reduced provision of diversion information since providing poor quality information can be much worse than providing no information at all (Arnott et al., 1991). 59% of the agencies surveyed use CMS to provide diversion information, the third most popular CMS message type.

CMS are also extensively used for providing fog and other adverse weather warnings, indicated by the 51% of agencies providing information of this type, with the expected benefit of increased safety. Though this benefit has not yet been quantified, it is generally believed that the reduction in traffic speeds observed during poor weather advisories would decrease the risk of accidents. These applications are also somewhat different in that they

are proactive to prevent problems before they occur compared to reactive strategies that respond to problems.

The remaining application types are either of the planned or special type. In the case of a planned capacity reduction, such as construction or special events, CMS is a redundant source of information to support portable signs and other information sources. Specialized CMS information, including reversible and high-occupancy vehicle lanes, are not applicable to most freeway segments and will not be elaborated here.

In summary, while CMS have been deployed to provide a wide range of information, the benefits of most information types have yet to be substantiated. Among the few applications that have proven beneficial are incident management and provision of diversion information. This suggests that any decision-making process relating to CMS installation should first focus on these two areas with other applications as secondary objectives, at least until reliable studies and models are developed.

1.3 The Research Problem

The discussion to this point has mainly focused on the studies of existing CMS deployments. The popularity, cost component, and application types of CMS have all been presented in addition to the role of CMS in the context of ITS freeway traffic management. Recent US and Canadian deployment studies have indicated that the number of CMS is expected to increase in the foreseeable future. However, an important issue that has not been fully addressed is the planning of these future CMS installation locations.

The planning of future locations of CMS represents a significant knowledge gap in the transportation systems planning process. Specifically, given a network and application environment, how many CMS should be installed and where should these installations be located? Existing guidelines merely indicate that CMS should be located upstream of

major diversion points, bottleneck locations, and frequent accident locations (Abbas and McCoy, 1999). There is no complete methodical approach to help senior planners decide how many CMS and where to install these CMS. Therefore, a need exists to develop a model that determines the optimal number of CMS to install and locates the CMS based on some system-wide benefit measure.

Currently, CMS are allocated in an ad-hoc method such as installing CMS on high-volume freeway segments close to points of diversion. The resulting decision may not be the optimal solution, however, resulting in poor and expensive decisions. An alternate approach of installing CMS at all freeway diversion points is also a poor policy for two reasons. First, the cost would be prohibitive considering the initial construction, maintenance, and energy requirements of these signs. The second reason relates to the diminishing effectiveness of placing too many additional signs. If the entire freeway network were instrumented with CMS it is likely that drivers would shift their attention away from the signs over time, since incidents are relatively rare. Urgent messages relating to recent events would be missed by motorists and the benefit of providing information could be lost.

1.4 Objectives and Scope

The main goal of this research is to develop a planning model for locating CMS in an optimal way that maximizes the benefit of the information delivered through CMS to users. The benefit would be a result of drivers diverting from an incident-induced highly congested freeway segment to a less congested diversion route. The specific objectives include:

1. Develop a method to estimate total vehicle delay both without and with CMS information.

2. Synthesize current driver behaviour models so that, for a specified incident condition, the aggregate diversion response can be estimated.

3. Develop an efficient optimization method that can be used as a planning tool for identifying the optimal locations of a given set of CMS.

4. Examine the sensitivity of the proposed model to stochastic variations and parameter estimation errors.

It should be noted that while the scope of this research is limited to the consideration of incident-induced delay, other CMS benefits can be incorporated into the model once reliable information and models become available for quantifying the effect of information provision.

1.5 Structure of this Document

The various aspects of the research are presented in seven chapters. Presented in Chapter 2 is an overview of past research relevant to this study. In Chapter 3, delay estimation using deterministic queuing is discussed. The network benefit model is described in Chapter 4 and the optimization procedure for maximizing that benefit is presented in Chapter 5. A case study of the Toronto freeway system is analyzed and sensitivity of the model to various input and parameter estimations is discussed in Chapter 6. Finally, relevant conclusions and future work are presented in Chapter 7.

CHAPTER 2
LITERATURE REVIEW

Transportation researchers have only recently proposed models to locate CMS to maximize the expected transportation system benefit. Most of these models synthesize several well-established component methods. Deterministic queuing theory, a method utilized in many scientific and engineering disciplines, is an excellent tool for input-output analysis and delay estimation resulting from capacity restrictions of a system. Utility theory, and more specifically choice models, are commonly used to estimate both aggregate and disaggregate choice behaviour. Lastly, genetic algorithms as an optimization tool are robust in their ability to handle computationally difficult and implicitly structured problems including location optimization. This section reviews each of these techniques followed by a critical examination of directly related research.

2.1 CMS Location Models

A thorough literature review revealed only two directly related models to locate CMS. Abbas and McCoy (1999) were the first in literature to study the problem of optimizing CMS locations in a road network. Their location optimization objective (Equation 2.1) was to maximize the potential reduction in vehicle delay due to traffic diversion to alternative routes in response to incident information provided by CMS. A simple deterministic queuing model was used to estimate delays, and the associated delay savings (DS), with and without CMS on a linear freeway network (Figure 2.1) considering a constant diversion rate. However, it was not clear how issues such as over-saturated conditions, incident rates on individual links, and dependency of diversion rate on potential savings were handled in their model.

$$B_i = \sum_j \sum_k \frac{\nu_{ijk}}{\sum_i \nu_{ijk}} DS_j \tag{2.1}$$

where

B_i = benefit of deploying a CMS at diversion point i (vehicle-hours);

ν_{ijk} = number of vehicle trips from diversion point i through downstream freeway section j during time period k;

k = time period ($k = 1$: peak period, $k = 2$: off-peak period);

DS_j = total estimated delay savings due to traffic diversion during incidents on freeway section j (vehicle-hours).

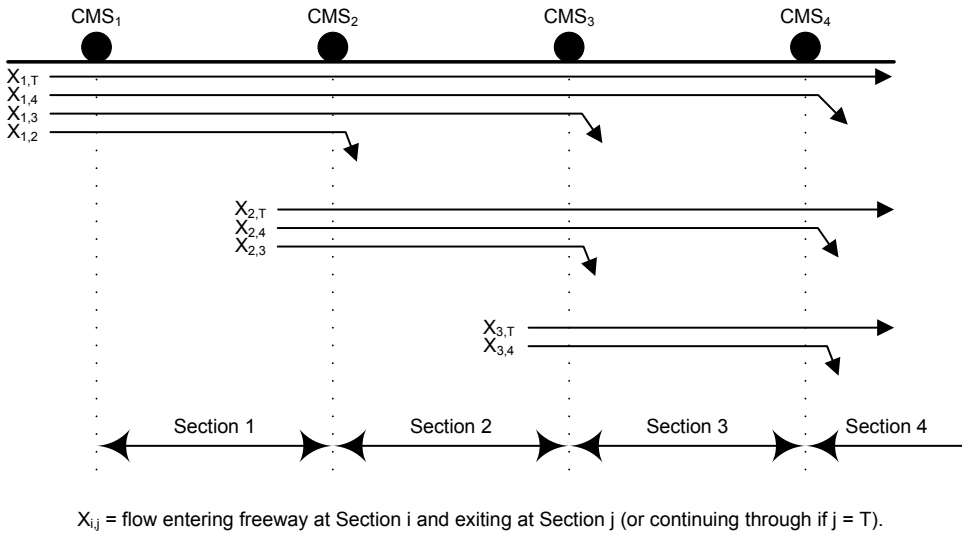


Figure 2.1: Freeway Origin-Destination Distribution in Abbas and McCoy (1999)

Another relevant study was initiated by Chiu et al. (2001) who proposed a bi-level stochastic integer programming model for the CMS location problem. The location optimization problem was realized at the upper level, seeking to maximize the total user benefit

of real-time information from CMS. The users' responses (route choices) to incident conditions and information were represented at the lower level as a user optimal dynamic traffic assignment problem. The expected total user benefit corresponding to a given location solution was calculated based on a sample of benefits, each of which was estimated by generating a random incident on a network link and solving the resulting dynamic traffic assignment problem. An overview of the Chiu et al. (2001) tabu-search location algorithm is as follows:

1. Determine the solution space by identifying all candidate CMS locations. A candidate CMS location is one on the freeway link with an off-ramp as one of its outbound links.

2. Find a new candidate solution by dropping a location from the current solution and add one based on probability of incident occurring in the activation region of the CMS. An activation region (geographical area within which incident information will be displayed for a specific CMS) is considered so that information regarding distant incidents is not provided to drivers. The algorithm begins using the highest ranked locations from the initial solution. This approach facilitates the search to converge to a plausible solution. The new candidate solution is adopted if it is not a tabu move (previously evaluated solution); otherwise, regenerate a new candidate solution.

3. Generate a random incident realization by drawing a random number from the uniform distribution, $U(0, 1)$, and map it to the corresponding cumulative distribution, which is constructed from the probability mass function of the ratio of link lane-miles and the total link lane-miles.

4. Activate CMS if the randomly generated incident occurs inside the CMS activation region.

5. Perform the time-dependent user equilibrium (UE) procedure and divert those vehicles that reach the CMS to their destinations via UE paths.

6. Simulate until the end of the horizon.

7. Repeat from step 3 for K number of incident realizations.

8. Update current solution and tabu list.

9. Repeat from step 2 until the specified maximum iteration is reached.

The Chiu et al. model suffers from several limitations. First, as acknowledged by the authors, the whole process is extremely computationally intensive because of the need to evaluate a large number of candidate location plans, consider sufficient number of incident realizations for each location solution, and perform a simulation-based dynamic traffic assignment procedure for each incident realization. Second, their location benefit model was based on a route choice assumption that all users have perfect knowledge and real-time information on the network and incident conditions and possess the ability to anticipate other users' choice of routes and choose their optimal routes accordingly. Third, they do not consider a variation of incident rates across the freeway network. Every lane-km of freeway is assumed to have the same incident rate, reducing the robustness and validity of the model. Additionally, the use of randomly generated incidents most likely would introduce sampling errors since insufficient incidents would be realized to match the distribution proposed. Finally, it is unclear if it is practical or necessary to apply such a complex model, seemingly designed for operational management and control purposes, for solving the CMS location problem, which is essentially a planning problem.

2.2 Deterministic Queuing Models

As discussed in the previous section, one of the key elements of a CMS location optimization system is how to estimate delay. Deterministic queuing theory is a well-known graphical technique to estimate delay in systems with input-output constraints, such as a highway section with an incident.

The general system can be defined in terms of an arrival stream and a server service rate (Figure 2.2). The arrival rate, usually denoted by λ, represents the number of arrivals over a given time period. Similarly, the number of departures during a period in time is called the service rate, generally denoted by μ.

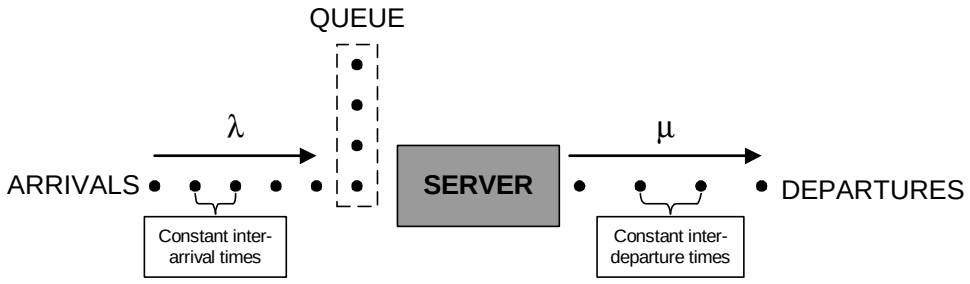


Figure 2.2: Deterministic Queuing System

The analysis of deterministic queuing systems is commonly performed using a graphical technique that requires the calculation of cumulative arrivals and departures from an initial reference time. The most accurate form of this technique considers all arrival and departure events as discrete (Figure 2.3). At each instance of time the cumulative curves represent a summation of all the events from the initial reference time to the current instance of time. To achieve this result, the arrival curve is incremented whenever an arrival event occurs. The departure curve is increased in a similar manner. It is notable that, by definition, the

cumulative number of departures cannot exceed the cumulative number of arrivals at any instance of time.

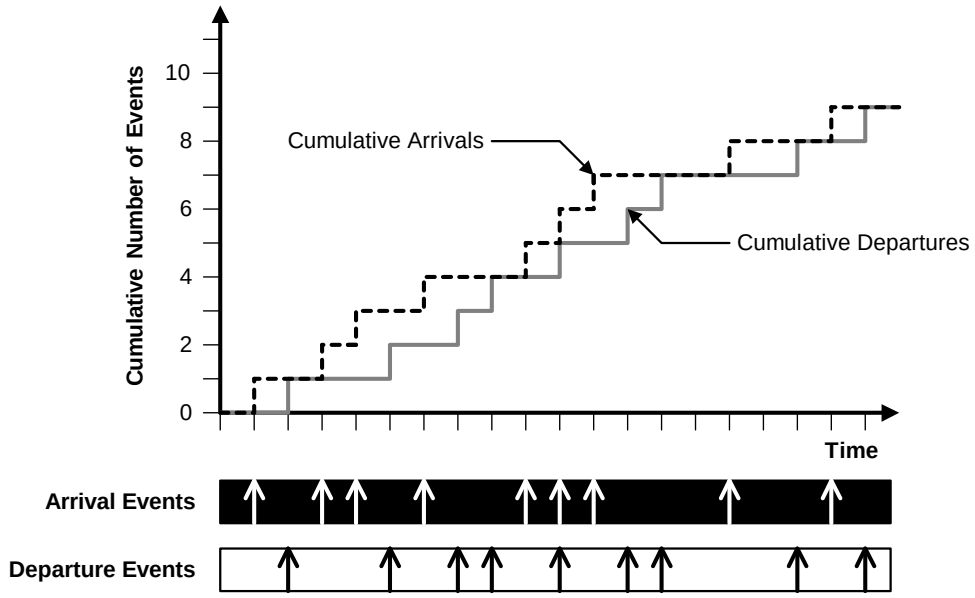


Figure 2.3: Discrete Queuing System

Considering arrivals and departures in a discrete form, although realistic, is generally inconvenient for applying mathematical operations such as derivatives and integrals. Besides, it is often difficult to accurately measure individual arrivals and departures in a traffic stream. Furthermore, queuing analysis is generally performed for future traffic analysis or systems without complete data. Therefore, a discrete representation of individual vehicle arrivals is not necessary and a continuous representation scheme is commonly adopted.

Figure 2.4 shows a continuous representation of a queuing process, assuming continuous arrival and departure rates. Several quantities may be derived from the interaction between the arrival and departure curves. As noted on the diagram, the area between the cumulative arrival and departure curves represents the system delay. The queue length at time t, in

vehicles, may be determined from the vertical distance between the arrival and departure curves at time t. Similarly, the delay experienced for a vehicle joining the queue at time t is represented by the horizontal distance between the arrival curve at time t and the departure curve. For illustrative purposes a capacity reduction caused by an incident is shown in Figure 2.4. For a more in-depth background specifically relating to traffic queuing refer to May (1990).

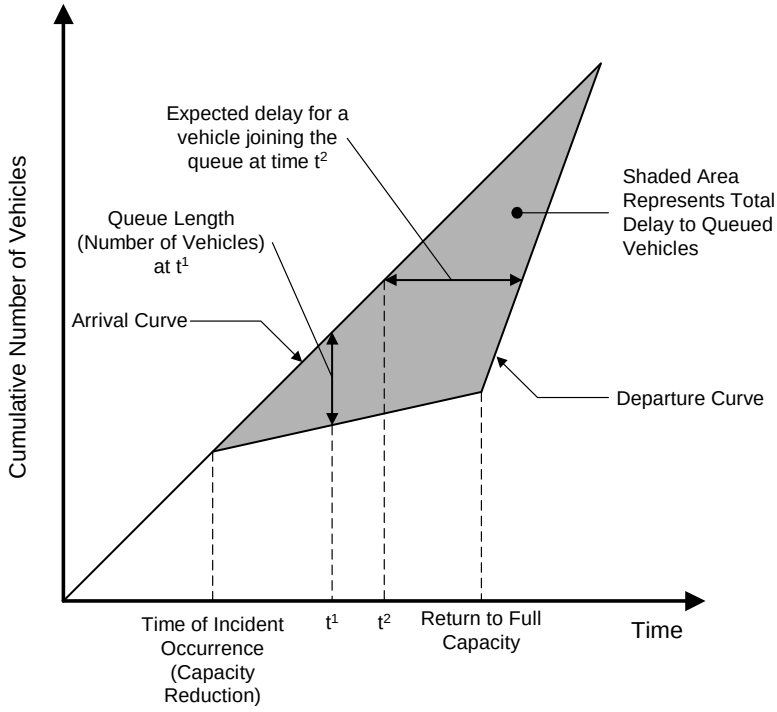


Figure 2.4: Deterministic Queuing Graphical Analysis

For traffic analysis studies, however, there are assumptions related to vehicle queuing that must be considered. Most importantly, it is assumed that any vehicle arriving to an over-saturated section, and not able to immediately depart, is stacked on a vertical pile at the location of the bottleneck. The major implication of this assumption lies in the

definition of an arrival, which now corresponds to the time at which a vehicle would have crossed the bottleneck had there not been a queue. This definition does not correspond with the actual arrival of a vehicle joining the end of a queue.

The shock-wave analysis technique (or simple hydrodynamic theory of traffic flow) first developed by Lighthill and Whitham (1955), has the mechanism to overcome the physical queue estimation limits imposed by deterministic queuing. The method utilizes macroscopic speed-flow-density relationships to identify flow states in a space-time graph. A shock-wave represents the transition between two flow states. Figure 2.5 illustrates the shock-wave technique for a freeway incident similar to the deterministic queuing example (Figure 2.4).

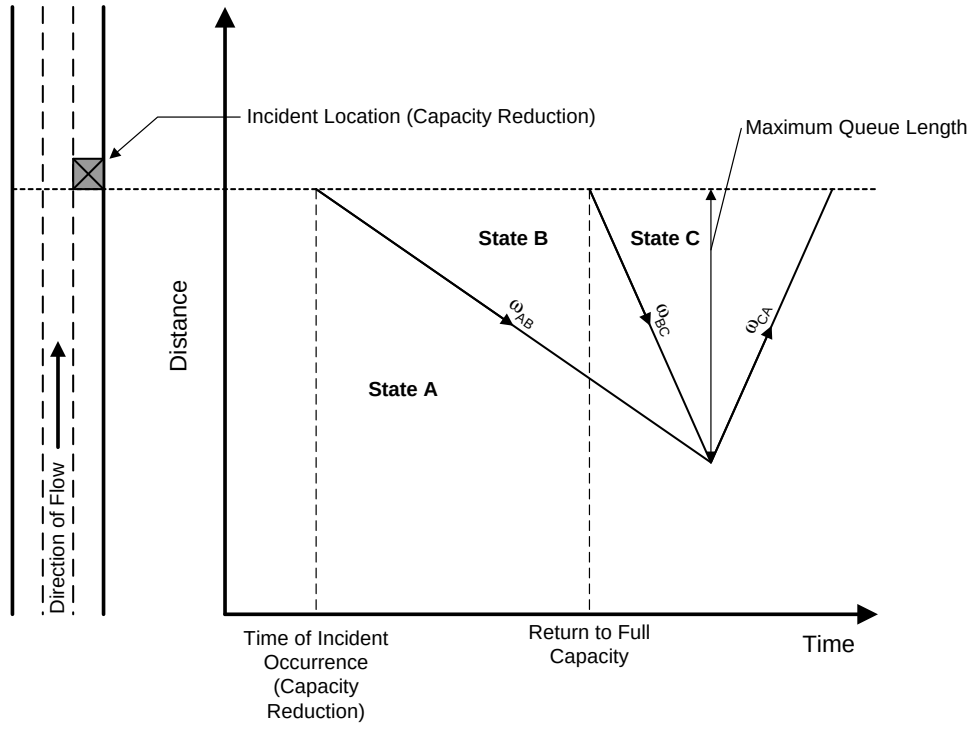


Figure 2.5: Shock-wave Graphical Analysis

Three flow states are introduced in the shock-wave example. State A is the original flow state that extends to areas unaffected by the incident. States B and C are queue flow states with State B as the queue forming state (flow equal to reduced capacity) and State C as the queue recovery state (flow equal to full capacity). Three shock-waves lie at the boundaries of these flow states: ω_{AB}, ω_{BC}, and ω_{CA}. The relative traffic flow (q) and density (k) for adjacent flow states are required to compute shock-wave speed (Equation 2.2).

$$\omega_{AB} = \frac{\Delta q}{\Delta k} \tag{2.2}$$

The computation of total vehicle travel time ($\sum TT$) utilizing traffic density (k), flow state areas (A), and the shock-wave space-time diagram is illustrated in Equation 2.3. The total vehicle delay (Equation 2.4) is derived by considering the total vehicle travel time with and without an incident occurrence.

$$\sum TT = k_A A_A + k_B A_B + \ldots + k_N A_N \tag{2.3}$$

$$Delay = \sum TT_{IC} - \sum TT_{NIC} \tag{2.4}$$

where

k_N = traffic density in flow state N;

A_N = area of flow state N;

TT = total travel time for area under consideration;

IC = incident conditions;

NIC = non-incident conditions.

Since the development of shock-wave analysis, there have been several arguments indicating that deterministic queuing and shock-wave analysis do not provide consistent results (e.g. Michalopoulos and Pisharody, 1981). These arguments are mostly a result of the difference between the physical queues determined by shock-wave analysis and the vertical

pile queues estimated by deterministic analysis. However, there are several notable references (Makigami and Newell, 1971; Daganzo, 1983; Lawson et al., 1997; Erera et al., 1998) proving the consistency of the two methods. These papers all conclude that deterministic queuing and shock-wave analysis provide the same estimate for vehicle delay. However, time-dependent queue length estimates are not consistent due to different interpretations for vehicle arrivals. Therefore, the consensus among researchers is that while delay estimation using deterministic queuing is valid, queue length interpretations have no physical meaning.

2.3 Driver Response to CMS Information

The second main component of a CMS location optimization system is to simulate drivers' route choice behaviour under CMS information. Several studies have been undertaken in an attempt to understand the reaction of drivers to information and, in this specific case, CMS information. The ultimate goal of these lines of research is to be able to predict the percentage of drivers who would divert to alternative routes, or driver diversion rates and/or route choices based upon the information specified.

Albrecht et al. (1978) performed the first of these studies relating driver behaviour to CMS information. The result of a traffic count study indicates that anywhere between 5% and 80% of drivers will voluntarily divert. This range is obviously too wide to be useful, but it does represent an attempt on the part of researchers to understand driver behaviour in the presence of CMS information. More detailed driver behaviour studies can generally be classified into: stated preference (SP) surveys, revealed preference (RP) surveys, route choice simulators and driving simulators.

A SP survey presents a variety of hypothetical situations to drivers, usually via a paper questionnaire, in an attempt to estimate drivers' preference within a set of transportation

options. The utility of each of these options (Equation 2.5) may be estimated based on the drivers' response. A variety of statistical techniques may be used to estimate the relative utility weights, with the most popular method in the transportation sector being the maximum likelihood logistic regression for multinomial logit (MNL) models.

$$U_k = \alpha_1 x_{k,1} + \alpha_2 x_{k,2} \ldots \alpha_n x_{k,n} \tag{2.5}$$

where

U_k	=	utility towards option k;
$x_{k,1}$ to $x_{k,n}$	=	values of factors 1 to n associated with option k;
α_1 to α_n	=	utility weights for factors 1 to n.

A simple example of utilizing a stated preference questionnaire to evaluate a particular bus service is provided in Kroes and Sheldon (1988). In the example three variables (journey time, availability of a seat, and fare) each with two levels (30|40 minutes, seats|no seats available, and 20|30 pence) are evaluated. Three of the sample alternative options presented to passengers are described:

Alternative 1	*Alternative 4*	*Alternative 7*
30 minute journey time	30 minute journey time	40 minute journey time
no seats available	seats available	no seats available
20 pence fare	30 pence fare	20 pence fare

To completely isolate factors all combinations of all levels of each factor need to be incorporated into the experimental design. This is generally referred to as "full factorial design" and can only be applied if there are few factors and levels under consideration. Conversely, "fractional factorial design" utilizes a subset of the full factorial design to capture the direct effects but not the interaction effects between factors. The difference

between the two design philosophies is illustrated, for the bus service example of Kroes and Sheldon (1988), in Table 2.1.

Table 2.1: Bus Service Example of Experimental Designs

	Factor 1	*Factor 2*	*Factor 3*
(i) *Full Factorial Design*			
Alternative 1	1	1	1
Alternative 2	1	1	2
Alternative 3	1	2	1
Alternative 4	1	2	2
Alternative 5	2	1	1
Alternative 6	2	1	2
Alternative 7	2	2	1
Alternative 8	2	2	2
(ii) *Fractional Factorial Design*			
Alternative 1	1	1	1
Alternative 4	1	2	2
Alternative 6	2	1	2
Alternative 7	2	2	1

The low cost of SP surveys has popularized the method as a way to collect data on a wide variety of driving situations, including en-route information. However, a recognized shortcoming of the SP method lies in the fact that drivers are not committed to behave in accordance with their stated preference responses (Bates, 1988). Fortunately, many studies have found that SP surveys generally correspond well with RP surveys (e.g. Louviere et al., 1980).

RP surveys consist of driver interviews conducted downstream of a specific investigation area, or more generally, an after-the-fact investigation. In most respects RP surveys are identical to SP surveys, with the notable exception being that RP surveys record past or observed behaviour and stated preference surveys measure projected behaviour under

hypothetical situations. RP surveys generally provide slightly better "real-world" correlation than SP surveys, but they suffer from a limiting range of situations that can be examined and the problem of unwanted factors being introduced into the analysis section. Additionally, RP methods cannot be used to directly evaluate situations that do not yet exist (Kroes and Sheldon, 1988).

Several stated and revealed preference studies (e.g. Abdel-Aty, 2000; Bonsall et al., 1995; Hato et al., 1995, 1999; Zhao et al., 1995) have identified factors that influence driver route choice behaviour in the presence of information, but not specifically CMS information. These studies show inconsistencies for several driver factors (age, education, income, etc.). However, they generally agree that the diversion rate is most highly influenced by drivers' network knowledge and the implied or actual travel time benefit of diversion. These studies will not be explicitly detailed, however it is notable that many of the important factors (Table 2.2) identified in these studies were used in the CMS SP studies of Wardman et al. (1998), Peeta et al. (2000), and Chatterjee et al. (2002) to reduce the number of factors considered.

Table 2.2: Transportation System Factors Influencing Individual Route Choice

Factor	*Factor*
overall expected journey time	safety and security hazards
delays	unfamiliar routes
congestion	accident information
signposted routes	delay information
tolls	congestion information

Wardman et al. (1998) conducted a route preference study including hypothetical scenarios at a junction between the main travel route to Manchester, UK (M62 motorway) and alternate routes (via M6 motorway). A highly visible CMS is located a short distance

upstream of the M62/M6 junction. Survey respondents were provided with a photograph representing a "through-the-windshield" view of the junction and CMS panel information (Table 2.3). Considering the visual and CMS information, it was expected that survey respondents had the following information:

- local traffic conditions on the M62 ahead (queuing or clear)
- local traffic conditions on the off-ramp to the M6 (queuing or clear)
- expected delays ahead on specified routes (notified via the CMS panel)
- cause of these delays (notified via the CMS panel)

Table 2.3: CMS Messages used in Wardman et al. (1998)

Message	*Message*	*Message*
All clear	20 Mins delay [accident]	Delays likely [congestion]
(No message)	30 Mins delay [accident]	Long delays [congestion]
5 Mins delay	Delays likely [accident]	5 Mins delay [roadworks]
10 Mins delay	Long delays [accident]	10 Mins delay [roadworks]
20 Mins delay	5 Mins delay [congestion]	20 Mins delay [roadworks]
30 Mins delay	10 Mins delay [congestion]	30 Mins delay [roadworks]
Delays likely	20 Mins delay [congestion]	Delays likely [roadworks]
Long delays	30 Mins delay [congestion]	Long delays [roadworks]
10 Mins delay [accident]		

Wardman et al. (1998) obtained 289 responses representing a total of 2304 choice observations. Several utility function forms were tested and the following power model (Equation 2.6) was found to provide the best fit.

$$U_k = \alpha_1 x_{k,1}^{\lambda} + \alpha_2 x_{k,2}^{\lambda} + \ldots + \alpha_n x_{k,n}^{\lambda} \tag{2.6}$$

This power model differs slightly from the more common linear utility function form (Equation 2.5). Results for the utility weights and factor powers are provided in Table 2.4. Validation of the model was based largely on the observed market share of each of the routes and value of time comparisons to previous studies. Wardman et al. (1998) found that their resulting range of $delay/free\ flow$ travel time ratios of 1.30 to 1.70 generally corresponded to the values of 1.43, 1.39, and 1.70 obtained in Wardman (1991), Oscar Faber TPA (1992), and Hensher (1992) respectively.

Table 2.4: Wardman et al. (1998) Preferred Utility Model

	Utility Weight (α)	Factor Power(λ)
Delays (Mins) Caused by Roadworks	-0.041	1.3
Delays (Mins) Caused by Congestion	-0.042	1.3
Delays (Mins) Caused by Accidents	-0.048	1.3
Delays (Mins) Unspecified Cause	-0.036	1.3
Likely Delays Message (Roadworks)	-0.595	1.0
Likely Delays Message (Congestion)	-1.867	1.0
Likely Delays Message (Accidents)	-2.100	1.0
Likely Delays Message (Unspecified Cause)	-0.835	1.0
Long Delays Message (Roadworks)	-2.732	1.0
Long Delays Message (Congestion)	-2.450	1.0
Long Delays Message (Accidents)	-3.337	1.0
Long Delays Message (Unspecified Cause)	-2.623	1.0
Alternate Route Clear Message	0.815	1.0
Alternate Route Visible Queuing	-0.043	1.0
Route Travel Time	-0.068	1.0

The major findings of Wardman et al. (1998) can be summarized as follows:

- additional delays are valued more highly than normally expected delays, at a ratio between 1.30 and 1.70 depending on the stated cause of the delay
- delays attributed to accidents had the biggest impact on route choice

- visible queues were found to have a significant effect on driver route choice
- those who had never used the alternate routes were less likely to be persuaded by the CMS panel advice

Peeta et al. (2000) conducted a similar stated preference survey in the Borman Expressway region of northwestern Indiana. The state department of transportation was in the process of developing an advanced traffic management system (ATMS) and, as such, there were no existing CMS within the study area. The survey presented hypothetical CMS messages to 248 respondents and collected responses on a five-point Likert scale (1 to 5), where 1 represented a low willingness to divert and 5 represented a high willingness to divert. Based on the responses, an aggregate effect of CMS message content was developed (Table 2.5). This was followed by the development of a logit model, similar to the Wardman et al. (1998) study, incorporating both the responses and socio-economic characteristics of drivers (Table 2.6).

Several conclusions were stated by Peeta et al. (2000), which can be summarized as follows:

- female and older drivers are, on average, more risk averse (less willing to divert) than males and younger drivers
- well-educated individuals are more likely to comply with the CMS messages than their less-educated counterparts under similar conditions
- incident location is significant in the diversion decision, however, it was not incorporated into this study
- truck drivers exhibit more resistance to diversion than other drivers

Table 2.5: Effect of CMS Content in Peeta et al. (2000)

CMS Message Type	Message Content	Relative Willingness to Divert				
		1 %	2 %	3 %	4 %	5 %
1	Occurrence of accident only	13.7	33.9	26.6	13.3	12.5
2	Location of the accident only	20.2	33.1	22.6	11.3	12.9
3	Expected delay only	9.3	12.9	39.5	23.8	14.5
4	The best detour strategy only	7.7	18.5	30.2	25.0	18.5
5	Location of the accident and the best detour strategy	2.0	4.0	22.6	35.1	36.3
6	Location of the accident and the expected delay	0.8	0.8	19.8	38.3	40.3
7	Expected delay and the best detour strategy	2.0	2.0	13.7	33.5	48.8
8	Location of the accident, expected delay, and the best detour strategy	1.2	2.0	5.6	19.8	71.4

Drivers attitude towards CMS information in London was studied by Chatterjee et al. (2002) through stated preference data. The mailed questionnaires provided a hypothetical incident situation, CMS message, and a map with which the respondents could indicate their diversion route if they would choose to divert. An example of a stated intention question used in this study is shown in Figure 2.6.

Chatterjee et al. (2002) developed four logit models, the two most comprehensive models (highest Nagelkerke R^2) are presented in Table 2.7. Further analysis revealed that the two most significant independent variables in these models are distance from the driver's origin to the Archway (Euclidean distance) and the indicator variable for a non-London origin. Other relevant factors revealed from the modeling process include the incident severity (magnitude of delay), if the incident occurs on the normal route, and if the driver is able to return to the normal route after diverting. The implication of the last point is that the

Table 2.6: Logit Model for Driver Response in Peeta et al. (2000)

Variable	Level 0	Level 1	Utility Weight (α)
Alternative specific constant			-1.897
Sex	female	male	0.433
Age	$<$ 40 years	$\geq$ 40 years	-0.458
Education	$\geq$ college	$\leq$ some college	-0.308
Regular driver in region	no	yes	0.207
Trust in information	otherwise	high	0.666
CMS message type 2	5 levels, see Table 2.5.		-0.090
CMS message type 3	$\vdots$		0.611
CMS message type 4	$\vdots$		0.842
CMS message type 5	$\vdots$		2.083
CMS message type 6	$\vdots$		2.490
CMS message type 7	$\vdots$		2.731
CMS message type 8	5 levels, see Table 2.5.		3.548

If, as you approached Archway junction, you had seen a CMS saying:

ISLINGTON ACCIDENT LONG DELAYS

Would you divert from the route you have marked on the map?

Yes, I would divert ☐

Yes, but not until encountering further problems ☐

No, I would not divert ☐

Figure 2.6: Example Stated Intention Question

probability of diversion is increased if the driver can return to their normal route.

Online evaluation of existing systems, either through traffic count or revealed preference

Table 2.7: Logistic Regression Models in Chatterjee et al. (2002)

Variable	Utility Weight (α)
CMS Messages (full data set)	
Constant	-2.24
Distance between Archway and problem (Euclidean)	-0.218
Problem on normal route	1.37
Non-London origin	-0.781
Problem cause (w.r.t. congestion)	
Accident or roadworks	0.676
Demonstration	1.38
Problem severity (w.r.t. delays/15 min delay)	
30 min delay/long delays/avoid area	0.686
CMS Messages (reduced data set)	
Constant	-1.16
Distance between Archway and problem (Euclidean)	-0.412
Non-London origin	-0.662
Problem cause (w.r.t. congestion)	
Accident or Roadworks	0.422
Demonstration	1.40
Problem severity (w.r.t. delays/15 min delay)	
30 min delay/long delays/avoid area	0.717
Distance from destination where alternative route merges back with normal route	0.195

data, has provided estimations of driver diversion rates. Table 2.8 provides a sample of some of these studies. There are a few other studies of this type, but the problem with this line of research is that results from one study site are not transferable to another site. So the only real value of these statistics is in providing bounds on reasonable diversion rates.

Table 2.8: Online Evaluations of Effects of CMS Systems on Diversion Rate

Study	*Item*	*Details*
Swann et al. (1995)	Location	A64/A1 Interchange, North Yorkshire, U.K.
	Method	Revealed preference questionnaires
	Results	16% diversion
Yim and Ygnace (1996)	Location	SIRUS System, Paris, France
	Method	Statistical and graphical analysis of loop detector data (ramps)
	Results [1]	Negligible (0.5 km queue)
		7% below capacity (1.0 km queue)
		10% below capacity (2.0 km queue)
		15% below capacity (3.0 km queue)
		30% below capacity (4.0 km queue)
Kraan et al. (1999)	Location	RIA System, Amsterdam, Netherlands
	Method	Statistical regression analysis of loop detector data
	Results [2]	0.8% to 1.6% diversion per additional km of queue
Chatterjee et al. (2002)	Location	London, U.K.
	Method	Analysis of SCOOT UTC detector data
	Results [3]	1^{st} experiment: 3.0% diversion rate
		2^{nd} experiment: no significant diversion
Levinson and Huo (2003)	Location	Minnesota, U.S.A.
	Method	Statistical analysis of loop detector data
	Results	0.13% to 0.15% diversion

[1]Basis of results is the degree to which the on-ramp to a congested freeway segment (A86) is under capacity, i.e. drivers divert to less congested D45 route. CMS messages indicate the length of queue. [2]Kraan et al. (1999) note a large proportion of traffic entering system has no route choice due to location of predetermined destination. [3]Experiments were performed during scheduled roadworks.

2.4 GA and its Application to Facility Location Problem

The CMS location planning problem belongs to the general class of facility location problems which have been studied extensively over the past forty years since the early work of Cooper (1963) and Hakimi (1964). The most general form of the problem can be simply stated as follows. Given a set of locations, $L = \{l_1, l_2, \ldots, l_n\}$, an objective function $f(L)$, and possibly a constraint on how many and where such facilities can be located, find the subset of L denoted by L^* such that $f(L^*)$ is maximized (or minimized). Most variants of the location problem are combinatory in nature defying optimal solution approaches. Because of their discrete nature and computational intractability, location problems have been found to be a natural fit for a Genetic Algorithm (GA) based solution approach. Background on GA are provided in Appendix A and Henderson and Fu (2004) with algorithm details shown in Section 5.3.

Hosage and Goodchild (1986) were the first to apply genetic algorithms to the location-allocation (p-median) problem, a well-studied form of the location problem. The objective is to minimize the transportation cost (travel time or distance) of allocating a set of demands to the nearest facility subject to a limit on the number of facilities (Equation 2.7).

$$\min Z = \sum_i \sum_j d_{ij} r_j \lambda_{ij} \tag{2.7}$$

where

d_{ij} = distance from node i to node j;

r_j = demand at the nodes, $j = 1 \ldots n$;

λ_{ij} = a binary variable such that node j is assigned to a facility at node i when $\lambda_{ij} = 1$, else $\lambda_{ij} = 0$.

A representation scheme was developed whereby each gene in the chromosome string is

used to represent a facility location, i.e. if the bit = 1 then the facility is chosen and not if the bit = 0. Though this coding scheme would allow illegal solutions (e.g., total number of facilities allocated exceeds the maximum number of facilities to be allocated), the authors contend that other more efficient representations may not permit all possible combinations. To partially deal with illegal solutions, a penalty function was introduced that multiplies the objective by $(\textit{number of facilities over maximum + 1})^2$. The reasoning behind this penalty function was not explained or justified other than the intuitive appeal that the use of this function lowers the likelihood of over allocation. They utilized a variant on the simple genetic algorithm with roulette wheel selection and non-overlapping generations. An inversion operator was used in lieu of the standard crossover techniques to reduce the number of infeasible solutions. Also, 50% of the chromosomes were chosen for mutation, which, as a result of the bit inversion, would most likely lead to an incorrect number of facilities. This problem is recognized by the authors and only noted that "considerable fine-tuning" of the genetic algorithm application should be undertaken. The authors note a 69% $\rightarrow$ 89% rate of finding the optimal solution and state that the algorithm failed in instances where the local optimum, in terms of the objective function, is very similar to the global optimum. However, there are many other good heuristic algorithms to solve this type of problem that were not used as a basis of comparison and consideration of these algorithms should have been made.

Jaramillo et al. (2002) studied the application of genetic algorithms to five classes of location problems and was the first to compare the performance of the GA-based heuristics against other traditional heuristics for the location problems. The same representation scheme as Hosage and Goodchild was used and a binary tournament selection method was chosen due to its effectiveness on problems in a similar class. A fitness-based uniform crossover operator based on earlier work by Beasley and Chu (1996) was modified to

include forced mutation when parents and offspring are identical. The fitness-based uniform crossover operator only differs from the uniform operator in that the selection of genes from either parent is proportional to their fitness value and not a completely random selection. Note also that in this implementation crossover and mutation are mutually exclusive events and as the algorithm converges the crossover rate will decrease and mutation rate will increase, a scheme of variable crossover and mutation rates. A population size equal to the chromosome length (which was set to the number of facilities to be allocated) was used and the algorithm was terminated when no improvement in the objective function value is experienced. Based on comparisons with well-known problem-specific heuristics, such as in Beasley (1993), the authors conclude that genetic algorithms produce better solutions, or at least not worse solutions, but the algorithms require much more computational effort.

CHAPTER 3
INCIDENT DELAY ESTIMATION

One of the main challenges in optimally locating CMS is how to estimate the benefit of a given set of CMS locations. In this research, we assumed that the benefit is expressed as expected total reduction in user delay under incident conditions due to diversion in response to congestion and route guidance information displayed on CMS. The expected delay is modeled as a function of many factors including network conditions, peak and off-peak demands, availability of alternative routes, route guidance information, compliance rates, etc. This chapter details the impact of incidents resulting in vehicle delay. First, the delay caused by a single incident on a single link is developed. Then, the expected annual incident induced delay on a single link is determined based on a probabilistic incident rate. Lastly, the network-wide expected annual incident delay is outlined.

3.1 Deterministic Queuing Justification

The major drawback for deterministic queuing, as outlined in the literature review, is the assumption that vehicles are stacked in a vertical pile. However, this is often overlooked considering that the method produces good estimates for delay and is relatively simple to implement. Alternatively, shock-wave analysis or stochastic queuing could be considered, but these methods are more restrictive.

Shock-wave analysis (Lighthill and Whitham, 1955) is able to estimate physical queue lengths and locations more accurately as compared to deterministic queuing. The determination of physical queue lengths however raises some issues. Traffic density (veh/km or veh/lane/km), required to estimate shock-waves, is difficult to predict in advance of the actual occurrence of a given traffic situation. Often a calibration procedure is required for each freeway segment and this would be extremely time consuming. The other problem

is that, to make physical queue length relevant, the location of the incident needs to be estimated. In this sense we would need to know the exact spatial distribution of incidents and not just the occurrence probabilities in relation to coded link segments. This increased complexity is not likely to improve the estimation without extensive additional effort.

Another alternative to deterministic queuing is stochastic queuing or Poisson processes. This method, though extremely useful in numerous other applications, is not applicable here for several reasons. Firstly, stochastic queuing models require that the arrival rate be less than the departure rate. This requirement is not usually met during incident conditions that experience excessive congestion. Secondly, the system must attain equilibrium with steady arrivals and departures over a long period of time. Again, it is unlikely for a traffic system to reach equilibrium given that traffic flows can fluctuate dramatically over a day. Lastly, independent arrivals are necessary for Poisson queuing to be valid and, especially during peak periods, drivers' behaviour may be based on that of other drivers.

3.2 Network Model

To determine shortest paths, shortest alternative routes, and ultimately project traffic volumes, or link arrival rates, under both incident and non-incident conditions a network model is developed. These arrival rate predictions are key is estimating vehicle delay utilizing deterministic queuing. Generally, any network model is defined by a set of nodes and a set of links. The starting and ending nodes geometrically define each road segment and corresponding road segment attributes are referenced to a directional link (Figure 3.1, Appendix B - NetLink & NetNode tables).

By performing the idealization for every road segment within the analysis area a network structure can be established (Figure 3.2). This complete structure also includes origin and destination nodes (or trip generators and attractors) where vehicle trips are generated and

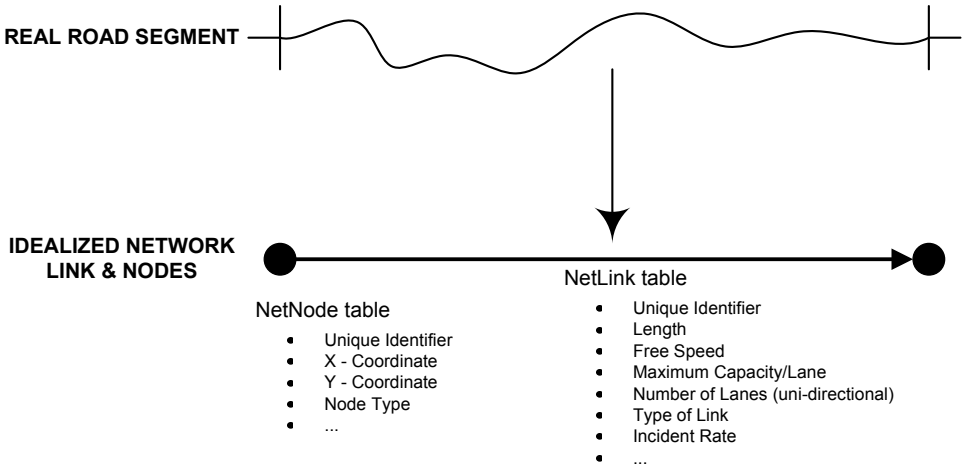


Figure 3.1: Network Representation

attracted and will be used to predict traffic volumes on road segments. The estimation of the magnitude of these trips is part of the four-step transportation planning process (Ortuzar and Willumsen, 1994).

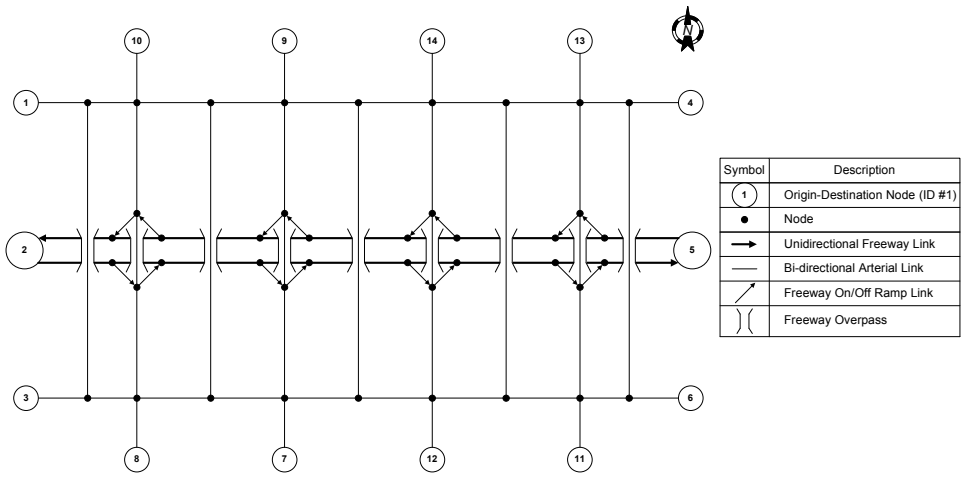


Figure 3.2: Network Representation with Generators

3.3 A Link-Based Incident Model

In order to quantify network-wide delay reduction benefit due to CMS information provision, a model for estimating incident delay on a given link must first be developed. The data required to construct the queuing diagram model are derived from databases. Link capacity, expected capacity reduction (incidents), expected incident duration, and detection time and processing time are all essential to the proposed model (NetLink Table, Appendix B). It is notable that incident duration is assumed to be a deterministic value, since it is expected that incident duration would be relatively consistent and reduced in length for freeway segments within the extents of an advanced traffic management system. The effects of stochastic variations in incident duration will be examined in Chapter 6. The original arrival rate for each incident link is determined based on trip rates (ODTrips Table, Appendix B) and the corresponding traffic assignment results.

A time-dependent model is used to estimate user delay on a specific link during an incident. The time of day is divided into several periods, depending on temporal variation of demand distribution (Figure 3.3). The total user delay for each of the time periods may be classified into one of three cases: Case I - off-peak, Case II - peak under-capacity, or Case III - peak over-capacity. The queuing diagrams for Cases I, II, and III are shown in Figures 3.4, 3.5, and 3.6 respectively. The shaded area represents the total user delay when an incident occurs on the specific link a during a specific period p, denoted by D_a^p.

The incident occurrence time, denoted by t^o, is defined for each of the three cases as follows:

$$t^o = \begin{cases} 0 & \text{Case I} \\ 0 \rightarrow t^p & \text{Case II, Case III} \end{cases} \tag{3.1}$$

Where t^p represents the end of the peak period under consideration, p, and start of

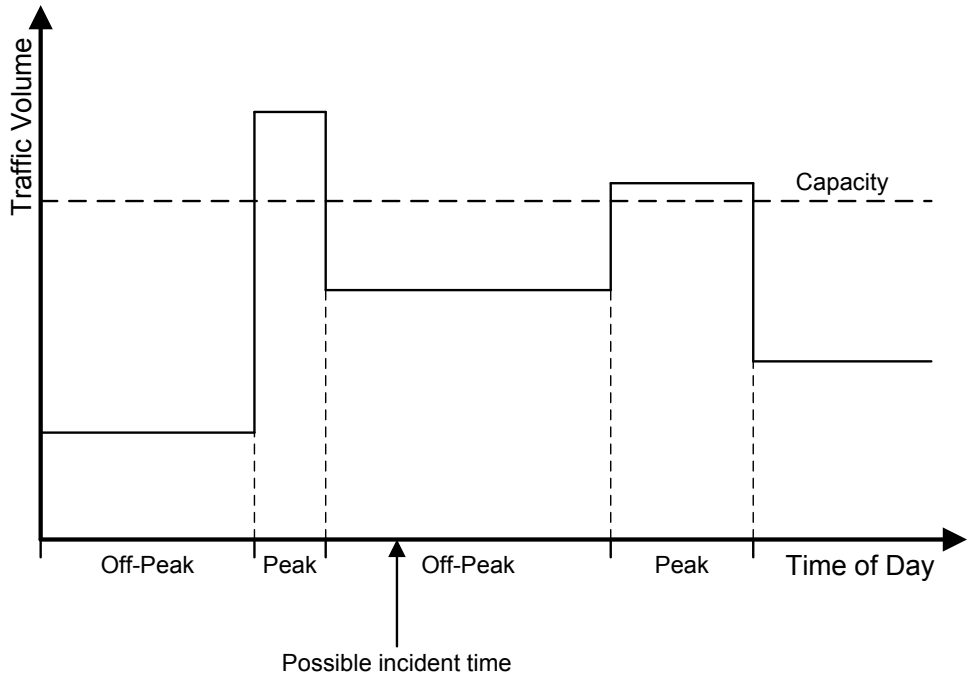


Figure 3.3: Time of Day Demand Distribution

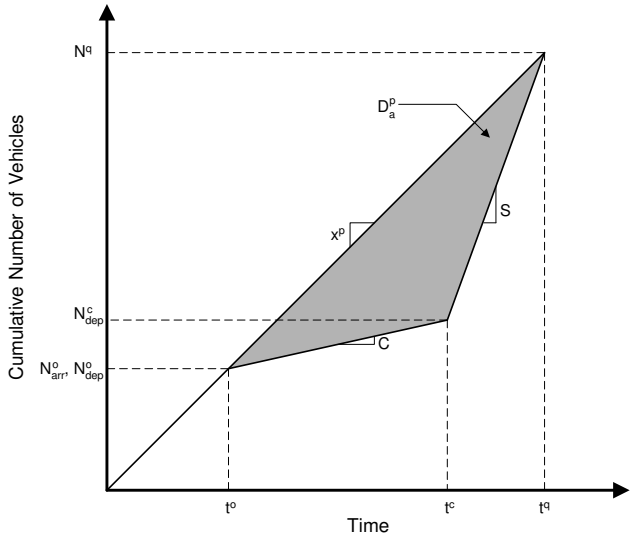


Figure 3.4: Vehicle Queuing Under Incident Conditions (Case I)

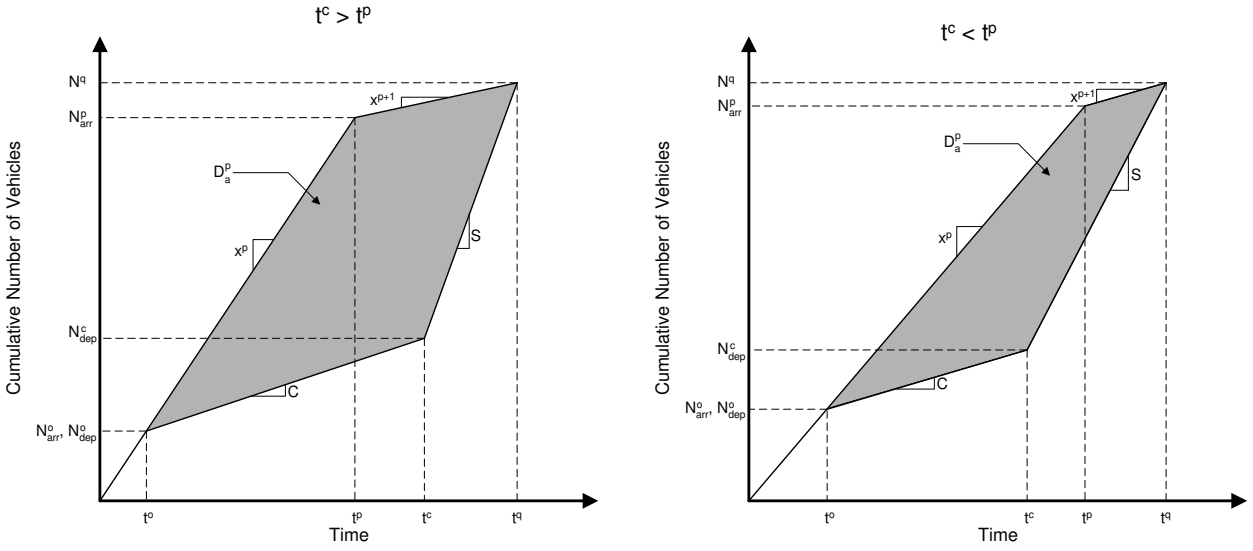


Figure 3.5: Vehicle Queuing Under Incident Conditions (Case II)

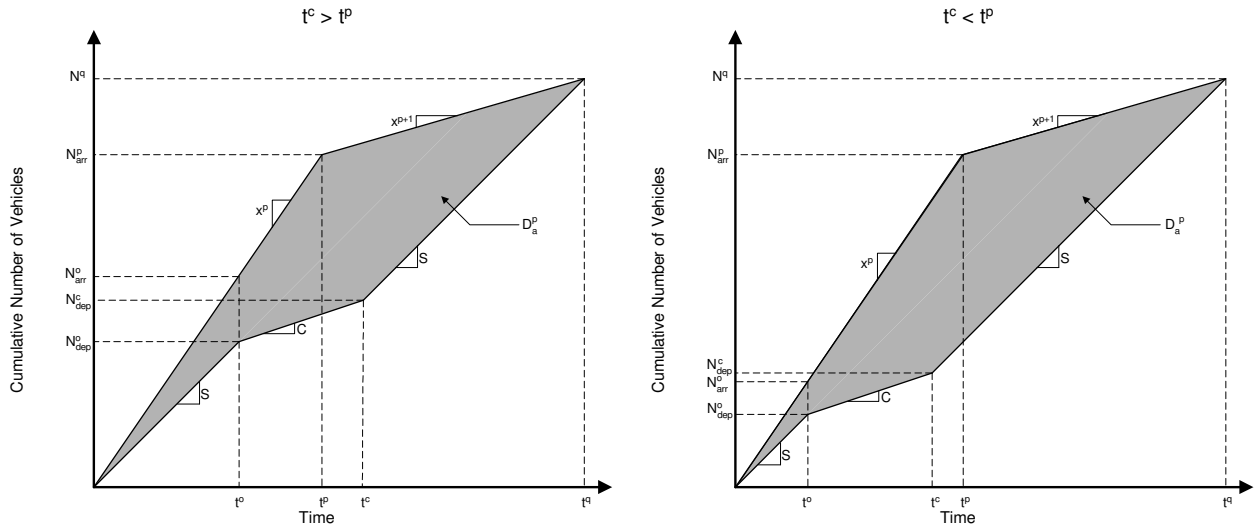


Figure 3.6: Vehicle Queuing Under Incident Conditions (Case III)

the off-peak period $p + 1$. Note that for the off-peak case (Case I), it is assumed that all links are under-saturated in normal traffic conditions and the incident occurrence time has negligible effect on the user delay. Therefore, to simplify calculations, only incidents that occur at the beginning of the time period, and cleared before the peak period, will be considered as Case I. If the computed time required to clear an off-peak incident queue exceeds the time to the beginning of the peak period, then Case II may be used in lieu of Case I conditions. It is additionally assumed that, for the peak period cases (Case II and III), the incident occurrence time is uniformly distributed from zero to the period duration under consideration. To determine a value for user delay for the peak period cases, a range of incident occurrence times must be considered and the user delay results averaged for each of these occurrence times. An arithmetic mean of user delays for a given number of incident occurrence times are evaluated. This same method is applied in the following section for estimating delay with information.

The cumulative arrivals and departures at the incident occurrence time (t^o), denoted by N^o_{arr} and N^o_{dep} respectively, can be calculated as follows:

$$N^o_{dep} = N^o_{arr} = 0 \qquad \text{Case I} \tag{3.2}$$

$$N^o_{arr} = x^p\, t^o \qquad \text{Case II, Case III} \tag{3.3}$$

$$N^o_{dep} = \begin{cases} x^p\, t^o & \text{Case II} \\ S\, t^o & \text{Case III} \end{cases} \tag{3.4}$$

where

x^p = flow rate on specific link during normal traffic conditions (non-incident) during period p. The arrival (flow) rate is assumed to be known for each of the time periods (veh/hour);

S = capacity of specific link during incident recovery period (veh/hour);

t^o = incident occurrence time (hours).

Note that, for the under-capacity cases (Cases I & II), the cumulative arrivals and departures are equivalent at the incident occurrence time. However, since the link is over capacity for Case III, the cumulative number of arrivals is greater than departures.

Similar to the occurrence time, the time of incident clearance (t^c) and the corresponding cumulative number of departures (N^c_{dep}) is defined for each of the three cases as follows:

$$t^c = t^o + \tau \qquad \text{Case I, Case II, Case III} \tag{3.5}$$

$$N^c_{dep} = N^o_{dep} + C\,\tau \qquad \text{Case I, Case II, Case III} \tag{3.6}$$

where

τ = time required to clear the incident (hours);

C = reduced capacity of the link during the incident (veh/hour);

t^o = incident occurrence time (hours);

N^o_{dep} = cumulative number of departures at the incident occurrence time.

Note that the cumulative number of arrivals at the time of incident clearance is not required to determine the user delay for any of the cases. It is also notable that the arrival rate to the incident link must be greater than the reduced capacity for Equation 3.6 to be valid. Otherwise a trivial case with zero delay must be considered when the arrival rate is less than the reduced capacity.

Recalling the assumption that the original arrival rate for a specific link (x^p) and the duration of each time period (t^p) are both known. Therefore, the cumulative number of arrivals at the end of the peak period p under consideration, denoted by N^p_{arr}, can be determined from the following equation:

$$N^p_{arr} = x^p\, t^p \qquad \text{Case II, Case III} \tag{3.7}$$

Similar to the cumulative number of arrivals at the time of incident clearance, the cumulative number of departures at t^p is not required to determine the user delay for any of the cases. Also recall that for the off-peak incident case, Case I, the incident is assumed to have occurred at the start of the time period under consideration. Therefore, for Case I under normal conditions, the end of the time period will occur after the incident queue clearance time (t^q).

The incident queue clearance time (t^q) and the cumulative number of vehicles at the queue clearance time (N^q) are determined from the following equations:

$$t^q = \begin{cases} \frac{(S-C)\,t^c}{(S-x^p)} & \text{Case I} \\ \frac{x^p\,(t^p-t^o)-x^{p+1}\,t^p+(S-C)\,t^c+C\,t^o}{(S-x^{p+1})} & \text{Case II} \\ \frac{(x^p-x^{p+1})\,t^p+(S-C)\,(t^c-t^o)}{(S-x^{p+1})} & \text{Case III} \end{cases} \tag{3.8}$$

$$N^q = \begin{cases} x^p\,t^q & \text{Case I} \\ x^p\,t^p + x^{p+1}\,(t^q - t^p) & \text{Case II, Case III} \end{cases} \tag{3.9}$$

It is important to note that the delay estimation methodology discussed above, while commonly used in literature such as the Highway Capacity Manual (Transportation Research Board, 2000), does not account for queue spill-back and its possible effects on delay estimation. Queue spillback may cause three possible effects on delay estimation. The first effect is that queue spillback may block neighbouring intersections, which would then reduce the capacity of neighbouring links and cause additional delays. Queue spillback may also induce traffic diversion from the incident link, which would lead to an arrival rate at the incident link lower than what would normally be expected under non-incident conditions. Lastly, queue spillback may block diversion access points such as freeway off ramps, preventing drivers from altering their routes. These effects are difficult to accurately represent in a delay estimation model, requiring knowledge of detailed network geometry and driver behaviour, and therefore are not treated in this research.

3.4 Expected Incident Delay

The queuing model developed to this point has focused on estimating delay for a single incident on a single link in the network. However, the focus of the proposed CMS location model is to plan for *future* incidents. Therefore, since the location and frequency of future incidents cannot be known beforehand, the CMS planning model will instead rely on his-

torical rates of incidents. The expected number of incidents on link a during time period p (n_a^p) may then be developed from the historical incident rate (Equation 3.10).

$$n_a^p = \Delta^p \, x^p \, L_a \, r_a \tag{3.10}$$

$$\sum_p \Delta^p = 24$$

where

n_a^p = expected number of incidents on link a during time period p (# incidents);

Δ^p = duration of time period p, which is assumed to be known (hours);

x^p = original link arrival rate (veh/hour);

L_a = length of link a (km);

r_a = incident rate for link a (# incidents/veh-km).

Equation 3.10 can now be combined with the incident delay estimation to determine the expected incident delay for a given link a. Recall that the area between the arrival and departure curves in the queuing diagram represents the magnitude of the delay estimation (Figure 3.4, 3.5 or 3.6). The expected incident delay for link a in a typical year is developed in Equation 3.11, with no distinction between weekend and weekday incident-induced delays. A more accurate analysis may be performed using weekend demand and incident scenarios.

$$D_a^{year} = 365 \sum_p D_a^p \, n_a^p \tag{3.11}$$

where

D_a^{year} = expected incident delay on link a for a typical year (veh-hours);

D_a^p = total vehicle incident delay on link a, during time period p, without CMS information calculated using Figures 3.4 $\rightarrow$ 3.6 and Equations 3.2 $\rightarrow$ 3.8 (veh-hours);

Equation 3.11 detailed calculations for expected incident delay for a single link. To extend this result to the entire traffic network an incident, and the corresponding rate, must be considered for every link. This straightforward total delay calculation is detailed in Equation 3.12.

$$D_{network}^{year} = \sum_a D_a^{year} \tag{3.12}$$

where

$D_{network}^{year}$ = expected delays caused by incidents on the entire traffic network during a typical year (veh-hours).

CHAPTER 4

INCIDENT DELAY WITH INFORMATION

The previous chapter described a method to estimate expected incident induced delay under the assumption that drivers are not informed of the incident. However, to optimize the locations of CMS across an entire freeway-arterial network the expected incident delay under the assumption that some drivers would be informed of the incidents by CMS must be estimated. Such estimation requires a set of models that can be used to predict driver's route diversion behaviour, identifying shortest paths and assign traffic to the underlying network. This chapter, derived from Henderson et al. (2004b), discusses these individual elements with a specific focus on driver diversion behaviour impact on incident induced delay.

4.1 Overview of Methodology

Providing information to drivers via CMS messages is expected to produce a system benefit by diverting vehicles from congested incident locations to less congested alternative routes. The benefit of CMS information can be illustrated using a deterministic queuing diagram (Figure 4.1). The straight, solid line represents the cumulative arrival of vehicles at an incident location under no information scenario, while the dashed line depicts the arrivals under real-time information from CMS. The arrival rate is reduced because some vehicles divert to alternative routes. The total delay is therefore reduced accordingly, with the corresponding delay benefit represented by the solid area in Figure 4.1. In order to estimate the reduced delay we need to predict the traffic diversion rate under each incident scenario. The former requires individual path flows be determined and potential benefit for each path be quantified. The following sections provide a detailed discussion on these components, including:

1. *Traffic Assignment* : Determine link traffic volumes and path flows utilizing a path-based assignment method (Section 4.3).

2. *Incident Delay* : Calculate incident induced delay without the benefit of CMS information based on the traffic volumes (Section 3.4).

3. *Traffic Diversion* : Estimate driver response to CMS information (Section 4.2).

4. *Delay Reduction* : Calculate the benefit of CMS information (Section 4.4).

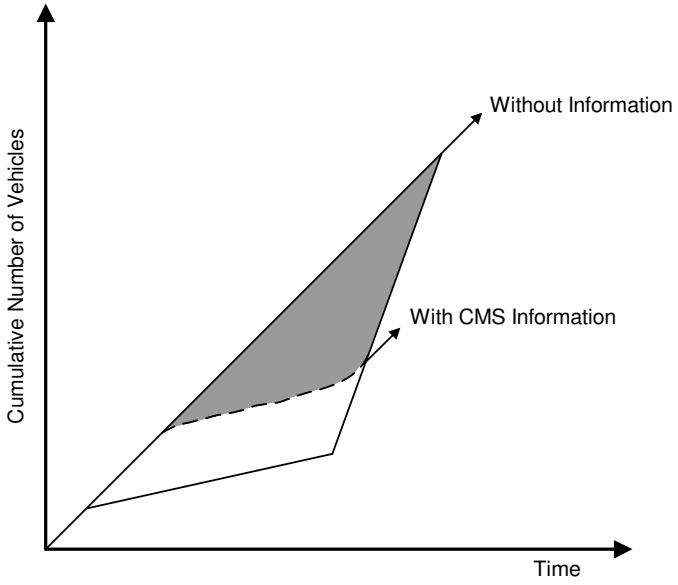


Figure 4.1: Benefit of Incident Information

4.2 Diversion Model

To be able to quantify the benefits of a given CMS, a diversion model is required to predict the number of vehicles that would divert to alternative routes due to message

activation of the CMS during incident conditions. When a driver is provided information from a CMS that an incident has occurred along the intended travel path, he/she would make a decision to either stay on the same route or divert to an alternative route. This decision depends on various factors such as severity of the incident, current extent of queue caused by the incident, the driver's experience and familiarity of the network, and incident characteristics delivered via the CMS. Therefore, modeling the underlying decisions is a significant challenge due to the behavioural complexity of the drivers' response to incidents and incident information (Wardman et al., 1998). In the two existing studies on the CMS location problem, Abbas and McCoy (1999) assumed a constant fixed diversion rate regardless of availability of alternative routes, severity of incidents and various other factors. In Chiu et al. (2001)'s simulation-based model, a bounded route choice model was applied, assuming a driver would divert to an alternative route if the expected travel time saving exceeds a certain threshold.

A simplified discrete choice model is proposed in this research to capture the major characteristics of drivers' common response behaviour under incident conditions. The model was motivated by the empirical work of Wardman et al. (1998), assuming that the probability for a driver to choose to divert depends on the expected travel time saving from diverting with the following logit form:

$$P_{k,m}(t) = \frac{1}{1 + e^{\alpha - \beta S_{k,m}(t)}} \tag{4.1}$$

where

$P_{k,m}(t)$ = probability for a vehicle, arriving at time t and traveling through CMS k on path m, to divert to an alternative route;

α, β = model parameters;

$S_{k,m}(t)$ = travel time savings ratio as defined in the next paragraph.

The travel time savings ratio, $S_{k,m}(t)$, is based on the expected delay that vehicles joining the incident queue will experience and the travel time through the shortest alternative route, as defined as follows:

$$S_{k,m}(t) = \frac{T_{k,m}(t) - T^*_{k,m}}{T^*_{k,m}} \tag{4.2}$$

where

$T_{k,m}(t)$ = expected travel time a vehicle joining the queue at time t will experience (hours);

$T^*_{k,m}$ = travel time of the shortest alternative route, not traversing the incident link.

The expected travel time experienced by a vehicle joining the queue, $T_{k,m}(t)$, is based on two components. The first component is the expected travel time from the CMS link, through the incident link, to the destination node of the path under incident-free conditions. The second component is based on the expected queuing delay for a vehicle if it were to continue on its original path.

It is possible for drivers to pass more than one CMS enroute to the incident link, however the diversion model does not distinguish between between single and multiple CMS information sources. Instead, the assumption is that drivers would delay their decision to divert until the closest CMS to the incident. Therefore, in this research, only the closest CMS for each path is considered in the diversion calculations of Equations 4.1 and 4.2. Further driver behaviour research is required before multiple CMS information sources can be incorporated into a diversion model.

The diversion model suggests that the proportion of vehicles that would divert increases as the travel-time savings increase. The relationship between diversion probability, or rate, and travel time savings is depicted in Figure 4.2 under three assumed combinations of

model parameters. The curves developed based on the logit model structure are intuitively correct: the higher the travel time savings, the higher the probability for a vehicle to divert; drivers are usually reluctant to change routes with a small percentage of savings. For example, if $\alpha = 5$ and $\beta = 5$, a 50% probability of diversion would correspond to a 50% travel time savings and 100% diversion for travel time savings of 100% or greater.

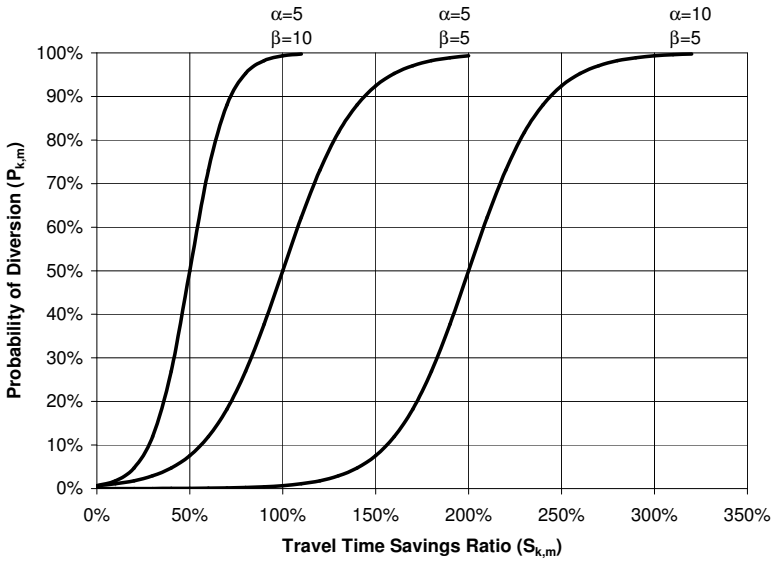


Figure 4.2: Diversion vs. Travel Time Savings

The parameters α and β essentially model variations in drivers' characteristics (e.g., aggressiveness) and information attributes (e.g., types, level of reliability, frequency etc.). Realistic estimates of these model parameters could be obtained through a statistical analysis of stated preference or revealed preference survey results as in Wardman et al. (1998). A sensitivity analysis is described in Chapter 6 to evaluate the potential impact of these parameters on the final CMS location solutions.

The reduced flow rate at an incident link due to a CMS may be determined by applying diversion probabilities, derived for a single path, to all paths (Equation 4.3).

$$\hat{x}_k(t) = \sum_m f_{k,m} \left[1 - P_{k,m}(t)\right] \tag{4.3}$$

where

$\hat{x}_k(t)$ = flow on the incident link, or reduced arrival rate, for vehicles that traverse both CMS k and the incident link (veh/hour);

$f_{k,m}$ = flow on the m^{th} path passing both CMS k and the incident link had there been no incident (veh/hour);

$P_{k,m}(t)$ = probability for a vehicle traveling through CMS k, on path m, to divert to an alternative route, as defined in Equation 4.1.

In determining the reduced arrival rate at a given link due to all CMS, it is assumed that drivers would defer their decision until they reach the CMS that is closest to the incident link. That is, for a path flow that traverses several CMS, only the CMS closest to the incident link has an effect on the diversion rate. An activating zone is also considered so that only CMS within a certain distance of the incident will display information.

4.3 Path-Based Traffic Assignment

In order to estimate the delay caused by incidents (with or without information), traffic volume estimates, for the time periods of interest, at individual network links must first be obtained. Typically, network traffic volumes are obtained using a link-based method (e.g. Frank-Wolfe method), which produces link flow estimates from a matrix of origin-destination flows based on Wardrop's user equilibrium (UE) assumption. This information is, however, not sufficient for the proposed benefit model as it requires not only the volume of traffic on a specific link but the individual path flows between origins and destinations of that traffic as well. This information is used both in the prediction of traffic diversion and

the alternative route travel times. A conceptual difference between the link-based method and path-based assignment methods is illustrated in Figure 4.3.

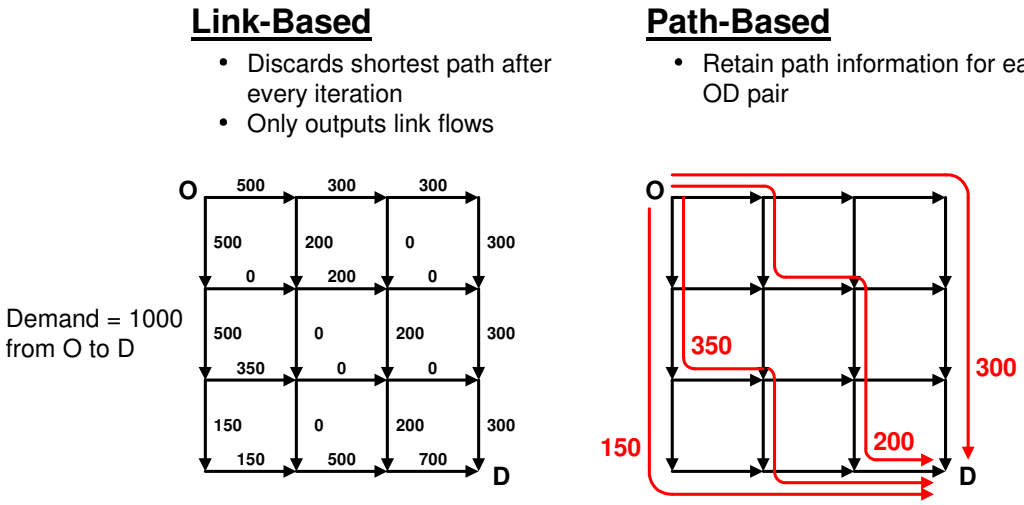


Figure 4.3: Conceptual Differences Between Assignment Methods

Generally, there are two classes of path-based assignment methods: incremental and user-equilibrium. The incremental method performs successive all-or-nothing assignments on a parsed OD trip matrix (Sheffi, 1985). Incremental assignment is not guaranteed to result in a UE condition and will not be considered further. Several user-equilibrium path-based methods for traffic assignment have been developed, including a modified Frank-Wolfe approach. However, the gradient projection (GP) method proposed by Jayakrishnan et al. (1994) for traffic assignment has thus far proven the most efficient (Chen and Lee, 1999). This formulation is based on Bertsekas (1976)'s more general version of GP for network assignment based on the Goldstein-Levitin-Polyak gradient projection method of non-linear programming. The complete algorithm is presented here, mainly following Chen and Lee (1999).

1. *Definitions*

$$
\begin{aligned}
q_{rs} &= \text{trip demand between origin } r \text{ and destination } s. \\
x_a(n) &= \text{flow on link } a \text{ during iteration } n. \\
t_a &= \text{travel time on link } a. \\
K_{rs}(n) &= \text{set of paths between } r \text{ and } s \text{ during iteration } n. \\
\bar{k}_{rs}(n) &= \text{shortest path between } r \text{ and } s \text{ during iteration } n. \\
f_k^{rs} &= \text{set of path flows between } r \text{ and } s \text{ on paths } k. \\
\delta_{ka}^{rs} &= \begin{cases} 1 & \text{if link } a \text{ is on path } k \text{ connecting } r \text{ and } s \\ 0 & \text{otherwise} \end{cases} \\
\alpha(n) &= \text{update stepsize that affects speed of convergence (usually = 1).}
\end{aligned}
$$

2. *Initialization* : Generate an initial path for each OD pair
   - *i)* Set $x_a(0) = 0$, $t_a = t_a[x_a(0)], \forall\, a$ and $K_{rs}(0) = \emptyset$.
   - *ii)* Set iteration counter $n = 1$.
   - *iii)* Solve the shortest path problem: $\bar{k}_{rs}(n), K_{rs}(n) = \bar{k}_{rs}(n) \cup K_{rs}(n-1), \forall r, s$.
   - *iv)* Perform AON assignment: $f^{rs}_{\bar{k}_{rs}(n)} = q_{rs}, \forall\, r, s$.
   - *v)* Assign path flows to links: $x_a(n) = \sum_{r \in R} \sum_{s \in S} \sum_{k \in K_{rs}(n)} f_k^{rs}(n) \delta_{ka}^{rs}, \forall\, a$.

3. *Column Generation* : Generate shortest path based on current link travel times and augment the set of generated paths if it is new.
   - *i)* Increment iteration counter: $n = n + 1$.
   - *ii)* Update link travel time: $t_a(n) = t_a[x_a(n-1)], \forall\, a$.
   - *iii)* Solve the shorted path problem: $\bar{k}_{rs}(n), \forall\, r, s$.
   - *iv)* Augment path $\bar{k}_{rs}(n)$ to the path set $K_{rs}(n-1)$ if it has not already existed:

     If $\bar{k}_{rs}(n) \notin K_{rs}(n-1)$, then $K_{rs}(n) = \bar{k}_{rs}(n) \cup K_{rs}(n-1)$.

     Otherwise, tag the shortest path among the paths in $K_{rs}(n-1)$ as $\bar{k}_{rs}(n)$ and set $K_{rs}(n) = K_{rs}(n-1)$.

4. *Equilibration* : Solve the path-formulated traffic assignment problem over the restricted set of paths generated thus far, $K_{rs}(n)$.

   i) Compute first and second derivative path costs: $d_k^{rs}(n)$, $d_{\bar{k}_{rs}(n)}^{rs}$, and $s_k^{rs}(n)$

$$\begin{aligned} d_k^{rs}(n) &= \sum_{a \in A} t_a(n)\, \delta_{ka}^{rs} && \forall\, k \in K_{rs}(n),\ k \neq \bar{k}_{rs}(n),\ r, s \\ d_{\bar{k}_{rs}}^{rs}(n) &= \sum_{a \in A} t_a(n)\, \delta_{\bar{k}_{rs}(n)a}^{rs} && \forall\, r, s \\ s_k^{rs}(n) &= \sum_{a \in A} t_a'(n)\, (\delta_{ka}^{rs} - \delta_{\bar{k}_{rs}(n)a}^{rs})^2 && \forall\, k \in K_{rs}(n),\ k \neq \bar{k}_{rs}(n),\ r, s \end{aligned}$$

   ii) Update non-shortest path flows:

$$f_k^{rs}(n+1) = \max \begin{cases} f_k^{rs}(n) - \frac{\alpha(n)}{s_k^{rs}(n)} (d_k^{rs}(n) - d_{\bar{k}_{rs}(n)}^{rs}) \\ 0 \end{cases} \qquad \forall k \in K_{rs}(n),\ k \neq \bar{k}_{rs}(n),\ r, s$$

   iii) If $f_k^{rs}(n+1) = 0$, then drop path k:

$$K_{rs}(n) = K_{rs}(n) \setminus k$$

   iv) Update shortest path flow:

$$f_{\bar{k}_{rs}(n)}^{rs}(n+1) = q_{rs} - \sum_{\substack{k \in K_{rs}(n) \\ k \neq \bar{k}_{rs}(n)}} f_k^{rs}(n+1) \qquad \forall\, r, s$$

   v) Update link flows:

$$x_a(n+1) = \sum_{r \in R} \sum_{s \in S} \sum_{k \in K_{rs}(n)} f_k^{rs}(n+1)\, \delta_{ka}^{rs} \quad \forall\, a$$

5. *Termination* : Terminate the algorithm if it satisfies the stopping criterion.

   i) If $\max_{rs} \sum_{\substack{k \in K_{rs}(n) \\ k \neq \bar{k}_{rs}(n)}} \frac{f_k^{rs}(n)}{q_{rs}} \left(\frac{d_k^{rs}(n) - d_{\bar{k}_{rs}(n)}^{rs}}{d_k^{rs}(n)} \right) \leq \epsilon$, then terminate.

   Otherwise, go to *Column Generation*.

The major concern when utilizing path enumeration methods is the realization that path flows are usually not unique. An example of non-unique path flows is shown in Figure 4.4. Current path assignment methods rely on initial path flow generation to determine paths,

i.e. the initial solution dictates the non-unique final solution. The impact of non-unique path flow solutions was not considered in this research and the extent to which optimal CMS locations depend on path flows is unknown.

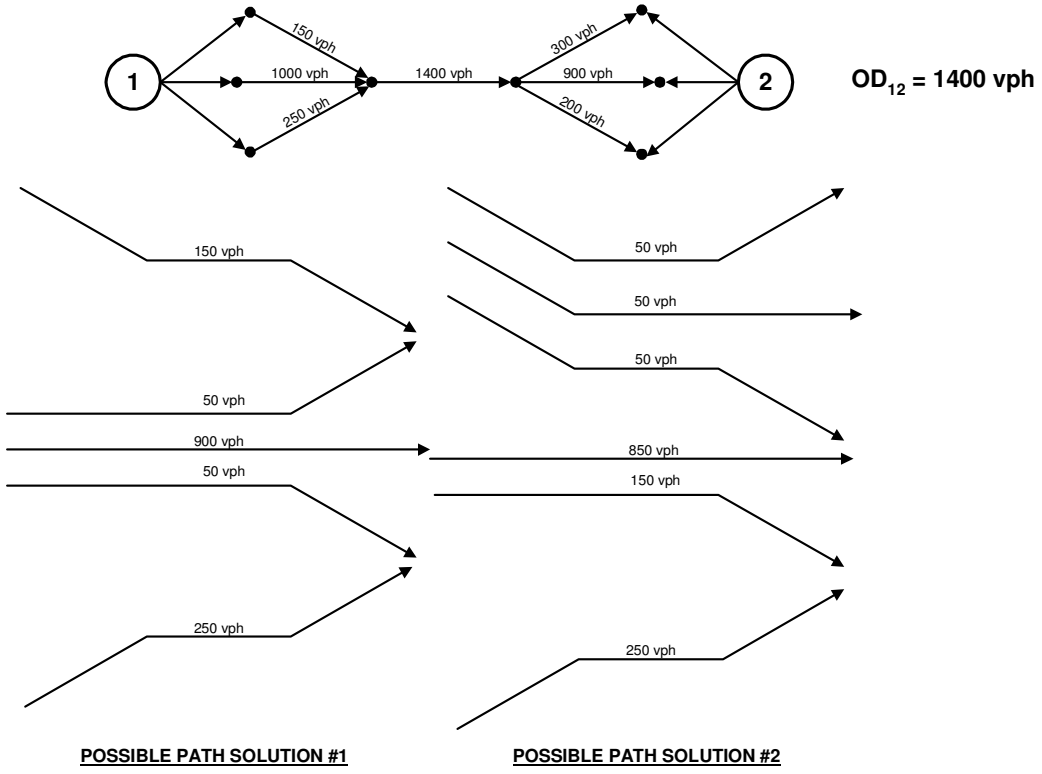


Figure 4.4: Non-Uniqueness of Path Assignment Solution

4.4 Delay Under Realtime CMS Information

The time-dependent deterministic queuing model may be combined with the diversion model to estimate the reduction in user delay for a given incident link and a given set of CMS. The set of CMS is ordered based on the shortest path travel time from the CMS link

to the incident link, T_k, with the shortest time at the start of the CMS set and the longest time at the end of the CMS set. The method of determining the reduction in incident delay may be best illustrated using an example based on the simplest off-peak case, Case I. Consider a network with three CMS (Figure 4.5), ordered from the closest to longest from the incident link, and an opportunity to divert to an alternative route after each of the CMS. The corresponding queuing diagram for this simple network is shown in Figure 4.6. Although this method is illustrated with three CMS, the extension to any number of CMS is straightforward.

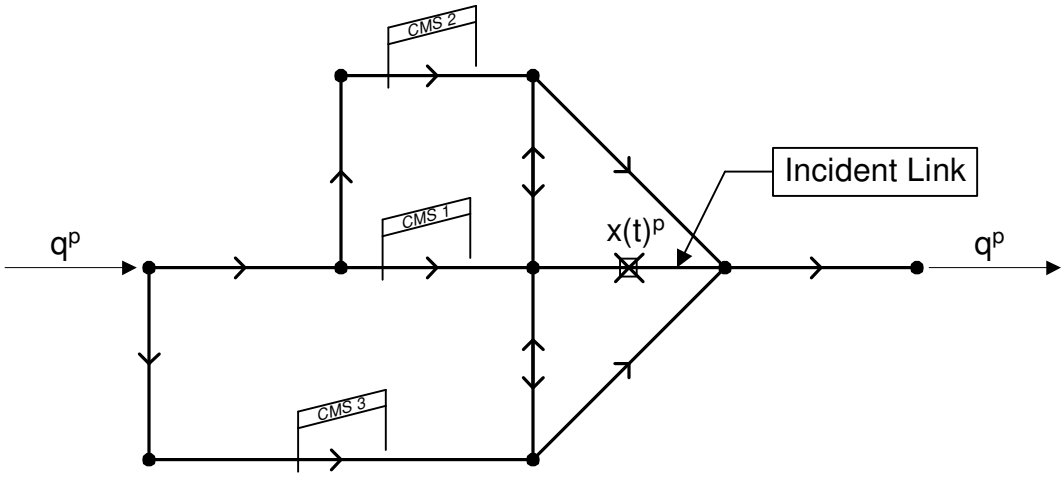


Figure 4.5: Three CMS Simple Network

Shown in Figure 4.6, an incident occurs at time t^o, and, after a time lag including three major components: incident detection time, information processing time, and CMS activating time, diversion due to CMS starts at time t^s. The incident link does not experience a reduction in arrival rate until $t^s + T_1$ due to the travel time component from the first CMS to the incident link. The same is true for additional reduction in arrivals caused by CMS 2 and CMS 3.

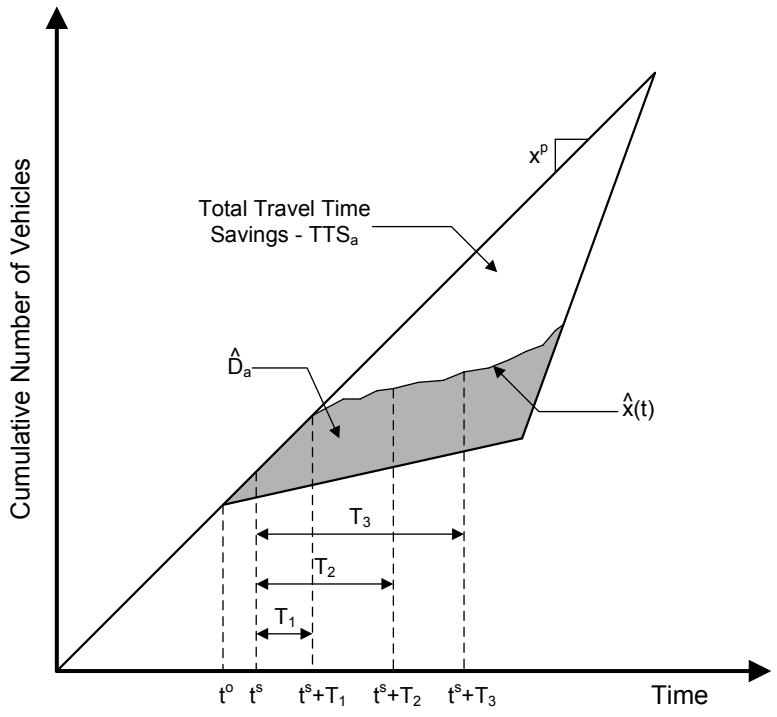


Figure 4.6: Simple Network Queuing Diagram

Determining the actual reduced flow rate at each interval between the diversion start times corresponding to two neighbouring CMS is not straightforward. This is because the reduction in flow rate depends on the proportion of traffic diverted due to CMS information, while traffic diversion is a function of travel time savings, which in turn depends on how much traffic is diverted. We solve this interdependency problem by constructing the reduced flow rate curve from left, starting at time $t^s + T_1$, to right at a small time interval (e.g. 5 minutes). At the start of each interval, the cumulative number of arrivals is calculated based on the reduced arrival rate of the previous interval. This cumulative arrivals is then used to estimate the expected delay for a vehicle arriving during this time instance and all vehicles arriving at the current interval are assumed to experience this same delay.

The diversion rate can then be estimated for the corresponding interval. This process continues until the reduced arrival curve, $\hat{x}(t)$, intersects with the cumulative departure curve.

The benefit estimation model can now be completely defined to estimate the expected travel time savings for a specific incident link a during time period p (TTS_a^p), as well as the expected savings for the entire traffic network during the year.

$$TTS_a^p(\mathcal{F}) = [D_a^p - \hat{D}_a^p(\mathcal{F})]\, n_a^p \tag{4.4}$$

$$TTS_{network}^{year}(\mathcal{F}) = 365 \sum_p \sum_a TTS_a^p(\mathcal{F}) \tag{4.5}$$

where

$\mathcal{F}$ = allocated CMS locations;

D_a^p = total vehicle incident delay on link a, during time period p, without CMS information (veh-hours) as in Equation 3.11;

$\hat{D}_a^p(\mathcal{F})$ = total vehicle incident delay on link a, during time period p, with CMS information (veh-hours);

n_a^p = expected number of incidents on link a during time period p (# incidents) as in Equation 3.10;

$TTS_{network}^{year}(\mathcal{F})$ = expected travel time savings for the entire network during a year (veh-hours).

4.5 Effect of Diversion on Alternative Routes

The diversion model presented in Section 4.2 approximates the rate at which drivers divert from their originally intended route to one or more alternatives during incident conditions. However, no prediction is made as to what those alternative routes are, since there are

presently no reliable route choice models during traffic equilibrium disruption. Additionally, the increased travel time experienced by drivers on these alternatives is not considered in the travel time savings calculation of Equation 4.5.

To partially account for this impact an alteration to the diversion rate and incident link-based travel time savings is proposed that re-assigns diverted traffic to the shortest alternative route between the CMS link and the trip destination. A diversion equilibrium is approximated through successive iterations of $T_{k,m}(t)$ and $T^*_{k,m}$ in Equation 4.2. During the iterations, link travel times are updated to reflect volume changes resulting from traffic diversion. These updated link travel times may then be included in Equation 4.5 to adequately reflect the negative impact of diverted traffic on otherwise unaffected links.

$$TTS^p_a(\mathcal{F}) = \left[D^p_a - \hat{D}^p_a(\mathcal{F}) - TTI^p_a(\mathcal{F})\right] n^p_a \tag{4.6}$$

where

$TTI^p_a(\mathcal{F})$ = travel time increase caused by diversion from incident link a during time period p;

CHAPTER 5

LOCATION OPTIMIZATION

The previous section focused on formulating a model to estimate the network benefit of providing incident information to drivers via messages at predetermined CMS locations. The next step is to extend the benefit model to include a CMS location optimization routine so that decisions can be made where to locate future CMS installations. The section will start with the location formulation with solution strategies following.

5.1 Problem Formulation

Many of the traditional facility location formulations in operations research literature seek to minimize a transportation cost objective. In the context of this research the location objective is to maximize the expected travel time savings over the duration of a CMS installation life. Furthermore, the candidate locations are links for the presented model as opposed to nodes used in traditional location models.

The CMS location problem can be described as follows. Given a maximum number of CMS to allocate N and a set of links A from which to choose candidate CMS locations a determine the optimal set of CMS $\mathcal{F}$ that maximizes the following objective within a cost constraint.

$$\max \quad TTS^{year}_{network}(\mathcal{F}) \tag{5.1}$$

subject to

$$\sum_{a \in \mathcal{F}} f_a \leq F \tag{5.2}$$

where

$TTS^{year}_{network}(\mathcal{F})$	=	travel time savings over all p time periods and all incident links a in a year (veh-hours);
f_a	=	equivalent annual construction and operating costs for a CMS installation on link a (\$);
F	=	maximum CMS costs (\$).

There are two major obstacles preventing the application of traditional combinatorial optimization techniques to this problem. First, the problem is implicitly structured given that diversion rates are a function of delay, while delay in turn depends on cumulative arrivals, which are also a function of diversion rates. Secondly, location problems involving interactions between distinct locations belong to a class of computationally difficult problem sets that, generally, cannot be solved to optimality. Therefore, this section will explore two heuristic strategies, namely, greedy allocation and genetic algorithms.

5.2 Greedy Allocation

The main idea behind greedy optimization techniques is that the problem at hand is broken into a set of incremental problems which are then solved sequentially. Generally, greedy solutions are best known for producing "good" solutions, which are also occasionally optimal solutions, in a reasonable amount of computational time.

A greedy algorithm as applied to the present CMS location problem entails sequential steps of locating one CMS at a time. Once a CMS is located, its location is assumed to be fixed in the subsequent steps of locating other CMS. Figure 5.1 gives the pseudocode of this algorithm.

```
for n = 1 to N do
  for all a ∈ A do
    if a ∉ L then
      temporarily add a CMS at location a to L
      calculate TTS^year_network (Equation 4.5)
      if TTS^year_network is maximized in this iteration then
        a^max ⇐ a
      end if
      remove CMS at location a from L
    end if
  end for
  add a^max to L
end for
```

Figure 5.1: Greedy Location Allocation

5.3 Genetic Optimization

Genetic algorithms (GA) as an optimization technique are outlined in Figure 5.2 and Appendix A. The reviewed research papers described in Section 2.4, specifically relating to location problems, contain some broad recommendations as to the most appropriate genetic algorithm parameters (Table 5.1). These parameters will be given the most consideration when choosing an appropriate variant of the simple genetic algorithm.

5.3.1 Encoding

Only the problem native encoding scheme will be considered as shown in Table 5.1. The main advantage of using this encoding scheme, as noted by GA researchers, is the explicit handling of the limitation on the number of facility locations. To accommodate this, each individual chromosome in the population, with a length corresponding to the number of CMS to add, contains genes that are references to *link* objects. Also, some algorithmic extensions have been added to the population generation, mutation, and crossover oper-

```
L ≡ population
λ ≡ population size
for i = 1 to λ do {randomly generate initial population}
   for j = 1 to sizeof(L) do
      l_j = random(A) {randomly select a link from A}
      L_i ⇐ l_j
   end for
   L ⇐ L_i
end for
mark best individual as elite
for i = 1 to max generations do {run GA max generations number of times}
   for all L ∈ L do
      evaluate fitness of individual L
   end for
   selection()
   crossover()
   mutation()
   generational replacement
end for
```

Figure 5.2: Genetic Algorithm

ators to prevent duplicate CMS at the same location. The binary encoding would have required a chromosome with the number of possible CMS locations as its length. Furthermore, limitations on the number of facility locations would be difficult to implement algorithmically.

5.3.2 Selection

The selection() function defined in Figure 5.2 can be performed utilizing the fitness proportionate, rank, or the tournament selection schemes. These schemes are detailed in Figures 5.3, 5.4, and 5.5. For each of these algorithms f_i denotes the fitness of individual i and F is the summation of the fitnesses of all individuals in the population.

Table 5.1: Suitable GA Parameter Options for CMS Location Optimization

Component	*Options*
Numerical Parameters [1]	Population Size Maximum Generations Crossover Probability [2] Mutation Probability [3] Elitism (Binary)
Encoding	Native
Selection	Fitness Proportionate (Figure 5.3, pg.63) Rank [4](Figure 5.4, pg.63) Tournament [5](Figure 5.5, pg.64)
Crossover	One point (Figure 5.6, pg.64) Uniform (Figure 5.7, pg.64) Fusion (Figure 5.8, pg.65)
Mutation	Simple (Figure 5.9, pg.65) Self-Adaptive (Equation 5.3, pg.66) Deterministic Increasing/Decreasing (Equations 5.4&5.5 , pg.66)

[1]Option is the real-value numerical quantity, other components involve selection from a discrete choice set.
[2]Not applicable to fusion crossover.
[3]Only applicable to simple mutation.
[4]Rank weight ($1 \leq \nu^{max} \leq 2$) also required.
[5]Tournament size (multiple of 2) also required.

5.3.3 Crossover

Three variants on the standard crossover operator have been considered for this research: one point (Figure 5.6), uniform (Figure 5.7), and fusion (Figure 5.8) which is a slight variant of the uniform crossover. It is notable that only the fusion operator prevents the duplication of CMS locations in a particular solution. Also, the fusion operator only produces one offspring compared to the two offspring for the other two operators. Additional algorithmic components were added to mutation operators for the one point and uniform crossover operators to remove duplication of solutions.

$F = \sum_i f_i$
$F^{rand} = rand[0,1]\, F$ {randomly generate a fitness}
$SUM = 0$
for $i = 1$ to λ **do** {add fitness values until F^{rand} reached}
  $SUM = SUM + f_i$
  if $SUM \geq F^{rand}$ **then**
    select individual i
  end if
end for

Figure 5.3: Fitness Proportionate Selection

Require: $1 \leq \nu^{max} \leq 2$
for $i = 1$ to λ **do**
  $P_i = \frac{1}{\lambda}\left(\nu^{max} - (\nu^{max} - \nu^{min})\,\frac{i-1}{\lambda-1}\right)$ $\{\nu^{min} = 2 - \nu^{max}\}$
end for
$sort(\mathcal{L})$ {sort the population in ascending order}
$\mathcal{P} = rand[0,1]$ {randomly generate a number on [0,1] interval}
$SUM = 0$
for $i = 1$ to λ **do**
  $SUM = SUM + P_i\,\lambda$
  while $SUM > \mathcal{P}$ **do**
    $\mathcal{C}(i) \Leftarrow i$
    $\mathcal{P} = \mathcal{P} + 1$
  end while
end for
Individual 1= $\mathcal{C}(rand[0, \lambda - 1])$ {1^{st} selected based on random integer index}
repeat
  Individual 2= $\mathcal{C}(rand[0, \lambda - 1])$
until Individual 2 $\neq$ Individual 1

Figure 5.4: Rank Selection

5.3.4 Mutation

All of the mutation schemes presented here perform that same mutation operation, only the rate at which the mutation occurs changes. The mutation operation randomly changes

Require: $SIZE$ be a multiple of 2
  $WINNER = rand(A)$ {randomly select initial winner}
  for $i = 1$ to *tournament size* -1 **do**
    $PICK = rand(A)$ {randomly select a contestant}
    if $f_{PICK} > f_{WINNER}$ **then**
      $WINNER \Leftarrow PICK$ {change winner if fitness improves}
    end if
  end for

Figure 5.5: Tournament Selection

  if $rand[0, 1] \leq p_c$ **then**
    $\mathcal{X}^{point} = rand[1, N - 1]$ {randomly generate a crossover point}
  else
    $\mathcal{X}^{point} = N$ {no exchange, crossover point at end of chromosome}
  end if
  Set 1^{st} child to 1^{st} parent's genes up to crossover point, 2^{nd} parent thereafter
  Set 2^{nd} child to 2^{nd} parent's genes up to crossover point, 1^{st} parent thereafter

Figure 5.6: One Point Crossover

  if $rand[0, 1] \leq p_c$ **then**
    for $i = 1$ to N **do** {consider exchanging genetic material at each gene in chromosome}
      if $rand[0, 1] < 0.5$ **then**
        copy chromosome i from parent 1 to child 1
        copy chromosome i from parent 2 to child 2
      else
        copy chromosome i from parent 2 to child 1
        copy chromosome i from parent 1 to child 2
      end if
    end for
  else
    copy entire parent chromosomes to children if no crossover
  end if

Figure 5.7: Uniform Crossover

$P(1) = \frac{f_1}{f_1+f_2}$ {probability of selecting parent 1's genes during crossover}
for $i = 0$ to N **do**
  if $rand[0,1] \leq P(1)$ **then**
    copy chromosome i from parent 1 to child
  else
    copy chromosome i from parent 2 to child
  end if
end for

Figure 5.8: Fusion Crossover

the CMS location to any link in the network, exclusive of those links already included in the chromosome. The simple algorithm with a static p_m is described in Figure 5.9. Details of non-static mutation rates are provided in Equations 5.3, 5.4, and 5.5.

for all $l \in L$ **do** {iterate through each gene in chromosome}
  if $rand[0,1] \leq p_m$ **then** {if mutation}
    repeat
      $l = rand(A)$ {randomly mutate chromosome l to another CMS location}
    until $l \notin L$ {iterate to remove duplicate locations}
  end if
end for

Figure 5.9: Simple Mutation

Self-Adaptive Mutation

The philosophy behind the self-adaptive mutation strategy is to relate a variant mutation rate to each individual in the population. The mutation rate changes or "learns" (Equation 5.3) over successive generations with the expectation of estimating superior mutation rates and not just superior solutions.

$$p'_m = \left(1 + \frac{1 - p_m}{p_m} e^{(-LR)\,N(0,1)}\right)^{-1} \tag{5.3}$$

where

LR = learning rate (usually LR = 0.22);

$N(0,1)$ = normal distribution (mean = 0, standard deviation = 1).

Deterministic Mutation Rate Changes

Some researchers argue that utilizing a high rate of mutation through part of the GA process will introduce more genetic diversity into the population and, subsequently, increase the quality of the generated solutions. To this end two changing mutation strategies have been described in the literature that increase, or decrease, in a predictable way the population mutation rate. The deterministically increasing mutation (Equation 5.4) starts at a rate of (*equivalent binary chromosome length*)$^{-1}$ and increases mutation up to a rate of 0.5. Alternatively, the decreasing rate (Equation 5.5) starts at 0.5 and goes down to (*equivalent binary chromosome length*)$^{-1}$ upon reaching the maximal number of generations.

$$p'_m = \left(2 + \frac{n-2}{T-1}(T - 1 - t)\right)^{-1} \tag{5.4}$$

$$p'_m = \left(2 + \frac{n-2}{T-1} t\right)^{-1} \tag{5.5}$$

where

n = equivalent binary chromosome length;

T = maximum number of generations;

t = current generation $(0 \ldots T-1)$.

5.4 OptimalCMS: A Decision Support Tool For Locating CMS

The methodologies outlined in this and previous chapters are calculation intensive and could not be performed, on a realistic traffic network, without the aid of a software program. To this end a software program, OptimalCMS, was developed to implement the delay benefit and optimization algorithms for locating CMS. OptimalCMS was written using the $C^{\#}$.NET programming language, with coding details presented in Appendix E. The data structure and logical program execution of OptimalCMS is illustrated in Figure 5.10 with further information contained in Henderson et al. (2004a).

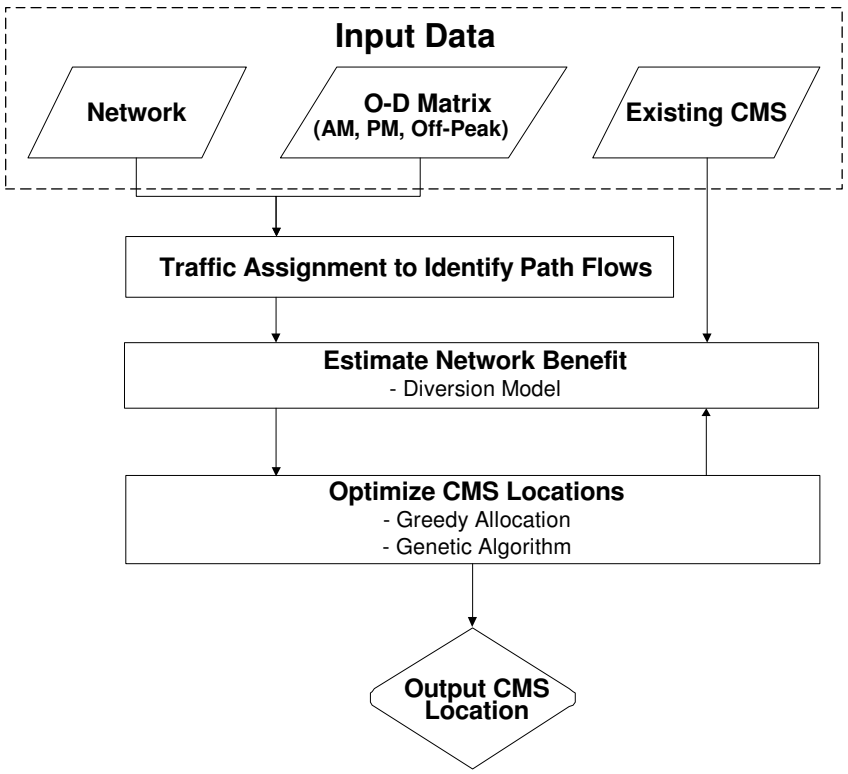


Figure 5.10: Algorithm Framework

CHAPTER 6
COMPUTATIONAL STUDY

The CMS location model presented in Chapters 3-5 is established on the basis of a number of assumptions and simplifications. For example, the proposed model assumes that the only benefit of installing CMS is in providing incident information. Also, the model is deterministic in nature, assuming perfect information on input parameters such as O-D demand, incident conditions and traffic diversion behaviour. In practice, however, most of these parameters are inherently uncertain due to various factors, such as insufficient relevant data to estimate the model parameters, errors in both the raw data and model specification, and errors in predicting future traffic demand and conditions. The objective of this chapter is therefore twofold: to demonstrate the application of the proposed model and its ability to solve realistic CMS location problems; to quantify the potential effects of these variations on CMS location.

Two datasets will be utilized in this computational study. The first dataset is relatively small, representing a hypothetical network. This dataset will be mainly applied to analyze the sensitivity of the model to variations in data and model parameters. The second dataset represents a section of the Highway 401 express-collector freeway and its surrounding arterials in the City of Toronto. This second dataset will be used to further substantiate the results obtained using the first network and to develop suitable genetic algorithm parameters. Unless otherwise stated, all optimization in this chapter is performed using the greedy allocation method.

6.1 Case A: Small Network

Figure 6.1 shows the layout of the road network used in the sensitivity analysis. The network represents a freeway-arterial system, consisting of 56 nodes and 124 links. The

longest of these links being 6 km and the shortest being 0.25 km. Also, the link free-flow speeds range from 60 km/h for the arterials and ramps to 110 km/h for the freeway links. An east-west freeway is located in the middle of the network surrounded by arterial links with a total east-west extension of 17.2 km and a north-south extension of 6.0 km.

Also shown in the network are 14 origin-destination zones as trip generators. Four time periods are considered: AM peak, midday off-peak, PM peak, and overnight off-peak with period durations of 2, 7, 3, and 12 hours respectively. Generally, a large number of trips travel eastbound (from west to east) during the AM peak period and, conversely, a large number of trips travel westbound (from east to west) during the PM peak period. The rate of AM demand is monotonically higher than the PM demand. Also, the demand matrix for modelling uncertainty in traffic demand and driver behaviour is different (larger demand) than the demand matrix for modelling uncertainty in incident attributes. The base demand matrix, from which all other analysis matrices are derived, is provided in Appendix C. Additionally, an example of the variation in demand experienced throughout 24 hours is shown in Figure 6.2.

The Bureau of Public Roads (BPR) link congestion function was chosen for traffic assignment purposes. The BPR function (Equation 6.1) is the most common travel congestion function used in transportation planning applications.

$$t = t_f \left(1 + \alpha \left(\frac{V}{C}\right)^{\beta}\right) \tag{6.1}$$

where

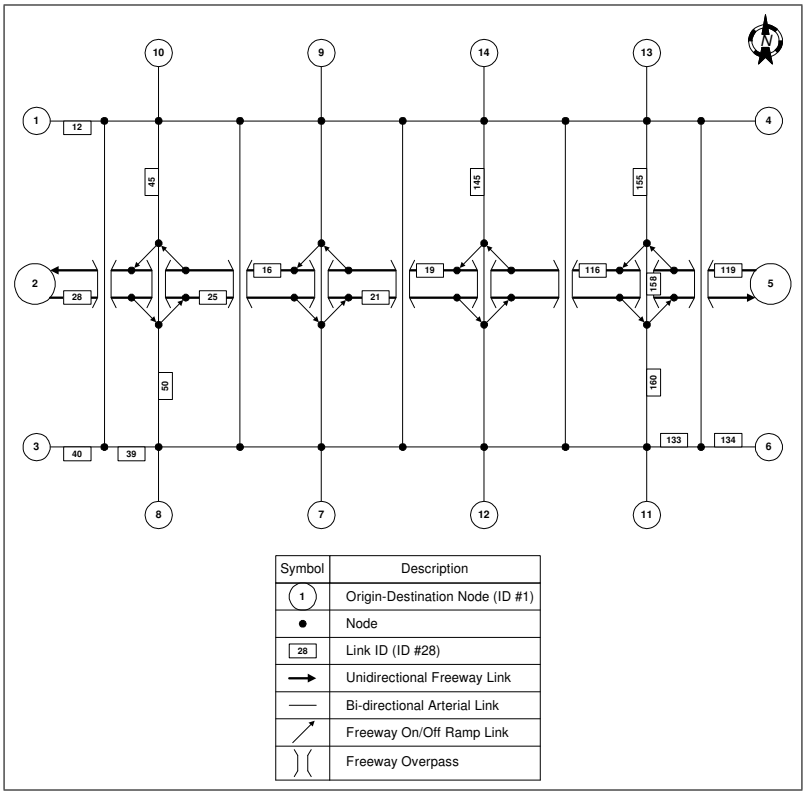


Figure 6.1: Sample Network

t = link travel time (min);

t_f = free-flow link travel time (min);

V = link volume (veh/hour);

C = link capacity (veh/hour);

α = function parameter, $\alpha = 0.15$ in this case;

β = function parameter, $\beta = 4$ in this case.

For the purpose of this analysis the base incident rate was set at 2.9 incidents per million-veh-km as suggested by a National Highway Traffic Safety Administration (2000)

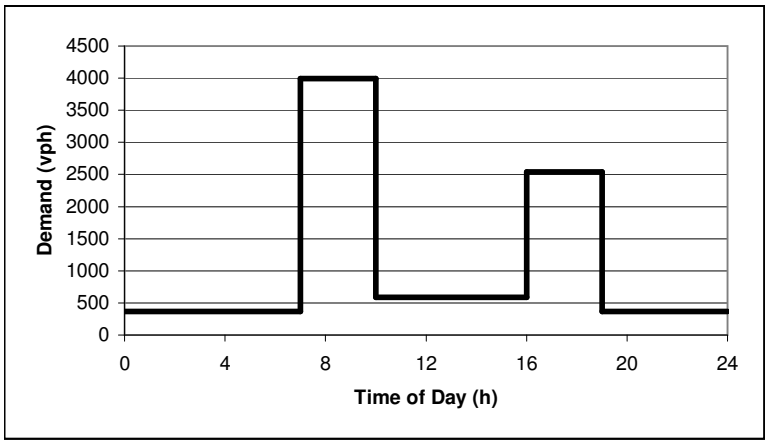


Figure 6.2: Time Varying Demand Example, Origin 2 to Destination 5

report. Equation 3.10 was used to convert this value, r_a, to incidents over a given time period, n_a^p. Other base values used for this sample network and applied to all incident links are: capacity reduction of 80%, incident duration of 30 min, detection time of 10 min, and processing and CMS activating time of 5 min. These values are similar to incident characteristics used in other relevant literature (Abbas and McCoy, 1999; Chiu et al., 2001).

6.1.1 Optimal CMS Locations and Marginal Benefits

The OptimalCMS software program was run using the data and parameters described in Section 6.1. The four best CMS locations are shown in Figure 6.3. The first two CMS are located at either end of the freeway, which is reasonable considering the high traffic volume on the freeway and the opportunity to divert at the most easterly and westerly interchanges. The second two CMS, located upstream of the next interchanges, also capture high volumes from zones 1 through 6 and provide a good opportunity for diversion.

Figure 6.4 shows the relationship between the total benefits of CMS and the total

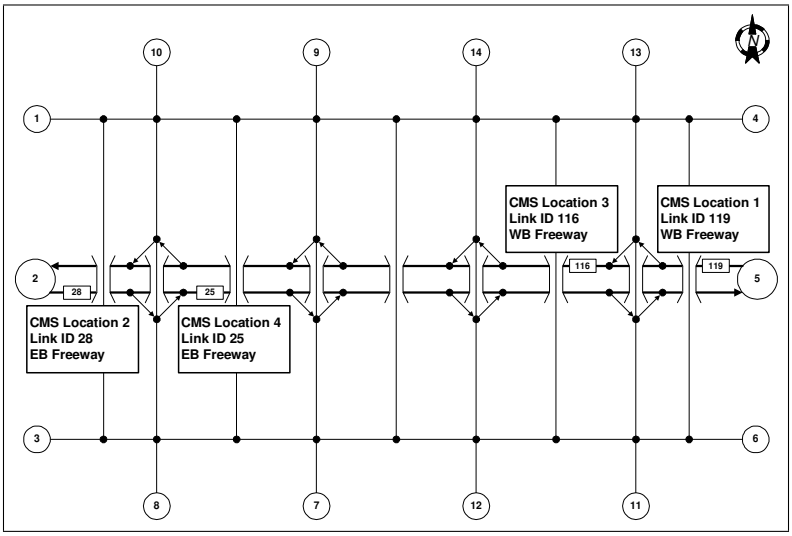


Figure 6.3: First Four Selected CMS Locations

number of CMS installed in the network. As seen in the figure, a significant increase in total network benefit resulted when the first 3 CMS were added to the network (an increased travel time saving of about 62% from 1 CMS to 2 CMS). The additional total network benefits from adding more CMS become less significant after the third CMS is added. The total travel time benefit tends to level off between 3 and 4 CMS. These results suggest the existence of optimal number of CMS for a given network, which could be obtained through a systematic cost-benefit analysis with the support of the proposed location model.

Also seen in Figure 6.4, the benefit of adding ten CMS to the freeway network is 3123 veh-hours/day. This benefit is the difference between the incident-induced delay without CMS information (22899 veh-hours/day) and the incident-induced delay in the presence of CMS information (19776 veh-hours/day).

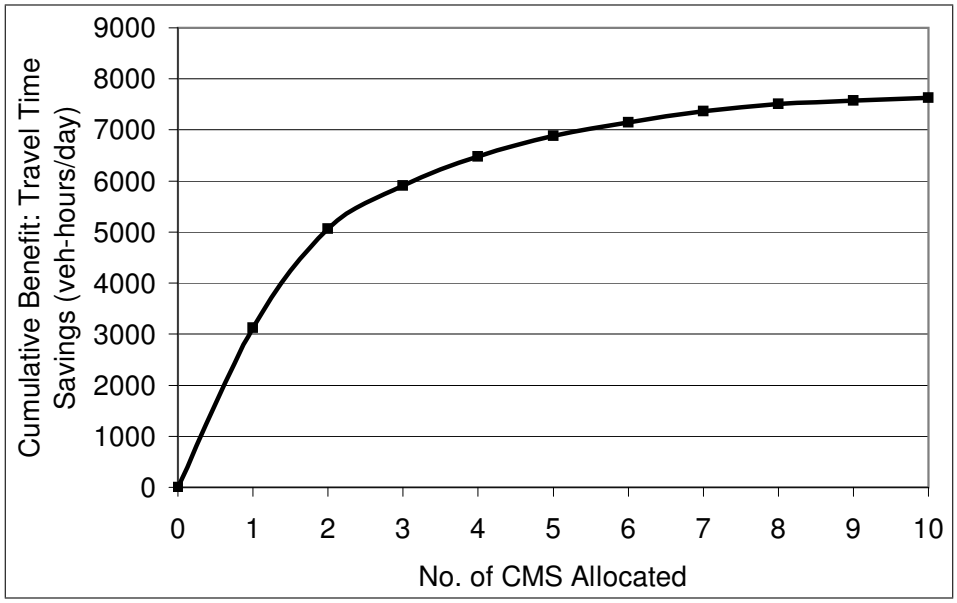


Figure 6.4: Benefit vs. Number of CMS Installed

6.1.2 Comparison to Heuristic Methodology

In the absence of a methodical approach for optimizing CMS locations, traffic managers usually locate CMS in a heuristic ad-hoc way based on freeway link traffic volume and diversion opportunities at downstream off-ramps. To illustrate the difference between the proposed model and this ad-hoc approach, we consider the greedy allocation results for the base case of the hypothetical sample network. Shown in Table 6.1 is the order of CMS allocation based on the proposed model, and the revised order based on link traffic volume. It is reasonable to assume that there is a good opportunity for traffic to divert to an alternative route from each of these links since they were all selected in the greedy allocation process.

As seen in Table 6.1, there is a significant difference between CMS allocation based on

Table 6.1: CMS Allocation Order, Proposed Model vs. Link Volume

Allocated CMS at Greedy Iteration #	*Link ID*	*Daily Volume (veh)*	*CMS Order (Volume)*
1	119	22500	6
2	28	21295	7
3	116	53310	1
4	25	47930	2
5	155	20345	8
6	45	17980	9
7	50	14350	10
8	158	24105	5
9	16	47790	3
10	21	46750	4

traffic volume and allocation based on the proposed model. Using the proposed model, freeway links with highest volume are not necessarily the first assigned with a CMS. A visual comparison of the two techniques may also be considered using Figure 6.3 for the proposed model and Figure 6.5 for the heuristic technique. As seen in the Figures, the two CMS at the end of the freeway are replaced with two CMS in the middle of the freeway with higher traffic volumes but inferior diversion opportunities.

6.1.3 Consideration of Multiple Time Periods

Most existing models for optimizing CMS locations consider only a single time period, AM or PM peak, and ignore the traffic exposure for the remainder of the day. The proposed model, however, is able to consider the entire day in the optimization process. The CMS allocation results based on a single time period compared to the results for all time periods are shown in Table 6.2.

It is evident that only considering a single time period in the optimization process does

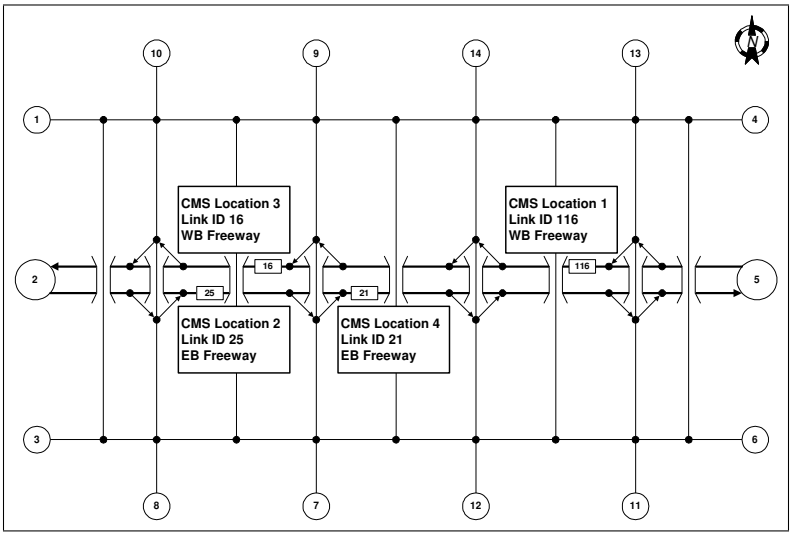


Figure 6.5: First Four Selected CMS Locations by Heuristic Methodology

not produce the best CMS locations since demand is directional and the highest volume freeway links during the AM peak period are not the highest volume freeway links during the PM peak period. Also, incidents during the off-peak period had a negligible impact on the final location solutions of the optimization process, as the estimated benefit was several orders of magnitude smaller than the benefit predicted during the peak periods.

6.1.4 Effects of Error in Traffic Demand Estimates

The location benefit model proposed is this research assumes O-D traffic demand as deterministic and known. The actual demand varies stochastically and can not be predicted accurately. This section attempts to quantify the effects of error in demand estimates on CMS location solutions. The uncertainty in traffic demand was modelled by considering several variations of the original O-D matrix. Variations of $+10\%$, -10%, $\pm 5\%$, $\pm 10\%$, and $\pm 20\%$ to the base origin-destination demand matrix were evaluated. The variation

Table 6.2: Greedy Allocation Results Based On Time Period

Allocated CMS at Greedy Iteration #	*All Periods*		*AM Peak*		*Midday*		*PM Peak*		*Overnight*	
	Link ID	*MS* [1]	*Link ID*	*MS* [1]	*Link ID*	*MS* [1]	*Link ID*	*MS* [1]	*Link ID*	*MS* [1]
1	119	3123	28	1941	45	3	119	3120	16	0
2	28	1942	25	556	16	2	116	841	45	0
3	116	849	45	223	155	1	133	407	125	0
4	25	560	39	142	50	1	155	266	155	0
5	155	406	21	55	160	1	19	63	50	0
6	45	268	145	30	21	0	176	28	160	0
7	50	224	75	29	116	0	55	20	105	0
8	158	142	125	9	125	0	58	14	174	0
9	16	64	12	7	25	0	16	9	68	0
10	21	55	35	4	106	0	145	5	80	0

[1]Marginal savings (veh-hours/day)

in each case was developed by increasing or decreasing each entry (O-D pair) in the base O-D matrix (AM, PM, and off-peak) by a certain amount. For example, for the case of +10% variation, each demand entry was increased by a random amount from 0% to 10% inclusive. The -10% variation in demand was determined in a similar manner. For the ±5% variations in demand, each demand entry was either increased or decreased, depending on the random value selector from -5% to +5% for each particular entry. The same stochastic changes are also applied to the ±10% and ±20% variation cases.

The results of the greedy allocation procedure for each of the demand variations are shown in Table 6.3. As seen in the Table, the error effect is relatively small since there is no effect to the first five CMS locations chosen or their respective order allocated. The order for some CMS locations are altered between the sixth and tenth CMS allocations, however only one further link (Link ID 50) is introduced into the solution sets.

Table 6.3: CMS Allocation Results for Variable Congestion Levels

Greedy Iteration #	*Base Case*		+10%		−10%		−5% → +5%		−10% → +10%		−20% → +20%	
	Link ID	*MS* [1]	*Link ID*	*MS* [1]	*Link ID*	*MS* [1]	*Link ID*	*MS* [1]	*Link ID*	*MS* [1]	*Link ID*	*MS* [1]
1	119	3123	119	3489	119	2997	119	3254	119	3628	119	2448
2	28	1942	28	2087	28	1675	28	1921	28	1918	28	2123
3	116	849	116	933	116	812	116	865	116	891	116	772
4	25	560	25	601	25	508	25	562	25	563	25	575
5	133	406	133	449	133	372	133	422	133	415	133	331
6	155	268	155	308	155	242	155	279	155	312	*45*	*248*
7	45	224	45	240	45	197	45	221	45	223	*155*	*222*
8	39	142	*50*	*144*	*50*	*134*	39	147	39	144	*50*	*129*
9	19	64	*21*	*72*	19	57	19	58	19	63	*21*	*70*
10	21	55	*19*	*66*	21	51	21	55	21	55	*19*	*63*

[1]Marginal savings (veh-hours/day)

CMS location identified as different from the base case

6.1.5 Effects of Uncertainty in Incident Conditions

Similar to O-D demand estimates, the proposed model also assumes deterministic and known incident attributes. Four incident attributes, including incident rate, incident duration, incident occurrence time, and capacity reduction, were modelled for uncertainty. Since the incident occurrence time, as mentioned earlier, is assumed to be distributed uniformly over the time period of concern it is not considered hereafter. The uncertainty in incident rate was modelled by considering random fluctuations in the link exposure-based incident rate. Fluctuations of +10%, - 10%, +50%, -50%, +100%, and -100% were considered and developed as previously described for other variations. The CMS location results for variations in link incident rates are shown in Table 6.4.

The optimal CMS locations are insensitive to smaller variations of 10% in the link inci-

Table 6.4: CMS Allocation Results for Variable Link Incident Rates

Greedy Iteration #	*Base Case*		+10%		+50%		−50%		+100%		−100%	
	Link ID	*MS* [1]	*Link ID*	*MS* [1]	*Link ID*	*MS* [1]	*Link ID*	*MS* [1]	*Link ID*	*MS* [1]	*Link ID*	*MS* [1]
1	119	559	119	545	119	735	119	549	119	605	119	283
2	28	417	28	444	28	375	28	381	28	461	28	218
3	116	138	116	134	116	173	116	114	116	160	*155*	*56*
4	25	103	25	107	25	94	25	83	25	116	*45*	*41*
5	155	49	155	48	155	65	155	62	*45*	*50*	*116*	*33*
6	45	43	45	46	45	39	45	44	*155*	*44*	*50*	*29*
7	50	25	50	27	*158*	*33*	*158*	*31*	50	26	*158*	*19*
8	158	24	158	24	*50*	*22*	*50*	*26*	*160*	*22*	*21*	*17*
9	16	15	*21*	*15*	*19*	*18*	16	13	16	22	*25*	*4*
10	21	15	*16*	*15*	21	16	21	12	21	17	*19*	*3*

[1]Marginal savings (veh-hours/day)

CMS location identified as different from the base case

dent rate, however, larger variations (±100%) seem to affect the allocated CMS locations. The reason for the effect on CMS locations is the linear relationship between incident rates and travel time savings.

An 80% reduction in link capacity was used for the base case. Four additional cases were generated: the first two cases involved monotonic reductions of 40% and 60% for all links; the other two cases had variations similar to the random fluctuations in congestion level. Random decreases in capacity with equal likelihood of a decrease from 50% to 70% inclusive for one case, and 40% to 80% for the other case were generated independently for each of the links. The CMS location results for capacity reduction variations were identical for the first six CMS allocated and only the order of allocation for the last four varied from the base case. This indicates that the optimal CMS location is insensitive to

uniform changes in capacity reduction and relatively insensitive to random fluctuations in capacity reduction. Again, a slight variation in CMS allocation order was observed but only for a few locations that produced a small marginal benefit.

Variations in incident duration were evaluated using 30 minutes for the base incident duration case, and uniform durations of 40 minutes and 50 minutes for additional uniform incident duration cases. Random fluctuations, similar to those discussed for capacity reduction, were modelled using two additional random duration cases with equal likelihood for durations between 30 minutes and 50 minutes inclusive for the first random case and between 20 minutes and 60 minutes inclusive for the second random case. The results of the allocation procedure, similar to those of variations in link capacity reduction, show identical CMS locations for the first six allocated and differing order for the last four allocated. Therefore, with the exception of a few locations with marginal benefit, the allocated CMS location is insensitive to both monotonic and random variations in incident duration.

6.2 Case B: Toronto Network

The Toronto case will now be considered for additional analysis. The complete dataset (Figure 6.6), provided by the Ministry of Transportation of Ontario, consists of 14160 nodes, 37386 links, and 69448 O-D pairs extracted from the emme/2 transportation planning software. This study area is prohibitively large for model execution, therefore, a smaller network (Figure 6.7) mainly comprised of Highway 401 and its nearby arterials was extracted for computational analysis. This reduced the number of nodes, links, and O-D pairs to 961, 2363, and 6149 respectively. The longest of these links being 3.45 km and the shortest being 0.04 km. Also, the link free-flow speeds range from 40 $\rightarrow$ 70 km/h for the arterials and ramps to 110 km/h for the freeway links. The smaller study area is approximately 23.2 km in the east-west direction, from west of Highway 410 to east of

Allen Road, and 17.4 km from the most southerly node to the most northerly node.

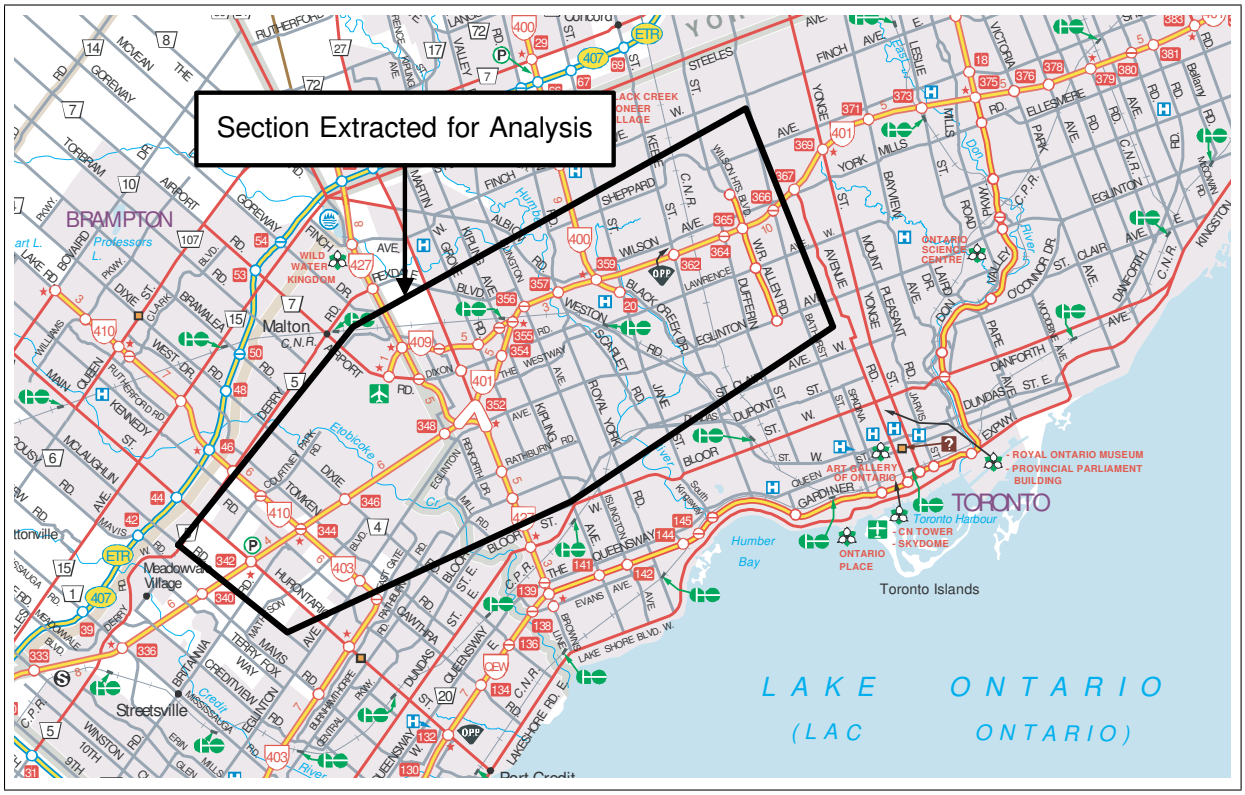


Figure 6.6: Toronto Street Network (Source: MTO Official Road Map)

The O-D demand for the smaller network (Appendix D) was not provided, only a sample of the larger demand matrix for the entire Greater Toronto Area was included. The smaller demand matrix was determined by performing path-based assignment for the complete dataset and saving the path information for each O-D pair. The paths, referenced as a succession of nodes, were truncated at the edges of the study area. The truncation was performed by eliminating all nodes up to the first node that was part of the smaller network, this node being the origin node, then continuing until the last node that appears

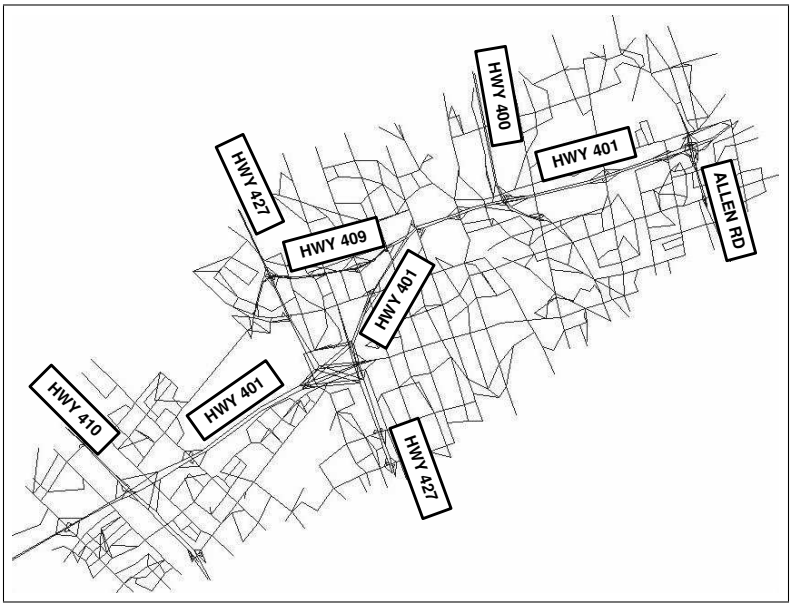


Figure 6.7: Highway 401 Network

in the network is found, this node being the complementary destination node. The demand for this new O-D pair was then set to the associated path flow. Demands with common origins and destinations were combined to reduce the size of the derived demand matrix. The peak hour demand (AM) for this matrix is approximately 152,000 vehicles and the associated daily demand is approximately 948,500 vehicles.

To increase the realism of these results, all calculations include the negative impact of diverted traffic on alternative routes in the travel time savings benefit results. However, no diversion rate iterations are performed as described in Section 4.5, except those described in Section 6.2.4.

6.2.1 Optimal CMS Locations and Marginal Benefits

Figure 6.8 illustrates the ten best CMS locations as chosen by the proposed model. Generally, most of the CMS have been allocated to freeway links upstream of an interchange. This can be expected from a good CMS location model since interchanges represent an excellent diversion opportunity. The first five of these locations are described below.

1. Link 7628 is located at the easterly end of westbound Highway 401. This location captures most of the westbound freeway traffic while providing a diversion opportunity to Allen Road.

2. Link 9980 is located on westbound Highway 401 just after the on-ramp from southbound Highway 400. Highway 400 as well as Highway 401 traffic pass this point and can be informed of conditions on the Highway 401 main route and Highway 409 $\rightarrow$ Highway 427 alternative route.

3. Link 7547 is located on the eastbound Highway 401 after the Highway 427 interchange. The Highway 401 and 427 traffic may divert at a minor interchange downstream of the CMS link.

4. Link 9986, located on eastbound Highway 401 at the Highway 400 interchange, has a high traffic volume but poorer diversion opportunities than the first three CMS allocated.

5. Link 29486, located at the beginning of the eastbound Highway 409, provides two excellent diversion opportunities through eastbound Highway 409 or southbound Highway 427. However, the traffic volume is much lower than the first four locations that CMS were located.

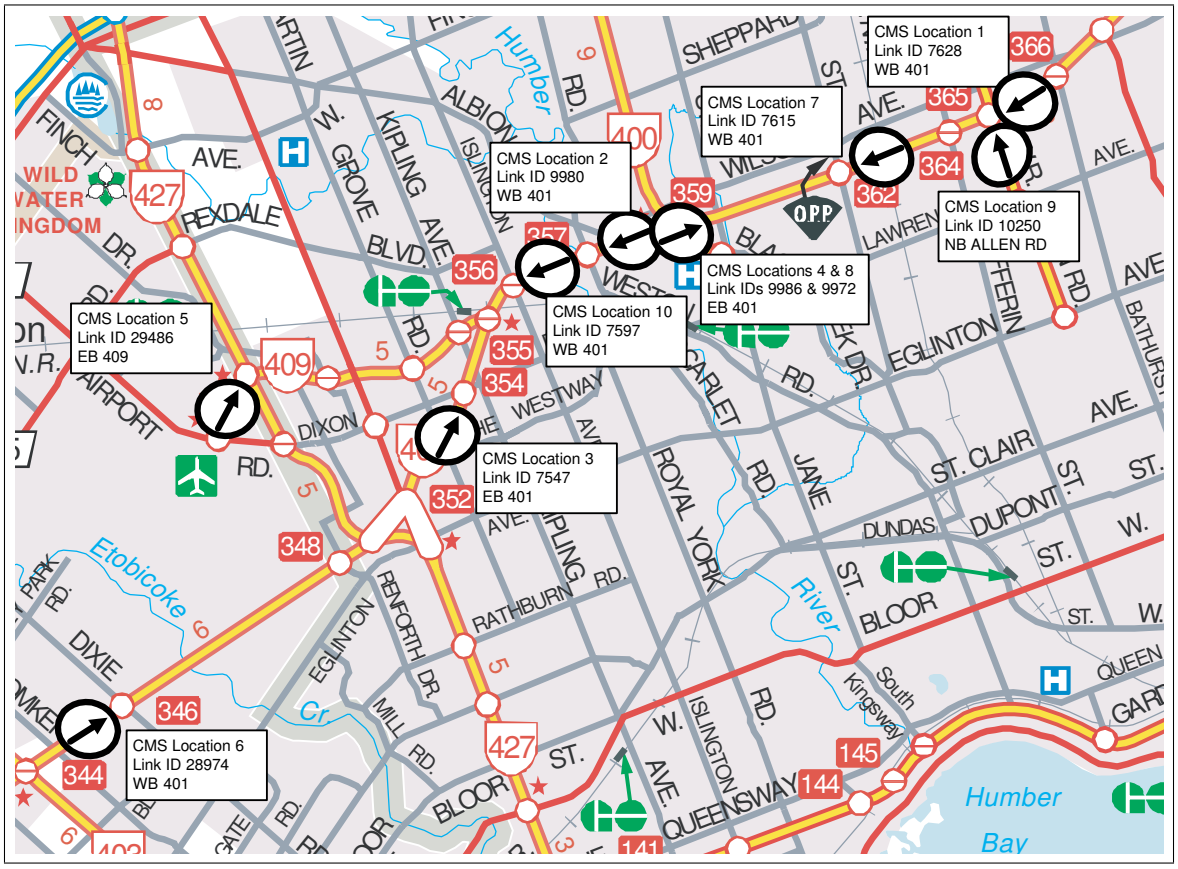


Figure 6.8: Model Selected Optimal CMS Locations

Similar to Figure 6.4 for the small network, Figure 6.9 shows the relationship between the total benefits of CMS and the total number of CMS installed in the GTA network for various diversion model parameters. The general trend is the same with a very high benefit for the first few CMS installed, followed by decreasingly marginal gains. Between the ninth and tenth CMS installation there is practically no further network benefit. The allocation of ten CMS would be excessive when considering the size of this network. A more reasonable number would be five CMS when considering the relatively small positive impact of CMS 6 $\rightarrow$ 10 and the limited opportunities to divert through interchanges.

The benefit of adding CMS to Highway 401 is approximately 1.52 million veh-hours/year when considering an α and β of 5. This benefit is the difference between the incident-induced delay without CMS information (4.44 million veh-hours/year) and the incident-induced delay in the presence of CMS information (2.92 million veh-hours/year).

The network benefit may be converted to a dollar value by assuming a value of time, e.g. \$10/hour. Utilizing this value of time the benefit attained by installing one CMS, approximately 640,000 $\rightarrow$ 1,010,000 veh-hours, may be converted to a range of \$6.4 $\rightarrow$ \$10.1 million. After the installation of the second CMS the benefit increases to \$8.5 $\rightarrow$ \$11.9 million. Again, a cost-benefit analysis would be complementary in determining the optimal number of CMS to install.

6.2.2 Comparison to Existing CMS

There are 21 existing CMS in the study area under consideration (Figure 6.10). A comparison was made between these existing locations and the model selected optimal locations. The most significant difference between the two sets of CMS locations is that the model allocated CMS upstream of interchanges while the existing CMS locations are mainly located upstream of the ramps connecting the express and collector routes. This is not unexpected

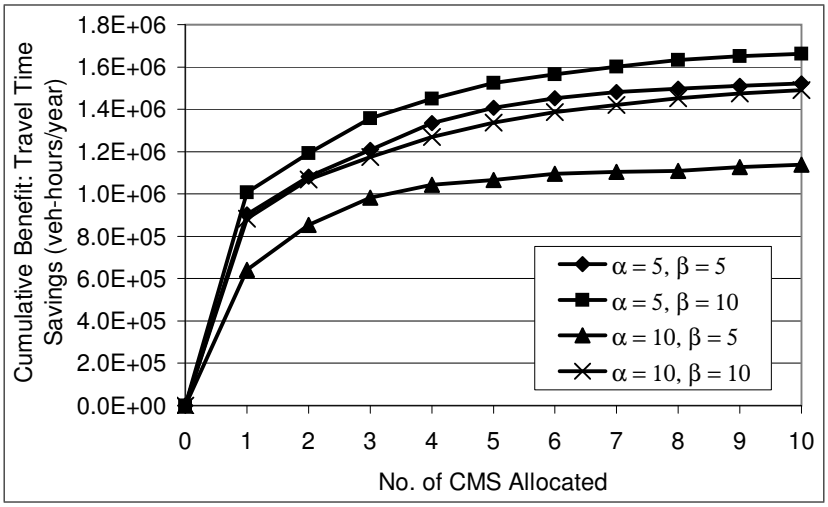


Figure 6.9: Benefit vs. Number of CMS Installed

when considering that the CMS in this section of Highway 401 are used to manage traffic between the express and collector routes and not to provide information relating to alternative routes off of the freeway system. Also, the network representation provided for this study did not distinguish between the express and collector systems, instead combining these capacities into a single link instead of parallel links for each of the express and collector options. This simplification impacts chosen CMS locations since diversion from express to collector and *vice versa* could not be considered in the model without a more detailed network.

Another major difference between the model results and the existing condition is the number of CMS installed. Only the first 5 CMS installed had any significantly positive effect according to the model, a substantially smaller number than the actual 21 installations. Additionally, these 21 existing locations did not achieve the model optimal travel time savings of approximately 1.52 million vehicle-hours and instead produced a slightly smaller value of 1.38 million vehicle-hours. However, the 21 installed CMS may not be

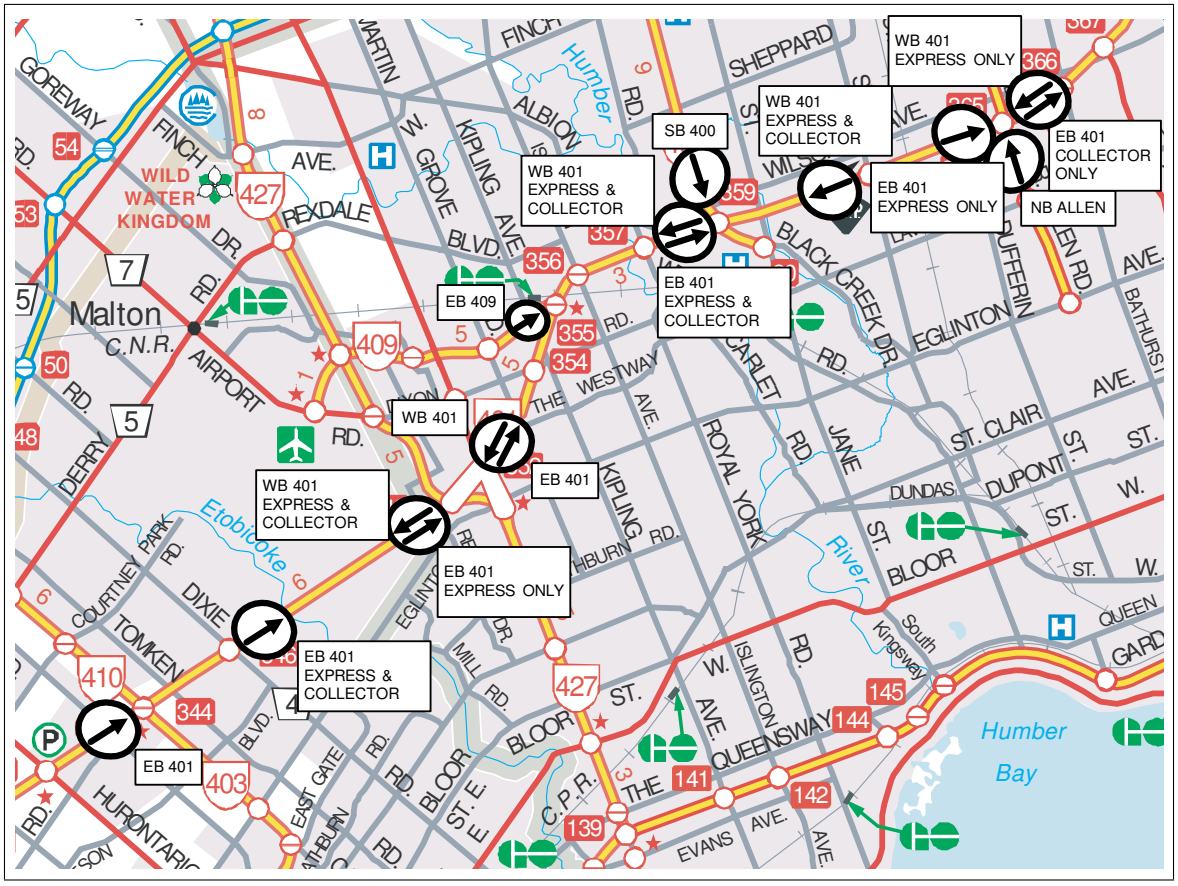


Figure 6.10: Existing CMS Locations

excessive when considering a positive repetition effect of passing multiple CMS which was not included in the proposed model.

6.2.3 Uncertainty in Diversion Model

The diversion model contains a high degree of uncertainty. To identify the impact of the diversion model on CMS allocation, CMS were allocated on the basis of different α and β values. Recall from Equation 4.1 that increasing α will monotonically decrease the diversion rate for all paths and increasing β will increase the diversion rate, depending on the travel time for the shortest alternative route. Therefore, a comparison of the marginal benefits of each allocation procedure is not relevant. Instead the focus will be on a comparison of the chosen locations (Table 6.5).

Table 6.5: CMS Allocation Order Based On Diversion Model Parameters

Allocated CMS at Greedy Iteration #	$\alpha = 5$ $\beta = 5$ *Link ID*	$\alpha = 5$ $\beta = 10$ *Link ID*	$\alpha = 10$ $\beta = 5$ *Link ID*	$\alpha = 10$ $\beta = 10$ *Link ID*
1	7628	7628	7628	7628
2	9980	7547	*29007*	9980
3	7547	9980	*7593*	7547
4	9986	9986	*7607*	9986
5	29486	*29469*	*9969*	29486
6	28974	28974	7615	28974
7	7615	10250	10250	*9774*
8	9972	*7559*	*10089*	*10421*
9	10250	7615	*28039*	*7575*
10	7597	7597	*25008*	*9966*

As seen in Table 6.5, the model selected CMS locations are highly dependent on the α and β parameters of Equation 4.1. Generally, the absolute values of α and β are not

as important as the ratio of the two parameters. Relatively higher values of α produce CMS locations with higher traffic volumes but, possibly, lower savings on alternate routes. Alternatively, a higher β parameter results in CMS locations where the value of travel time savings is more important than link traffic volumes. The relative values of these two parameters should, therefore, be a result of an accurate utility model for route preference.

6.2.4 Inclusion of Alternative Path Travel Time

As indicated in Section 4.5, the increased delay incurred on alternative routes may be a factor in CMS location decisions. Therefore, a comparison was made between chosen CMS locations with and without the inclusion of the negative impact diversion has on alternative route travel times. Recall that both methods include the travel times on alternate routes, but the consideration of alternate path travel times are used along with diverted traffic information in determining diversion rates. As seen in Figure 6.11 the inclusion of alternate path travel time slightly increases the travel time benefit. This can be expected since the iterative approach to determining diversion rates would prevent excessive re-routing of traffic to congested alternate paths, however the change is not enough to significantly alter the solution benefit, especially after the fifth allocation.

The locations of these two approaches were also compared and the results are presented in Table 6.6. The results with and without the consideration of alternate path travel time are quite different as seen in the Table. Generally, after an examination of network characteristics, the available diversion paths are slightly less congested when including the effects of alternate path travel time. This result is reasonable when considering that the optimal solution may not be unique. That is, there may be many different combinations of solutions that would produce the same objective function value. So while the solutions may be different, the actual end result in terms of travel time savings is the same.

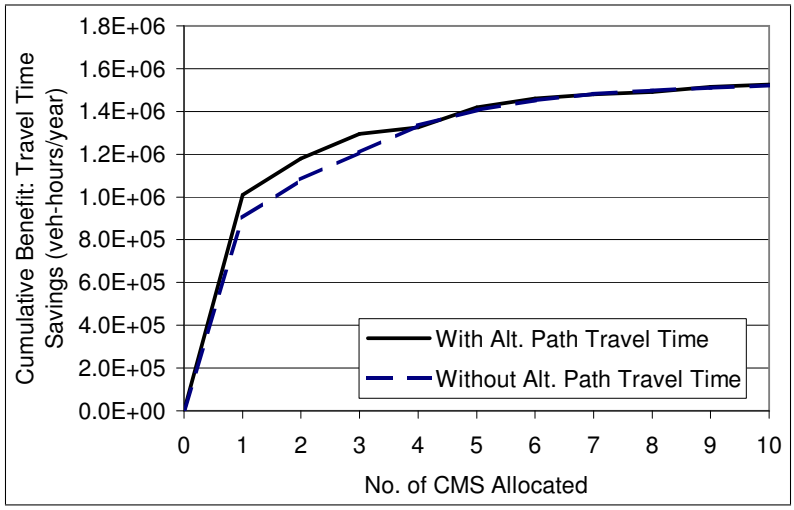


Figure 6.11: Cumulative Benefit Comparison: With and Without Consideration of Alternate Path Travel Time

6.3 Greedy vs. Genetic Optimization

All of the computation analysis performed in this Section use the Greedy optimization method. However, this method is not guaranteed to produce the optimal CMS locations. Therefore, a comparison was made between results obtained using both the Greedy and Genetic optimization techniques for both the hypothetical network and the Toronto network.

The computational time required for GA can be much greater than the greedy method. Using a computer with a 2.0 GHz processor and 512MB of RAM, the benefit evaluations for Cases A and B required 10.32 and 590.56 seconds of CPU time respectively. These times are not significant, however the greedy and genetic algorithms require many iterations of benefit evaluations. Applying the greedy algorithm to the Toronto Highway 401 network results in approximately 1500 iterations, but each iteration does not require the same

Table 6.6: Optimal CMS Locations: With and Without Consideration of Alternate Path Travel Time

Allocated CMS at Greedy Iteration #	*Without Alt. Path Time* *Link ID*	*With Alt. Path Time* *Link ID*
1	7628	7628
2	9980	7547
3	7547	9980
4	9986	7615
5	29486	9986
6	28974	*28039*
7	7615	29486
8	9972	*25008*
9	10250	*7559*
10	7597	9972

CMS location not included in base solution.

computational effort due to code optimization. The CPU time of GA are highly dependant on population size and number of generations. For example, a population size of 100 with 100 generations requires 10,000 iterations of benefit calculations.

6.3.1 GA Parameters

As indicated in Section 5.3 there are many parameters required to specify how the GA will execute. The parameters chosen for the analysis of both Case A and Case B are shown in Table 6.7. Each of the parameter values was determined by the general rules-of-thumb outlined in Appendix A. The Case A network contains 124 candidate locations, which is approximately equivalent to a chromosome length of 100. Using this value both the population size (100) and mutation rate ($\frac{1}{100} = 0.01$) were determined. Similarly, Case B with 157 candidate locations was assigned a population size of 150 and a mutation rate of

0.0067. Other parameter values were set based on recommendations of relevant literature in location theory.

Table 6.7: Genetic Algorithm Parameters used in Computational Analysis

GA Parameter	*Case A Value*	*Case B Value*
Population Size	100	150
Maximum Number of Generations	Determined during algorithm execution	
Crossover Rate	0.8	
Mutation Rate	0.01 ($\frac{1}{100}$)	0.0067 ($\frac{1}{150}$)
Elitism	*ON*	
Selection Method	Binary Tournament	
Crossover Method	Fusion Operator	
Mutation Method	Simple	

6.3.2 Results

The Case A network was run five times using the previously described GA parameters for 500 generations. One of the GA runs is shown in Figure 6.12 with corresponding maximum, average, and minimum statistics for each generation. All of the GA runs produced superior results to the greedy method, however the difference was marginal. In the case illustrated in Figure 6.12 the best GA result (3181 veh-hours/year) does not significantly improve on the greedy result (3123 veh-hours/year).

The larger Case B was run for 100 generations but the results did not exceed the greedy results. A better solution than the greedy result may exist, but the solution space explored, which depends on the population size and number of generations considered, was not large enough to explore sufficient location schemes and find a superior solution. Therefore, the application of GA to larger networks is inconclusive at this point without further study and longer CPU run times.

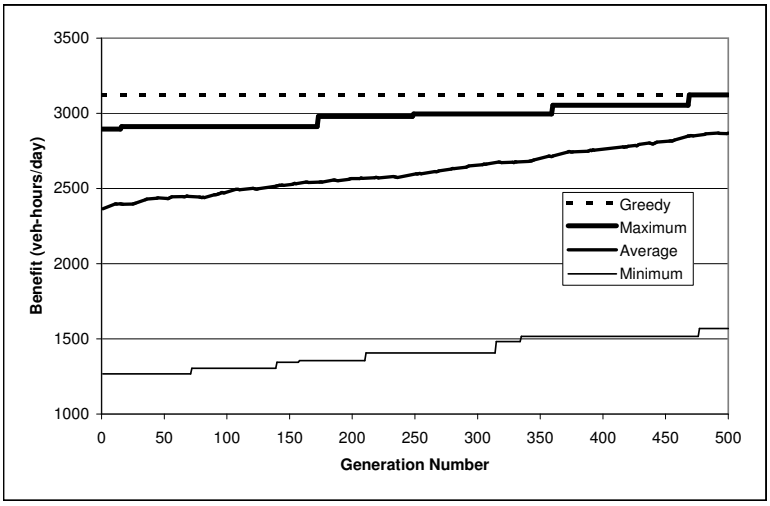


Figure 6.12: Case A Genetic Algorithm Results

CHAPTER 7
CONCLUSIONS AND FUTURE RESEARCH

Changeable Message Signs (CMS) are becoming an important component of ITS applications such as Advanced Traffic Management and Traveler Information Systems (ATMS/ATIS). By providing travelers with accurate, timely and reliable traffic information, safety and efficiency of the road network can be improved. The effectiveness of CMS, however, depend on how many CMS are installed and where the CMS are located. This thesis presents an optimization model that can be used to systematically locate CMS. This chapter summarizes the major contributions and findings of this research and suggests future research in the CMS location area.

7.1 Contributions

The following contributions have been made in this study:

- A multi-period, user-equilibrium traffic assignment procedure was implemented to estimate traffic volumes on individual links and path flows between individual O-D pairs. This improves existing methods that only consider a single time period, the peak period. The use of a user-equilibrium path-based assignment procedure allowed for diversion calculations on a complete traffic network as compared to a simplified linear freeway network model.

- A dynamic diversion model was proposed that relates the probability for a vehicle to divert from an incident path to an alternative route with the potential for travel time savings. This model is based on utility theory, well accepted in the transportation research community, and is considered superior to static diversion rates or choices based on a delay upper-limit tolerance (threshold).

- A time-dependent queuing model is developed to estimate delays with and without the presence of CMS information. This model expands on current methods by synchronizing the arrival curve with the dynamic diversion model to continually update queue length and expected delay. Additionally, the drop-off in arrivals at the end of peak periods is introduced as a consideration.

- A hybrid genetic algorithm was implemented to identify the best locations for future CMS installations. Appropriate parameters are developed based on extensive computational analysis of real-world data.

- A computational study was conducted on a hypothetical as well as a real traffic network. The case study indicates that the assumption of deterministic input parameters is appropriate for many, but not all, input parameters. Additionally, the CMS locations chosen by the proposed model were upstream of good diversion opportunities in reasonable locations.

7.2 Findings

The computational study yielded the following findings:

- The chosen CMS locations were found to be relatively insensitive to uncertainties in traffic demand and incident conditions. Only large changes were found to impact the model selected optimal locations.

- Including the negative impact of diverting traffic on alternative route travel time did not significantly alter the CMS locations for the Toronto case that was considered.

- The allocated CMS locations were found to be highly sensitive to the diversion model

parameters indicating the importance of careful selection of these parameters, ideally based on field calibrated values.

- The cumulative benefit curves, for both cases, indicate that there exists a point where adding additional CMS would not be significantly beneficial. This point, along with a cost-benefit analysis, may be used to determine the optimal number of CMS to allocate in addition to the optimal locations.

- Links directly upstream of major interchanges were the most frequently chosen for CMS locations. This is reasonable considering the diversion opportunities.

7.3 Suggestions for Future Research

Suggestions for modifications to the proposed model are listed as follows:

- More research is needed to accurately predict utility model parameters under incident conditions with CMS information.

- The benefit of other CMS uses (e.g. environmental information) needs to be quantified before they are included in a CMS location model.

- Other traffic diversion models that take into account repetition effect of vehicles passing multiple CMS should be considered.

- Further computational analysis using GA as an optimization tool for locating CMS is required. More specifically, it is still unknown whether or not the genetic method will produce significantly superior results as compared to the greedy method.

REFERENCES

Abbas, M. and McCoy, P. (1999). Optimizing variable message sign locations on freeways using genetic algorithms. In *Transportation Research Board 78th Annual Meeting Proceedings*. Transportation Research Board.

Abdel-Aty, M. (2000). Using ordered probit modeling to study the effect of ATIS on transit ridership. *Transportation Research Part C*, 9:265–277.

Albrecht, H., Everts, K., Heusch, H., and Boesefeldt, J. (1978). Bewertung einer zentalen überwachung und steuerung des verkehrs durch verkehrsstromfuhrung mit hilfe von wechselwegweisern. *Forschung Strassenbau und Strassenverkehrstechnick*, 251.

Arnott, R., Palma, A., and Lindsey, R. (1991). Does providing information to drivers reduce traffic congestion. *Transportation Research Part A*, 25(5):309–318.

Bäck, T., Hammel, U., and Schwefel, H. (1997). Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1(1):3–17.

Bates, J. (1988). Econometric issues in stated preference analysis. *Journal of Transport Economics and Policy*, 22:59–69.

Beasley, J. (1993). Lagrangean heuristics for location problems. *European Journal of Operational Research*, 65:383–399.

Beasley, J. and Chu, P. (1996). A genetic algorithm for the set covering problem. *European Journal of Operational Research*, 94:392–404.

Bertsekas, D. (1976). On the Goldstein-Levitin-Polyak gradient projection method. *IEEE Transactions on Automatic Control*, 21(2):174–183.

Bonsall, P., Whelan, G., and Page, M. (1995). Stated preference experiments to determine the impact of information and guidance on drivers' route choice. In *European Transport Forum, Seminar E*. PTRC.

Chatterjee, K., Hounsell, N., Firmin, P., and Bonsall, P. (2002). Driver response to variable message sign information in London. *Transportation Research Part C*, 10:149–169.

Chen, A. and Lee, D. (1999). Path-based algorithms for large scale traffic equilibrium problems: A comparison between DSD and GP. In *Transportation Research Board 78th Annual Meeting Proceedings*. Transportation Research Board.

Chiu, Y., Huynh, N., and Mahmassani, H. (2001). Determining optimal locations for VMS's under stochastic incident scenarios. In *Transportation Research Board 80th Annual Meeting Proceedings*. Transportation Research Board.

Cooper, L. (1963). Location-allocation problems. *Operations Research*, 11:331–344.

Daganzo, C. (1983). Derivation of delays based on input-output analysis. *Transportation Research*, 17A(5):341–342.

Dudek, C. (1997). Changeable message signs: A synthesis of highway practice. Technical Report NCHRP Synthesis 237, Transportation Research Board.

Erera, A., Lawson, T., and Daganzo, C. (1998). A simple generalized method for analysis of a traffic queue upstream of a bottleneck. *Transportation Research Record 1646*, pages 132–140.

Fu, L., Henderson, J., and Hellinga, B. (2003). Inventory of Intelligent Transportation Systems (ITS) projects in Canada. Technical report, Intelligent Transportation Systems Society of Canada.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley.

Gordon, S. and Trombley, J. (2001). Tracking the deployment of the integrated metropolitan intelligent transportation systems infrastructure in the USA: FY 2000. Technical Report OP-O1-136, FHWA, US Department of Transportation.

Hakimi, S. (1964). Optimum distribution of switching centers in a communication network and some related graph theoretic problems. *Operations Research*, 13:462–475.

Hato, E., Taniguchi, M., and Sugie, Y. (1995). Influence of traffic information on driver's route choice. In *World Conference on Travel Research*.

Hato, E., Taniguchi, M., Sugie, Y., Kuwahara, M., and Morita, H. (1999). Incorporating an information acquisition process into a route choice model with multiple information sources. *Transportation Research Part C*, 7:109–129.

Henderson, J. and Fu, L. (2004). Genetic algorithms in transportation engineering. In *Transportation Research Board 83th Annual Meeting Proceedings*. Transportation Research Board.

Henderson, J., Fu, L., and Li, S. (2004a). OptimalCMS: A decision support system for locating Changeable Message Signs. In *8th International Conference and Exhibition on Application of Advanced Technologies in Transportation*. American Society of Civil Engineers.

Henderson, J., Fu, L., and Li, S. (2004b). A planning model for determining optimal CMS locations on a freeway-arterial network. *Submitted to Transportation Research Part B*.

Hensher, D. (1992). Integrating revealed preference and stated preference data into a jointly estimated hierarchical model. Working Paper ITS-WP-92-9, Institute of Transport Studies, Graduate School of Business, University of Sydney.

Holland, J. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

Hosage, C. and Goodchild, M. (1986). Discrete space location-allocation solutions from genetic algorithms. *Annals of Operations Research*, 6:35–46.

Jaramillo, J., Bhadury, J., and Batta, R. (2002). On the use of genetic algorithms to solve location problems. *Computers and Operations Research*, 29:761–779.

Jayakrishnan, R., Tsai, W., Prashker, J., and Rajadhyaksha, S. (1994). A faster path-based algorithm for traffic assignment. *Transportation Research Record 1443*, pages 75–83.

Kraan, M., van der Zijpp, N., Tutert, B., Vonk, T., and van Megen, D. (1999). Evaluating networkwide effects of variable message signs in the netherlands. *Transportation Research Record 1689*, pages 60–67.

Kroes, E. and Sheldon, R. (1988). Stated preference methods. *Journal of Transport Economics and Policy*, 22:11–25.

Lawson, T., Lovell, D., and Daganzo, C. (1997). Using input-output diagram to determine spatial and temporal extents of a queue upstream of a bottleneck. *Transportation Research Record 1572*, pages 140–147.

Levinson, D. and Huo, H. (2003). Effectiveness of effectiveness of VMS using empirical loop detector data. In *Transportation Research Board 2003 Annual Meeting CD-ROM*.

Lighthill, M. and Whitham, G. (1955). On kinematic waves II, a theory of traffic flow on long crowded roads. *Proceedings of the Royal Society A*, 229:317–345.

Lockheed Martin Federal Systems and Odetics Intelligent Transportation Systems (1997). ITS cost analysis. Technical report, FHWA, US Department of Transportation.

Louviere, J., Henley, D., Woodworth, G., Meyer, R., Levin, I., Stoner, J., Curry, D., and Anderson, D. (1980). Laboratory simulation versus revealed preference methods for estimating travel demand models. *Transportation Research Record 794*, pages 42–51.

Makigami, Y. and Newell, G. (1971). Three-dimensional representation of traffic flow. *Transportation Science*, 5(3):302–313.

May, A. (1990). *Traffic Flow Fundamentals*. Prentice-Hall.

Michalopoulos, P. and Pisharody, V. (1981). Derivation of delays based on improved traffic models. *Transportation Research*, 15B:299–317.

Ortuzar, J. and Willumsen, L. (1994). *Modelling Transport*. John Wiley and Sons.

Oscar Faber TPA (1992). Trans-pennine rail strategy. Technical report, GMPTE, Merseytravel, SYPTE, WYPTE, Cheshire County Council, Derbyshire County Council, Humberside County Council, Lancashire County Council and Peak Park.

Peeta, S., Ramos, J., and Pasupathy, R. (2000). Content of variable message signs and on-line driver behavior. *Transportation Research Record 1725*, pages 102–108.

Schrank, D. and Lomax, T. (2002). The 2002 urban mobility report. Technical report, Texas Transportation Institute.

Sheffi, Y. (1985). *Urban transportation networks : equilibrium analysis with mathematical programming methods*. Prentice-Hall.

Swann, J., Routledge, I., Parker, J., and Tarry, S. (1995). Results of practical applications of variable message signs (VMS): A64/A1 accident reduction scheme and forth estuary driver information and control system (FEDICS). In *Traffic Management and Road Safety. Proceedings of Seminar G held at the 23rd PTRC European Transport Forum, University of Warwick*, pages 149–167.

Tarry, S. and Graham, A. (1995). The role of evaluation in ATT development. *Traffic Engineering and Control*, 36(12):688–693.

Transportation Research Board (2000). Highway capacity manual: 2000 update. Technical Report 209, National Research Council.

Wardman, M. (1991). Stated preference methods and travel demand forecasting: an examination of the scale factor problem. *Transportation Research Part A*, 25:79–89.

Wardman, M., Bonsall, P., and Shires, J. (1998). Driver response to variable message signs: A stated preference investigation. *Transportation Research Part C*, 5(6):389–405.

Yim, Y. and Ygnace, J.-L. (1996). Link flow evaluation using loop detector data: Traveler response to variable-message signs. *Transportation Research Record 1550*, pages 58–64.

Zhao, L., Wren, A., and Kwan, R. (1995). Development of a driver duty estimator. *Journal of the Operational Research Society*, 46:1102–1123.

APPENDIX A: GENETIC ALGORITHMS AND THEIR APPLICATION TO TRANSPORTATION ENGINEERING

Optimization problems arise frequently in many transportation systems engineering processes such as planning, design, management, and operations control. Because of the complex nature of transportation systems, most of these problems are difficult to formulate and solve, characterized by complex objective functions (e.g., multiple competing objectives, multiple peaks and no closed form), large number of variables with mixed solution space (e.g., integer and continuous variables) and exponential number of potential solutions. Because of these challenges, these problems are not amenable to traditional optimization and computational techniques such as linear and non-linear programming and greedy search algorithms that often require an explicit formulation of the problem with a convex and differentiable objective function. Heuristic solution methods such as genetic algorithms (GA) have therefore attracted a great deal of attention and interest in transportation research community.

GA (Holland, 1975; Goldberg, 1989) are fundamentally an optimization technique. Different from traditional optimization techniques, GA seeks an optimal solution through a course mimicking the natural evolution process. In GA, each candidate solution is coded as a chromosome string and the search process starts from a group of these chromosomes, referred to as populations, and generates new solutions or offspring by alternating (mutation) and recombining (crossover) those chromosomes in the population through many iterative steps or generations. The algorithm is terminated after a specified number of generations or the change in the fitness of the population after several generations between successive generations becomes acceptably small. A representation of the simple genetic algorithm (SGA) is shown in Figure A.1.

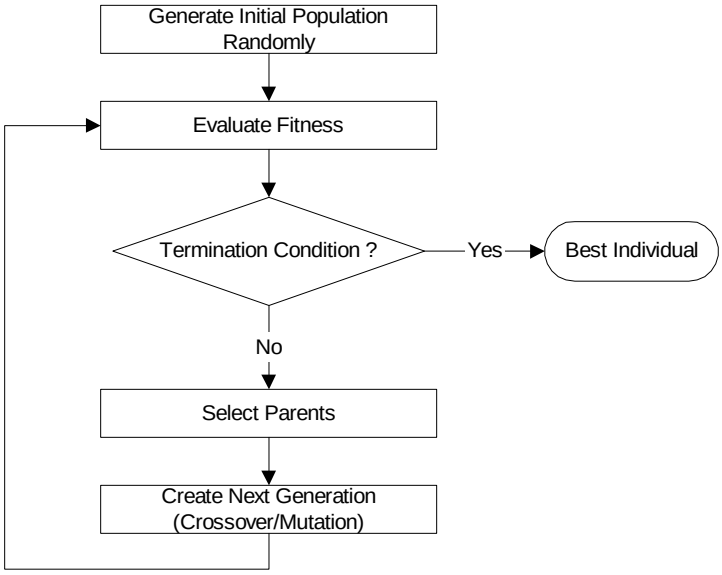


Figure A.1: Overview of Simple Genetic Algorithm

Encoding

Application of GA to a specific problem requires the development of a fitness function and representation of candidate solutions in a chromosome string. The fitness function generally involves three problem specific components: the original or variant on the problem objective function, a scaling function, and possibly a penalty function for invalid solutions or violation of constraints. Adding a scaling function to the fitness function can be useful at the initial and final stages of algorithm execution to deal with premature convergence.

Despite many possible representation schemes, binary encoding of chromosome genes (bits) is still preferred by the majority of researchers (Bäck et al., 1997). The main argument for binary encoding relates to schema theory (or solution templates) as discussed in Goldberg (1989) but beyond the intended scope. Additionally, binary encoding presents a more simplified problem representation. As shown in Figure A.2, binary encoding repre-

sents a candidate solution as a string of 0-1 bits.

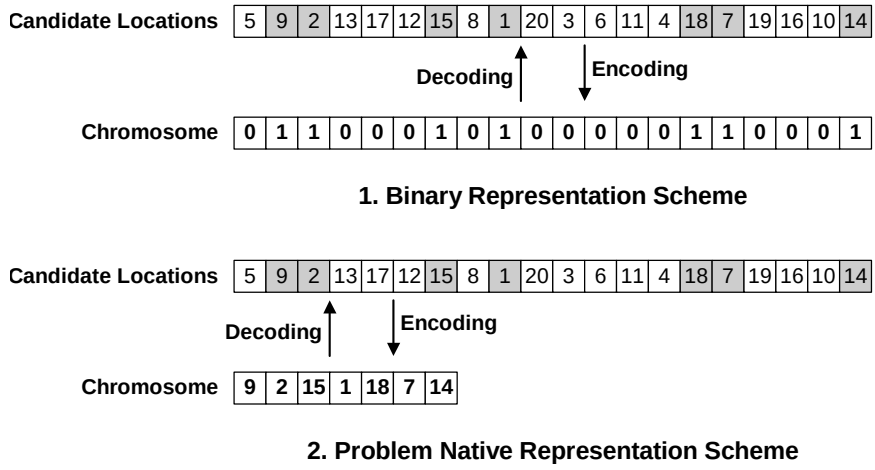


Figure A.2: GA Encoding Schemes

A string of bits may also be used to represent an integer or real-valued vector (Goldberg, 1989). This mapping is more complex, however, as details such as parameter precision and decision variable bounds must be worked out beforehand. Also, these encoding techniques may encounter problems involving loss of valuable information if crossover operations are not performed in appropriate locations between decision variables, or, mutation is applied to bits that produce large changes in parameter values. For example, if left-most bit of a 5-bit string (0→31 range) is mutated a relatively large variation of ± 16 would occur.

Incorporating native problem specific encoding to a chromosome is an alternative to binary representation schemes. Usually, using more natural representations allows constraints to be considered explicitly, which limit invalid solutions and subsequently the need for penalty functions. Figure A.2 illustrates the same solution using both binary representation and native encoding.

Population

The set of candidate solutions, retained during each iteration (generation) of the algorithm, is referred to as the population. The size of population is an important parameter in the effectiveness of the genetic algorithm. Larger populations create a more diverse gene pool and a greater likelihood of achieving the globally optimum solution, but require more computational time. Smaller populations contain a less diverse gene pool and run the risk of premature convergence. Therefore, a trade-off must be made between larger populations, with more substantial computational efforts, and smaller populations that may converge to sub-optimal solutions but require less computing time. Unfortunately there are no universal rules for determining the optimal population size for a specific problem or application. A general rule-of-thumb has however been developed that indicates a population size at least as large as the chromosome string length in an *equivalent binary* representation. Therefore, if we were using a binary chromosome length of 20 for our sample problem, we would choose a population size of at least 20.

Selection

Numerous selection schemes have been proposed for GA; however many of them can be classified as variants of fitness proportionate selection (roulette wheel selection), rank selection or tournament selection. All methods rely on the fitness of individual members of the population and explicit requirements that all fitness values are positive and larger magnitude fitness values are superior to smaller magnitude fitness values. Populations that do not meet these requirements must have their fitness values mapped. Details will not be provided here but several references are available. An example of each of the selection mechanisms is shown in Figure A.3.

Elitist selection is also often used to retain the best member in the population for sub-

			Fitness Proportionate Selection		Rank Selection	
Individual #	**Chromosome**	**Fitness (f_i)**	**Probability (p_i)**	**Cumulative Probability**	**Rank (r_i)**	**Probability (p_i)**
1	0 1 1 0 0	16	16/80 = 0.2	0.2	3	0.2
2	1 0 0 1 0	31	0.3875	0.5875	1	0.36
3	0 1 0 1 0	4	0.05	0.6375	5	0.04
4	0 1 0 0 1	9	0.1125	0.75	4	0.12
5	1 0 1 0 0	20	0.25	1.0	2	0.28
		$\sum f_i = 80$	$\sum p_i = 1.0$			$\sum p_i = 1.0$

Fitness Proportionate Selection

Step 0: Determine fitness values for population.

Step 1: Sum all fitness values. ($\sum f_i$).

Step 2: Calculate selection probability as $p_i = f_i / \sum f_i$.

Step 3: Calculate cumulative probability for all $j \leq i$ ($\sum p_j, j \leq i$).

Step 4: Generate a random number on the interval {0,1}. Select the individual with the greatest cumulative probability that does not exceed the random number generated.
Ex. random number = 0.611, Individual 3 selected.

Rank Selection

Step 0: Determine fitness values and rank individual from best to worst fitness (r_i).

Step 1: Choose a distribution from which selection probabilities will be calculated. Ex. Linear -0.08i+0.44.

Step 2: Calculate probabilities using selected distribution. Note that $\sum p_i$ must equal 1.

Step 3: Same as Steps 3&4 for Fitness Proportionate Selection.

Tournament Selection

Step 0: Randomly select a group of individuals from the population. Ex. Individuals 1, 4 and 5 in the example.

Step 1: Select the individual with the highest fitness. In the random group Individual 5 has the highest fitness (20) and would be selected.

Figure A.3: GA Selection Schemes

sequent generations. When elitism is applied to a genetic algorithm the best individual survives to the next generation. Though there is a risk of being trapped in a local optimum solution, this method is useful to preserve the best individual through subsequent generations.

Crossover

Crossover, or recombination, is the process of transferring well adapted "building blocks" from fit individuals to offspring generations for the further successful evolution. Holland (1975) noted that it was crossover, and not random point mutations, which separated genetic algorithms from other evolutionary computation methods. For example, crossover operations have the "ability" to identify those successful "building blocks" (subset strings) of individual chromosomes and use them to invent new and innovative individuals. This feature of a crossover operator is in contrast to the mutation operator that rarely produces these types of successful "innovations".

Most of the crossover operations are variants on three basic recombination schemes: one-point crossover, multiple-point crossover, and uniform crossover. Each of these schemes is shown in Figure A.4. The one-point crossover (most basic and common) and multiple-point crossover operations are self-explanatory. For the uniform crossover operation, each offspring bit is taken from either parent with an equal probability. The decision on whether or not to perform a crossover operation on two selected parents is determined randomly based on the GA parameter called crossover probability (p_c). Because of the effectiveness of the operation, a large crossover probability is commonly used in literature (e.g., $p_c \geq 0.8$).

Mutation

Mutations are small changes to the genetic code usually achieved by altering one or more of the genes in the chromosome. The mutation operation (Figure A.5) is generally considered

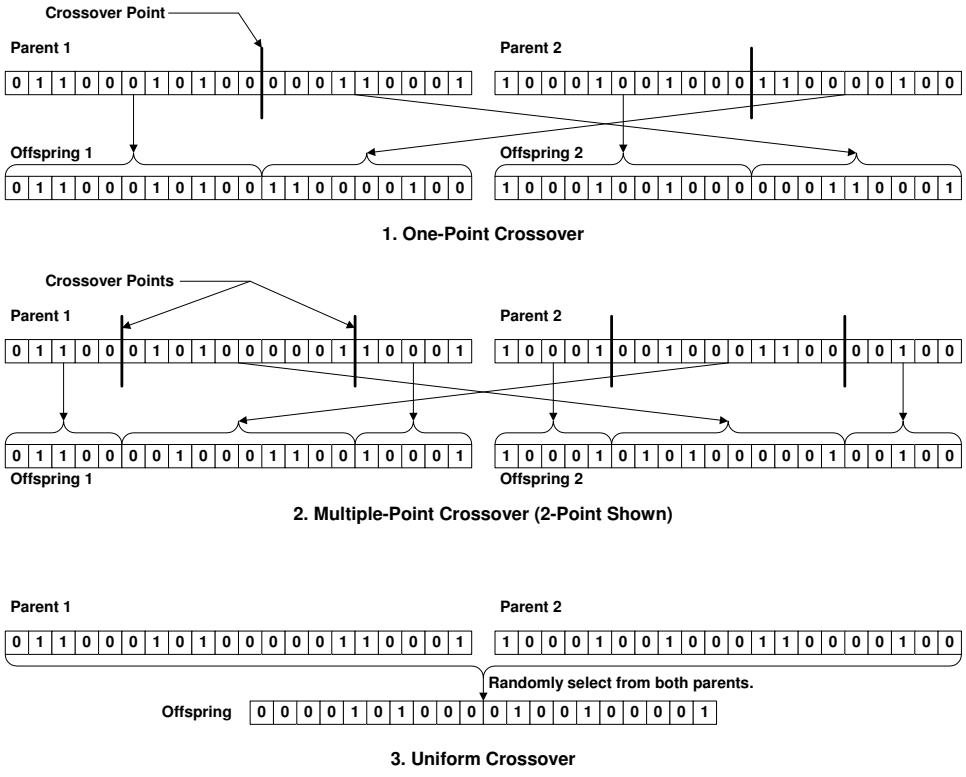


Figure A.4: GA Crossover Operators

as a method to recover lost genetic material rather than to search for better solutions. The decision on whether or not a given gene should be mutated is decided on the basis of a GA parameter called mutation probability (p_m). Past research has suggested that a low mutation probability should be used ($p_m = 0.001 \rightarrow 0.01$) and that a good rule of thumb is to set $p_m = (\textit{number of bits in the chromosome})^{-1}$. Also, close attention should be paid to the mutation probability convention used in literature. Some researchers use the probability that a bit will change (deterministic flip) while other researchers refer to mutation as the probability that a bit could change, i.e. even if a bit is selected for mutation there is a 50% probability (for binary strings) that the value will not change.

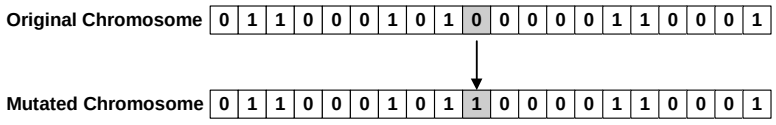


Figure A.5: GA Simple Mutation Operation

APPENDIX B: DATABASE SPECIFICATION

Link Information Table (Netlink)

Field	*Type*	*Description*
LinkID	Long Integer	Unique link identifier
FromNodeID	Long Integer	Unique node identifier for start node of link
ToNodeID	Long Integer	Unique node identified for end node of link
LinkLength	Double	Length of link (km)
Speed	Double	Speed used in link congestion function (km/h)
Capacity	Integer	Capacity used in link congestion function (vehicles/h/lane)
RecoveryCapacity	Double	Reduced capacity during recovery to full capacity after incident
NumberLanes	Byte	Number of lanes for this link direction
LinkType	Byte	1 = zone centroid connector, 2 = other
IncidentRate	Double	Exposure based incident rate (incidents/veh-km)
CapacityReduction	Double	Percentage reduction of capacity during incidents (%)
IncidentDuration	Double	Average length of incident on link (minutes)
DetectionTime	Double	Time to detect incident on link (minutes)
ProcessingTime	Double	Time to active CMS message (minutes)

Node Information Table (Netnode)

Field	*Type*	*Description*
NodeID	Long Integer	Unique node identifier
CoordinateX	Double	X-Coordinate
CoordinateY	Double	Y-Coordinate
NodeType	Long Integer	1 = zone centroid, 2 = other
Notation	Text (5)	Short description of NodeType

Origin-Destination Trips Table (ODTrips)

Field	*Type*	*Description*
ID	Long Integer	Unique trip identifier
FromZone	Long Integer	Trip start node (zone)
ToZone	Long Integer	Trip end node (zone)
AMPeakTripRate	Double	Trip rate during AM peak (veh/h)
MiddayTripRate	Double	Trip rate during midday (veh/h)
PMPeakTripRate	Double	Trip rate during PM peak (veh/h)
OvernightTripRate	Double	Trip rate during overnight period (veh/h)

CMS Information Table (CMS)

Field	*Type*	*Description*
CMSID	Long Integer	Unique identifier for CMS installation
LinkID	Long Integer	Unique link identifier where CMS is installed

APPENDIX C: SMALL NETWORK (CASE A) DATA

Link Data Note

The fields *IncidentRate, CapacityReduction, IncidentDuration, DetectionTime, and Processing Time* contained the same data: 0.0000029, .8, 30, 10, and 5 respectively for each link in the Toronto network case study. These fields were omitted from this Appendix to save space.

Link Table

Link ID	*From NodeID*	*To NodeID*	*Link Length*	*Speed*	*Capacity*	*Number Lanes*	*Link Type*
1	111	1	2.50	60.00	1000	2	2
2	112	111	0.50	60.00	1000	2	2
3	113	112	1.00	60.00	1000	2	2
4	114	113	1.00	60.00	1000	2	2
5	115	114	0.50	60.00	1000	2	2
6	212	115	0.25	60.00	1000	2	2
7	115	212	0.25	60.00	1000	2	2
8	114	115	0.50	60.00	1000	2	2
9	113	114	1.00	60.00	1000	2	2
10	112	113	1.00	60.00	1000	2	2
11	111	112	0.50	60.00	1000	2	2
12	1	111	2.50	60.00	1000	2	2
13	117	2	6.00	110.00	2000	3	2
15	119	117	0.30	110.00	2000	3	2
16	121	119	1.70	110.00	2000	3	2
18	123	121	0.30	110.00	2000	3	2
19	217	123	0.60	110.00	2000	3	2
21	135	232	0.60	110.00	2000	3	2
23	134	135	0.30	110.00	2000	3	2
25	133	134	1.70	110.00	2000	3	2
26	132	133	0.30	110.00	2000	3	2
28	2	132	6.00	110.00	2000	3	2
29	125	3	2.50	60.00	2000	2	2
30	126	125	0.50	60.00	2000	2	2
31	127	126	1.00	60.00	2000	2	2
32	128	127	1.00	60.00	2000	2	2
33	129	128	0.50	60.00	2000	2	2
34	226	129	0.25	60.00	2000	2	2
35	129	226	0.25	60.00	2000	2	2
36	128	129	0.50	60.00	2000	2	2
37	127	128	1.00	60.00	2000	2	2
38	126	127	1.00	60.00	2000	2	2
39	125	126	0.50	60.00	2000	2	2
40	3	125	2.50	60.00	2000	2	2
41	111	125	1.00	60.00	1000	2	2

Link ID	*From NodeID*	*To NodeID*	*Link Length*	*Speed*	*Capacity*	*Number Lanes*	*Link Type*
42	125	111	1.00	60.00	1000	2	2
45	112	118	0.35	60.00	1000	2	2
46	118	112	0.35	60.00	1000	2	2
47	118	130	0.30	60.00	1000	2	2
48	130	118	0.30	60.00	1000	2	2
49	130	126	0.35	60.00	1000	2	2
50	126	130	0.35	60.00	1000	2	2
51	113	127	1.00	60.00	1000	2	2
52	127	113	1.00	60.00	1000	2	2
55	114	122	0.35	60.00	1000	2	2
56	122	114	0.35	60.00	1000	2	2
57	122	131	0.30	60.00	1000	2	2
58	131	122	0.30	60.00	1000	2	2
59	131	128	0.35	60.00	1000	2	2
60	128	131	0.35	60.00	1000	2	2
61	115	129	1.00	60.00	1000	2	2
62	129	115	1.00	60.00	1000	2	2
65	118	117	0.30	60.00	1000	2	2
66	132	130	0.30	60.00	1000	2	2
67	130	133	0.30	60.00	1000	2	2
68	119	118	0.30	60.00	1000	2	2
69	122	121	0.30	60.00	1000	2	2
70	134	131	0.30	60.00	1000	2	2
71	131	135	0.30	60.00	1000	2	2
72	123	122	0.30	60.00	1000	2	2
73	112	10	0.25	60.00	1000	2	2
74	9	114	2.50	60.00	1000	2	2
75	8	126	2.50	60.00	1000	2	2
76	7	128	2.50	60.00	1000	2	2
77	128	7	2.50	60.00	1000	2	2
78	126	8	2.50	60.00	1000	2	2
79	114	9	2.50	60.00	1000	2	2
80	10	112	2.50	60.00	1000	2	2
103	213	212	1.00	60.00	1000	2	2
104	214	213	1.00	60.00	1000	2	2

Link ID	*From NodeID*	*To NodeID*	*Link Length*	*Speed*	*Capacity*	*Number Lanes*	*Link Type*
105	215	214	0.50	60.00	1000	2	2
106	4	215	2.50	60.00	1000	2	2
107	215	4	2.50	60.00	1000	2	2
108	214	215	0.50	60.00	1000	2	2
109	213	214	1.00	60.00	1000	2	2
110	212	213	1.00	60.00	1000	2	2
115	219	217	0.30	110.00	2000	3	2
116	221	219	1.70	110.00	2000	3	2
118	223	221	0.30	110.00	2000	3	2
119	5	223	6.00	110.00	2000	3	2
121	235	5	6.00	110.00	2000	3	2
123	234	235	0.30	110.00	2000	3	2
125	233	234	1.70	110.00	2000	3	2
126	232	233	0.30	110.00	2000	3	2
131	227	226	1.00	60.00	2000	2	2
132	228	227	1.00	60.00	2000	2	2
133	229	228	0.50	60.00	2000	2	2
134	6	229	2.50	60.00	2000	2	2
135	229	6	2.50	60.00	2000	2	2
136	228	229	0.50	60.00	2000	2	2
137	227	228	1.00	60.00	2000	2	2
138	226	227	1.00	60.00	2000	2	2
145	212	218	0.35	60.00	1000	2	2
146	218	212	0.35	60.00	1000	2	2
147	218	230	0.30	60.00	1000	2	2
148	230	218	0.30	60.00	1000	2	2
149	230	226	0.35	60.00	1000	2	2
150	226	230	0.35	60.00	1000	2	2
151	213	227	1.00	60.00	1000	2	2
152	227	213	1.00	60.00	1000	2	2
155	214	222	0.35	60.00	1000	2	2
156	222	214	0.35	60.00	1000	2	2
157	222	231	0.30	60.00	1000	2	2
158	231	222	0.30	60.00	1000	2	2
159	231	228	0.35	60.00	1000	2	2

Link ID	*From NodeID*	*To NodeID*	*Link Length*	*Speed*	*Capacity*	*Number Lanes*	*Link Type*
160	228	231	0.35	60.00	1000	2	2
161	215	229	1.00	60.00	1000	2	2
162	229	215	1.00	60.00	1000	2	2
165	218	217	0.30	60.00	1000	2	2
166	232	230	0.30	60.00	1000	2	2
167	230	233	0.30	60.00	1000	2	2
168	219	218	0.30	60.00	1000	2	2
169	222	221	0.30	60.00	1000	2	2
170	234	231	0.30	60.00	1000	2	2
171	231	235	0.30	60.00	1000	2	2
172	223	222	0.30	60.00	1000	2	2
173	212	14	2.50	60.00	1000	2	2
174	13	214	2.50	60.00	1000	2	2
175	12	226	2.50	60.00	1000	2	2
176	11	228	2.50	60.00	1000	2	2
177	228	11	2.50	60.00	1000	2	2
178	226	12	2.50	60.00	1000	2	2
179	214	13	2.50	60.00	1000	2	2
180	14	212	2.50	60.00	1000	2	2

NodeID	*CoordinateX*	*CoordinateY*	*NodeType*	*Notation*
1	250.00	1500.00	1	zone
10	1000.00	1750.00	1	zone
9	3000.00	1750.00	1	zone
7	3000.00	250.00	1	zone
8	1000.00	250.00	1	zone
3	250.00	500.00	1	zone
2	250.00	1000.00	1	zone
111	500.00	1500.00	2	
112	1000.00	1500.00	2	
113	2000.00	1500.00	2	
114	3000.00	1500.00	2	
115	4000.00	1500.00	2	
117	850.00	1010.00	2	
118	1000.00	1150.00	2	
119	1150.00	1010.00	2	
121	2850.00	1010.00	2	
122	3000.00	1150.00	2	
123	3150.00	1010.00	2	
125	500.00	500.00	2	
126	1000.00	500.00	2	
127	2000.00	500.00	2	
128	3000.00	500.00	2	
129	4000.00	500.00	2	
130	1000.00	850.00	2	
131	3000.00	850.00	2	
132	850.00	990.00	2	
133	1150.00	990.00	2	
134	2850.00	990.00	2	
135	3150.00	990.00	2	
14	5000.00	1750.00	1	zone
13	7000.00	1750.00	1	zone
4	7750.00	1500.00	1	zone
5	7750.00	1000.00	1	zone
6	7750.00	500.00	1	zone
11	7000.00	250.00	1	zone

Node Table

NodeID	*CoordinateX*	*CoordinateY*	*NodeType*	*Notation*
12	5000.00	250.00	1	zone
212	5000.00	1500.00	2	
213	6000.00	1500.00	2	
214	7000.00	1500.00	2	
215	7500.00	1500.00	2	
217	4850.00	1010.00	2	
218	5000.00	1150.00	2	
219	5150.00	1010.00	2	
221	6850.00	1010.00	2	
222	7000.00	1150.00	2	
223	7150.00	1010.00	2	
226	5000.00	500.00	2	
227	6000.00	500.00	2	
228	7000.00	500.00	2	
229	7500.00	500.00	2	
230	5000.00	850.00	2	
231	7000.00	850.00	2	
232	4850.00	990.00	2	
233	5150.00	990.00	2	
234	6850.00	990.00	2	
235	7150.00	990.00	2	

Trips Table

ID	*FromZone*	*ToZone*	*AMPeakRate*	*MiddayRate*	*PMPeakRate*	*OvernightRate*
1	2	4	1916.00	104.00	111.00	59.00
2	2	6	2073.00	102.00	98.00	47.00
3	2	5	3994.00	586.00	2538.00	370.00
4	5	2	2102.00	413.00	3583.00	257.00
5	5	1	111.00	97.00	3382.00	54.00
6	5	3	95.00	110.00	1379.00	41.00
11	4	1	90.00	106.00	93.00	50.00
12	4	2	119.00	108.00	1051.00	54.00
13	4	3	101.00	104.00	115.00	51.00
16	1	4	90.00	111.00	109.00	58.00
17	1	5	1742.00	96.00	105.00	53.00
18	1	6	101.00	90.00	81.00	45.00
26	3	4	88.00	101.00	103.00	59.00
27	3	5	1798.00	82.00	110.00	44.00
28	3	6	116.00	87.00	81.00	53.00
32	6	1	110.00	108.00	119.00	55.00
39	6	2	116.00	93.00	2367.00	44.00
40	6	3	81.00	82.00	109.00	58.00
41	10	8	104.00	110.00	85.00	51.00
42	10	7	106.00	114.00	96.00	59.00
43	10	11	103.00	100.00	87.00	56.00
44	10	12	110.00	110.00	87.00	48.00
45	9	8	85.00	114.00	96.00	57.00
46	9	7	115.00	104.00	94.00	52.00
47	9	11	112.00	117.00	107.00	46.00
48	9	12	84.00	119.00	104.00	46.00
49	13	8	86.00	90.00	111.00	53.00
50	13	7	84.00	92.00	90.00	58.00
51	13	11	83.00	97.00	101.00	59.00
52	13	12	110.00	97.00	117.00	59.00
53	14	8	85.00	107.00	99.00	56.00
54	14	7	89.00	104.00	108.00	54.00
55	14	11	83.00	108.00	88.00	51.00
56	14	12	92.00	120.00	101.00	47.00

ID	*FromZone*	*ToZone*	*AMPeakRate*	*MiddayRate*	*PMPeakRate*	*OvernightRate*
57	7	9	108.00	103.00	119.00	54.00
58	7	10	113.00	93.00	91.00	51.00
59	7	13	110.00	86.00	100.00	53.00
60	7	14	92.00	96.00	105.00	56.00
61	8	9	95.00	86.00	103.00	43.00
62	8	10	104.00	117.00	89.00	45.00
63	8	13	114.00	96.00	94.00	53.00
64	8	14	116.00	83.00	94.00	45.00
65	11	9	93.00	101.00	85.00	43.00
66	11	10	99.00	111.00	84.00	51.00
67	1	11	110.00	118.00	107.00	60.00
68	1	12	118.00	94.00	118.00	49.00
69	4	8	116.00	87.00	91.00	46.00
70	4	7	93.00	83.00	95.00	58.00
71	4	11	111.00	97.00	81.00	43.00
72	4	12	96.00	111.00	80.00	52.00

APPENDIX D: SAMPLE TORONTO (CASE B) DATA

Link Data Note

The fields *IncidentRate, CapacityReduction, IncidentDuration, DetectionTime, and Processing Time* contained the same data: 0.0000029, .8, 30, 10, and 5 respectively for each link in the Toronto network case study. These fields were omitted from this Appendix to save space.

Link Table

Link ID	*From NodeID*	*To NodeID*	*Link Length*	*Speed*	*Capacity*	*Recovery Capacity*	*Number Lanes*	*Link Type*
9724	11953	11954	0.11	20.00	1400	700	1	2
9725	11954	11955	0.07	20.00	1400	700	1	2
9726	11955	11956	0.11	20.00	1400	700	1	2
9727	11956	11947	0.30	110.00	1800	900	7	3
9728	11957	11506	0.27	60.00	800	400	2	2
9729	11957	11958	0.35	50.00	1400	700	1	2
9730	11957	11963	0.22	60.00	800	400	2	2
9731	11957	14051	0.33	20.00	500	250	1	2
9732	11958	11087	0.79	110.00	1800	900	7	3
9733	11959	10153	0.30	60.00	800	400	2	2
9734	11959	11071	0.51	50.00	1400	700	1	2
9735	11959	11967	0.19	60.00	800	400	2	2
9736	11960	11958	0.21	110.00	1800	900	7	3
9737	11960	11961	0.11	20.00	1400	700	2	2
9738	11961	11962	0.08	20.00	1400	700	2	2
9739	11962	11963	0.07	20.00	1400	700	2	2
9740	11963	11957	0.22	60.00	800	400	2	2
9741	11963	11967	0.22	60.00	800	400	2	2
9742	11964	11965	0.10	20.00	1400	700	2	2
9743	11965	11967	0.10	20.00	1400	700	2	2
9744	11966	11071	0.43	110.00	1800	900	7	3
9745	11966	11964	0.13	20.00	1400	700	2	2
9746	11967	11959	0.19	60.00	800	400	2	2
9747	11967	11963	0.22	60.00	800	400	2	2
9748	11968	10143	0.11	60.00	800	400	2	2
9749	11968	10186	0.15	60.00	800	400	2	2
9750	11968	11969	0.07	20.00	1400	700	1	2
9751	11969	11970	0.08	20.00	1400	700	1	2
9752	11970	11971	0.11	20.00	1400	700	1	2
9753	11971	11076	0.24	110.00	1800	900	3	3
9754	11972	10134	0.22	50.00	700	350	2	2
9755	11972	11973	0.36	50.00	1400	700	1	2
9756	11972	11976	0.10	50.00	700	350	2	2
9757	11973	11222	1.13	110.00	1800	900	3	3

Link ID	*From NodeID*	*To NodeID*	*Link Length*	*Speed*	*Capacity*	*Recovery Capacity*	*Number Lanes*	*Link Type*
9758	11974	11975	0.48	50.00	1400	700	2	2
9759	11974	11983	0.42	110.00	1800	900	3	3
9760	11975	11074	0.45	50.00	1400	700	1	2
9761	11975	11980	0.11	50.00	700	350	2	2
9762	11975	11993	0.46	50.00	700	350	2	2
9763	11976	11972	0.10	50.00	700	350	2	2
9764	11976	11977	0.09	20.00	1400	700	1	2
9765	11976	11980	0.07	50.00	700	350	2	2
9766	11977	11978	0.09	20.00	1400	700	1	2
9767	11978	11979	0.10	20.00	1400	700	1	2
9768	11979	11973	0.38	110.00	1800	900	3	3
9769	11980	11975	0.11	50.00	700	350	2	2
9770	11980	11976	0.07	50.00	700	350	2	2
9771	11980	11981	0.10	20.00	1400	700	1	2
9772	11981	11982	0.08	20.00	1400	700	1	2
9773	11982	11983	0.08	20.00	1400	700	1	2
9774	11983	11074	0.43	110.00	1800	900	3	3
9775	11984	11709	0.15	50.00	700	350	1	2
9776	11984	11710	0.31	50.00	700	350	1	2
9777	11985	10131	0.23	50.00	700	350	1	2
9778	11985	10133	0.29	50.00	700	350	1	2
9779	11985	11222	0.24	50.00	1400	700	1	2
9780	11986	11078	0.69	110.00	1800	900	5	3
9781	11986	11987	0.90	50.00	1400	700	2	2
9782	11987	11988	0.27	50.00	1400	700	1	2
9783	11987	11989	0.22	60.00	800	400	3	2
9784	11987	13028	0.30	60.00	800	400	3	2
9785	11988	11072	1.63	110.00	1800	900	5	3
9786	11989	11987	0.22	60.00	800	400	3	2
9787	11989	11990	0.07	20.00	1400	700	1	2
9788	11989	11998	0.08	60.00	800	400	3	2
9789	11990	11991	0.10	20.00	1400	700	1	2
9790	11991	11078	0.06	20.00	1400	700	1	2
9791	11992	11994	0.27	50.00	1400	700	2	2

Node Table

NodeID	*CoordinateX*	*CoordinateY*	*NodeType*	*Notation*
25	614030	834534	1	zone
26	614678	835031	1	zone
30	615349	837312	1	zone
31	616179	837360	1	zone
37	616890	838297	1	zone
38	615897	838112	1	zone
39	616438	839415	1	zone
40	617201	839737	1	zone
48	613964	837641	1	zone
49	613409	838406	1	zone
50	613167	839504	1	zone
59	613949	840981	1	zone
60	614357	839291	1	zone
61	615364	839622	1	zone
62	615498	840829	1	zone
63	614960	841386	1	zone
68	616743	841851	1	zone
69	618949	840910	1	zone
70	617929	842187	1	zone
71	618726	842502	1	zone
79	619747	842296	1	zone
80	620991	840689	1	zone
83	622213	841089	1	zone
84	622704	841657	1	zone
91	621051	842411	1	zone
92	622293	842846	1	zone
99	621386	842212	1	zone
100	623649	843124	1	zone
101	623930	841657	1	zone
106	625453	843982	1	zone
124	620032	840092	1	zone
283	628239	843010	1	zone
293	627020	842449	1	zone
294	625933	843101	1	zone
295	624581	842598	1	zone
296	624641	843333	1	zone

NodeID	*CoordinateX*	*CoordinateY*	*NodeType*	*Notation*
297	626787	843703	1	zone
298	626274	844745	1	zone
299	627639	844003	1	zone
300	628480	845165	1	zone
301	629580	845835	1	zone
302	629231	844409	1	zone
306	631149	846049	1	zone
307	627640	846569	1	zone
320	628085	847032	1	zone
322	629289	847336	1	zone
323	628414	846614	1	zone
324	630752	847532	1	zone
330	632660	847800	1	zone
331	632426	848682	1	zone
339	633319	847848	1	zone
340	634653	848018	1	zone
341	634082	848626	1	zone
344	633423	846741	1	zone
354	634694	846873	1	zone
377	637267	849432	1	zone
378	637493	848846	1	zone
379	638193	848954	1	zone
382	638541	849932	1	zone
385	635320	848644	1	zone
388	636598	849174	1	zone
389	636035	848289	1	zone
390	636193	846929	1	zone
416	638294	847751	1	zone
417	639268	847971	1	zone
425	640174	848537	1	zone
426	641294	848771	1	zone
429	638948	849260	1	zone
430	639857	849637	1	zone
431	641020	850119	1	zone
439	642573	851355	1	zone
440	642921	850601	1	zone
446	644974	851670	1	zone
447	646974	851688	1	zone

Trips Table

ID	*FromZone*	*ToZone*	*AMPeakRate*	*MiddayRate*	*PMPeakRate*	*OvernightRate*
56	10054	11356	6.07	1.77	38.01	5.69
57	10054	11728	57.50	16.41	35.23	5.40
58	10054	13016	28.75	8.30	0.00	0.00
59	13016	10054	0.00	0.00	27.35	8.04
60	10054	1505	46.57	14.24	0.00	0.00
61	1505	10054	0.00	0.00	39.67	13.02
62	10054	1508	17.41	4.37	0.00	0.00
63	1508	10054	0.00	0.00	14.86	4.21
64	10054	1509	21.06	6.25	0.00	0.00
65	1509	10054	0.00	0.00	20.98	5.54
66	10054	1582	6.07	1.74	0.00	0.00
67	1582	10054	0.00	0.00	5.58	1.82
68	10054	1593	19.44	5.48	0.00	0.00
69	1593	10054	0.00	0.00	17.21	6.29
70	10054	1596	23.49	6.42	0.00	0.00
71	1596	10054	0.00	0.00	23.54	6.19
72	10054	1598	23.49	7.55	0.00	0.00
73	1598	10054	0.00	0.00	25.71	7.45
74	10054	1599	4.86	1.29	0.00	0.00
75	1599	10054	0.00	0.00	5.01	1.43
76	10054	1600	36.45	12.07	9.72	1.56
77	10054	1601	30.37	9.87	0.00	0.00
78	1601	10054	0.00	0.00	26.37	9.32
79	10054	1602	52.25	16.70	0.00	0.00
80	1602	10054	0.00	0.00	43.54	16.90
81	10054	1603	41.31	11.29	8.10	1.23
82	10054	1604	26.73	6.98	0.00	0.00
83	1604	10054	0.00	0.00	23.00	6.26
84	10054	1609	70.06	19.04	17.01	2.63
85	10054	1610	34.02	10.13	0.00	0.00
86	1610	10054	0.00	0.00	30.05	10.82
87	10054	1613	24.30	6.18	0.00	0.00
88	1613	10054	0.00	0.00	24.54	5.33
89	10054	25	46.98	12.50	14.58	2.27
90	10054	26	30.37	8.77	42.12	6.48
91	10054	27	12.55	3.91	14.58	2.30

APPENDIX E: OptimalCMS SOURCE CODE

Objects

File	*Page*	*Description*
CMS.cs	132	Changeable Message Sign object
Link.cs	133	Link object
Network.cs	139	Representation of traffic network including nodes and links
Node.cs	161	Node object
ODTrips.cs	165	Representation of origin-destination trips by an object
Path.cs	167	Sequential set of links used for traffic (re)assignment
QueuingDiagram.cs	175	Abstract representation of a deterministic queuing diagram with and without CMS information
Settings.cs	195	Miscellaneous precision settings and GA parameters stored in object

Modules

File	*Page*	*Description*
Import.cs	210	Functions to import data from source database
Save.cs	214	Save functions for traffic assignment and optimization results

CMS.cs

```
using System;
using System.Collections;

namespace OptimalCMS
{
   /// <summary>
   /// Summary description for CMS.
   /// </summary>
   ///

   public class CMS
   {
      private Link link;

      public CMS(Link lk)
      {
         link = lk;
      }
```

```
      public Link Link
      {
         get
         {
            return link;
         }
         set
         {
            link = value;
         }
      }
   }
}
```

Link.cs

```
using System;
using System.Collections;

namespace OptimalCMS
{
   /// <summary>
   /// Summary description for Link.
   /// </summary>
   public class Link
   {
      #region Local variables to hold property values
      private long linkID;
      private Node fromNode;
      private Node toNode;
      private double length;
      private double freeFlowTime;
      private double capacity;
      private double recoveryCapacity;
      private double incidentRate;
      private double incidentDuration;
      private double reducedCapacity;
      private double detectAndProcess;
      private double[] timeAtLink;
      private double[] flowAtLink;
```

```
//Store yearly incident rate for each link
//(depends on link length and volume)
private double yearlyIncidentRate;

//Store yearly incident rate/time period
private double[] yearlyIncidentRateByTimePeriod;

//Store reference to paths for faster indexing
private ArrayList[] paths;
#endregion

#region Constructor logic
//Import constructor
public Link(long numTimePeriods, object lLinkID, Node lFromNode,
   Node lToNode, object lLength, object lFreeFlowSpeed, object
   lCapacity, object lrecoveryCapacity, object lNumberLanes, object
   lLinkType, object lIncidentRate, object lCapacityReduction,
   object lIncidentDuration, params object[] lDetectAndProcess)
{
   linkID = Convert.ToInt32(lLinkID);
   fromNode = lFromNode;
   toNode = lToNode;
   length = Convert.ToDouble(lLength);
   double freeFlowSpeed = Convert.ToDouble(lFreeFlowSpeed);
   freeFlowTime = length/freeFlowSpeed;

   long numberLanes = Convert.ToInt32(lNumberLanes);
   for (int i = 0; i < numberLanes; i++)
   {
      capacity += Convert.ToDouble(lCapacity);
      recoveryCapacity += Convert.ToDouble(lrecoveryCapacity);
   }

   incidentRate = Convert.ToDouble(lIncidentRate);

   //Calculations for capacity of link during incident
   double capacityReduction = Convert.ToDouble(lCapacityReduction);
   if (capacityReduction >= 1)capacityReduction = 0D;
   reducedCapacity = capacity * (1D - capacityReduction);
```

```
   //Convert minutes to hours
   incidentDuration = Convert.ToDouble(lIncidentDuration)/60D;
   detectAndProcess = (Convert.ToDouble(lDetectAndProcess[0])+
      Convert.ToDouble(lDetectAndProcess[1]))/60D;

   //Initialize array sizes
   timeAtLink = new double[numTimePeriods];
   flowAtLink = new double[numTimePeriods];
   paths = new ArrayList[numTimePeriods];
}
#endregion

#region Define get/set methods for value type properties
public long LinkID
{
   get
   {
      return linkID;
   }
}

public Node FromNode
{
   get
   {
      return fromNode;
   }
}

public Node ToNode
{
   get
   {
      return toNode;
   }
}

public double Capacity
{
```

```
   get
   {
      return capacity;
   }
}

public double RecoveryCapacity
{
   get
   {
      return recoveryCapacity;
   }
}

public double IncidentDuration
{
   get
   {
      return incidentDuration;
   }
}

public double ReducedCapacity
{
   get
   {
      return reducedCapacity;
   }
}

public double DetectAndProcess
{
   get
   {
      return detectAndProcess;
   }
}

public double[] TimeAtLink
{
```

```
   get
   {
      return timeAtLink;
   }
   set
   {
      timeAtLink = value;
   }
}

public double[] FlowAtLink
{
   get
   {
      return flowAtLink;
   }
   set
   {
      flowAtLink = value;
   }
}

public double YearlyIncidentRate
{
   get
   {
      return yearlyIncidentRate;
   }
}

public double[] YearlyIncidentRateByTimePeriod
{
   get
   {
      return yearlyIncidentRateByTimePeriod;
   }
}

public ArrayList[] Paths
{
```

```
   get
   {
      return paths;
   }
}
#endregion

#region Define add/get/delete methods for reference type properties

public void SetTimeAtLink(long timeIndex, double alpha,
   double beta)
{
   timeAtLink[timeIndex] = freeFlowTime * (1 + alpha *
      Math.Pow((flowAtLink[timeIndex]/capacity), beta));
}

public void SetYearlyIncidentRate(double[] tripRateDurations)
{
   //Store daily vehicle-kms
   double dailyVehKms = 0D;
   int numPeriods = tripRateDurations.Length;
   for(int i = 0; i < numPeriods; i++)
      dailyVehKms += tripRateDurations[i] * this.FlowAtLink[i];

   //Multiply by days/year & incident rate & length
   yearlyIncidentRate = dailyVehKms * 365D * this.incidentRate
      * this.length;

   //
   //Now do calculations for each time period
   yearlyIncidentRateByTimePeriod = new double[numPeriods];
   for(int j = 0; j < numPeriods; j++)
      yearlyIncidentRateByTimePeriod[j] = tripRateDurations[j]
         * yearlyIncidentRate / 24D;
}

public double ReturnSecondDerivativeCost(long timeIndex,
   double alpha, double beta)
{
   return freeFlowTime * alpha * Math.Pow(capacity, (-1D * beta))
```

```
            * beta * Math.Pow(flowAtLink[timeIndex], (beta-1D));
      }
      #endregion
   }

   public enum LinkTypes
   {
      CentroidConnector = 1,
      Other = 2,
   }
}
```

Network.cs

```
using System;
using System.Collections;

namespace OptimalCMS
{
   /// <summary>
   /// Summary description for Network.
   /// </summary>
   public class Network
   {
      #region Delegates
      public delegate void ProgressSet();
      public delegate void ProgressSetMax(int val);
      public delegate void ChartProgress(double val);
      public delegate void ChartGeneticProgress(double max,
         double min, double avg);
      private ProgressSet ps1;
      #endregion

      #region Local variables for GA
      //Local variables
      public double delayWithoutCMS;
      public double delayWithCMS;
      public double travelTimeSavings;
      private Settings cmsCalcSet;
```

```
private SortedList availableCMSLocations;

private ArrayList GApermanentCMSList;

#endregion

#region Local variables to hold property values
private double bprAlpha;
private double bprBeta;
private double cmsAlpha;
private double cmsBeta;
private SortedList nodeList;
private SortedList originNodeList;
private SortedList linkList;
private ArrayList odTripsList;
private ArrayList cmsList;
private ArrayList[] pathLists;
private double[] tripRateDurations;
private long numTimePeriods;

//Variables to store link characteristics
private SortedList linkFlows;

//Temporary list of candidate CMS locations during calculations
private ArrayList cmsTempList;

//Statistics for diversion rate
private double totalDiversion;
private double diversionCounter;

#endregion

#region Constructor logic
public Network(long numberPeriods)
{
   bprAlpha = 0.15;
   bprBeta = 4;
   cmsAlpha = 5;
   cmsBeta = 5;
   nodeList = new SortedList();
```

```
   originNodeList = new SortedList();
   linkList = new SortedList();
   odTripsList = new ArrayList();
   cmsList = new ArrayList();

   numTimePeriods = numberPeriods;
   pathLists = new ArrayList[numTimePeriods];
   tripRateDurations = new double[numTimePeriods];
}

public void AddSettings(Settings CMSCalcSet)
{
   cmsCalcSet = CMSCalcSet;
}

#endregion

#region Define get/set methods for value type properties
public double BPRalpha
{
   get
   {
      return this.bprAlpha;
   }
   set
   {
      bprAlpha = value;
   }
}

public double BPRbeta
{
   get
   {
      return this.bprBeta;
   }
   set
   {
      bprBeta = value;
   }
```

```
}

public double CMSalpha
{
   get
   {
      return this.cmsAlpha;
   }
   set
   {
      cmsAlpha = value;
   }
}

public double CMSbeta
{
   get
   {
      return this.cmsBeta;
   }
   set
   {
      cmsBeta = value;
   }
}

public SortedList NodeList
{
   get
   {
      return this.nodeList;
   }
}

public SortedList LinkList
{
   get
   {
      return this.linkList;
   }
```

```
}

public ArrayList CMSList
{
   get
   {
      return this.cmsList;
   }
}

public long NumTimePeriods
{
   get
   {
      return numTimePeriods;
   }
}

public ArrayList[] PathLists
{
   get
   {
      return pathLists;
   }
   set
   {
      pathLists = value;
   }
}

public double[] TripRateDurations
{
   get
   {
      return tripRateDurations;
   }
   set
   {
      tripRateDurations = value;
   }
```

```
}


public ArrayList CMSTempList
{
   get
   {
      return cmsTempList;
   }
   set
   {
      cmsTempList = value;
   }
}

public double MeanDiversion
{
   get
   {
      return totalDiversion/(double)diversionCounter;
   }
}

#endregion

#region Define add/get/delete methods for reference type properties
public Node GetNode(long NodeID)
{
   return (Node)this.nodeList[NodeID];
}

public Link GetLink(long LinkID)
{
   return (Link)linkList[LinkID];
}

public long NumberLinks()
{
   return this.linkList.Count;
}
```

```
public void AddNode(Node node)
{
   if(!nodeList.ContainsKey(node.NodeID))
   {
      nodeList.Add(node.NodeID, node);

      //If origin node add to origin node (used for faster assignment)
      if (node.NodeType == NodeTypes.Centroid &&
         !originNodeList.ContainsKey(node.NodeID))
            originNodeList.Add(node.NodeID, node);
   }
}

public void AddLink(Link link)
{
   if(!linkList.ContainsKey(link.LinkID))
      linkList.Add(link.LinkID, link);
}

public void AddODTrip(ODTrips odTrip)
{
   odTripsList.Add(odTrip);
}

public void AddCMS(CMS cms)
{
   cmsList.Add(cms);
}
#endregion

#region Network calculations/functions
public void FindSP_LC(Node orignode, long index)
{
   //create a dqueue for storing the scan eligible node set
   Stack q1 = new Stack();      //High priority list
   Stack q2 = new Stack();      //Low priority list

   //temp variables
   Node iNode, jNode;
```

```
double newCost = 0D;

//Step 1: Initialization - set information for the other nodes
foreach (Node node in this.nodeList.Values)
{
   node.CostAtNode[index] = System.Single.MaxValue;
   node.AppearedInQueue[index] = SPAState.NeverAppeared;
}
//set information for the origin node
orignode.CostAtNode[index] = newCost;
orignode.InLink[index] = null;
q1.Push(orignode);

//Step 2: Node selection
do
{
   //Get and remove the first element
   iNode = (Node)q1.Pop();
   iNode.AppearedInQueue[index] = SPAState.AppearedNotInQueue;
   //appeared, but removed (no longer in queue)

   //Step 3: Node expansion
   foreach (Link ijLink in iNode.OutLinkList.Values)
   {
      jNode = ijLink.ToNode;
      newCost = iNode.CostAtNode[index] + ijLink.TimeAtLink[index];
      if (newCost < jNode.CostAtNode[index])
      {
         //update cost and pointer
         jNode.CostAtNode[index] = newCost;
         jNode.InLink[index] = ijLink;

         //insert node j into Q
         if (jNode.AppearedInQueue[index] == SPAState.NeverAppeared)
            q2.Push(jNode);
         else
            q1.Push(jNode);

         jNode.AppearedInQueue[index] = SPAState.AppearedInQueue;
      }
```

```
      }

      //If no nodes in high priority queue -> copy low priority
      //nodes to high
      if(q1.Count == 0)
      {
         q1 = (Stack)q2.Clone();
         q2.Clear();
      }
   }
   //Step 4:Stop?
   while(q1.Count != 0);
}

public Path ReturnShortestPath(Node oNode, Node dNode, long timeIndex)
{
   //Create new path
   Stack createPath = new Stack();

   //Update each link along path
   Node iNode = dNode;
   Link link;
   do
   {
      link = iNode.InLink[timeIndex];
      createPath.Push(link);
      iNode = link.FromNode;
   }
   while (iNode != oNode);

   return new Path(createPath);
}

private void ResetLinkTravelTimes()
{
   foreach (Link link in this.linkList.Values)
   {
      for (long index = 0; index < this.numTimePeriods; index++)
      {
         link.FlowAtLink[index] = 0D;
```

```
            link.SetTimeAtLink(index, this.bprAlpha, this.bprBeta);
         }
      }

      for (long index = 0; index < this.numTimePeriods; index++)
         this.pathLists[index] = new ArrayList();

   }

   private void UpdateLinkTravelTime()
   {
      foreach (Link link in this.linkList.Values)
         for (long index = 0; index < this.numTimePeriods; index++)
            link.SetTimeAtLink(index, this.bprAlpha, this.bprBeta);
   }

   private void CopyPathsToMaster()
   {
      //Copy paths back to roadNetwork pathlist
      for (long index = 0; index < this.numTimePeriods; index++)
      {
         //Remove old paths from links
         foreach(Link link in this.linkList.Values)
            link.Paths[index] = new ArrayList();

         foreach (ODTrips trip in this.odTripsList)
         {
            foreach(Path path in trip.PathList[index])
            {
               this.pathLists[index].Add(path);

               //Add path to all links on the path
               foreach(Link link in path.Links)
                  link.Paths[index].Add(path);

               //Add destination node of the OD trip to the path
               path.DestinationNode = trip.ToZone;
            }
         }
      }
```

```
}

private void AssignPathFlowsToLinks(long index)
{
   //Constant double 0
   const double ZERO = 0D;

   //Set link flows to zero before assigning path flows
   foreach (Link link in this.linkList.Values)
      link.FlowAtLink[index] = ZERO;

   //Then assign path flows
   foreach (ODTrips trip in this.odTripsList)
      foreach(Path path in trip.PathList[index])
         foreach(Link link in path.Links)
            link.FlowAtLink[index] += path.Flow;
}

public void GradientProjectionAssignment()
{
   const double GPalpha = 0.2D;
   const double StoppingCriterion = 0.01D;
   double terminationVar = 1D;
   double newSPFlow;
   Stack removeStack = new Stack();
   Path newPath, shortestPath;
   bool firstSP;

   //INITIALIZATION
   ResetLinkTravelTimes();

   long index = 0;

   //for (long index = 0; index < this.numTimePeriods; index++)
   //{
   //Force SP calculations during first run
   firstSP = true;

   //Iterate through each entry in the O-D matrix
   foreach (ODTrips trip in this.odTripsList)
```

```
{
   //Delete old paths
   trip.PathList[index] = new ArrayList();


   //Perform shortest path routing
   if(trip.FromZone.CostAtNode[index] != 0 || firstSP)
   {
      firstSP = false;
      FindSP_LC(trip.FromZone, index);
   }

   //Create new path
   newPath = ReturnShortestPath(trip.FromZone, trip.ToZone, index);
   newPath.Flow = trip.TripRate[index];

   //Add path
   trip.PathList[index].Add(newPath);

   //Flag this path as the shortest path
   trip.ShortestPath[index] = newPath;
}

//Assign path flows to links
AssignPathFlowsToLinks(index);
//}

//for (long index = 0; index < this.numTimePeriods; index++)
//{
//Counter for number of iterations
int counter = 0;

do
{
   counter++;

   //COLUMN GENERATION
   //Update link travel time
   this.UpdateLinkTravelTime();
```

```
//Force SP calculations during first run
firstSP = true;

//Iterate through each entry in the O-D matrix
foreach (ODTrips trip in this.odTripsList)
{
   //Perform shortest path routing
   if(trip.FromZone.CostAtNode[index] != 0 || firstSP)
   {
      firstSP = false;
      FindSP_LC(trip.FromZone, index);
   }

   //Create new path
   newPath = ReturnShortestPath(trip.FromZone,
      trip.ToZone, index);
   trip.ShortestPath[index] = newPath;

   //Add path
   trip.PathList[index].Add(newPath);
}

//EQUILIBRATION
foreach (ODTrips trip in this.odTripsList)
{
   //Get the shortest path object
   shortestPath = trip.ShortestPath[index];

   //Compute shortest path cost
   shortestPath.PathCostD = 0D;
   foreach(Link link_D in shortestPath.Links)
      shortestPath.PathCostD += link_D.TimeAtLink[index];

   foreach (Path path in trip.PathList[index])
   {
      if(path != shortestPath)
      {
         //Reset path costs
         path.PathCostD = 0D;
         path.PathCostS = 0D;
```

```
            //Compute non-shortest path costs
            foreach (Link link in path.Links)
            {
               path.PathCostD += link.TimeAtLink[index];
               if (shortestPath.Links.Contains(link) != true)
                  path.PathCostS += link.ReturnSecondDerivativeCost
                     (index, this.bprAlpha, this.bprBeta);
            }

            //Update non-shortest path flows
            path.PreviousFlow = path.Flow;
            if (path.PathCostS != 0D)
               path.Flow = path.PreviousFlow - (GPalpha
                  /path.PathCostS) *(path.PathCostD
                  - shortestPath.PathCostD);
            else path.Flow = -1D;

            if (path.Flow <= 0D)
               removeStack.Push(path);
         }
      }

      //Remove paths with zero/negative flows
      while(removeStack.Count != 0)
         trip.PathList[index].Remove(removeStack.Pop());

      //Update shortest path flow
      shortestPath.PreviousFlow = shortestPath.Flow;
      shortestPath.Flow = 0D;
      newSPFlow = trip.TripRate[index];

      //Remove non-shortest path flows
      foreach(Path setpath in trip.PathList[index])
         newSPFlow -= setpath.Flow;
      shortestPath.Flow = newSPFlow;
   }
   //Update link flows
   this.AssignPathFlowsToLinks(index);
```

```
      //TERMINATION
      terminationVar = 0D;
      foreach (ODTrips trip in this.odTripsList)
      {
         terminationVar = 0D;
         foreach(Path path in trip.PathList[index])
            terminationVar += (path.PreviousFlow/
               (double)trip.TripRate[index]) *
               (path.PathCostD -
               trip.ShortestPath[index].PathCostD) /
               path.PathCostD;

         //Check for violations
         if (terminationVar > StoppingCriterion)
            break;
      }

      //Terminate if too many iterations
      if (counter > 99) terminationVar = 0D;
   }
   while(terminationVar > StoppingCriterion);
   //}
   this.UpdateLinkTravelTime();
   this.CopyPathsToMaster();
   this.SetYearlyLinkIncidentRate();
}

public void IncrementalAssignment(long Increment)
{
   //Reset the link flows to zero
   this.ResetLinkTravelTimes();

   //Incremental Assignment
   for (long i = 0; i < Increment; i++)
   {
      //Update the link travel times
      this.UpdateLinkTravelTime();

      //Iterate through each entry in the O-D matrix
      foreach (ODTrips trip in this.odTripsList)
```

```
      {
         //Calculate for all time periods
         for (long index = 0; index < 4; index++)
         {
            //Perform shortest path routing
            FindSP_LC(trip.FromZone, index);

            //Create new path
            Path newPath = this.ReturnShortestPath(trip.FromZone,
               trip.ToZone, index);
            newPath.Flow = trip.TripRate[index]/(double)Increment;

            //Assign path flows to links
            this.AssignPathFlowsToLinks(index);

            //Add path
            trip.PathList[index].Add(newPath);
         }
      }
   }
   this.CopyPathsToMaster();
   this.SetYearlyLinkIncidentRate();
}

private void SetYearlyLinkIncidentRate()
{
   //Must be done after traffic assignment
   foreach(Link lk in this.linkList.Values)
      lk.SetYearlyIncidentRate(this.tripRateDurations);
}
#endregion

#region Save/Restore UE link characteristics
public void SaveUESolution()
{
   linkFlows = new SortedList();
   double[] flows;
   double copy;

   //Save link flows
```

```
   foreach(Link lk in this.linkList.Values)
   {
      flows = new double[lk.FlowAtLink.Length];
      for(int i = 0; i < flows.Length; i++)
      {
         copy = lk.FlowAtLink[i];
         flows[i] = copy;
      }
      linkFlows.Add(lk.LinkID, flows);
   }
}

public void RestoreUESolution()
{
   //Now reset the flows and times
   foreach(Link lk in this.linkList.Values)
   {
      double[] copyArray = (double[])linkFlows[lk.LinkID];
      for(int i = 0; i < copyArray.Length; i++)
      {
         double copy = copyArray[i];
         lk.FlowAtLink[i] = copy;
      }
   }
}
#endregion

#region CMS benefit calculations
public void CMSBenefit()
{
   //Save the UE solution for link flows and times
   this.SaveUESolution();

   //Reinitialize the calculation variables
   delayWithoutCMS = delayWithCMS = travelTimeSavings = 0D;

   //Create object that represents the queuing diagram
   QueuingDiagram QD;

   //Iterate through all links and simulate incident
```

```
   foreach(Link lk in linkList.Values)
   {
      //Iterate through each time period
      for(int i = 0; i < numTimePeriods; i++)
      {
         //Create new diagram
         QD = new QueuingDiagram(lk, i, cmsCalcSet.NumUpdateIntervals,
            cmsCalcSet.NumIncidentStarts, this,
            cmsCalcSet.IncludeAltTravelTime);

         //Temporarily release system resources for other tasks
         System.Windows.Forms.Application.DoEvents();

         //Calculate delays
         QD.CalculateDelay();
         QD.CalculateCMSDelay();

         //Scale by time period duration & incident rate
         //and then store benefit
         delayWithoutCMS += QD.Delay *
            lk.YearlyIncidentRateByTimePeriod[i];
         delayWithCMS += QD.CMSDelay < 0 ? 0D : QD.CMSDelay *
            lk.YearlyIncidentRateByTimePeriod[i];

         //Add statistical data
         if(QD.TotalDiversion > 0D)
         {
            totalDiversion += QD.TotalDiversion;
            diversionCounter += QD.DiversionCounter;
         }
      }
   }
   //Calculate the benefit of CMS
   travelTimeSavings = delayWithoutCMS - delayWithCMS;

   //Restore the UE Solution if it has not been done already
   this.RestoreUESolution();
}
```

```
#endregion

#region Optimization Calculations
private void InitializeStatistics()
{
   totalDiversion = 0D;
   diversionCounter = 0L;
}


public void GreedyCMSOptimization(ProgressSet ps, ProgressSetMax psMax,
   ChartProgress chartP, ChartGeneticProgress chartGP)
   //Adds one CMS incrementally at a time.
{
   InitializeStatistics();

   //Use a new list or current list depending on user choice
   if(cmsCalcSet.AddLocations)
      cmsTempList = cmsList;
   else
      cmsTempList = new ArrayList();

   //Create a set of possible locations for CMS
   SortedList availableCMSLocations = AvailableCMSLocations(cmsTempList);

   //Set up the progress bar
   psMax((int)cmsCalcSet.CMSToAllocate);

   //Initial size of the cmsTempList
   int initSize = cmsTempList.Count;

   //Loop through greedy allocation of each additional CMS
   for(int i = 0; i < cmsCalcSet.CMSToAllocate; i++)
   {
      //Store the highest travel time savings/CMS
      double TTSOptimal = 0D;
      Link optimalLink = null;

      //Add a placeholder
      cmsTempList.Add(null);
```

```
      foreach(Link link in availableCMSLocations.Values)
      {
         cmsTempList[i + initSize] = new CMS(link); //new CMS(link);

         //Do the calculations
         CMSBenefit();

         //Check if benefit is greater than current optimal
         if(travelTimeSavings > TTSOptimal)
         {
            TTSOptimal = travelTimeSavings;
            optimalLink = link;
         }
      }

      //Finally set the optimal link
      cmsTempList[i + initSize] = new CMS(optimalLink);

      //Do benefit calculations once more
      CMSBenefit();

      //Update the progress bar display
      ps();

      //Update the chart display
      chartP(travelTimeSavings);

      //Remove the optimal link from the available locations list
      if(availableCMSLocations.Count != 0)
         availableCMSLocations.Remove(optimalLink.LinkID);
   }
}

public void GeneticOptimization(ProgressSet ps, ProgressSetMax psMax,
   ChartProgress chartP, ChartGeneticProgress chartGP)
{
   InitializeStatistics();

   //Use a new list or current list depending on user choice
```

```
ArrayList permanentCMSlist = cmsCalcSet.AddLocations ?
       cmsList : new ArrayList();

//
GApermanentCMSList = permanentCMSlist;

//Create a set of possible locations for CMS
availableCMSLocations = this.AvailableCMSLocations(permanentCMSlist);

//Send the listing of possible locations as a static value
//to the Allele class
SGA.availLinkSet = new Link[availableCMSLocations.Count];
availableCMSLocations.Values.CopyTo(SGA.availLinkSet, 0);

//Set the length of the chromosome
int lchrom = (int)cmsCalcSet.CMSToAllocate;

//Set up the progress bar
psMax((int)(cmsCalcSet.MaximumGenerations + 1)
   *(int)cmsCalcSet.PopulationSize);

ps1 = ps;

//Start the algorithm
SGA sga = new SGA(cmsCalcSet,
   new SGA.EvaluateObjectiveFunction(this.Decode),
   new SGA.ChartGeneticProgress(chartGP));

//Get the best chromosome
Individual bestLocations = sga.Optimize();

//Set the travel time savings
this.travelTimeSavings = bestLocations.Fitness;

//Save the results
cmsTempList = new ArrayList();
CMS cmsSave;
for(int i = 0; i < bestLocations.Links.Length; i++)
{
   cmsSave = new CMS(bestLocations.Links[i]);
```

```
      cmsTempList.Add(cmsSave);
   }
}

private double Decode(Link[] Chromosome, long lchrom)
{
   //Set the network temp CMS list for decoding
   cmsTempList = (ArrayList)GApermanentCMSList.Clone();

   //Add each CMS from the chromosome
   CMS cmsAdd;
   for(int i = 0; i < lchrom; i++)
   {
      cmsAdd = new CMS(Chromosome[i]);
      cmsTempList.Add(cmsAdd);
   }

   //Do the calculations
   CMSBenefit();

   //Update the progressbar
   ps1();

   //Return the benefit
   return this.travelTimeSavings;
}

public SortedList AvailableCMSLocations(ArrayList permanentCMSlist)
{
   SortedList locations = new SortedList();

   //Add possible locations
   foreach(Link lk in linkList.Values)
      locations.Add(lk.LinkID, lk);

   //Remove permanent/fixed locations
   foreach(CMS cms in permanentCMSlist)
      locations.Remove(cms.Link.LinkID);

   //Remove locations with no benefit
```

```
            CMS cmsCheck = new CMS(null);
            cmsTempList = new ArrayList();
            cmsTempList.Add(cmsCheck);
            //
            Stack inefficientLocations = new Stack();
            foreach(Link lk in locations.Values)
            {
               cmsCheck.Link = lk;
               this.CMSBenefit();

               //Mark the inefficient location for removal
               if(travelTimeSavings == 0)
                  inefficientLocations.Push(lk.LinkID);
            }

            //Finally remove the locations
            while(inefficientLocations.Count != 0)
               locations.Remove(inefficientLocations.Pop());

            //Return the list
            return locations;
         }

         #endregion
      }
}
```

Node.cs

```
using System;
using System.Collections;

namespace OptimalCMS
{
   /// <summary>
   /// Summary description for Node.
   /// </summary>
   public class Node
   {
      #region Local variables to hold property values
```

```
private long nodeID;
private double x;
private double y;
private NodeTypes nodeType;
private SortedList outLinkList;

//shortest path algorithm variables
private double[] costAtNode;
private Link[] inLink;
private SPAState[] appearedInQueue;
#endregion

#region Constructor logic
//Import constructor
public Node(object lNodeID, object lCoordinateX, object lCoordinateY,
   object lNodeType, long numTimePeriods)
{
   nodeID = Convert.ToInt32(lNodeID);
   x = Convert.ToDouble(lCoordinateX);
   y = Convert.ToDouble(lCoordinateY);

   if (Convert.ToInt32(lNodeType) == 1)
      nodeType = NodeTypes.Centroid;
   else
      nodeType = NodeTypes.Other;

   outLinkList = new SortedList();

   costAtNode = new double[numTimePeriods];
   inLink = new Link[numTimePeriods];
   appearedInQueue = new SPAState[numTimePeriods];
}
#endregion

#region Define get/set methods for value type properties
public long NodeID
{
   get
   {
      return nodeID;
```

```
   }
}

public double X
{
   get
   {
      return x;
   }
}

public double Y
{
   get
   {
      return y;
   }
}

public NodeTypes NodeType
{
   get
   {
      return nodeType;
   }
}

public double[] CostAtNode
{
   get
   {
      return costAtNode;
   }
   set
   {
      costAtNode = value;
   }
}

public Link[] InLink
```

```
{
   get
   {
      return inLink;
   }
   set
   {
      inLink = value;
   }
}

public SPAState[] AppearedInQueue
{
   get
   {
      return appearedInQueue;
   }
   set
   {
      appearedInQueue = value;
   }
}

public SortedList OutLinkList
{
   get
   {
      return outLinkList;
   }
}
#endregion

#region Define add/get/delete methods for reference type properties
public void AddOutLink(Link newLink)
{
   if(!outLinkList.ContainsKey(newLink.LinkID))
   {
      outLinkList.Add(newLink.LinkID, newLink);
   }
}
```

```
      #endregion
   }

   public enum SPAState
   {
      NeverAppeared,
      AppearedInQueue,
      AppearedNotInQueue,
   }

   public enum NodeTypes
   {
      Centroid = 1,
      Other = 2,
   }
}
```

ODTrips.cs

```
using System;
using System.Collections;

namespace OptimalCMS
{
   /// <summary>
   /// Summary description for ODTrips.
   /// </summary>
   public class ODTrips
   {
      #region Local variables to store property values
      private Node fromZone;
      private Node toZone;
      private double[] tripRate;
      //Path-based traffic assignment variable
      private ArrayList[] pathList;
      private Path[] shortestPath;
      #endregion

      #region Constructor logic
      //Import Constructor
```

```
public ODTrips(Node FromNode, Node ToNode, params object[] TripRate)
{
   long numberTimePeriods = TripRate.Length;
   fromZone = FromNode;
   toZone = ToNode;

   //Set size of arrays
   tripRate = new double[numberTimePeriods];
   pathList = new ArrayList[numberTimePeriods];

   for(int i = 0; i < numberTimePeriods; i++)
   {
      tripRate[i] = Convert.ToDouble(TripRate[i]);
      pathList[i] = new ArrayList();
   }

   shortestPath = new Path[numberTimePeriods];
}
#endregion

#region Define get/set methods for value type properties
public Node FromZone
{
   get
   {
      return fromZone;
   }
}

public Node ToZone
{
   get
   {
      return toZone;
   }
}

public double[] TripRate
{
   get
```

```
            {
                return tripRate;
            }
        }

        public Path[] ShortestPath
        {
            get
            {
                return shortestPath;
            }
            set
            {
                shortestPath = value;
            }
        }

        public ArrayList[] PathList
        {
            get
            {
                return pathList;
            }
            set
            {
                pathList = value;
            }
        }
        #endregion
    }
}
```

Path.cs

```
using System;
using System.Collections;

namespace OptimalCMS
{
    /// <summary>
```

```
/// Summary description for Path.
/// </summary>
public class Path
{
   #region Local variables to hold property values
   private double flow;
   private ArrayList links;

   //Traffic Assignment variables
   private double pathCostD;
   private double pathCostS;
   private double previousFlow;
   private Node destinationNode;

   //Store already calculated alternative routes
   private Hashtable alternateTimes;

   //Used for extra delay on alternate routes
   private double exDelayAlt;
   #endregion

   #region Constructor logic
   public Path(ICollection lks)
   {
      links = new ArrayList(lks);
      flow = pathCostD = pathCostS = previousFlow = 0D;
      alternateTimes = new Hashtable();
   }
   #endregion

   #region Define get/set methods for value type properties
   public double Flow
   {
      get
      {
         return flow;
      }
      set
      {
         flow = value;
```

```
   }
}

public double PathCostD
{
   get
   {
      return pathCostD;
   }
   set
   {
      pathCostD = value;
   }
}

public double PathCostS
{
   get
   {
      return pathCostS;
   }
   set
   {
      pathCostS = value;
   }
}

public double PreviousFlow
{
   get
   {
      return previousFlow;
   }
   set
   {
      previousFlow = value;
   }
}

public Node DestinationNode
```

```
{
   set
   {
      destinationNode = value;
   }
}

public ArrayList Links
{
   get
   {
      return links;
   }
   set
   {
      links = value;
   }
}

#endregion

#region Define function for diversion calculations
public double DivertWithAltTravelTime(long timeIndex, double delay,
   Node divertNode, Link incidentLink, Network network)
{
   //Travel time of alternative path
   double TkmAlt = 0D;

   //Keep the original path
   ArrayList originalPath = new ArrayList();
   ArrayList originalPathFlows = new ArrayList();
   int indexOfOrigStart = 0;

   //Add the travel times of the links before the diversion point
   //to the alternate route
   for(int i = 0; i < links.Count; i++)
   {
      Link lk = (Link)links[i];
      TkmAlt += lk.TimeAtLink[timeIndex];
      if(lk.ToNode.NodeID == divertNode.NodeID)
```

```
   {
      indexOfOrigStart = i+1;
      break;
   }
}

//Travel time through the original path -> include links before
//diversion and delay. Links after diversion will be added later.
double Tkm = TkmAlt + delay;

//Store the original path after the diversion point
for(int i = indexOfOrigStart; i < links.Count; i++)
{
   Link lk = (Link)links[i];
   originalPath.Add(lk);
   originalPathFlows.Add(lk.FlowAtLink[timeIndex]);
}

//Find the shortest alternative route
network.FindSP_LC(divertNode, timeIndex);

//Check for possibility that alternate path does not exist
if(destinationNode.CostAtNode[timeIndex] > 0.95*System.Single.MaxValue)
   return 0D;

//Create a path object for that route
Path alternatePath = network.ReturnShortestPath(divertNode,
   destinationNode, timeIndex);

//Store the alternate path flows
ArrayList alternatePathFlows = new ArrayList();
for(int i = 0; i < alternatePath.links.Count; i++)
{
   Link lk = (Link)alternatePath.links[i];
   alternatePathFlows.Add(lk.FlowAtLink[timeIndex]);
}

//Create a temporary alternate path travel time
double TkmAlternate = 0D;
```

```
//The actual diversion that will occur
double divertedFlow = 0D;

//Calculate the original travel time on the alternate path
exDelayAlt = 0D;
for(int j = 0; j < alternatePath.links.Count; j++)
{
   Link lk = (Link)alternatePath.links[j];
   exDelayAlt -= lk.TimeAtLink[timeIndex];
}

//Loop until convergence
for(int j = 0; j < 10; j++)
{
   //Partial path travel time first ...
   TkmAlternate = TkmAlt;

   //... then alternate path travel time
   for(int k = 0; k < alternatePath.links.Count; k++)
   {
      Link lk = (Link)alternatePath.links[k];
      lk.SetTimeAtLink(timeIndex, network.BPRalpha, network.BPRbeta);
      TkmAlternate += lk.TimeAtLink[timeIndex];
   }

   //Restore original alternate path flow
   for(int k = 0; k < alternatePath.links.Count; k++)
   {
      Link lk = (Link)alternatePath.links[k];
      lk.FlowAtLink[timeIndex] = (double)alternatePathFlows[k];
   }

   //Original path travel time
   Tkm = TkmAlt + delay;

   //Add times of links after diversion point
   foreach(Link lk in originalPath)
   {
      lk.SetTimeAtLink(timeIndex, network.BPRalpha, network.BPRbeta);
      Tkm += lk.TimeAtLink[timeIndex];
```

```
      }

      //Restore the original flow on the original path
      for(int k = 0; k < originalPath.Count; k++)
      {
         Link lk = (Link)originalPath[k];
         lk.FlowAtLink[timeIndex] = (double)originalPathFlows[k];
      }

      //Calculate diversion
      divertedFlow = flow * (1 / (1 + Math.Exp(network.CMSalpha -
         network.CMSbeta * ((Tkm - TkmAlternate)/TkmAlternate))));

      //Update flows and recalculate travel times for alternate path
      for(int k = 0; k < alternatePath.links.Count; k++)
      {
         Link lk = (Link)alternatePath.links[k];
         lk.FlowAtLink[timeIndex] += divertedFlow;
         lk.SetTimeAtLink(timeIndex, network.BPRalpha, network.BPRbeta);
      }

      //Update flows and recalculate travel times for original path
      for(int k = 0; k < originalPath.Count; k++)
      {
         Link lk = (Link)originalPath[k];
         lk.FlowAtLink[timeIndex] -= divertedFlow;
         lk.SetTimeAtLink(timeIndex, network.BPRalpha, network.BPRbeta);
      }
   }

   //Now calculate travel time on the alternate path with diversion
   for(int j = 0; j < alternatePath.links.Count; j++)
   {
      Link lk = (Link)alternatePath.links[j];
      exDelayAlt += lk.TimeAtLink[timeIndex];
   }

   //Return the diverted flow
   return divertedFlow;
}
```

```
public double ExtraDelay(double timeInterval)
{
   return exDelayAlt*timeInterval;
}


public double Divert(long timeIndex, double delay, Node divertNode,
   Link incidentLink, Network network)
{
   //pathCostD is the travel time of the path -> computed in
   //traffic assignment.
   double Tkm = this.pathCostD + delay;

   //Travel time of alternative path
   double TkmAlt = 0D;

   //Find the shortest alternative route
   string keyForCosts = "N" + divertNode.NodeID.ToString() + "L"
      + incidentLink.LinkID.ToString();
   if(this.alternateTimes.ContainsKey(keyForCosts))
   {
      TkmAlt = (double)alternateTimes[keyForCosts];
   }
   else
   {
      //Add the travel times of the links before the diversion point
      //to the alternate route
      for(int i = 0; i < links.Count; i++)
      {
         Link lk = (Link)links[i];
         TkmAlt += lk.TimeAtLink[timeIndex];
         if(lk.ToNode.NodeID == divertNode.NodeID)
            break;
      }

      network.FindSP_LC(divertNode, timeIndex);
      TkmAlt += destinationNode.CostAtNode[timeIndex];
      alternateTimes.Add(keyForCosts, TkmAlt);
   }
```

```
         //Return the diverted flow
         return flow * (1 / (1 + Math.Exp(
            network.CMSalpha - network.CMSbeta * ((Tkm - TkmAlt)/TkmAlt))));
      }
      #endregion
   }
}
```

QueuingDiagram.cs

```
using System;
using System.Collections;
using System.IO;

namespace OptimalCMS
{
   /// <summary>
   /// Summary description for QueuingDiagram.
   /// </summary>
   public class QueuingDiagram
   {
      #region Local variables to hold queuing diagram parameters
      private Link incidentLink;
      private long timeIndex;
      private long numberIncrements;
      private long numberIncidentTimeInstances;
      private Network network;
      private bool includeAltPathTravelTime;
      #endregion

      #region Local variables to hold calculated values
      private double delay;
      private double cmsDelay;
      private long arraySizes;
      private CMS[] cmsList;
      private bool[] cmsFlagList;
      private double[] cmsIncidentDist;
      private Path[][] cmsPathsCur;
      private Path[][] cmsPathsNext;
```

```
private long nextIndex;
private bool isOverSaturated;
private QPoint[] arrivalCurve;
private QPoint[][] departureCurve;
private QueueCase queueCase;
private double incidentStartInterval;
#endregion

#region Local variables to hold statistical quantities
private double totalDiversion;
private long diversionCounter;
#endregion

#region Constructor logic
public QueuingDiagram(Link IncidentLink, long TimeIndex,
   long NumberIncrements, long NumberIncidentTimeInstances,
   Network QDNetwork, bool IncludeAltPathTravelTime)
{
   //Set link where the incident is simulated
   incidentLink = IncidentLink;

   //Set the time period of the simulated incident
   timeIndex = TimeIndex;

   //Set the number of increments used in calculating the
   //diverted arrival curve
   numberIncrements = NumberIncrements;

   //Set the number of incident time considered
   numberIncidentTimeInstances = NumberIncidentTimeInstances;

   //Network data
   network = QDNetwork;

   //Does the savings include a penalty for impact to drivers
   //on alternate routes?
   includeAltPathTravelTime = IncludeAltPathTravelTime;

   //Initialize statistical counters
   totalDiversion = 0D;
```

```
   diversionCounter = 0L;
}
#endregion

#region Calculations section (Public)
public void CalculateDelay()
{
   //
   //First do calculations without the presence of CMS
   //

   //Check if incident link is already oversaturated
   isOverSaturated = (incidentLink.FlowAtLink[timeIndex] >
      incidentLink.Capacity);
   nextIndex = timeIndex + 1L < network.NumTimePeriods ?
      timeIndex + 1L : 0L;

   //Do calculations based on queuing case
   if(incidentLink.FlowAtLink[timeIndex] < incidentLink.ReducedCapacity)
   {
      //Flow is less than the reduced capacity case -- no delay predicted
      delay = 0D;
   }
   else if(isOverSaturated)
   {
      //Case 3 - oversaturated case
      queueCase = QueueCase.CaseIII;
      delay = this.CaseIII();
   }
   else if(incidentLink.FlowAtLink[timeIndex] <
      incidentLink.FlowAtLink[nextIndex] )
   {
      //Case 1 - regular undersaturated case
      queueCase = QueueCase.CaseI;
      delay = this.CaseI();
   }
   else //if(!isRatioSmaller)
   {
      //Case 2 - close to saturation, but still undersaturated
      queueCase = QueueCase.CaseII;
```

```
      delay = this.CaseII();
   }
}

public void CalculateCMSDelay()
{
   //
   //textWriter.WriteLine("Starting to calculate delay with CMS...");


   //Reset delay
   this.cmsDelay = 0D;

   //If arrival rate is too low -> deterministic queuing predicts no delay
   if(incidentLink.ReducedCapacity >= incidentLink.FlowAtLink[timeIndex])
      return;

   //CMS distance calcs
   if(this.CMSDistance() == 0L)
   {
      this.cmsDelay = this.delay;
      return;
   }

   //Set the travel time on the incident link to some large value
   double[] incTimes = (double[])incidentLink.TimeAtLink.Clone();
   incidentLink.TimeAtLink[timeIndex] = System.Double.PositiveInfinity;
   incidentLink.TimeAtLink[nextIndex] = System.Double.PositiveInfinity;

   //Create delay calculation for each incident time instance
   double[] itiDelay = new double[this.numberIncidentTimeInstances];

   //Jagged arrays store current and next time period path lists
   cmsPathsCur = this.BuildPathFlowByCMSAndIncidentLink(timeIndex);
   cmsPathsNext = this.BuildPathFlowByCMSAndIncidentLink(nextIndex);

   //If no CMS are effective -> delay is the same as before
   if(this.FlagCMSWithoutPaths() == 0)
   {
      this.cmsDelay = this.delay;
```

```
   incidentLink.TimeAtLink = incTimes;
   return;
}

//Define the number of increments to use during incident
long numIncidents = queueCase == QueueCase.CaseI ? 1L :
   this.numberIncidentTimeInstances;

//Variable to hold current incident time
double divertTime = incidentLink.DetectAndProcess + cmsIncidentDist[0];

//The interval for which the reduced arrival curve is calculated
double updateIncrement, curTime, dissipationTime;

//Temp t, N
double t, N;

//The current index of the arrival curve
int k = 1;

//Variable to indicate next time period has been reached
bool nextPeriodReached = false;

//Resize the arrival curve array (increments + zero pt
//+ 1st diversion pt + non-cms queue cleared pt)
arrivalCurve = new QPoint[this.numberIncrements + 3];

//Add the zero point
arrivalCurve[0] = new QPoint(0D, 0D);

//Do calculations for each simulated incident start time
for(int j = 0; j < numIncidents; j++)
{
   //Calculate the diversion time
   divertTime += incidentStartInterval;

   //Calculate diversion start point
   t = divertTime - network.TripRateDurations[timeIndex];
   N = t < 0 ? divertTime * incidentLink.FlowAtLink[timeIndex]
      : network.TripRateDurations[timeIndex]
```

```
   * incidentLink.FlowAtLink[timeIndex]
   + t * incidentLink.FlowAtLink[nextIndex];
arrivalCurve[1] = new QPoint(divertTime, N);

//Calculate the update increment
dissipationTime = departureCurve[j][departureCurve[j].Length-1].t;
updateIncrement = (dissipationTime-divertTime)/this.numberIncrements;

//Set the current time to the diversion start time
curTime = divertTime;

//Default - next time period has not been reached
nextPeriodReached = false;

//Reset the arrival curve index
k = 1;

//Store the reduced current flow rate
double currentFlow = 0D;

while(curTime <= dissipationTime)
{
   //First calculate the expected delay
   double expectedDelay = this.ExpectedDelay(j, arrivalCurve[k]);

   //If expected delay == 0 then curve has been reached
   if(expectedDelay <= 0D)
   {
      while(k < this.numberIncidentTimeInstances + 1)
      {
         k++;
         arrivalCurve[k] = arrivalCurve[k-1];
      }

      if(queueCase == QueueCase.CaseIII)
      {
         //Check to see if incident clearance point should be added
         if(arrivalCurve[k].N < departureCurve[j][2].N)
            arrivalCurve[k] = departureCurve[j][2];
```

```
      //Finally add the dissipation point
      k++;
      arrivalCurve[k] = departureCurve[j][3];
   }
   else //Same calculations as above, but for Cases I & II.
   {
      if(arrivalCurve[k].N < departureCurve[j][1].N)
         arrivalCurve[k] = departureCurve[j][1];

      k++;
      arrivalCurve[k] = departureCurve[j][2];
   }

   //Break from the while statement
   break;
}
else
{
   //Then calculate the diverted vehicles
   if(!nextPeriodReached &&
      curTime > network.TripRateDurations[timeIndex])
         nextPeriodReached = true;
   currentFlow = this.ArrivalRateWithDiversion(arrivalCurve[k],
      nextPeriodReached, expectedDelay, updateIncrement);

   //Increment k, curTime, N
   k++;
   curTime += updateIncrement;
   N += updateIncrement * currentFlow;

   //Then add a point to the arrival curve
   arrivalCurve[k] = new QPoint(curTime, N);
}

//If diversion not significant, ignore the effect of CMS
//--CMS too far away/not enough flow!
if(k == 2 && currentFlow > 0.95D * incidentLink.FlowAtLink[timeIndex])
{
   this.cmsDelay = this.delay;
   incidentLink.TimeAtLink = incTimes;
```

```
            return;
         }
      }
      //Store delay
      itiDelay[j] = this.CalculateDelayArea(j);
   }
   //Set the travel time on the incident link back to normal
   incidentLink.TimeAtLink = incTimes;

   //Set average delay with CMS
   this.cmsDelay += this.MeanDelay(itiDelay);
}
#endregion

#region Calculations section (Private)
private double CaseI()
{
   //Define some times
   incidentStartInterval = 0D;

   //Calculate time to dissipate incident queue
   double dissipationTime = incidentLink.IncidentDuration*
      (incidentLink.FlowAtLink[timeIndex]-incidentLink.ReducedCapacity)/
      (incidentLink.Capacity-incidentLink.FlowAtLink[timeIndex]);

   //Resize arrays
   arrivalCurve = new QPoint[2];
   if(queueCase == QueueCase.CaseI)
   {
      departureCurve = new QPoint[1][];
      departureCurve[0] = new QPoint[3];
   }

   //Add points to arrival curve
   //
   //Origin point
   arrivalCurve[0] = new QPoint(0D, 0D);

   //Queue clearance time
   double t = incidentLink.IncidentDuration + dissipationTime;
```

```
   arrivalCurve[1] = new QPoint(t, incidentLink.FlowAtLink[timeIndex]*t);

   //Add points to departure curve
   //Origin Point
   departureCurve[0][0] = arrivalCurve[0];

   //Incident Removed
   t = incidentLink.IncidentDuration;
   departureCurve[0][1] = new QPoint(t, incidentLink.ReducedCapacity*t);

   //Queue clearance time
   departureCurve[0][2] = arrivalCurve[1];

   //Return area
   return this.CalculateDelayArea(0);
}

private double CaseII()
{
   //Create delay calculation for each incident time instance
   double[] itiDelay = new double[this.numberIncidentTimeInstances];

   //Define some times
   incidentStartInterval = network.TripRateDurations[timeIndex]
      /this.numberIncidentTimeInstances;
   double incidentTimeToPeriodEnd = network.TripRateDurations[timeIndex];

   double t, N;
   double dissipationTime;
   QPoint qpt;
   departureCurve = new QPoint[this.numberIncidentTimeInstances][];

   //Do calculation for each time instance
   for(int i = 0; i < this.numberIncidentTimeInstances; i++)
   {
      //Resize/Reinitialize arrays
      arrivalCurve = new QPoint[3];
      departureCurve[i] = new QPoint[3];

      //First set the time from the incident start to the end
```

```
//of the time period
incidentTimeToPeriodEnd -= incidentStartInterval;

//Calculate the queue dissipation time
dissipationTime = (incidentTimeToPeriodEnd*
   (incidentLink.FlowAtLink[timeIndex]
   - incidentLink.FlowAtLink[nextIndex])
   + incidentLink.IncidentDuration*(incidentLink.Capacity
   -incidentLink.ReducedCapacity))/
   (incidentLink.Capacity-incidentLink.FlowAtLink[nextIndex]);

//Simple Case I queuing if time to period end is greater than
//the calculated dissipation time.
if(incidentTimeToPeriodEnd >= dissipationTime)
{
   //No need to do calculations again since Case I
   //queuing does not depend on incident start time
   queueCase = QueueCase.CaseI;
   return this.CaseI();
}
else   //Case II
{

   //Add first point to both curves
   arrivalCurve[0] = departureCurve[i][0] = new QPoint(0D, 0D);

   //Add points to arrival & departure curves
   t = incidentTimeToPeriodEnd;
   N = incidentLink.FlowAtLink[timeIndex] * t;
   qpt = new QPoint(t, N);
   arrivalCurve[1] = qpt;

   //Check for queue clearance during incident -> departures
   //greater than arrivals
   if(incidentTimeToPeriodEnd < incidentLink.IncidentDuration &&
      incidentLink.ReducedCapacity*incidentLink.IncidentDuration >
      N + (incidentLink.IncidentDuration - t)*
      incidentLink.FlowAtLink[nextIndex])
   {
      dissipationTime = (N - t * incidentLink.FlowAtLink[nextIndex])
```

```
                /(incidentLink.ReducedCapacity -
                incidentLink.FlowAtLink[nextIndex]);

            N = dissipationTime * incidentLink.ReducedCapacity;
            qpt = new QPoint(dissipationTime, N);
            departureCurve[i][1] = qpt;
            arrivalCurve[2] = departureCurve[i][2] = departureCurve[i][1];
         }
         else
         {
            N += incidentLink.FlowAtLink[nextIndex] * (dissipationTime - t);
            t = dissipationTime;
            qpt = new QPoint(t, N);
            arrivalCurve[2] = qpt;

            //Add points to the departure curve
            t = incidentLink.IncidentDuration;
            N = incidentLink.ReducedCapacity * t;
            qpt = new QPoint(t, N);
            departureCurve[i][1] = qpt;
            departureCurve[i][2] = arrivalCurve[2];
         }

         //Store delay
         itiDelay[i] = this.CalculateDelayArea(i);
      }
   }
   //Return average delay
   return this.MeanDelay(itiDelay);
}

private double CaseIII()
{
   //Create delay calculation for each incident time instance
   double[] itiDelay = new double[this.numberIncidentTimeInstances];

   //Define some times
   incidentStartInterval = network.TripRateDurations[timeIndex]
      /this.numberIncidentTimeInstances;
   double incidentTime = 0D;
```

```
double t, N, dissipationTime;
QPoint qpt;
departureCurve = new QPoint[this.numberIncidentTimeInstances][];

//Dissipation time does not depend on incident start time
//for the oversaturated case.
dissipationTime = (network.TripRateDurations[timeIndex]*
   (incidentLink.FlowAtLink[timeIndex]-
   incidentLink.FlowAtLink[nextIndex])+
   incidentLink.IncidentDuration*(incidentLink.Capacity
   -incidentLink.ReducedCapacity))/(incidentLink.Capacity
   -incidentLink.FlowAtLink[nextIndex]);

//Arrival curve also does not depend on incident start time
//for the oversaturated case.
arrivalCurve = new QPoint[3];
arrivalCurve[0] = new QPoint(0D, 0D);

t = network.TripRateDurations[timeIndex];
N = t * incidentLink.FlowAtLink[timeIndex];
qpt = new QPoint(t, N);
arrivalCurve[1] = qpt;

N += (dissipationTime - t)*incidentLink.FlowAtLink[nextIndex];
qpt = new QPoint(dissipationTime, N);
arrivalCurve[2] = qpt;

//Do calculation for each time instance
//for(int i = 0; i < this.numberIncidentTimeInstances; i++)

//Only need to do calculation for first time instance.
//Difference in delay is constant!!!
for(int i = 0; i < this.numberIncidentTimeInstances; i++)
{
   //Resize/Reinitialize departure array
   departureCurve[i] = new QPoint[4];

   if(i == 0)
   {
```

```
      //First set the time of the incident start
      incidentTime = ((double)i) * incidentStartInterval;

      //Add points to the departure curve
      departureCurve[i][0] = arrivalCurve[0];

      N = incidentTime*incidentLink.Capacity;
      qpt = new QPoint(incidentTime, N);
      departureCurve[i][1] = qpt;

      t = incidentTime+incidentLink.IncidentDuration;
      N += incidentLink.IncidentDuration*incidentLink.ReducedCapacity;
      qpt = new QPoint(t, N);
      departureCurve[i][2] = qpt;

      departureCurve[i][3] = arrivalCurve[2];
   }
   else
   {
      departureCurve[i][0] = departureCurve[i-1][0];
      departureCurve[i][3] = departureCurve[i-1][3];

      //Difference in arrivals
      N = incidentStartInterval*incidentLink.Capacity;

      qpt = departureCurve[i-1][1];
      qpt.t += incidentStartInterval;
      qpt.N += N;
      departureCurve[i][1] = qpt;

      qpt = departureCurve[i-1][2];
      qpt.t += incidentStartInterval;
      qpt.N += N;
      departureCurve[i][2] = qpt;
   }

   //Store delay
   if(i < 2)
      itiDelay[i] = this.CalculateDelayArea(i);
}
```

```
   //Return average delay
   return itiDelay[0]+0.5*(itiDelay[1]-itiDelay[0])
      *(this.numberIncidentTimeInstances-1);
}

private double MeanDelay(double[] values)
{
   //Calculation of simple mean for double values in an array
   int numberObs = values.Length;
   double tot = 0D;
   for(int i = 0; i < numberObs; i++)
   {
      tot += values[i];
   }
   return tot/(double)numberObs;
}

private double ExpectedDelay(int inc, QPoint qpt)
{
   //Get the index of the last pt on the departure curve
   int lastIndex = queueCase == QueueCase.CaseIII ? 3 : 2;

   //If number of arrivals is greater than that at the queue clearance
   //then use the recovery curve, else use the reduced capacity curve
   if(qpt.N < departureCurve[inc][lastIndex-1].N )
      lastIndex--;

   //Now use similar triangles property to calculate delay
   QPoint qpt1 = departureCurve[inc][lastIndex-1];
   QPoint qpt2 = departureCurve[inc][lastIndex];

   //Calculate intersection of horizontal delay line with
   //reduced/recovery curve
   double timeOfIntersect = qpt2.t - (qpt2.t - qpt1.t) *
      (qpt2.N - qpt.N) / (qpt2.N - qpt1.N);

   return timeOfIntersect - qpt.t;
}

private double ArrivalRateWithDiversion(QPoint qpt, bool isNextTimePeriod,
```

```
   double delay, double updateIncrement)
{
   Path[][] pathLists;
   Path[] pathList;
   double flow;

   //Make sure original link flows and travel times are used
   if(includeAltPathTravelTime)
      network.RestoreUESolution();

   //Set the variables according to time period
   if(isNextTimePeriod)
   {
      flow = incidentLink.FlowAtLink[nextIndex];
      pathLists = cmsPathsNext;
   }
   else   //Current time period.
   {
      flow = incidentLink.FlowAtLink[timeIndex];
      pathLists = cmsPathsCur;
   }

   double origFlow = flow;

   //Loop through each CMS
   for(int i = 0; i < this.arraySizes; i++)
   {
      //Check first to see if the CMS is effective
      if(this.cmsFlagList[i] = false || cmsIncidentDist[i] >= qpt.t)
              break;

      //Otherwise start decrementing the flow
      pathList = pathLists[i];

      //Loop through each path
      for(int j = 0; j < pathList.Length; j++)
      {
         if(this.includeAltPathTravelTime)
         {
            flow -= pathList[j].DivertWithAltTravelTime(timeIndex,
```

```
                delay, cmsList[i].Link.ToNode, incidentLink, network);

             //Add extra delay to alternative routes
             this.cmsDelay += pathList[j].ExtraDelay(updateIncrement);
          }
          else
             flow -= pathList[j].Divert(timeIndex, delay,
                cmsList[i].Link.ToNode, incidentLink, network);
       }
    }

    //Add some statistical information
    double diversion = (origFlow-flow)/origFlow;
    totalDiversion += diversion;
    diversionCounter++;

    //Return the reduced flow
    return flow;
}

private double CalculateDelayArea(int dI) //short for departureIndex
{
    double area = 0D;

    for(int i = 0; i < arrivalCurve.Length-1; i++)
    {
       area += (arrivalCurve[i+1].t - arrivalCurve[i].t)
          * (arrivalCurve[i+1].N + arrivalCurve[i].N) * 0.5;
    }
    for(int i = 0; i < departureCurve[dI].Length-1; i++)
    {
       area -= (departureCurve[dI][i+1].t - departureCurve[dI][i].t)
          * (departureCurve[dI][i+1].N + departureCurve[dI][i].N) * 0.5;
    }

    return area;

}

private long CMSDistance()
```

```
{
   //List used so sort does not have to be programmed
   SortedList sortedCMS = new SortedList();
   double distance;

   //Deal with duplicate distances--arbitrarily make slightly different
   double duplicate = 0D;
   const double DUPLICATE_SCALE = 0.0001D;

   foreach(CMS cms in network.CMSTempList)
   {
      //Find the shortest path from the CMS link end node to all other nodes
      network.FindSP_LC(cms.Link.FromNode, timeIndex);

      //Now add to list with distance as key
      distance = incidentLink.FromNode.CostAtNode[timeIndex];
      if(distance != 0D && distance < System.Single.MaxValue)
      {
         try
         {
            sortedCMS.Add(distance, cms);
         }
         catch
         {
            duplicate++;
            distance *= (1 + duplicate * DUPLICATE_SCALE);
            sortedCMS.Add(distance, cms);
         }
      }
   }

   //Set up local arrays
   arraySizes = sortedCMS.Count;  //network.CMSTempList.Count;
   cmsList = new CMS[arraySizes];
   cmsIncidentDist = new double[arraySizes];
   sortedCMS.Values.CopyTo(cmsList, 0);
   sortedCMS.Keys.CopyTo(cmsIncidentDist, 0);

   //Return arraySizes
   return arraySizes;
```

```
}

private Path[][] BuildPathFlowByCMSAndIncidentLink(long TimePeriod)
{
   //Lists for storing paths
   ArrayList[] cmsPathLists = new ArrayList[(int)arraySizes];

   //Temporary variables
   long IndexOfCMS = -1L;
   long IndexOfIncLink = -1L;
   Path path;
   for(int k = 0; k < arraySizes; k++)
      cmsPathLists[k] = new ArrayList();

   //Build path lists
   for(int j = 0; j < incidentLink.Paths[TimePeriod].Count; j++)
   {
      path  = (Path)incidentLink.Paths[TimePeriod][j];
      IndexOfIncLink = path.Links.IndexOf(incidentLink);
      for(int i = 0; i < arraySizes; i++)
      {
         IndexOfCMS = path.Links.IndexOf(cmsList[i].Link);
         if(IndexOfCMS > -1L && IndexOfCMS < IndexOfIncLink)
         {
            cmsPathLists[i].Add(path);

            //Only add the path once -- IMPORTANT!!!
            break;
         }
      }
   }

   //Create array to return
   Path[][] returnPaths = new Path[(int)arraySizes][];
   int jaggedIndex;
   for(int j = 0; j < arraySizes; j++)
   {
      jaggedIndex = cmsPathLists[j].Count;
      returnPaths[j] = new Path[jaggedIndex];
      for(int k = 0; k < jaggedIndex; k++)
```

```
            returnPaths[j][k] = (Path)cmsPathLists[j][k];
      }
      return returnPaths;

   }

   private long FlagCMSWithoutPaths()
   {
      //Count the number of CMS with paths
      long cmsWithPaths = 0L;

      //Initialize the cms flag
      this.cmsFlagList = new bool[this.arraySizes];

      //Check to see if there are CMS with no paths
      for(int i = 0; i < this.arraySizes; i++)
      {
         if(cmsPathsCur[i].Length == 0)
         {
            this.cmsFlagList[i] = false;
         }
         else
         {
            this.cmsFlagList[i] = true;
            cmsWithPaths++;
         }
      }

      //Return the count
      return cmsWithPaths;
   }
   #endregion

   #region Return values
   public double Delay
   {
      get
      {
         return delay;
      }
```

```
   }

   public double CMSDelay
   {
      get
      {
         return cmsDelay;
      }
   }

   public double TotalDiversion
   {
      get
      {
         return totalDiversion;
      }
   }

   public long DiversionCounter
   {
      get
      {
         return diversionCounter;
      }
   }
   #endregion

}

public struct QPoint
{
   public double t, N;

   public QPoint(double t1, double N2)
   {
      t = t1;
      N = N2;
   }
}
```

```
   public enum QueueCase
   {
      CaseI,
      CaseII,
      CaseIII,
   }
}
```

Settings.cs

```
using System;
using System.Text;
using System.IO;

namespace OptimalCMS
{
   /// <summary>
   /// Summary description for Settings.
   /// </summary>
   public class Settings
   {
      //File paths
      private const string settingsFile = "\\OptimalCMS.set";
      private string settingsFilePath = "";

      //Output labels
      private string[] outLabels = {   "Number of Update Intervals = ",
                                 "Number of Incident Starts = ",
                                 "Include Alternate Path Travel Time = ",
                                 "Optimization Type = ",
                                 "CMS To Allocate = ",
                                 "Add/Delete = ",
                                 "Population Size = ",
                                 "Maximum Generations = ",
                                 "Crossover Probability = ",
                                 "Mutation Probability = ",
                                 "Elitism = ",
                                 "Selection Type = ",
                                 "Crossover Type = ",
```

```
                        "Mutation Type = ",
                        "Rank Weight = ",
                        "Tournament Size = "};

//Notes
private string[] outNotes = {   "",
"",
"",
"Notes",
"-----",
"1) Number of Update Intervals: Number of intervals to update arrival
   rate with diversion due to CMS (default = 10).",
"",
"2) Number of Incident Starts: Number of simulated incident start
   times (default = 10).",
"",
"3) Include the effect of travel time on alternate routes?: 0 = No,
   1 = Yes",
"",
"4) Optimization Type: 0 = Greedy Optimization, 1 = Genetic Optimization.",
"",
"5) CMS to allocate: Number of additional CMS.",
"",
"6) Add/Delete: 0 = Add to existing locations, 1 = Delete current
   locations.",
"",
"7-10) Genetic Algorithm parameters.",
"",
"11) Elitism: Genetic algorithm parameter = 1 if elitist strategy, = 0
   if no elitist strategy.",
"",
"12) Selection Type: 1 = Roulette Wheel Selection, 2 = Rank Selection,
   3 = Tournament Selection",
"",
"13) Crossover Type: 1 = One-Point Crossover, 2 = Uniform Crossover,
   3 = Fusion Crossover",
"",
"14) Mutation Type: 1 = Simple, 2 = Deterministic Decreasing,
   3 = Deterministic Increasing, 4 = Self-Adaptive",
"",
```

```
"15) Rank Weight: 1 < Rank Weight < 2. Defined as the ratio of the
   probability of selecting the best individual compared to an average
   individual (Rank Selection).",
"",
"16) Tournament Size: Size of selection pool for tournament selection.
   (= 2 for binary tournament)"};


#region Stored settings
//Queuing Diagram Settings
private long numUpdateIntervals;
private long numIncidentStarts;
private bool includeAltTravelTime;

//Optimization Settings
private bool greedyOptimization;
private bool geneticOptimization;
private long cmsToAllocate;
private bool addLocations;
private bool deleteLocations;

//Genetic Algorithm Settings
private long populationSize;
private long maximumGenerations;
private double pCrossover;
private double pMutation;
private bool elitism;

//More Genetic Algorithm Settings
private SelectionType selectType;
private CrossoverType crossType;
private MutationType mutateType;
private double rankWeight;
private long tournamentSize;


#endregion

#region Constructor
public Settings(string applicationPath)
```

```
{
   //Set the full application path
   settingsFilePath = applicationPath + settingsFile;

   //Load settings from a text file
   this.Load();
}

#endregion

#region Define access
public long NumUpdateIntervals
{
   get
   {
      return numUpdateIntervals;
   }
   set
   {
      numUpdateIntervals = value;
   }
}

public long NumIncidentStarts
{
   get
   {
      return numIncidentStarts;
   }
   set
   {
      numIncidentStarts = value;
   }
}

public bool IncludeAltTravelTime
{
   get
   {
      return includeAltTravelTime;
```

```
   }
   set
   {
      includeAltTravelTime = value;
   }
}

public bool GreedyOptimization
{
   get
   {
      return greedyOptimization;
   }
   set
   {
      greedyOptimization = value;
   }
}

public bool GeneticOptimization
{
   get
   {
      return geneticOptimization;
   }
   set
   {
      geneticOptimization = value;
   }
}

public long CMSToAllocate
{
   get
   {
      return cmsToAllocate;
   }
   set
   {
      cmsToAllocate = value;
```

```
   }
}

public bool AddLocations
{
   get
   {
      return addLocations;
   }
   set
   {
      addLocations = value;
   }
}

public bool DeleteLocations
{
   get
   {
      return deleteLocations;
   }
   set
   {
      deleteLocations = value;
   }
}

public long PopulationSize
{
   get
   {
      return populationSize;
   }
   set
   {
      populationSize = value;
   }
}

public long MaximumGenerations
```

```
{
   get
   {
      return maximumGenerations;
   }
   set
   {
      maximumGenerations = value;
   }
}

public double PCrossover
{
   get
   {
      return pCrossover;
   }
   set
   {
      pCrossover = value;
   }
}

public double PMutation
{
   get
   {
      return pMutation;
   }
   set
   {
      pMutation = value;
   }
}

public bool Elitism
{
   get
   {
      return elitism;
```

```
   }
   set
   {
      elitism = value;
   }
}

public SelectionType SelectType
{
   get
   {
      return selectType;
   }
   set
   {
      selectType = value;
   }
}

public CrossoverType CrossType
{
   get
   {
      return crossType;
   }
   set
   {
      crossType = value;
   }
}

public MutationType MutateType
{
   get
   {
      return mutateType;
   }
   set
   {
      mutateType = value;
```

```
   }
}

public double RankWeight
{
   get
   {
      return rankWeight;
   }
   set
   {
      rankWeight = value;
   }
}

public long TournamentSize
{
   get
   {
      return tournamentSize;
   }
   set
   {
      tournamentSize = value;
   }
}

#endregion

#region Save/Load
private void Load()
{
   //Open the settings file for reading
   StreamReader inStream = new StreamReader(settingsFilePath);

   //Read in data sequentially
   numUpdateIntervals = GetLongData(inStream.ReadLine());
   numIncidentStarts = GetLongData(inStream.ReadLine());
   //Get whether to include the alternative path travel time impact
   switch(GetLongData(inStream.ReadLine()))
```

```
{
   case 1:
      includeAltTravelTime = true;
      break;

   case 0:
      includeAltTravelTime = false;
      break;

   default:
      includeAltTravelTime = true;
      break;
}
//Get optimization type
switch(GetLongData(inStream.ReadLine()))
{
   case 0:
      greedyOptimization = true;
      break;

   case 1:
      greedyOptimization = false;
      break;

   default:
      greedyOptimization = true;
      break;
}
geneticOptimization = !greedyOptimization;
cmsToAllocate = GetLongData(inStream.ReadLine());
//Get add to current or delete current locations
switch(GetLongData(inStream.ReadLine()))
{
   case 0:
      addLocations = true;
      break;

   case 1:
      addLocations = false;
      break;
```

```
   default:
      addLocations = true;
      break;
}
deleteLocations = !addLocations;
//Finally, get GA settings
populationSize = GetLongData(inStream.ReadLine());
maximumGenerations = GetLongData(inStream.ReadLine());
pCrossover = GetDoubleData(inStream.ReadLine());
pMutation = GetDoubleData(inStream.ReadLine());
switch(GetLongData(inStream.ReadLine()))
{
   case 0:
      elitism = false;
      break;

   case 1:
      elitism = true;
      break;

   default:
      elitism = true;
      break;
}
switch(GetLongData(inStream.ReadLine()))
{
   case 1:
      selectType = SelectionType.RouletteWheel;
      break;

   case 2:
      selectType = SelectionType.Rank;
      break;

   case 3:
      selectType = SelectionType.Tournament;
      break;

   default:
```

```
      selectType = SelectionType.RouletteWheel;
      break;
}
switch(GetLongData(inStream.ReadLine()))
{
   case 1:
      crossType = CrossoverType.OnePoint;
      break;

   case 2:
      crossType = CrossoverType.Uniform;
      break;

   case 3:
      crossType = CrossoverType.Fusion;
      break;

   default:
      crossType = CrossoverType.OnePoint;
      break;
}
switch(GetLongData(inStream.ReadLine()))
{
   case 1:
      mutateType = MutationType.Simple;
      break;

   case 2:
      mutateType = MutationType.DeterministicDecreasing;
      break;

   case 3:
      mutateType = MutationType.DeterministicIncreasing;
      break;

   case 4:
      mutateType = MutationType.SelfAdaptive;
      break;

   default:
```

```
            mutateType = MutationType.Simple;
            break;
      }
      rankWeight = GetDoubleData(inStream.ReadLine());
      tournamentSize = GetLongData(inStream.ReadLine());

      //Close the file
      inStream.Close();
   }

   public void Save()
   {

      //Save the settings
      StreamWriter outStream = File.CreateText(this.settingsFilePath);

      //Write the settings
      WriteSettings(outStream);

      //Write notes
      for(int i = 0; i < this.outNotes.Length; i++)
         outStream.WriteLine(outNotes[i]);

      //Close the file
      outStream.Close();
   }

   public void WriteSettings(StreamWriter outStream)
   {
      int i = 0;
      outStream.WriteLine(outLabels[i++] + numUpdateIntervals.ToString());
      outStream.WriteLine(outLabels[i++] + numIncidentStarts.ToString());
      outStream.Write(outLabels[i++]);
      if(includeAltTravelTime)
         outStream.WriteLine("1");
      else
         outStream.WriteLine("0");
      outStream.Write(outLabels[i++]);
      if(greedyOptimization)
         outStream.WriteLine("0");
```

```
else
   outStream.WriteLine("1");
outStream.WriteLine(outLabels[i++] + cmsToAllocate.ToString());
outStream.Write(outLabels[i++]);
if(addLocations)
   outStream.WriteLine("0");
else
   outStream.WriteLine("1");
outStream.WriteLine(outLabels[i++] + populationSize.ToString());
outStream.WriteLine(outLabels[i++] + maximumGenerations.ToString());
outStream.WriteLine(outLabels[i++] + pCrossover.ToString());
outStream.WriteLine(outLabels[i++] + pMutation.ToString());
outStream.Write(outLabels[i++]);
if(elitism)
   outStream.WriteLine("1");
else
   outStream.WriteLine("0");
outStream.Write(outLabels[i++]);
if(selectType == SelectionType.RouletteWheel)
   outStream.WriteLine("1");
else if(selectType == SelectionType.Rank)
   outStream.WriteLine("2");
else if(selectType == SelectionType.Tournament)
   outStream.WriteLine("3");
outStream.Write(outLabels[i++]);
if(crossType == CrossoverType.OnePoint)
   outStream.WriteLine("1");
else if(crossType == CrossoverType.Uniform)
   outStream.WriteLine("2");
else if(crossType == CrossoverType.Fusion)
   outStream.WriteLine("3");
outStream.Write(outLabels[i++]);
if(mutateType == MutationType.Simple)
   outStream.WriteLine("1");
else if(mutateType == MutationType.DeterministicDecreasing)
   outStream.WriteLine("2");
else if(mutateType == MutationType.DeterministicIncreasing)
   outStream.WriteLine("3");
else if(mutateType == MutationType.SelfAdaptive)
   outStream.WriteLine("4");
```

```
   outStream.WriteLine(outLabels[i++] + rankWeight.ToString());
   outStream.WriteLine(outLabels[i++] + tournamentSize.ToString());
}

private long GetLongData(string inLine)
{
   StringBuilder streamIn = new StringBuilder(inLine);
   streamIn.Remove(0, streamIn.ToString().LastIndexOf("=") + 1);
   return Convert.ToInt32(streamIn.ToString());
}

private double GetDoubleData(string inLine)
{
   StringBuilder streamIn = new StringBuilder(inLine);
   streamIn.Remove(0, streamIn.ToString().LastIndexOf("=") + 1);
   return Convert.ToDouble(streamIn.ToString());
}



#endregion

#region enums

public enum SelectionType
{
   RouletteWheel,
   Rank,
   Tournament,
}

public enum CrossoverType
{
   OnePoint,
   Uniform,
   Fusion,
}

public enum MutationType
{
```

```
         Simple,
         DeterministicDecreasing,
         DeterministicIncreasing,
         SelfAdaptive,
      }

      #endregion

   }
}
```

Import.cs

```
using System;
using System.Data;
using System.Data.OleDb;
using System.Collections;

namespace OptimalCMS
{
   /// <summary>
   /// Summary description for Import.
   /// </summary>
   public class Import
   {
      public static void FromDatabase(string dataFileName, out Network roadNetwork)
      {
         //Static table names from database
         string[] tablenames = {"Netnode", "NETLINK", "ODTrips", "CMS"};
         //
         DataSet ds = new DataSet();

         //Clean up existing network
         roadNetwork = new Network((long)4);

         //Open the database and write data to dataset
         OpenDatabase(dataFileName, ds, tablenames);
         if (ds.Tables.Count == 0) {return;}

         //Import the node information
```

```
   ImportNodes(ds, tablenames[0], roadNetwork);

   //Import the link information
   ImportLinks(ds, tablenames[1], roadNetwork);

   //Import the O-D trips information
   ImportODTrips(ds, tablenames[2], roadNetwork);

   //Import the CMS information
   ImportCMS(ds, tablenames[3], roadNetwork);
}
private static void OpenDatabase(string datafile, DataSet ds,
   string[] tablenames)
{
   const string STR_ACCESS_SELECT = "SELECT * FROM ";

   //Create a dataset copy of the database
   OleDbConnection myAccessConn = new OleDbConnection(
      "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" + datafile);

   //Open the database
   try
   {
      myAccessConn.Open();
   }
   catch
   {
      return;
   }

   try
   {
      for (int i = 0; i < tablenames.Length; i++)
      {
         //Add tables to dataset
         ds.Tables.Add(tablenames[i]);
         OleDbCommand myAccessCommand = new OleDbCommand(
            STR_ACCESS_SELECT + tablenames[i],myAccessConn);
         OleDbDataAdapter myDataAdapter = new OleDbDataAdapter
            (myAccessCommand);
```

```
            myDataAdapter.Fill(ds,tablenames[i]);
         }
      }
      finally
      {
         myAccessConn.Close();
      }
   }

   private static void ImportNodes(DataSet ds, string nodeTableName,
      Network roadNetwork)
   {
      DataRowCollection dra = ds.Tables[nodeTableName].Rows;
      Node node;

      foreach (DataRow dr in dra)
      {
         //Create new node and add information
         node = new Node(dr[0], dr[1], dr[2], dr[3], (long)4);

         //Add node to node collection
         roadNetwork.AddNode(node);
      }
   }

   private static void ImportLinks(DataSet ds, string linkTableName,
      Network roadNetwork)
   {
      DataRowCollection dra = ds.Tables[linkTableName].Rows;
      Node fNode;
      Node tNode;
      Link link;
      foreach (DataRow dr in dra)
      {
         //Get the from and to nodes
         fNode = roadNetwork.GetNode((int)dr[1]);
         tNode = roadNetwork.GetNode((int)dr[2]);

         //Create new link and add information
         link = new Link((long)4, dr[0], fNode, tNode, dr[3], dr[4],
```

```
            dr[5], dr[6], dr[7], dr[8], dr[9], dr[10], dr[11],
            dr[12], dr[13]);

         //Add link to link collection
         roadNetwork.AddLink(link);

         //Update the incoming and outgoing links of the nodes
         fNode.AddOutLink(link);
         //tNode.AddInLink(link);
      }
   }

   private static void ImportODTrips(DataSet ds, string odTableName,
      Network roadNetwork)
   {
      DataRowCollection dra = ds.Tables[odTableName].Rows;
      ODTrips odTrips;
      bool firstrow = true;

      foreach (DataRow dr in dra)
      {
         //The first row in the OD matrix is the trip rate durations
         if (firstrow)
         {
            for (int i = 0; i < 4; i++)
            {
               roadNetwork.TripRateDurations[i] = Convert.ToDouble(dr[i+3]);
            }
            firstrow = false;
         }
         else
         {
            //Create the new OD trips objects
            odTrips = new ODTrips(roadNetwork.GetNode(Convert.ToInt32(dr[1])),
               roadNetwork.GetNode(Convert.ToInt32(dr[2])), dr[3], dr[4],
                  dr[5], dr[6]);

            //Add the OD trips to the OD trips list
            roadNetwork.AddODTrip(odTrips);
         }
```

```
         }
      }

      private static void ImportCMS(DataSet ds, string cmsTableName,
         Network roadNetwork)
      {
         DataRowCollection dra = ds.Tables[cmsTableName].Rows;
         CMS cms;
         Link link;

         foreach (DataRow dr in dra)
         {
            link = roadNetwork.GetLink(Convert.ToInt32(dr[1]));

            //Create the new CMS object
            cms = new CMS(link);

            //Add the new CMS to the CMS list
            roadNetwork.AddCMS(cms);
         }
      }
   }
}
```

Save.cs

```
using System;
using System.IO;
using System.Collections;
using System.Windows.Forms;
using System.Text;

namespace OptimalCMS
{
   /// <summary>
   /// Summary description for Save.
   /// </summary>
   public class Save
   {
      public static void OptimizationResults(Settings settings, double ttSavings,
```

```
   ArrayList optimalLinkResults, double[] coordY)
{
   //Save the optimization results
   SaveFileDialog saveDlg = new SaveFileDialog();
   saveDlg.Filter = "Text Files (*.txt)|*.txt|All Files (*.*)|*.*";
   saveDlg.FileName = "OptimizationResults.txt";
   saveDlg.DefaultExt = ".txt";
   saveDlg.CheckPathExists = true;

   //Output
   if(saveDlg.ShowDialog() == DialogResult.OK)
   {
      StreamWriter outStream = File.CreateText(saveDlg.FileName);

      //Write the date
      outStream.WriteLine(System.DateTime.Now.ToString());
      outStream.WriteLine();

      //Write the general settings
      settings.WriteSettings(outStream);
      outStream.Write(outStream.NewLine + outStream.NewLine);

      //Write the optimization results
      outStream.Write("Travel Time Savings : ");
      outStream.Write(ttSavings.ToString() + outStream.NewLine
         + outStream.NewLine);

      //Write the optimal CMS link locations
      outStream.Write("Optimal Link Locations : ");
      for(int i = 0; i < optimalLinkResults.Count; i++)
      {
         CMS cms = (CMS)optimalLinkResults[i];
         outStream.Write(cms.Link.LinkID.ToString());

         if(i != optimalLinkResults.Count - 1)
            outStream.Write(", ");
      }
      outStream.Write(outStream.NewLine + outStream.NewLine);

      //Write the incremental results
```

```
      string blankSpace;
      if(settings.GreedyOptimization)   //Greedy Optimization
      {
         outStream.WriteLine("CMS #  Benefit");
         outStream.WriteLine("-----  -------");
         blankSpace = new String(' ', 4);
      }
      else   //Genetic Optimization
      {
         outStream.WriteLine("Generation  Benefit");
         outStream.WriteLine("----------  -------");
         blankSpace = new String(' ', 9);
      }

      outStream.WriteLine();
      int k;
      for(int j = 0; j < coordY.Length; j++)
      {
         //Write the index
         k = j+1;
         outStream.Write(k.ToString());
         if(k < 10)
         {
            outStream.Write(blankSpace + "  ");
         }
         else if(k < 100)
         {
            outStream.Write(blankSpace + " ");
         }
         else //(k < 1000)
         {
            outStream.Write(blankSpace);
         }

         //Then write the benefit
         outStream.Write(coordY[j].ToString() + outStream.NewLine);
      }
      outStream.Close();
   }
}
```

```
public static void TrafficAssignmentResults(Network roadNetwork,
   bool writePaths)
{
   //Time period labels
   string[] PeriodString = {"AM Peak Period", "Midday Period",
      "PM Peak Period", "Overnight Period"};
   const string linkheader = "LinkID, FromNode, ToNode,
      TotalCapacity (vph), TotalFlow (vph), TravelTime (s)";
   const string pathheader = "Path Information: Flow,
      Link(i)[source], Link(i+1), ..., Link(j)[sink]";

   //Save the traffic assignment results
   SaveFileDialog saveDlg = new SaveFileDialog();
   saveDlg.Filter = "Text Files (*.txt)|*.txt|All Files (*.*)|*.*";
   saveDlg.FileName = "TrafficAssignmentResults.txt";
   saveDlg.DefaultExt = ".txt";
   saveDlg.CheckPathExists = true;

   //Output
   if(saveDlg.ShowDialog() == DialogResult.OK)
   {

      StreamWriter outStream = File.CreateText(saveDlg.FileName);
      //for (long j = 0; j < roadNetwork.NumTimePeriods; j++)
      //{
      long j = 0;
         outStream.WriteLine(PeriodString[j]);
         outStream.WriteLine(linkheader);

         foreach (Link link in roadNetwork.LinkList.Values)
            outStream.WriteLine("{0}, {1}, {2}, {3}, {4:F1}, {5:F1}",
               link.LinkID, link.FromNode.NodeID, link.ToNode.NodeID,
               link.Capacity, link.FlowAtLink[j], link.TimeAtLink[j]
               * 3600D);

         outStream.WriteLine("");

         if(writePaths)
```

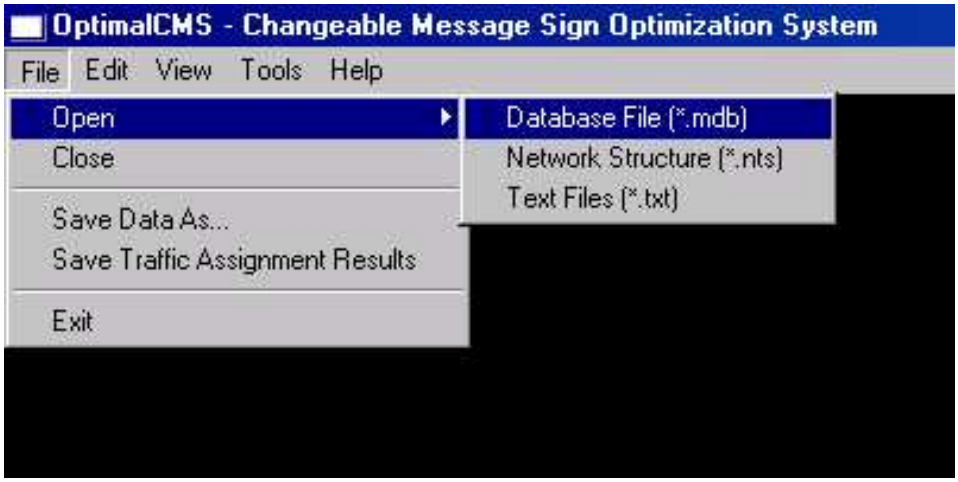
```
            {
               outStream.WriteLine(pathheader);

               IEnumerator pathIEnum = roadNetwork.PathLists[j].GetEnumerator();
               Path path; Link link;
               while(pathIEnum.MoveNext())
               {
                  path = (Path)pathIEnum.Current;
                  outStream.Write(path.Flow);
                  IEnumerator linkIEnum = path.Links.GetEnumerator();
                  while(linkIEnum.MoveNext())
                  {
                     link = (Link)linkIEnum.Current;
                     outStream.Write(", " + Convert.ToString(link.LinkID));
                  }
                  outStream.WriteLine("");
               }
            }

            outStream.WriteLine("");
            outStream.WriteLine("");
         //}
         outStream.Close();
      }
   }

   }
}
```

APPENDIX F: OptimalCMS MANUAL

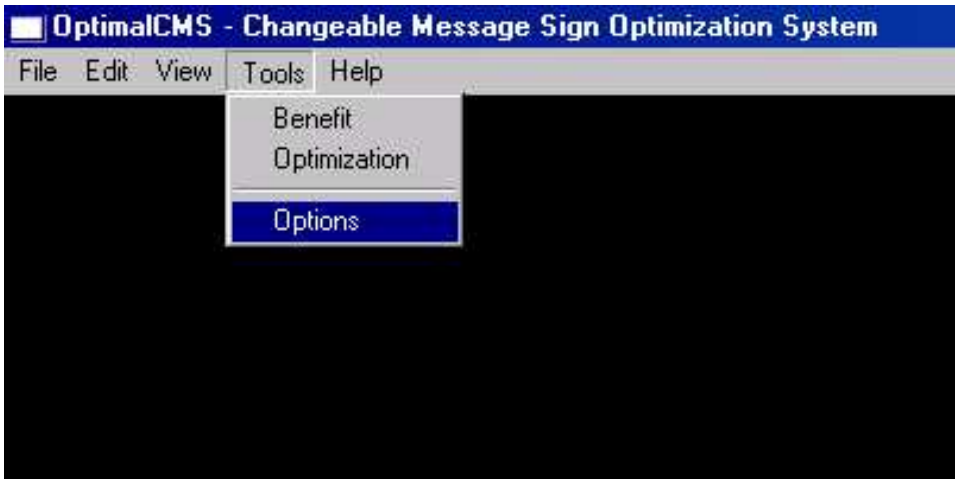
1. *Prepare Database :* Import data from a transportation planning software dataset to the nodes, links, and trips tables in the program database. Use the sample database distributed with the application to get started or create a new database conforming to the specifications of Appendix B. Make sure to include existing CMS locations in the appropriate table.

2. *Run Application :* Run OptimalCMS by browsing to the appropriate directory and double clicking on "OptimalCMS.exe".

3. *Import Data :* Click **File→Open Database File (*.mdb)** then select the database prepared in **1** from the Open File dialog. This step both imports the data and performs traffic assignment. Click **File→Save Traffic Assignment Results** once assignment has completed to save the traffic assignment results to a text file.



4. *Settings GUI :* A graphical interface is provided to modify queuing diagram and genetic algorithm parameters. Click **Tools→Options** and modify the settings, if necessary, before running the network benefit or optimization routines.



5. *Network Benefit :* Click **Tools→Benefit** to estimate the network benefit provided by the CMS locations listed in the database. Results are shown on a pop-up form.

6. *Location Optimization :* Click **Tools→Optimize Locations** to perform the optimization routines. A pop-up form displays results (graphically) at each iteration and the final results are shown, but not saved to the database, in the **Optimal CMS Link Locations** frame. Upon completion of the optimization calculations the user will be asked if they wish to save the optimization results in a text file, click **Yes** and choose a file location, otherwise click **No**.