UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

A STOCHASTIC SIMULATION MODEL TO SUPPORT DESIGNS OF EXPERIMENTS ON TRANSPORTATION EVACUATION PLANNING

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

DOCTOR OF PHILOSOPHY

By

HUONG T. L. PHAM
Norman, Oklahoma
2010

A STOCHASTIC SIMULATION MODEL TO SUPPORT DESIGNS OF
EXPERIMENTS ON TRANSPORTATION EVACUATION PLANNING

A DISSERTATION APPROVED FOR THE
SCHOOL OF INDUSTRIAL ENGINEERING

BY

____
Dr. Mary Court, Chair

____
Dr. Theodore Trafalis

____
Dr. Suleyman Karabuk

____
Dr. Amy McGovern

____
Dr. Guoqiang Shen

ACKNOWLEDGEMENTS

I owe my gratitude to many people who have given me the opportunity to make this dissertation possible. My heartfelt thanks go to Dr. Mary Court for her invaluable support and guidance throughout my graduate studies. Her outstanding intellectual works and enthusiasm inspired me to pursue higher academic goals; and her kindness and encouragement helped me overcome all challenges to finish what I set out to do. She is not only a great advisor but also a wonderful friend, like a family member to me.

I am thankful to Dr. Suleyman Karabuk, Dr. Amy McGovern, Dr. Guoqiang Shen, and Dr. Theodore Trafalis for being great mentors. This dissertation would not have been completed without their valuable comments, suggestions, and advice.

I acknowledge the faculty and staff of the School of Industrial Engineering for providing me an outstanding and well-rounded education. Thanks to Dung Phan for his help and suggestions for a more beautiful programming code. My appreciation also goes to all of my relatives and friends who supported me during the completion of this dissertation.

Last but not least, I would like to express my deepest thanks to my beloved family whom this dissertation is dedicated to:

- Thanks to my parents, Mr. Khang Pham and Mrs. Lieu Nguyen, for giving me life, raising me, supporting me, and loving me unconditionally. My parents and my sister Thuy Pham give me the happiest family I could ever dream of. I am also grateful to my mother for taking care of me and my baby Uyen Hoang when I needed her the most. I thank my grandmother for her devotion to her

children and grandchildren. Even though she is not with us anymore, my love for her is forever.

- My special thanks must go to my parents-in-law, Mr. Ky Hoang and Mrs. Thi Nguyen, and my brother-in-law Lam Hoang for their enormous support. Many thanks to my mother-in-law for looking after my baby Uyen during the toughest time of my graduate study, and for all of her help and understanding.
- Most of all, I am indebted to my husband Son Hoang for always being there by my side. His loving support keeps me motivated under any circumstances. Thanks to my sweet baby Uyen for joyful times as well as tough times that give me the blessing, pride, and happiness of being a mother.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

This research presents a new discrete-event simulation model, the DOE_EVAC, that can (i) effectively simulate alternative modes of transportation during evacuations, (ii) support designs of experiments, thus, provide the users (e.g., emergency planners and traffic engineers) with means to investigate "what-if" scenarios with sound statistical analysis capabilities, and (iii) allow the users to build and execute these models without having to know complex simulation or coding languages.

The contributions of this research are threefold. First, this research adopts designs of experiments to furnish users with statistical support to investigate "what-if" scenarios. Second, the DOE_EVAC model resolves existing issues of current simulation transportation evacuation modeling approaches by improving the initial system setup and supporting the stochastic traffic loading process. It allows users to implement and analyze various traffic management strategies, and is capable of rerouting traffic due to critical infrastructure failures during evacuation. DOE_EVAC also supports user-interruptions during simulation runs so that changes on the system can be executed. Finally, the DOE_EVAC model does not require the user to have any knowledge of specialized simulation or coding language as it relies only on four required (and one optional) data files supplied by the users to execute. A sample design of experiment is illustrated to show the multiple simulation run capabilities of the DOE_EVAC model and the ability of the DOE_EVAC to allow users to manipulate data that are typically inaccessible in existing evacuation models.

CHAPTER 1

INTRODUCTION

Attempts to develop large-scale simulation transportation evacuation systems to support emergency decision makers have been extensive over the last few decades (Sheffi, Mahmassani, & Powell, 1982; McLean, Moeller, Desrosiers, & Urbanik, 1983; Urbanik, Moeller, & Barnes, 1988; Hobeika & Kim, 1998; Hobeika, Kim, & Beckwith, 1994; Hiramatsu, 1983; Han, 1990; Tufekci & Kisko, 1991; Rathi, 1994; Pidd, Silva, & Eglese, 1996; Wu et al., 2007). While these existing decision support systems are commendable, they fail to provide emergency authorities the means to analyze the impact large populations and transportation infrastructure have on disaster mitigation and evacuation strategies. Experimental designs cannot be and were not conducted to find critical factors that influence the output performance measures of the simulation models, hence, "what-if" investigations cannot be performed.

Similarly, decisions in existing models are made on deterministic simulation approach and results; thus, they cannot reflect the uncertainties and the randomness occurring during real-life evacuation. Conclusions are drawn upon one observation; and no confidence intervals are offered to confirm the reliability of the final outputs/results. No conclusions should be drawn on a stochastic system when only one observation is available to study.

Additionally, most existing models do not support real-time simulation. They run the simulation and update the traffic conditions of the evacuation at fixed-time intervals. This approach does not capture the system state when instantaneous events occur and does not skip over inactive periods of time. It is far less efficient than discrete-event simulation in which the state of the system only changes when an event occurs at a real-valued time point.

In this research, a discrete-event simulation transportation evacuation model, DOE_EVAC, has been developed. The new model has the capability to effectively simulate alternative modes of transportation during evacuations. It also supports designs of experiments; thus, provide users (e.g., emergency planners and traffic engineers) with means to investigate "what-if" scenarios with robust statistical analysis capabilities. Finally, DOE_EVAC enables users to build and execute these models without having to know complex simulation or coding languages. The ultimate goal is to make parameters of interest readily accessible and easily changeable so that designs of experiments on these parameters can be performed; thus, provides users with the statistical support to identify the most important factors influencing time to evacuate.

This research contributes a number of substantial improvements to large-scale simulation transportation evacuation modeling. First, "what-if" scenarios can finally be investigated owing to the implementation of designs of experiments. Second, the DOE_EVAC model has resolved the existing issues of current simulation transportation evacuation modeling approaches by improving the initial system setup and supporting the stochastic traffic loading process. Realistic and desirable features such as allowing users to implement and analyze various traffic management strategies, rerouting traffic due to critical infrastructure failures during evacuation, or user-interruptions and system modifications during simulation runs have also been incorporated. In addition, the DOE_EVAC relies on four required (and one optional) data files supplied by the users to execute, and does not require the user to have in-depth knowledge of specialized simulation or coding languages. A sample design of experiment is illustrated to show the multiple simulation run capabilities of the DOE_EVAC model and the ability of the DOE_EVAC to allow users to manipulate data that are typically inaccessible in other evacuation models.

The dissertation is outlined as follows:

1. Chapter 2 presents the literature review of existing large-scale simulation transportation evacuation models. The content of this chapter has been published in a review article by Pham, Pittman, and Court (2008).
2. Chapter 3 describes the DOE_EVAC model and its model logic.
3. Chapter 4 provides the validation of DOE_EVAC's behavior on a real-world transportation network.
4. Chapter 5 shows how to use DOE_EVAC's supplied parameters of interest to perform designs of experiments and to draw statistically significant conclusions on the simulation model output.
5. Chapter 6 contains the summary, highlights the contributions, and recommends some potential future research points.

CHAPTER 2

LITERATURE REVIEW*

Most early literature on large-scale simulation transportation evacuation models primarily deal with civil emergency defense scenarios such as nuclear power facility incidents. They are *Network Evacuation I* (Sheffi et al., 1982), *Calculates Logical Evacuation and Response* (McLean et al., 1983), *Interactive Dynamic Network Evacuation* developed by KLD Associates in 1984 (Urbanik et al., 1988), *Mass Evacuation* (Hobeika & Jamei, 1985; Hobeika & Kim, 1998), and *Transportation Evacuation Decision Support System* (Hobeika et al., 1994).

Other applications include *Net Structure Analyzing System IV* (Hiramatsu, 1983) for earthquake evacuation, *Transportation Evacuation System* (Han, 1990), *Regional Evacuation Modeling System* (Tufekci & Kisko, 1991), *Oak Ridge Evacuation Modeling System* (Rathi & Solanki, 1993; Rathi, 1994), *Configurable Emergency Management and Planning System* (Pidd et al., 1996), and *Dynamic Discrete Disaster Decision Simulation System* (Wu et al., 2007).

The 11 existing evacuation models are now assessed via their three simulation approaches: macroscopic, microscopic and mesoscopic. The purpose of this review is to evaluate the modeling methodologies, identify the gaps and issues of these models, and show the pitfalls of current data analysis trends.

---

* Adopted from Pham, Pittman, & Court (2008)

2.1 Macroscopic Simulation

Macroscopic simulation operates based on the deterministic relationships of aggregate speed/density and demand/capacity of the traffic stream. It does not track individual vehicles, but "considers platoons of vehicles and simulates traffic flow in brief time increments" (Jeannotte, Chandra, Alexiadis, & Skabardonis, 2004). Hence, macroscopic models require less computer memory and storage; and they are suitable for scenarios with large-scale networks and long time periods. However, this aggregate-flow characteristic adversely impacts the ability to analyze transportation improvements in detail. Further discussion of macroscopic model characteristics can be found in May (1990).

Macroscopic simulation models include *Network Evacuation I* (Sheffi et al., 1982), *Net Structure Analyzing System IV* (Hiramatsu, 1983), *Mass Evacuation* (Hobeika & Jamei, 1985; Hobeika & Kim, 1998), *Transportation Evacuation System* (Han, 1990), *Regional Evacuation Modeling System* (Tufekci & Kisko, 1991), and *Transportation Evacuation Decision Support System* (Hobeika et al., 1994).

2.1.1 Network Evacuation 1 (NETVAC1)

NETVAC1 (Sheffi et al., 1982) is a fixed time simulation model initially developed to estimate traffic patterns and network-evacuation clearance time on a road network surrounding nuclear power plant sites. The model is anchored in mathematical/analytical relationships among significant traffic variables such as flow, speed, density, and queue length. The route selection mechanism of NETVAC1, however, is dynamic by means of (i) driver's choice of outbound link (turning movements) while approaching intersections with respect to his/her prior knowledge of the network as well as a myopic view of forefront traffic conditions, and (ii) user-specified priority at unsignalized intersections.

The preference factors, PF_k , and speeds $U_k(t)$ of outbound links set up the probability of driver's choice $P_j(t)$ of link *j* over others links at time *t*:

$$P_j(t) = \frac{PF_j \cdot U_j(t)}{\sum_k PF_k \cdot U_k(t)} \tag{1}$$

where *k* represents all outbound links including link *j*, PF_j is preference factor of link *j* and $U_j(t)$ is speed of link *j*.

Unsignalized intersections are treated according to a priority scheme pertaining to either a primary or secondary approach. Vehicles from a secondary approach can be emitted into a primary approach if residual intersection capacity exists from the primary approach; otherwise, NETVAC1 allows small capacity for vehicles from the secondary approach to sneak into the primary approach.

The NETVAC1 simulator includes the *link pass* which calculates the number of vehicles moving along the link; and the *node pass* which calculates flow from each inbound link to each outbound link at a given intersection. At each simulation interval, the node pass and the link pass are executed once for every node and link in the network. Note that the simulation interval is user-specified, but its maximum is strictly less than the minimum free flow link travel time.

The vehicle average speed in the link pass process $U(t)$ are obtained via the current density of moving vehicles $K(t)$, the jam density per lane *KJ* and the free flow speed *UF*:

$$U(t) = UF \cdot \left(1 - \frac{K(t)}{KJ}\right) \tag{2}$$

This speed is then used to compute the *link flow*—the number of vehicles reaching the downstream node of the link and the excess vehicular capacity available for the next iteration. The flow transferred from inbound link *i* into outbound link *j* at a given intersection, $M_{ij}(t)$, is subjected to two constraints: total flow that can be moved out of link *i*, $VI_i(t)$, and total flow that can be moved into link *j*, $VO_j(t)$.

$$M_{ij}(t) = VI_i(t) \cdot P_{ij}(t) \cdot \frac{VO_j(t)}{\sum_j VI_i(t) \cdot P_{ij}(t)} \tag{3}$$

where $P_{ij}(t)$ is the share of drivers coming from a given link *i* who choose to move into link *j*. Each $P_{ij}(t)$ can be calculated using Equation 1.

At each simulation interval, the moving time at a given intersection is determined via (i) the "green time" of each incoming direction for the signalized intersection, or (ii) the "equivalent green time", which is calculated as the fraction each incoming flow over the total incoming flows, for the unsignalized intersection.

Despite of its dynamic route selection approach, several shortcomings remain in NETVAC1 that can diminish the integrity of the evacuation model. First, all vehicles simulated by NETVAC1 are assumed to simultaneously enter the network at the beginning of the evacuation. This loading pattern which is not a time-dependent loading pattern can 'blow up' the system with inflated congestion, queues, and so forth. Second, vehicles egress the affected area without consideration of their desired final destination. Finally, link preference factor must be determined by the users; thus, "introducing considerable subjectivity into the process" (Abkowitz & Meyer, 1996).

2.1.2 Net Structure Analyzing System IV (NESSY-IV)

Application of the macroscopic simulation model NESSY-IV (Hiramatsu, 1983) is not limited to, but primarily associated with earthquake emergency. Population in NESSY-IV is classified into four groups (Office, Depart, Street and Subroad) due to their recent activities and can be evacuated via walk/foot, bus, car, or metro modes.

NESSY-IV's transportation network is modified so that the simulation model only needs to deal with nodes, not links. Each node in the network contains information on its node level, two level thresholds, inflow threshold, and modify type. The *node level* shows the current flow at the node; and the *upper* and *lower level thresholds* describe the maximum and minimum capacities of the node. If the *node level* is greater than the *upper level threshold*, the node is overflow; and if the *node level* is smaller than the *lower level threshold*, the node is underflow. The allowable increase of inflow within a unit of time is bound by the *inflow threshold*. The *modify type* determines how the node status is changed due to thru-traffic, for example type *0* associates with "no change", and type *1* associates with "inflows to the node are cut if the node is overflow".

The network flow model works exclusively based on the relationship between adjacent nodes in the network including node preceding status, mass flow or information flow transfer, and node following status. The mass flow transfer from node A to node B at time *i* is calculated through two types of output functions: a linear type (Eq. 4) and a saturated type (Eq. 5).

$$\begin{cases} A(i+1) = A(i) - A(i) * p \\ B(i+1) = B(i) + A(i) * p \end{cases} \tag{4}$$

$$\begin{cases} A(i+1) = A(i) - dv \\ B(i+1) = B(i) + dv \end{cases} \text{ in which } \begin{cases} dv = P(i) & \text{if } A(i) \geq \sum P \\ dv = A(i) * P(i) / \sum P & \text{otherwise} \end{cases} \tag{5}$$

where $P(i)$ represents the threshold of flux from node A to node B and p is a rate of P to the total flow from node A. The information flow transfer from node A to node B at time i is computed as follows:

$$\begin{cases} A(i+1) = A(i) \\ B(i+1) = B(i) + A(i) * p \end{cases} \tag{6}$$

The network flow model of NESSY-IV is simple and does not include the traffic control and traffic management strategies as other evacuation models. Additionally, because of the generation of dummy nodes to adjust delay time between two nodes, significant computer storage is required. Thus, NESSY-IV is suitable for small area evacuations only.

2.1.3 Mass Evacuation (MASSVAC)

MASSVAC (Hobeika & Jamei, 1985; Hobeika & Kim, 1998) simulation model is designed for providing assessment and analysis of urban area evacuation plans. The MASSVAC construct contains three interrelated modules:

1. *Community and disaster type module* delineates the geometric shape of the affected area, its neighborhood (urban or rural), and the characteristics of the disaster (natural or man-made).
2. *Population distribution module* categorizes population into permanent and transient population. Population density of the permanent population, who dwells inside the affected area, is classified by age and household size. For the transient population who are traveling through the affected area, if the evacuation is long range—when people have a long time to evacuate—the transient population may be ordered to return home; and if evacuation time is

a short range, they may be directed to situ shelters. In this way, the population density and destinations can be obtained for the transient population. The vehicle utilization and transportation modes (auto, mass transit, etc.) are also clarified in this module.

3. *Network evacuation module* includes detailed descriptions of highway network topology and traffic management strategies such as traveling restrictions (vehicles must emanate from origin to the closest exit without traveling on links toward the nuclear plant), shelter deficiency management (the user must specify additional shelters), intersection control, usages of reserved lanes for special vehicles (for example shoulder lanes for emergency management vehicles or HOV lanes for high occupancy vehicles) and contraflow implementation.

The simulation clock time in MASSVAC is advanced by a finite simulation interval of 15 to 60 min. Once the simulation starts, vehicles enter the network following an S-shaped curve (Figure 1) of the logit-based function:

$$\text{Current Trip}\left(Origin,\ Destination\right) = \frac{Trip(Origin, Destination)}{1 + e^{-Z*(ID-H)}} \tag{7}$$

where Z represents the slope of the logit curve, *ID* is the simulation interval, and *H* is the time at which half of the population is loaded. Basically, a portion of population at each origin is loaded onto the network in each simulation interval and the S-shaped curve shows the cumulative population loaded at particular time. It is obvious that the loading rates are the same at all origins. This loading pattern does not accurately capture the stochastic features of the evacuation departure process in which the departure time of evacuees at each origin instantly varies and the loading rates are absolutely different among origins.

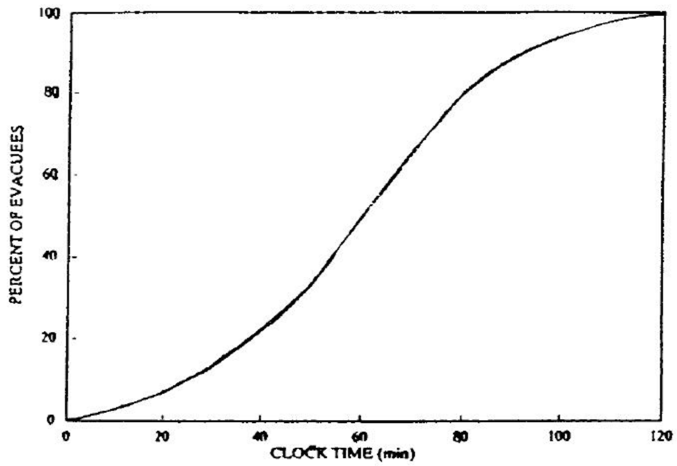


Figure 1: Logit Curve (Hobeika & Kim, 1998)

The Bureau of Public Roads (BPR) function (BPR 1964),

$$t = t_f \left[1 + 0.15 \left(\frac{\upsilon}{c} \right)^4 \right] \tag{8}$$

where t_f is link free-flow travel time, υ is link volume and c is link capacity, is adopted to calculate travel times on each link:

$$Travel\ \ Time = \begin{cases} t & if\ \upsilon/c < 1 \\ \min\left(t,\ 8t_f\right) & if\ 1 \le \upsilon/c \le 1.5 \\ \min\left(t,\ 10t_f\right) & if\ \upsilon/c > 1.5 \end{cases} \tag{9}$$

MASSVAC facility is limited in handling unsignalized intersections as well as beyond-4-leg intersections. For signalized intersections, vehicular volume Q_a discharged to outbound link (a) on major road 1 at the given intersection is calculated as follow:

$$Q_a = \frac{Q_1}{Q_1 + Q_2} * 1{,}800 \tag{10}$$

where Q_1 and Q_2 are critical volumes on major road 1 and minor road 2, respectively. The saturation flow rate is assumed to be 1,800 vehicles per hour. The vehicles then dissipate on the links following a regression model for mixed vehicles:

$$Flow(Q) = 74.3 * Density - 0.75 * Density^2 \tag{11}$$

where flow is the number of travel units (vehicles or passengers) traversing a given facility in a unit time (e.g., vehicles per hour) and density is the number of travel units traversing in a unit distance (e.g., vehicles/mile).

MASSVAC 3.0 implements Dial's probabilistic traffic assignment to handle evacuation route selection. MASSVAC 3.0 (1985) was updated to MASSVAC 4.0 (1998) to provide additional modeling features of the enhanced user equilibrium traffic assignment methodology.

Advances in modeling population socioeconomic characteristics and traffic management strategies increase MASSVAC's integrity and level of application. However, its brief touch on these subjects cannot exclusively demonstrate the evacuation process, which requires in-depth features and process analysis. Also, like most of the other macroscopic simulation models, MASSVAC is not a stochastic analysis tool. Randomization is not taken into consideration, thus the uncertainty that occurs during real-world evacuations cannot be demonstrated in MASSVAC.

2.1.4 Transportation Evacuation System (TEVACS)

At the core of TEVACS system (Han, 1990) is an enhanced version of the NETVAC1 simulation model, developed for different styles of transportation infrastructure. The distinctions between the two models are reflected in the vehicle usage manner and the deployment of public evacuation routes and transport stations. Since the dominant transportation modes in TEVACS researched networks are public transportation and motorcycles, TEVACS deals with multiple types of vehicles (car, pick-up van, bus, truck, motorcycle and bicycle) rather than homogeneous types (vehicle) as in NETVAC1, and the origins in TEVACS include public transits or stations. Note that different attributes of multiple vehicles extremely affect traffic conditions as well as trip loading rates. Also, TEVACS's improved features such as trip generation, GUI, traffic control and traffic management strategies provide a more useful tool for emergency management. However, TEVACS's algorithms are the same as those of NETVAC1. So, it too suffers from the impractical traffic loading pattern and the unreasonable destination choice.

2.1.5 Regional Evacuation Modeling System (REMS)

REMS (Tufekci & Kisko, 1991) is a decision support system software mainly used for traffic emergency control and management. In order to achieve the objectives established for the system, the regular transportation network is modified to accommodate REMS's designated network analysis as follows:

1. First, to ensure the intolerance of link's static capacity, a new node is placed in the middle of the link to split the link into two parts of which the front contains the link's original characteristics and the back has the infinite static and dynamic capacities. The intersection node performance is then converted to link behavior by being duplicated and connected to each other via a "dummy" link. This "dummy" link contains the dynamic capacity of the

intersection and the traverse time of zero. The modified transportation network is called the Intersection Augmented Network (IAN).

2. Second, a time unit is established as a benchmark of the time dimension to reveal the updated system state. All traffic variables, such as link travel time, capacity and flow rate, are converted to this time period. Consider expanding the network in T time periods, each node in the network is replicated T times and each copy of a node is connected to its next time copy. If there is a link between node A and node B in original network or IAN with link travel time of *t* time periods, the j^{th} copy of A is connected to the $(j+t)^{th}$ copy of B until (j+t) is larger than T. Figure 2 illustrates the logic of this Time Expanded Network.

The transportation network once modified can alleviate model complexity to handle time dimension, determine evacuation routes, and incorporate network adjustments, such as blocking links or closing intersections during the evacuation process.

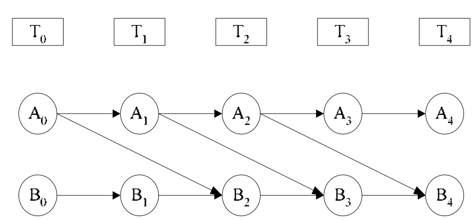


Figure 2: Time Expanded Network with t = 2

REMS provides three different methodologies to calculate network clearing time: discrete event simulation, linear programming (LP) model and network flow model using dynamic network representation. All of these methods control the vehicle movements in an aggregate level. The LP model, which is actually a multiobjective optimization LP, exploits the analytical user equilibrium assumptions to find the smallest network clearing time. The network flow model, on the other hand, applies the same dynamic loading rate as MASSVAC to estimate the number of vehicles entering the network at each time period and estimate the time minimizing flow pattern of the network for each Origin-Destination (O-D) pair. The advantage of the dynamic model is in its capability to identify bottlenecks and to simply block roads or intersections just by removing unwanted copies of nodes and links. Note that no formulations or detail algorithms are provided for the LP and network flow model in the literature.

The drawback of REMS's LP and the network flow models is that both are heuristic methods, which only yield near optimal solutions. Also, the nature of the REMS modified transportation network can exceed the demand of computer storage; hence the software can only be used for analyzing small-scale transportation networks.

2.1.6 Transportation Evacuation Decision Support System (TEDSS)

Strongly based on the MASSVAC 3.0 approach, TEDSS (Hobeika et al., 1994) is a microcomputer software package to assist in the development of evacuation plans for the Surry and North Anna nuclear power stations in Virginia. TEDSS employs Dial's algorithm (1971) to assign traffic, and estimate traffic bottlenecks as well as evacuation time. Except for its improved graphical user interface (GUI), and add-in socioeconomic characteristics (such as size of labor force, number of school attendees), features of TEDSS are the same as those of MASSVAC 3.0. Consequently, it too does not provide a stochastic environment for factor analysis to be performed.

2.2 Microscopic Simulation

Microscopic simulation or micro-simulation models are perhaps the most applied dynamic traffic models nowadays. The core characteristic that makes microscopic simulation prevail over other simulation approaches is its ability to mimic the behavior of every individual vehicle entering the transportation system realistically. Microscopic simulation modeling can be used to analyze systems with the highest level of traffic details including disaggregate relations among vehicles and traffic control. The disadvantage of microscopic simulation is its excessive requirements of computer time and storage for running and calibrating the model. This constrains the size of analyzed transportation network as well as the possible number of simulation runs. Detail discussion of microscopic model characteristics can be found in May (1990).

Generally, microscopic simulation approach consists of two different types: fixed-time-interval simulation and discrete-event simulation. The fixed-time-interval simulation divides a simulation run into very small time intervals of seconds or sub-seconds. After each interval, all vehicles are determined for possible behavior and movements and then moved to a new position. The total required computation per link is proportional to the product of the number of time intervals and the number of vehicles that traverse through the link. This procedure is far less efficient than that of discrete-event simulation of which computation per link is only proportional to the number of vehicles that traverse through the link. The state of the system in discrete-event simulation only changes when an event occurs at a real-valued time point. For example, when a vehicle arriving into the system or a change of signal phase at a controlled intersection.

Reviewed microscopic simulation models include *Calculates Logical Evacuation and Response* (McLean et al., 1983), *Configurable Emergency Management and Planning System* (Pidd et al., 1996), and *Dynamic Discrete Disaster Decision Simulation System* (Wu et al., 2007).

2.2.1 Calculates Logical Evacuation and Response (CLEAR)

CLEAR (McLean et al., 1983) was developed for the U.S. Nuclear Regulatory Commission (NRC) to provide a means of simulating vehicle movements and estimating network clearing time during an evacuation due to a nuclear plant emergency. This microscopic simulation model analyzes individual vehicles on only the primary road network of the *Emergency Planning Zone* (EPZ) – the area surrounding the nuclear plant that is possibly contaminated by the incident. Three buffers of 2 miles, 5 miles and 10 miles are generated in the vicinity of the nuclear plant. CLEAR then divides each buffer into 8 identical sectors by geographical direction: north, northeast, east, southeast, south, southwest, west and northwest. The roads eventually are divided into several road segments characterized by road attributes such as length, number of lanes, free flow speed, consecutive segments, and population density. Interacting road segments form an evacuation tree (highlighted in Figure 3), which can spread in multiple zones.

Note some of the drawbacks to the model. The possible number of vehicles served by one road segment is proportional to the length of that road segment. In other words, the vehicles appear evenly spaced in road segments; and the longer the road segment, the more the possible number of loaded vehicles. By using a random number generator, vehicles are arbitrarily assigned starting positions and spaces along the segment. This loading pattern does not indicate the actual starting positions for real situations. For example, when vehicle movement starts at the vehicle's initial resting position (parking slot or drive way in front of the house). It is also time-constrained given that all vehicles must enter the network during the maximum allowable departure time (the sum of maximum notification time and maximum individual preparation time).

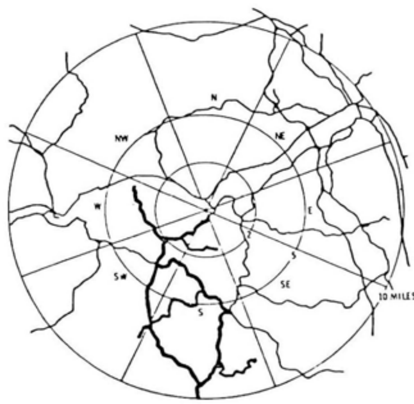


Figure 3: Designation of Evacuation Trees (McLean et al., 1983)

The movement of vehicles is driven by the traffic relationships between a road segment and its next road segment—called the *link*. These relationships—which are affected by loading rate, queuing system, vehicle capacity, vehicle density, and velocity of travel—are simple and transform in short increments of time (nearly 12 s).

CLEAR has three types of queues:

1. *The random queue* lists vehicles with a designated starting position.
2. *The loading queue* lists loaded vehicles yet not moved.
3. *The back-up queue* handles traffic jams based on link capacity and maximum allowable density (number of vehicles traversing on link).

In every increment of time, the road segment is loaded and vehicles are replaced from the random queue to the loading queue of the corresponding road segment.

Whenever a vehicle is advanced from a road segment to a full link, the vehicle is placed in the back-up queue of that link. The total queue of each road segment chronologically lists vehicles from the loading queue and then the back-up queue. Vehicles can only be released from a queue when adequate space on the road segment is available.

The velocity of movement on a road segment is affected by the density of traffic, which is altered in every vehicle movement:

$$V = \frac{f * l * d}{n} \tag{12}$$

where V—velocity of travel, f—free-flow rate, l—number of lanes, d—length of segment, and n—current link density. As long as the free flow presentation of the road segment exists, the travel speed is at its nominal value. Otherwise, it decreases linearly and traffic jams happen when it goes down to minimum speed of regular travel (15 mph). Vehicles, in that case, are added to the back-up queue.

CLEAR operates intersections via relative vehicle densities—not signalized intersections, i.e. vehicles on road segment with higher densities are allowed to move to the next link prior to those from a lower density road segment. This approach is appropriate for intersections controlled by traffic officers who must have full information of vehicle densities along road segments. However, it eliminates the ability to handle traffic logic involving green/red time, stop signs, or conflict approaches and so forth.

The advantages of CLEAR lies in its ability to direct traffic via a traffic management approach and to identify traffic difficulties while lessening computer memory requirements. Conversely, simplifying the assumptions for traffic handling diminishes model reliability and renders detailed modeling for large-scale transportation network inapplicable via CLEAR analysis.

2.2.2 Configurable Emergency Management and Planning System (CEMPS)

Not yet a full evacuation model, CEMPS (Pidd et al., 1996) is a proposed prototype system which links a Geographic Information System (GIS) to discrete-event simulation models to provide an aid for evacuation planning management. A microscopic approach is applied to simulate individual vehicle movement on roads of which structures are built by making use of a C++ linked list mechanism. CEMPS also applies GIS to establish the database and initial conditions of the simulation model, and display facilities as the simulation runs. Even though CEMPS is proclaimed to be a feasible solution for spatial decision support system for emergency evacuation, further constructions of traffic control and traffic route selection are needed for a complete evaluation. However, very little literature is available on the system to date.

2.2.3 Dynamic Discrete Disaster Decision Simulation System (D4S2)

D4S2 (Wu et al., 2007) is an application evacuation model, which makes use of available software and techniques to implement specific evacuation strategies. This microscopic simulation model is induced by integrating geographic information system ArcGIS with the simulation software ARENA and Microsoft's SQLServer database to simulate the evacuation process, the deployment of emergency resources, and the transport of casualties to safe facilities. Although D4S2 is still in its infancy stage, Wu et al. (2007) claim that the system when completed will be able to support emergency planning, training and research in simulation and optimization. Wu et al. (2007) also claim that the future system will be applicable for fifteen disaster event types and features, such as identifying traffic bottlenecks and allowing damaged infrastructure to be inserted during model executions.

2.3 Mesoscopic Simulation

Mesoscopic simulation models contain characteristics of both microscopic models in terms of modeling individual vehicle behavior and macroscopic models in terms of aggregate presentation of traffic dynamics. The detail level of traffic operations in mesoscopic simulation model is still limited compared to that of microscopic simulation. As such, adopting this approach will reduce the fidelity of microscopic simulation tools. Nevertheless, mesoscopic simulation models have the ability to present large-scale networks with less network coding and computer storage requirements.

Mesoscopic simulation models include *Interactive Dynamic Network Evacuation* developed by KLD Associates in 1984 (Urbanik et al., 1988), and *Oak Ridge Evacuation Modeling System* (Rathi & Solanki, 1993; Rathi, 1994).

2.3.1 Interactive Dynamic Network Evacuation (I-DYNEV)

Similar to CLEAR, the use of I-DYNEV was endorsed by the NRC to fulfill the requirement of providing an aid to evaluate plans in terms of their ability for safely evacuating populations away from a nuclear power plant should an incident occur. The core of I-DYNEV is TRAFLO, a system of models including a mesoscopic urban network model NETFLO, a macroscopic freeway model FREFLO, and an equilibrium traffic assignment model TRAFFIC.

NETFLO operates the evacuation process at three levels: (1) individual vehicles microscopically, (2) macroscopic groupings of vehicles (by flow statistical histograms), and (3) traffic flow in terms of traffic parameters (while FREFLO computes traffic as a function of flow rate, density and space-mean speed on freeway sections (Jaske, 1985)). The traffic assignment model TRAFFIC then generates turn movements/percentages from the O-D trip table at each intersection within the permitted time interval of the simulation model.

I-DYNEV's traffic loading rate is specified for each origin and this rate can be impacted due to traffic congestion. Vehicles cannot be discharged into a link if no space is available; thus, these vehicles are added into the link's queue. "Under these conditions, queues will grow and extend upstream along a congested path" (Jaske, 1985).

Even though I-DYVEV was widely used by the U.S. government, a benchmark study of I-DYNEV conducted for NRC indicates that the system "underestimates roadway capacity when the roadway does not have any congestion-induced capacity reduction" (Urbanik et al., 1988). Also, little information exists in the literature about validation and practicality of the model.

2.3.2 Oak Ridge Evacuation Modeling System (OREMS)

Developed to support the Federal Emergency Management Agency and U.S. Army for the Chemical Stockpile Emergency Preparedness Program, OREMS (Rathi & Solanki, 1993; Rathi, 1994) claims to exhibit advancement in evacuation modeling via an improved graphical user interface (GUI). The application-oriented software includes three major components:

1. *The input data manager IEVAC* handles the topology and characteristics of transportation network, traffic volumes (O-D matrix by transportation modes), and traffic controllers (intersection and lane control). Features such as driver performance characteristics are also added into the program. However, these features are simple and do not reflect the complexity of traffic and human behaviors during evacuation.
2. *The Fortran-based simulation analysis ESIM* operates OREMS's traffic flow simulation, trip distribution (destination selection), and traffic assignment models. The simulation and traffic assignment algorithms are the same as those in I-DYNEV. The trip distribution model emits evacuees via (i) pre-specified destinations, (ii) the nearest destination in terms of distance or time regardless of initial traffic conditions, or (iii) the closest destination (also in terms of distance or time) but based on the traffic conditions at their departure time. Sets of pseudo-links and supernodes are added to the original network to form a supernetwork of which pseudo-links connect different destinations to the supernodes.
3. *The output display program SIMOD* claims to show the input data and the output statistics produced by ESIM for every specific link, as well as an aggregate of the entire network.

OREMS claims that its advances in GUI allow the user to perform data manipulation readily and visualize digital traffic conditions (including operational characteristics and bottleneck identification). To date OREMS has had limited application for real time large-scale evacuation modeling due to its simulation approach, which constrains the modeling of “the route choice behavior of the drivers responding to different levels of traffic information” (Kwon & Pitt, 2005).

2.4 Summary and Drawbacks

Table 1 lists some of the common characteristics and desired features for simulation transportation evacuation model. Since CEMPS and D4S2 are under development, they are not included in the summary table. Of the models reviewed, MASSVAC, TEDSS, TEVAC and OREMS have the most number of desired features. However, the following observations are also made:

1. Most of the existing evacuation models follow a macroscopic simulation approach of which vehicles are treated in group and traffic flows are treated at an average level. This approach can diminish the complexity of analytical computation and computer storage requirements, but it will offset the elaborate characteristics of the evacuation progress.
2. Very little information exists on how to estimate the number of vehicles or people (rather than vehicles) entering the transportation network, which is especially vital to produce an accurate prediction of total evacuation time and traffic performance. Although population is classified in dissimilar groups, group handling is not specified in existing models.
3. Most of the evacuation models do not categorize transportation modes and even if they do, no modal split process is recorded. All vehicle types are treated in the same manner, even when they possess different attributes and characteristics.

4. Evacuees are not provided chances to select their destinations, except in OREMS. Currently, evacuees are assigned to destinations so that the trip assignment is optimized for minimal evacuation time.
5. None of these models, except for TEDSS, investigate the initial conditions of the traffic network. Vehicles are loaded into an empty network, which is unrealistic. Only in TEDSS, daily normal traffic is assumed at the beginning of the simulation.
6. All of the models load vehicles onto the network all at the beginning or following the cumulative S-shaped curve. As mentioned in Section 2.1.3, this loading pattern does not truly capture the stochastic features of the evacuees' departure process. In reality, the evacuees can leave at any time and it is not necessary that a certain number of evacuees have to enter the network at a pre-determined time. Therefore, it is possible that the loading rates are completely different among origins, and each origin has a unique loading rate.
7. How to effectively implement intersection control and traffic management strategies during an evacuation is still a big challenge. Labor and lead time requirements are the biggest concerns such that implementation plans of these strategies are still under debate among many US state planning authorities (Wolshon et al., 2005).
8. Most of the existing models lack user-friendly interfaces for data manipulation and output analysis. Consequently, processing large numbers of collected data and interpreting output data are difficult tasks—however, these tasks are critical for supporting decision analysis.

9. Risk assessment of infrastructure failure and network vulnerability has not been associated in existing models. Damage of critical structures such as bridges and tunnels will detrimentally impact the traffic flow and induce the possibility of human casualties and route closures. Additional research on how to include these factors in evacuation models is needed.
10. No user-interrupted changes are allowed during the simulation process of these models. Once the simulation starts, users cannot interact with the running model other than stop it and start another run. In other words, there is no way for the user to specify a sudden incident such as a broken bridge or impose damage to the infrastructure during the simulation.
11. Traffic conditions in reviewed models are analyzed and updated at fixed time intervals. This fixed time interval simulation does not capture the system state at instantaneous events such as traffic incidents, and does not skip over inactive periods of the time. Discrete-event simulation is a better choice for transportation evacuation modeling (see Section 2.2).
12. A major problem with these models is the lack of realistic driver behavior and crowd analysis. Additional research and investigation of this subject is substantially required for future evacuation models.
13. Last but not least, none of the models provide statistical validation of their modeling approaches. Conclusions are drawn upon one observation of a random process; and no confidence intervals are offered to prove the reliability of the final outputs/results. No designs of experiments are supported so that "what-if" analysis can be performed; and no statistical comparisons are

made to ensure the models are credible representations of the real-world systems. Furthermore, decisions in these models are made on deterministic (non-stochastic) simulation approaches and results.

13 gaps and issues are addressed in the research approach. Point 12 is not included—it is outside the scope of this research and is reserved for future research.

Table 1: Features of Existing Large-scale Simulation Evacuation Models/Softwares

Features		**NETVAC1**	**NESSY-IV**	**MASSVAC**	**TEVACS**	**REMS**	**TEDSS**	**CLEAR**	**I-DYNEV**	**OREMS**
Simulation model type	Macroscopic	X	X	X	X	X	X			
	Microscopic							X		
	Mesoscopic								X	X
Population characterized	Age			X						
	Labor force						X			
	School attendee						X			
	Vehicle utilization						X			
	Household size			X			X			
	Dwelling			X						
Modes	Auto	X	X	X	X	X	X	X	X	X
	Walk		X							
	Bus		X	X	X		X			X
	Carpool									X
	Truck				X					X
	Commercial vehicles			X						
	Mass Transit		X	X						
	Motocycle				X					
	Bicycle				X					
Destination Selection										X
Traffic initial conditions							X			
Traffic intersection control	Approach/density priority	X						X		
	Turning %									X
	Unsignalized	X								X
	Signal control	X		X	X		X		X	X
Traffic management strategies	Lane closure						X			
	Contraflow			X			X			
	Reserved lanes			X			X			
Loading pattern		All at the beginning		Logit-based		Logit-based	Logit-based	Fraction		
Infrastructure failure risk				X						
Identify traffic bottlenecks				X	X	X	X		X	X
Driver behavior										Simple
Graphical network					X		X		X	X
Statistical Analysis and Designs of Experiments										

CHAPTER 3

THE DOE_EVAC MODEL

3.1 Research Goals and DOE_EVAC Capabilities

The goals of this research are to develop a new discrete-event simulation model, the Designs of Experiments Evacuation (DOE_EVAC) model, that can (i) effectively simulate alternative modes of transportation during evacuations, (ii) support designs of experiments, thus, provide the users (e.g., emergency planners and traffic engineers) with means to investigate "what-if" scenarios with sound statistical analysis capabilities, and (iii) allow the users to build and execute these models without having to know complex simulation or coding language.

The DOE_EVAC model will bridge the gaps of current simulation transportation evacuation modeling approaches by its ability to:

- Treat alternative vehicle modes differently based on their characteristics (lengths).
- Allow evacuees to select their own destinations, but also allow users to implement pre-defined evacuation routes.
- Warm up the system in order to provide realistic initial conditions of the traffic network (avoid empty-and idle initial conditions).
- Implement and analyze various traffic management strategies, for example intersection control (adjust green, red, and yellow time at intersections).
- Reroute traffic if critical infrastructure is damaged.
- Allow users to interrupt the simulation for the purpose of changing:

- Entity attributes such as vehicle capacity.
- Traffic management strategies.

DOE_EVAC also has capabilities of:

- Supporting designs of experiments and confidence interval generation by its ability to perform multiple simulation runs and to obtain important traffic performance measures so that alternative plans/strategies can be analyzed to identify the “best” evacuation plan. None of today’s existing evacuation models have provided users with these capabilities.
- Furnishing users with ease of use. That is users do not need to know any specialized computer language or data structure. Data is manipulated in table formats and there is no need to reformat data to run the model as in other evacuation models. In addition, DOE_EVAC has various flexibility in its input modeling capabilities so that the use of probability distributions and mathematical expressions can be incorporated. Users will now be able to alter parameters directly within the model at anytime without having to reload the data files. This capability does not exist in today’s other models.

3.2 DOE_EVAC Model

DOE-EVAC was developed using Visual Basic.Net (VB.Net) and consists of the model’s GUI, data accessing and processing, and Arena models and Arena outputs. The code is in Appendix D and this code is open source. Thus, users can change the code anytime to fit their own modeling purposes. They also have the capability of manipulating the Arena code if they are familiar with that software language.

DOE_EVAC can create simulation models in any version of Arena (13.0 is the newest version). There are no limits to the size of the investigated transportation network. However, the Arena professional or commercial version is needed to build and run large-scale transportation network, and limits the execution of DOE_EVAC on personal computers.

Figure 4 demonstrates the architecture of DOE_EVAC. The following sections – the user supplied data, the graphical user interface, the data processor – explain important components of the model. The programming code will be referred to frequently by phrases such as "from line *i* to line *j*" meaning that the reader may refer to the code from line *i* to line *j* found in Appendix D.

3.2.1 User Supplied Data

3.2.1.1 Data Sources for the Tables

For the simulation transportation evacuation model presented, the following components are defined along with their corresponding data sources:

1. *Risk area* is the geographical area that might be hit or affected by the disaster. The boundary between dangerous and safe regions has to be pre-specified and updated by the users, for example, the emergency planners. The risk area is usually identified by group of zip codes or traffic analysis zones (TAZ). For example, Figure 5 shows the possible affected area along the Houston Galveston Area Council (HGAC)'s coast (Houston TranStar, 2010). Should a hurricane hit the

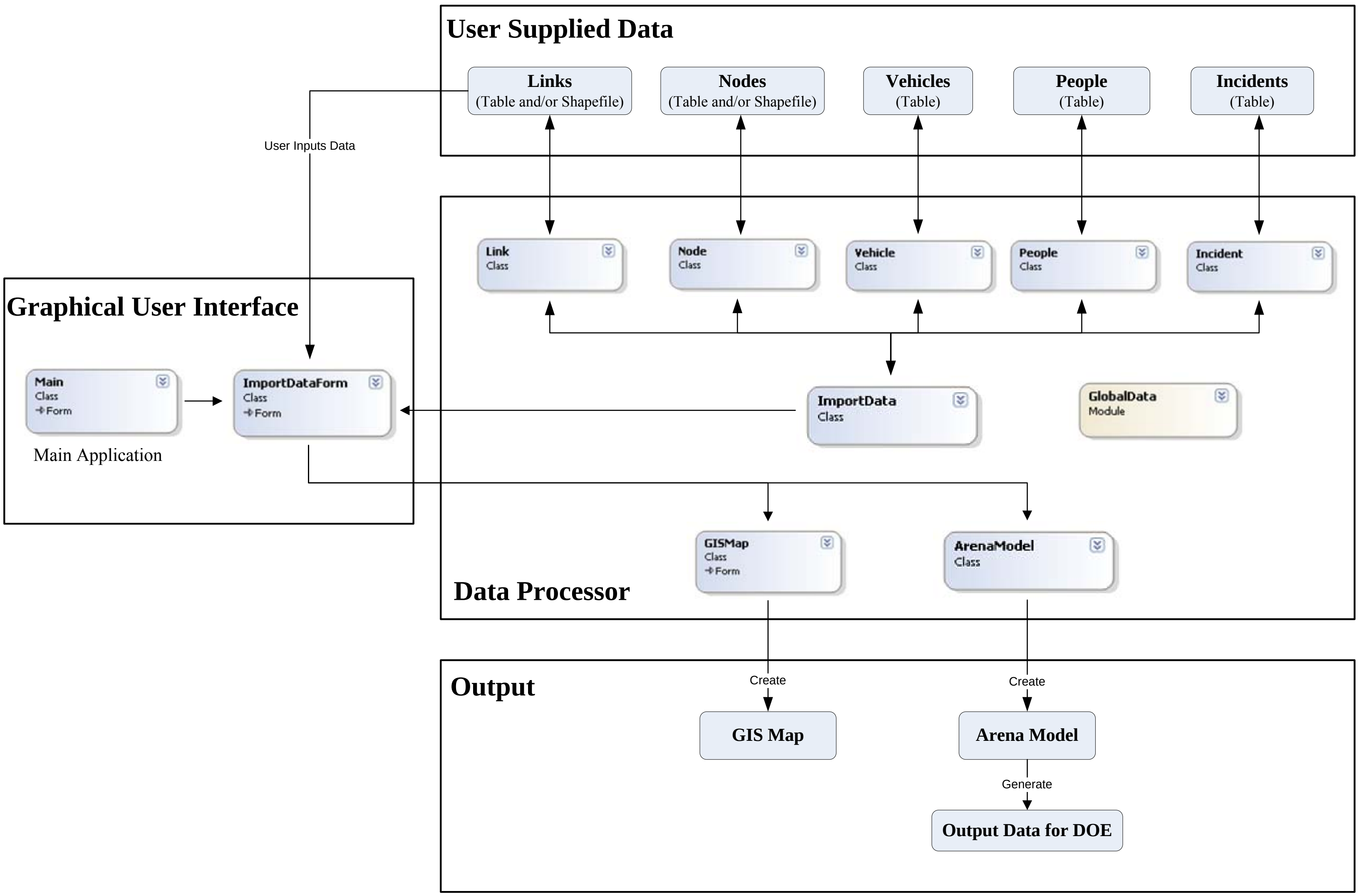


Figure 4: DOE_EVAC Architecture

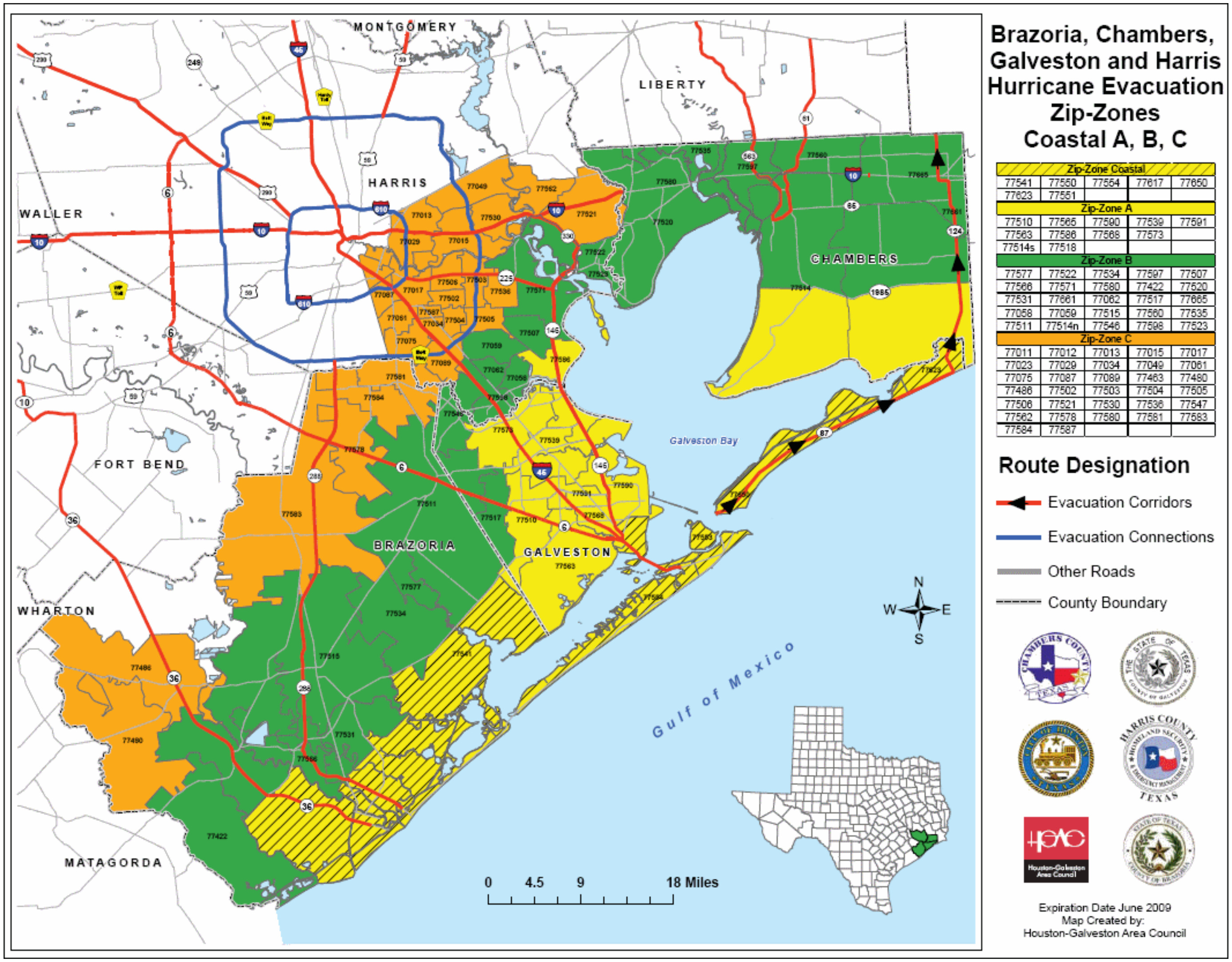


Figure 5: Houston Hurricane Evacuation Route Map (Houston TranStar, 2010)

gulf coast, a list of the zip-zones considered part of the evacuation corridor are listed in the table to the right of the map and shaded according to their categories A, B, or C.

2. *Transportation infrastructure network* is represented by a graph of nodes and links. Nodes are defined as joint traffic streams such as intersections, origins, destinations, upstream point of off-ramp, and downstream point of on-ramp. Links are the road segments connecting nodes. Links can be unidirectional or bidirectional. Note that a mesoscopic simulation approach is taken, so there is no need to investigate lanes separately since traffic flows are treated at the aggregate level. However, the number of lanes of each link must be specified in order to obtain the available capacity of link and to implement intersection control strategies. Here, the transportation network, including evacuation routes, are attainable via GIS shape files and maps. Figure 6 shows a street network of city within the Houston, Texas downtown in a GIS shape file as downloaded from the Tiger Line of U.S. Census Bureau (2009). Note that the nodes in Figure 6 are generated by using TransCAD (Caliper Corporation, 2005). Also, many counties, cities, and states now post their own data in the public domain. For example, the transit center locations and the TAZs of Houston Galveston Area Council (HGAC) can be downloaded from the HGAC database (2009).

3. *Origins and Destinations*: The origins are the centroids of geographical units. Evacuees who have cars depart directly from origins; and ones who do not have cars first walk or bike to transit centers (or pick-up points specified by emergency planners), and then can transit out of the risk area. In addition, the centroid

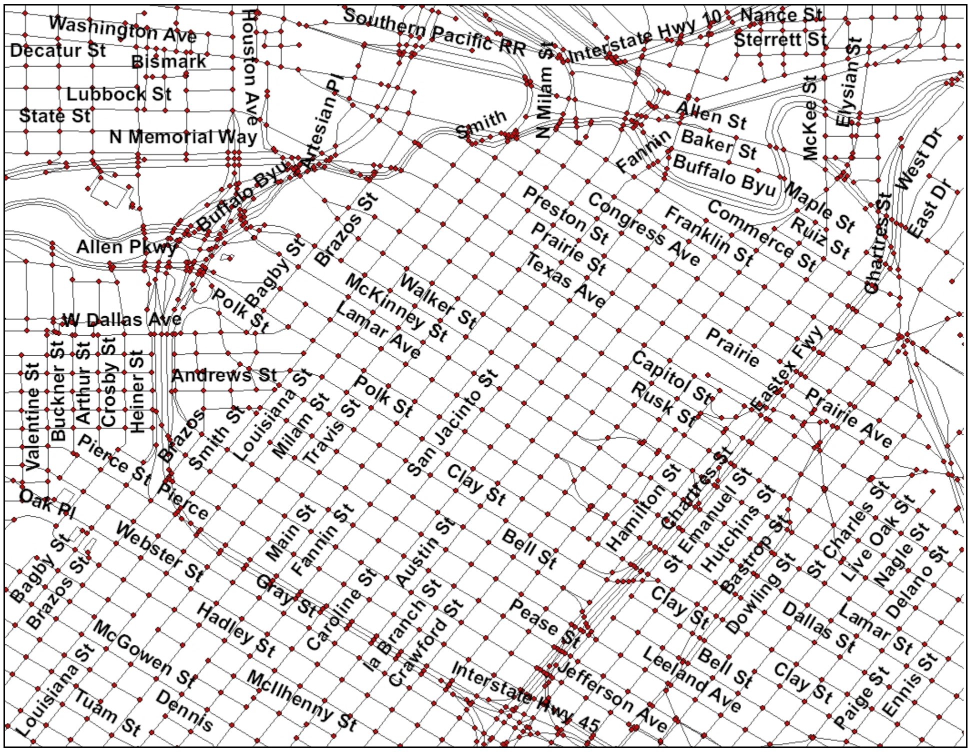


Figure 6: Houston Downtown's Street Network

connectors, which connect the centroids of geographical units to the transportation network, are not physical links. Thus, they should only be traversed once when vehicles are loaded onto the network. Locations of destinations must be assigned by users. They can be any points that lie outside the risk areas. Origins (source node) have to be different from destinations (target node).

Different from traditional transportation model, since the evacuees choose their own destinations, there is no need to pre-define travel demands for each O-D pair in this approach. In other words, no O-D matrix is needed or generated. This advantage allows DOE_EVAC to accurately represent the real-world – evacuation managers usually can determine the locations of destinations, but they do not know who or how many evacuees will go to which destinations.

4. *Populations at risk and vehicle utilization* describes the spatial distribution of the population and vehicles in the risk area. Geographical units can be the census block, census track, zip codes, TAZ, sub-county-division, county, state, and nation. Low mobility populations (populations without auto) must be identified so that planners can provide sufficient means of public transportation. Some downloadable sources for populations and vehicle utilization are the online database of the American Fact Finder for the U.S. Census Bureau (http://www.factfinder.census.gov/) and the Census Transportation Planning Package (CTPP) (http://www.fhwa.dot.gov/ctpp/). Population data can also be obtained from local databases, for example the HGAC database (http://www.h-gac.com/rds/gis/clearinghouse/default.aspx). Further discussion of data sources can be found in Section B.1 of Appendix B.

   At each origin i of the network, the total number of individual vehicles entering the network N_i can be estimated via trip production generation methods (see Section B.1 of Appendix B for a complete literature search). However, if other socioeconomic data are not available, the number of vehicles, N_i, can be calculated by the number of available private vehicles, NA_i, and the number of public transportation vehicles required, NP_i. The model generates NP_i based on the number of people who possess no vehicles $POP\_NO\_VEH_i$. Recall that the emergency planners are assumed to sufficiently provide public transportation to evacuees. Thus, the generation of required public transportation stops when all evacuees are served. In other words, the public means of transportation generating mechanism has to satisfy the following constrains where j stands for different types of public transportation modes, NP_{ij} and NP_CAP_{ij} are the quantity and the capacity of transportation modes j.:

$$
\begin{aligned}
&POP\_NO\_VEH_i \leq \sum_j NP_{ij} * NP\_CAP_{ij} \\
&\min\left(NP_i = \sum_j NP_{ij} \right)
\end{aligned}
\qquad (13)
$$

According to Wolshon et al. (2005), there are people who refuse to evacuate (e.g., elderly or ones who stay to protect their properties) and people who evacuate even though threats are not directly exposed to them. Thus, users themselves can assume a percentage of population who evacuate and recalculate the number of vehicles entering the network.

5. *Evacuee and authority emergency responses in terms of time* determine the egress pattern or loading rate of evacuees over time. As mentioned above, the logit S-shaped traffic loading rate cannot feature the stochastic characteristic of an evacuation. Furthermore, no existing data have been found for evacuation loading rate. Thus, statistical distributions can be explored by the user as possible traffic loading rates at each origin.
6. *Traffic management strategies* control traffic during evacuation.
   a. Incidents due to infrastructure failures must be specified by users so that links' characteristics can be modified to adapt those strategies. For example, if link A-B has space capacity of 30 vehicles with three lanes, but one lane is blocked till the end of the evacuation, then DOE_EVAC will modify the capacity of the link A-B to 20 vehicles (which is two thirds of the original capacity). Infrastructure failures such as broken

bridge or flooded streets usually take weeks, months, or even years to be repaired. Thus, in DOE_EVAC, once a link has an incident occurring on it, the link capacity is changed permanently.

b. Two types of intersections are handled in the model: signalized and unsignalized intersections (such as an intersection with a stop sign). For signalized intersections, signal phases (green and yellow time) must be specified. Further discussion of traffic signal control can be found in Section 3.4.3.

3.2.1.2 Tables and GIS Shapefiles

DOE_EVAC accepts data inputted in table forms such as dBASE (.dbf), Excel (.xls) and Access (.mdb). Thus, no pre-defined data formats or structures are required to build and run DOE_EVAC, except the input data files must at least include some required fields with the exact field names (see below).

There are only four tables needed: *Nodes*, *Links*, *Vehicles*, and *People*. The *Incidents* table as well as the *Nodes* GIS shapefile, and the *Links* GIS shapefile are optional. Note that the term “vehicle” represents the transportation modes and the term “people” represents pedestrians or any other type of evacuees that use paved streets or bicycle trails for their evacuation.

In order to illustrate the data input modeling, a sample GIS network with two TAZs (1055 and 1061, which are corresponding with Origin 7405 and 7533, respectively) is selected from the Houston, TX (Figure 7). This network is used as an example throughout this context. The total population and available vehicles per TAZ were downloaded from CTPP2000 Part 1 Table 47 and Table 74 (U. S. Department of Transportation, 2010).

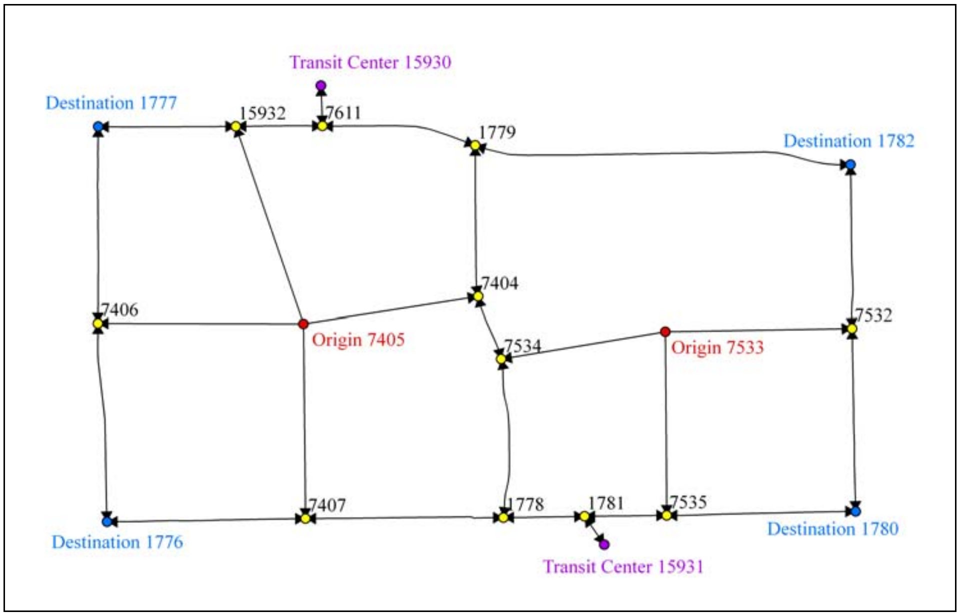


Figure 7: Example GIS Network

Table 2 - Table 6 show the data files of this network that are necessary to run DOE_EVAC. All required data fields of these files are summarized and described in Table 7. The table can be used as a convenient check list for users to prepare the required data for generating the tables of DOE_EVAC. Note that the field *DestDist* (destination distribution at each origin) must be in the *Nodes* file, but its values can be null. If so, the model will automatically generate the destination distribution.

Other required input parameters are the average public transportation (bus) capacity, the average public transportation length, the maximum time that people have to wait at transit centers before the bus can depart, and the minimum gap between vehicles in queues. The users will have opportunity to input and manipulate these data when they start running DOE_EVAC.

Table 2: Links

Dataview1 - Links

ID	Length	Dir	FromID	ToID	ABLanes	BALanes	ABFlowTime	BAFlowTime	ABSpeed	BASpeed
11905	0.53	2	7404	1779	2	2	0.89	0.89	36.00	36.00
11907	0.69	2	7406	1777	3	3	0.64	0.59	65.00	70.00
12135	0.58	2	7532	1782	2	2	0.96	0.96	36.00	36.00
12358	0.60	2	7661	1782	2	2	1.01	0.91	36.00	40.00
11899	0.31	2	7401	1894	2	2	0.53	0.53	35.00	35.00
12123	0.30	2	7525	7395	1	1	0.45	0.51	40.00	35.00
12132	0.42	1	7528	7530	1	0	0.71	0.00	35.00	0.00
12133	0.53	2	7531	1883	2	2	0.88	0.88	36.00	36.00
12139	0.51	1	7533	7534	2	0	0.86	0.00	36.00	0.00
12140	0.25	2	7535	1781	4	4	0.41	0.41	37.00	37.00
2492	0.64	2	1881	1882	3	3	0.54	0.54	70.00	70.00
11891	0.62	1	7396	7395	2	0	1.03	0.00	36.00	0.00
11893	0.62	1	7396	7397	2	0	1.07	0.00	35.00	0.00
26667	0.18	2	15924	7385	2	2	0.30	0.30	35.00	35.00
2491	0.28	2	1880	7372	3	3	0.24	0.24	70.00	70.00
11871	0.58	2	7385	1880	2	2	1.00	1.00	35.00	35.00
11849	0.33	2	7372	1881	3	3	0.28	0.28	70.00	70.00
2494	0.28	2	1884	7398	2	2	0.47	0.47	35.00	35.00
11896	0.56	1	7396	7398	2	0	0.95	0.00	35.00	0.00
11895	0.17	2	7398	15924	2	2	0.30	0.30	35.00	35.00
26668	0.14	2	15925	15924	2	2	0.25	0.25	35.00	35.00

Table 3: Nodes

Dataview1 - Nodes

ID	Type	Green	Yellow	PeoTime	VehTime	People	Vehicle	DestDist
7404	3	0.00	0.00			--	--	
1779	3	0.50	0.10			--	--	
7406	3	0.30	0.05			--	--	
1777	1	0.00	0.00			--	--	
7532	3	0.50	0.10			--	--	
1782	1	0.00	0.00			--	--	
7533	0	0.00	0.00	EXPO(0.04)	EXPO(0.01)	315	7785	
7534	3	0.00	0.00			--	--	
7535	3	0.50	0.10			--	--	
1781	3	0.50	0.10			--	--	
15931	2	0.00	0.00			--	--	
1776	1	0.00	0.00			--	--	
1778	3	0.50	0.05			--	--	
7405	0	0.00	0.00	EXPO(0.025)	EXPO(0.03)	16	715	
7407	3	0.50	0.10			--	--	
1780	1	0.00	0.00			--	--	
7611	3	0.50	0.10			--	--	
15932	3	0.30	0.05			--	--	
15930	2	0.00	0.00			--	--	

Table 4: Vehicles

Dataview1 - Vehicles

TYPE	LENGTH
Car	13.50
SUV	16.40
Truck	60.00

Table 5: People

Dataview1 - People

TYPE	SPEED
Pedestrian	2.50
Bicycle	12.00

Table 6: Incidents

Dataview1 - Incidents

ID	FROMNODE	TONODE	STARTTIME	CAPPERCENT
1	1778	1781	40.00	0.00

Table 7: Required Data Fields

Files	Field Names	Field Descriptions	Field Types	Values
Nodes	ID	Node identification	Integer	
	Type	Node type	Integer	0: Centroid 1: Destination 2: Transit Center 3: Intersection
	Green	Green time for Type 3	Double	0: if unsignalized
	Yellow	Yellow time for Type 3	Double	0: if unsignalized
	PeoTime	Interarrival time of people for Type 0	String	e.g., EXPO(0.01) (minutes)
	VehTime	Interarrival time of vehicles for Type 0	String	e.g., EXPO(0.04) (minutes)
	People	Number of people without vehicle for Type 0	Integer	
	Vehicle	Number of vehicles for Type 0	Integer	
	DestDist	Percentage of vehicle going to destinations, generated automatically if null or empty	String	e.g., DISC(0.5,1,1,2)
Links	ID	Link identification	Integer	
	Length	Length	Double	
	Dir	Direction	Integer	1: one-way 2: two-way
	FromID	Topological starting node ID	Integer	
	ToID	Topological ending node ID	Integer	
	ABLanes	Number of lanes on forward topological direction	Integer	
	BALanes	Number of lanes on backward topological direction	Integer	
	ABFlowTime	Free-flow travel time on forward topological direction	Double	(minutes)
	BAFlowTime	Free-flow travel time on backward topological direction	Double	(minutes)
	ABSpeed	Value or probability distribution of speed on forward topological direction	String	(miles per hour)
	BASpeed	Value or probability distribution of speed on backward topological direction	String	(miles per hour)
Vehicles (on road)	Type	Vehicle type	String	e.g., car, truck
	Length	Vehicle length	Double	
People (on pavement)	Type	Type	String	e.g., pedestrian, bicycle
	Speed	Value or probability distribution of speed	String	(miles per hour)
Incidents (*Optional*)	FromNode	Start node of link	Integer	
	ToNode	End node of link	Integer	
	StartTime	Start time	Double	(minutes)
	CapPercent	Available capacity percentage	Double	0 to 100

3.2.2 Graphical User Interface

The GUI or the main application of DOE_EVAC (Figure 8) is designed for the users' ease of use. The *File* menu contains the *Import Data* menu item which allows users to input the data into the model. The *Import Data* window contains five dialogs to open five data files (*Incidents* file is optional) and to input four other required parameters (Figure 9). The method to import data will be described in section 3.2.3.

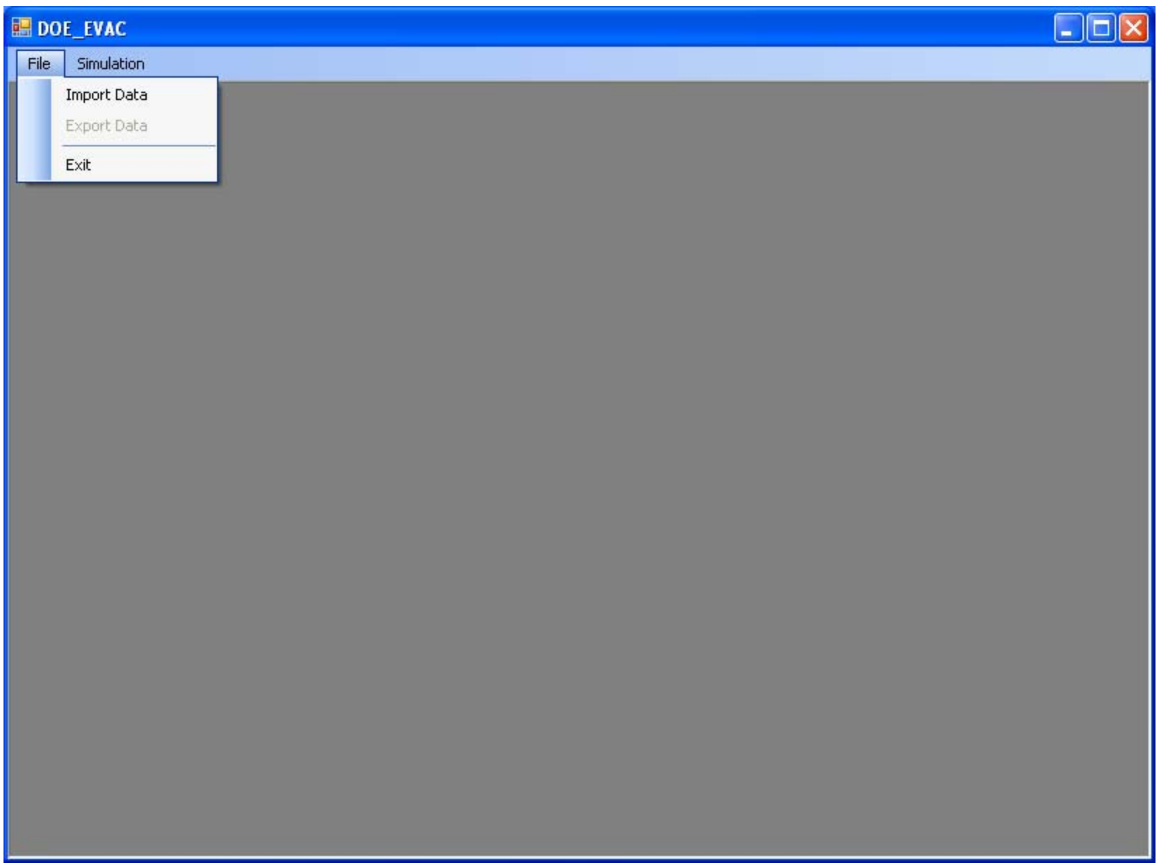


Figure 8: Evacuation Model GUI

Once the data from the tables are imported into the model, if the *Nodes* and *Links* GIS shapefiles are available, DOE_EVAC will display the GIS map docked in the main application (Figure 10). DOE_EVAC will ask the users if they would like to create an Arena model for the imported network. If the answer is "Yes", the Arena application will be opened and DOE_EVAC generates an Arena model for the network. Otherwise, the menu item *Create Model* under *Simulation* menu is enabled (initially, the menu items under *Simulation* and *Analysis* are disabled) and the users can later on create the Arena model. After the Arena model is generated, the menu item *Export* under the *File* menu is enabled to allow users the ability to access the Arena output files. Note that users can only use this menu item after running the Arena model.

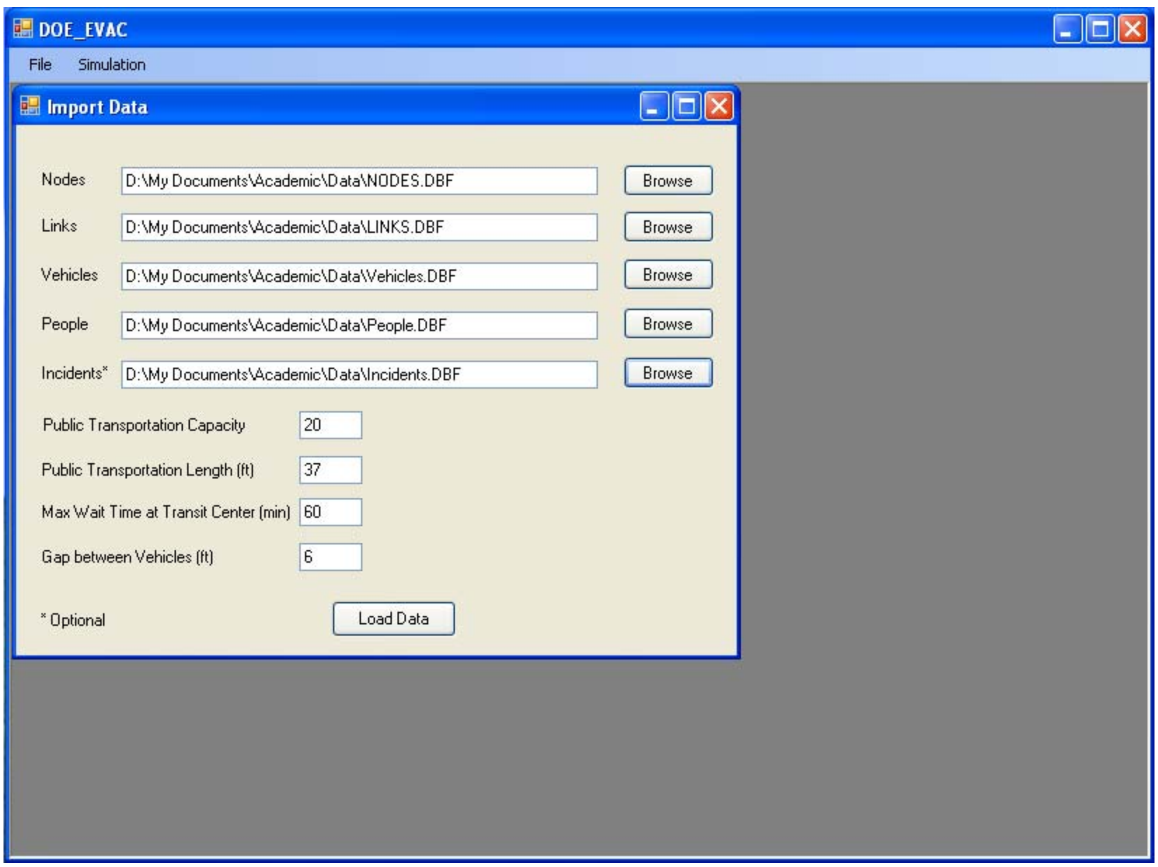


Figure 9: Import Data Window

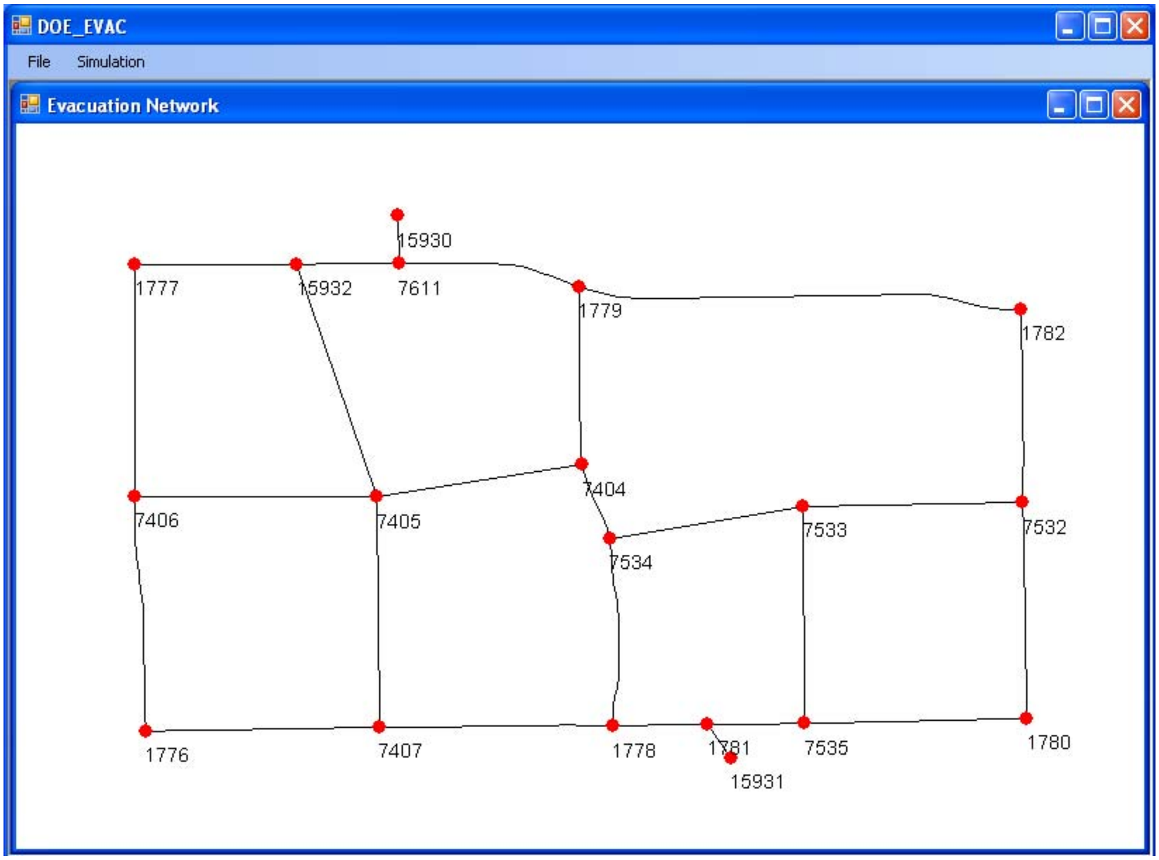


Figure 10: GIS Network Window

3.2.3 Data Processor

When the *Load Data* command (on *Import Data* window) is issued, the model retrieves and processes the imported data (from line 479 to line 852 in Appendix D). The model first establishes connections to the data files (from line 813 to line 851). The required data to generate an Arena model then are selected from data files by using the structured query language (SQL). For example, the syntax to obtain link data is as follow:

sql = "SELECT Length, Dir, FromID, ToID, ABLanes, BALanes, ABFlowTime, BAFlowTime, ABSpeed, BASpeed FROM " & linksTable & ""

The model stores each data type in one separated enumerable list of which each row contains an instance object of the corresponding data type. For instance, the origin nodes are stored in *OriginList* and each row of *OriginList* is an instance object of *Node*. All of the nodes and links are also added into a graph *g* to obtain a shortest path afterward. The open source library QuickGraph (Microsoft Corporation, 2010) was used to calculate the shortest paths in the network. The applied algorithm is Dijkstra's shortest path algorithm (Dijkstra, 1959). The basic command to find shortest path is *g.ShortestPathsDijkstra(edgeCost, source)* where edgeCost contains travel costs (distance or flow time) on links and source is the source node.

The MapWindow (Ames et al., 2010), a GIS open source library, was used to draw the transportation network. The code to generate GIS map is from line 2,741 to line 2,792. The stored data is now ready to create the Arena model.

3.3 Execution Assumptions

The following assumptions are made during the execution of the run:

1. The disaster is eminent (e.g., hurricane).
2. The geophysical risk or affected area is known at particular time.
3. Evacuation routes and location of accessible destinations must be indicated prior to the evacuation.
4. Advance warning is issued and evacuees have sufficient preparation time prior to the evacuation.
5. Users (emergency planners) can access a high level of information (disaster, network…)

6. Evacuees know their destinations prior to the evacuation.
7. Evacuees owning vehicles take as many vehicles as they own.
8. Destinations have infinite capacity.
9. Emergency planners provide enough means of public transportation for evacuees who do not have access to private vehicles.
10. Non-vehicular evacuees who go to transit centers follow their designated paths and not "disturb" the other traffic flows.
11. The initial condition of the simulation is normal daily traffic.
12. Vehicles traveling inside the network are well behaved and follow all traffic rules. That is vehicles only traverse intersections under green traffic signals, and vehicles run at assigned speeds, and so forth.

Assumption (11) can be relaxed so that various levels of the initial condition's effects can be investigated.

3.4 DOE_EVAC Model Logic

DOE_EVAC has characteristics of microscopic models in terms of keeping track of individual vehicles. However, DOE_EVAC does not include the lane-changing behavior and the acceleration or deceleration of vehicles. Other than that it behaves similarly to that of a microscopic model while it still can simulate a large-scale transportation network as macroscopic models (i.e., it has lower computation and computer memories requirements). Figure 4 provides an overview of the DOE_EVAC model logic including the traffic loading approach and the traffic en route and operation control.

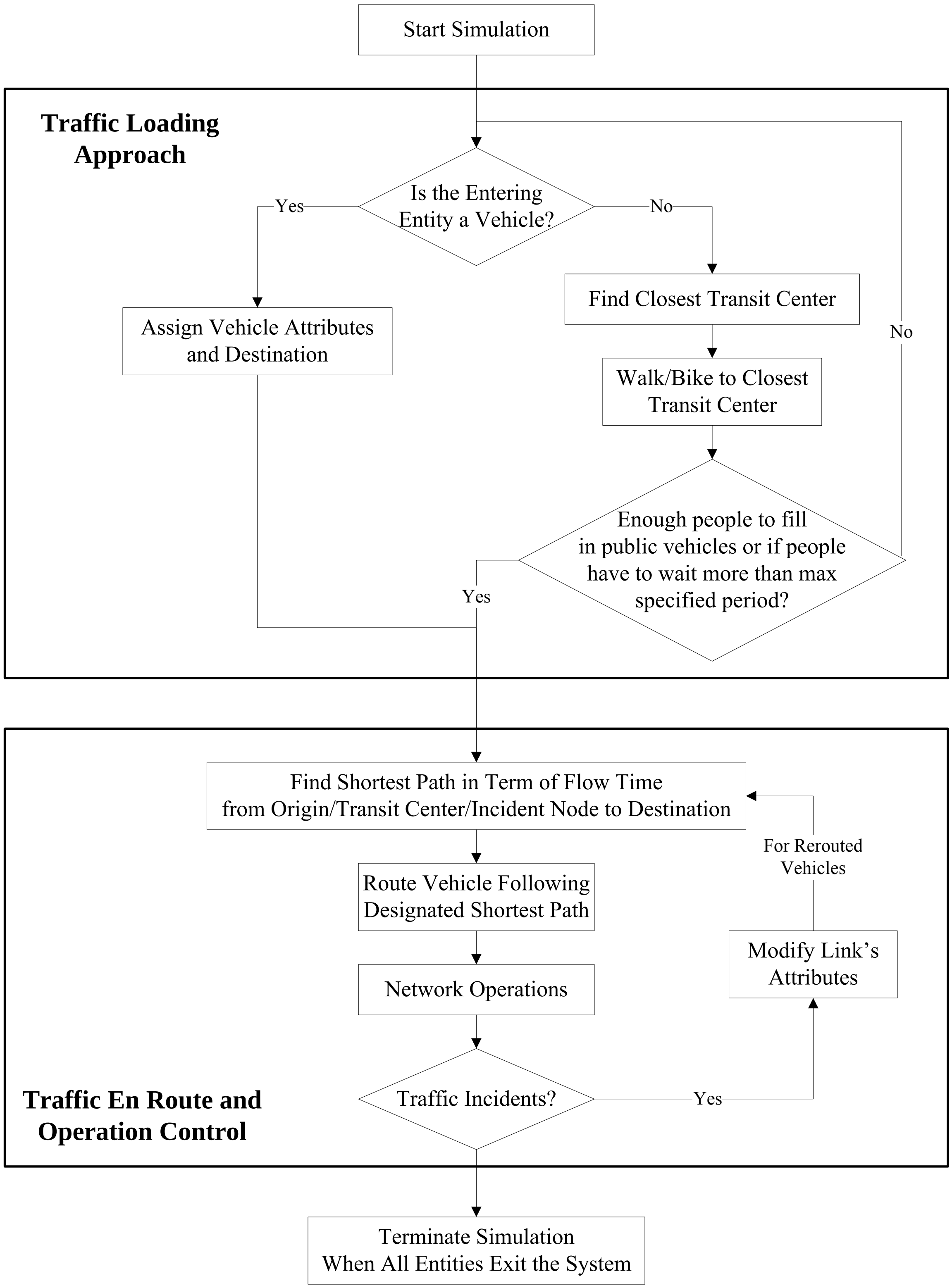


Figure 11: DOE_EVAC Model Logic

3.4.1 Traffic Generation

Once the simulation starts, vehicles arrive to the network via origin nodes by following user-defined interarrival time expressions. The interarrival time truly is the inverse of the traffic loading rate. The loading process stops when all evacuees leave the origins. The traffic loading in the DOE_EVAC is very flexible; that is DOE_EVAC supports all types of expressions including mathematical expressions and probability distributions. Traffic managers/engineers can build their own complicated expressions of traffic loading and still can apply DOE_EVAC to run their applications. This ability makes DOE_EVAC more advanced than other simulation models of which traffic loading is a fixed process with limited allowable expressions.

Table 8 summarizes available Arena's probability distribution (Kelton, Sadowski, & Sturrock, 2007). Each distribution has one or more expressed ways. For instance, exponential distribution can be expressed as EXPONENTIAL(Mean) or EXPO(Mean). Descriptions of useful probability distributions can also be found in Law (2007) or Montgomery and Runger (2007).

Table 8: Arena's Probabiliy Distributions

Distribution		Parameters	Example
Beta	BETA	Beta, Alpha	BETA(2, 5)
Continuous	CONT	$CumP_1, Val_1, \ldots, CumP_n, Val_n$	CONT(0.5, 1, 0.7, 2, 1, 3)
Discrete	DISC	$CumP_1, Val_1, \ldots, CumP_n, Val_n$	DISC(0.5, 1, 0.7, 2, 1, 3)
Erlang	ERLA	ExpMean, k	ERLA(2, 3)
Exponential	EXPO	Mean	EXPO(2)
Gamma	GAMM	Beta, Alpha	GAMM(1, 2)
Johnson	JOHN	Gamma, Delta, Lambda, Xi	JOHN(1, 3, 2, 5)
Lognormal	LOGN	LogMean, LogStd	LOGN(3, 1)
Normal	NORM	Mean, StdDev	NORM(3, 1)
Poisson	POIS	Mean	POIS(5)
Triangular	TRIA	Min, Mode, Max	TRIA(2, 5, 7)
Uniform	UNIF	Min, Max	UNIF(4, 10)
Weibull	WEIB	Beta, Alpha	WEIB(2, 5)

Each vehicle entering the network is randomly assigned its vehicle type and destination. Users are able to modify the vehicle list and select multiple types of vehicles such as car, truck, and so forth. The vehicle length associated with the vehicle type is used to calculate the occupied spaces on a link and available spaces for incoming vehicles. The model randomly picks a destination for each vehicle from the destination list. The users also have the option to specify destinations such as shelters for the public transportation population.

Recall that evacuees ("people" type) who use the paved streets or the bicycle trails to evacuate are assumed to travel on their designated lanes without disrupting vehicular traffic. They move according to an average speed determined and set by the users.

3.4.2 Route Choice

The questions now are what routes the vehicles follow and what traffic algorithm drives the route choice mechanism? The en-route assignment mechanism which is a set of behavioral rules is applied to determine drivers' reactions during the evacuation. Whenever a vehicle n enters the network, a pre-trip route is selected from the set of current shortest routes (C_{ij}) that connect the vehicle's origin i to its destination j in terms of minimum travel time. Since it is possible that more than one route has the same travel time, a set of shortest routes is also possible. The behavioral rules for route choice are as follows:

- If C_{ij} contains only one route, vehicle n is assigned to follow that route.

- If C_{ij} has more than one element, the route with the shortest distance is chosen to be vehicle n's route. If a few elements have the same shortest distances, the route is randomly chosen from these elements.

Even though each vehicle is assigned a destination, if the shortest path of a vehicle to a designated destination contains another destination in that path, the vehicle exits the network through the first destination it reaches.

The route choice of walkers and bikers is practically the same as that of vehicles. Walkers and bikers from origin i follow the shortest path to the closest transit center or pick-up point in the vicinity of i. However, the shortest path, which is actually the shortest distance path, is calculated from the lengths of links connecting origin i to the designated transit center instead of the total travel time on the links.

3.4.3 Network Operations

Figure 12 shows the schematic of a link A-B (A is an upstream node and B is an downstream node). A link consists of two parts: the running part and the queue part. The queue part is only formed when the ingoing flow exceeds the outgoing flow at the downstream node. Thus, the boundary between these two parts varies over time.

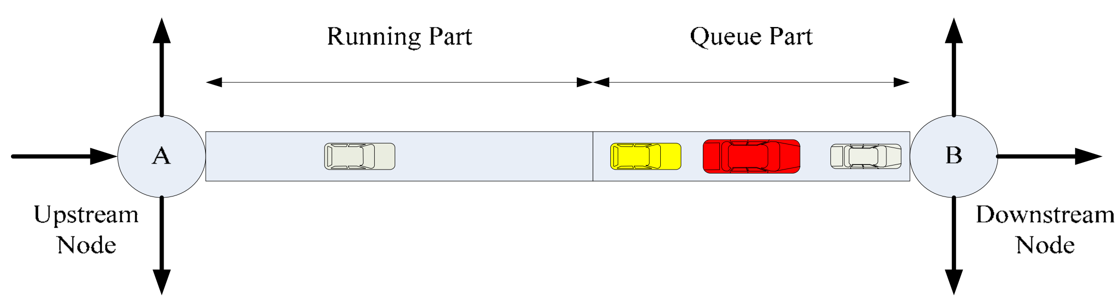


Figure 12: Link Diagram

In the previous mesoscopic simulation models, the travelling speed of vehicles $V(k)$ on a link can be computed by various speed-density relationships (Del Castillo & Benitez, 1995; Burghout, 2004):

$$V(k) = V_{free}\left(1 - \frac{k}{k_{jam}}\right) \text{(Greenshields, 1935)} \tag{14}$$

where *k* is the density on the current running part of the link, *V(k)* is the speed assigned to the vehicle, V_{free} is the free flow speed, and k_{jam} is the jam density

$$V(k) = V_c \ln\left(\frac{k}{k_{jam}}\right) \text{ (Greenberg, 1959)} \tag{15}$$

where V_c is the speed at maximum flow

$$V(k) = V_{free}\left(1 - \left(\frac{k}{k_{jam}}\right)^a\right)^b \text{ (Gazis, Herman, \& Potts, 1959)} \tag{16}$$

where a, b is the model parameter

$$V(k) = V_{free} \exp\left(-\frac{k}{k_c}\right) \text{ (Underwood, 1961)} \tag{17}$$

where k_c is the density at maximum flow

$$V(k) = V_{free} \exp\left(-\frac{1}{2}\left(\frac{k}{k_{jam}}\right)^2\right) \text{ (Drake, Schofer, \& May, 1967)} \tag{18}$$

The speed *V(k)* is then used to calculate the link travel time. Two problems of formulae (16) – (20) have been pointed out in Burghout (2004): "Firstly, the speed of traffic loaded at densities approaching jam density will be approaching 0,…, which would mean that link travel times would approach infinity. Secondly, at low densities, the

speed has been shown empirically not to depend on density, but to remain around V_{free}."
In other words, one cannot apply these formulae to simulate the extreme slow traffic such as queues during traffic jam and there is no need to apply these formulae when link's density is low.

Burghout (2004) and Burghout et al. (2006) proposed a new generalization speed-density relationship to overcome those shortcomings:

$$V(k) = \begin{cases} V_{free} & \text{if } k < k_{min} \\ V_{min} + \left(V_{free} - V_{min}\right)\left(1 - \left(\dfrac{k - k_{min}}{k_{max} - k_{min}}\right)^{a}\right)^{b} & \text{if } k \in \left[k_{min}, k_{max}\right] \\ V_{min} & \text{if } k > k_{max} \end{cases} \tag{19}$$

where V_{min} is the minimum speed, k_{max} and k_{min} are the maximum and minimum densities where speed is still a function of density, *a* and *b* are the user-specified calibration parameters. As observed, the authors introduce two new parameters of V_{min} and k_{min}. V_{min} has to be positive to ensure the computation of link's travel time. The new formulation, however, forces the vehicles in the network to always move during the simulation. This type of simulation does not represent the movement of vehicles in real situations.

Thus, in DOE_EVAC, the speed applied for each vehicle entering the running part of the link is not computed by following any speed-density relationships, but is a generalized speed, which can be in the form of a single value (e.g., free flow speed) or a generated value of a statistical distribution (see Table 8). Users can estimate the link speed via history data and apply it into their models. If the vehicle speed follows a distribution, each vehicle entering the link will be randomly assigned a speed bounded to that distribution. The achieved speed is then used to calculate the time needed for the vehicle to reach the back of the queue or the incoming intersection if there is no queue ahead (Equation 20). This approach guarantees the natural variability of vehicle speeds as well as maintains the integrity of vehicle movement and queuing as in a real-world system.

$$t = \frac{\text{Available Space of Link}}{V_{Link}} \tag{20}$$

When the queue begins to dissolve from the link's downstream, the vehicles in queue move through the downstream node to the next link. The queue discipline is first-in-first-out (FIFO). Figure 13 shows the schematic of node B with two incoming approaches (AB and EB) and two outgoing approaches (BD and BC).

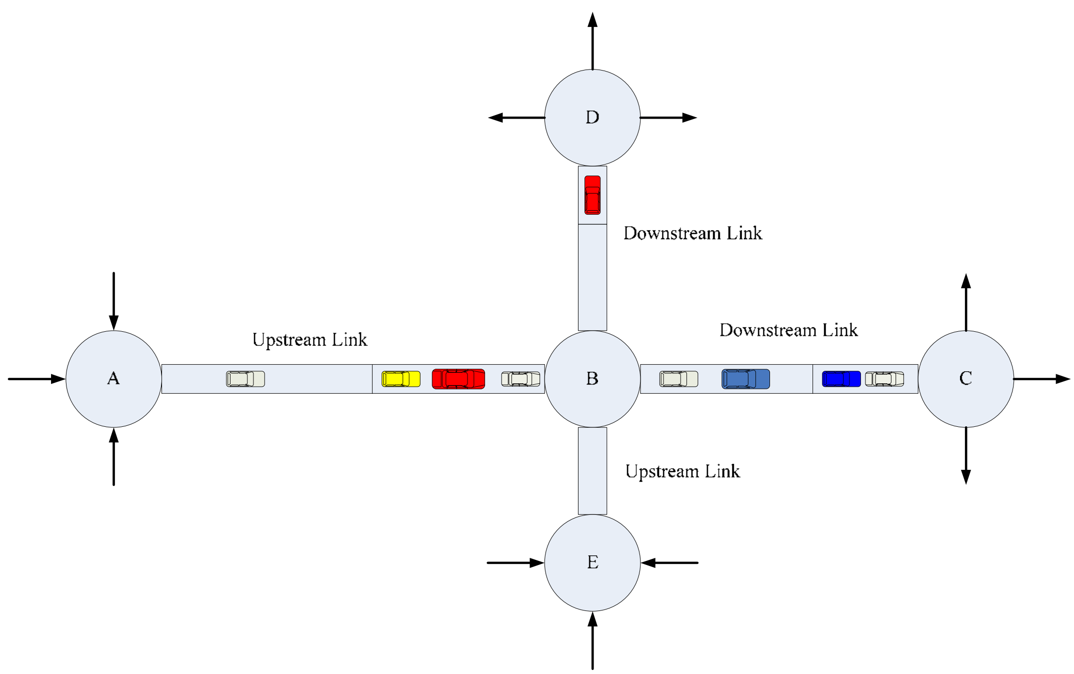


Figure 13: Node Diagram

A vehicle can only enter its next link if the next link has sufficient space and the movement is possible, i.e. the intersection is clear for the movement. Since lanes are not investigated separately as in a mesoscopic simulation model, the capacity of a link is the space capacity of the whole corresponding street segment. It equals the product of the link length and the number of lanes on that link. The space occupied by a vehicle includes vehicle length and the user-specified gap between vehicles. This parameter is used to estimate the occupied space on a link at an instant of time.

Again, in previous mesoscopic models, the speed of the vehicle moving from the queue of link A-B to other links B-*l* (where *l* is node C, D, and E) via node B, ω_{AB-Bl}, can be calculated using the density upstream $k_{B_{in}}$, the density downstream $k_{B^l_{out}}$, the flow upstream $q_{B_{in}}$, and the flow downstream $q_{B^l_{out}}$ of node B (May, 1990):

$$\omega_{AB-Bl} = \frac{q_{B_{in}} - q_{B^l_{out}}}{k_{B_{in}} - k_{B^l_{out}}} \tag{21}$$

The moving time through node B then can be determined by dividing the intersection travel distance to ω_{AB-Bl}. This practice is only possible if one can obtain the intersection travel distance. However, the parameter is not always available. Thus, in DOE_EVAC, the required time for a vehicle moving through an intersection follows a user-specified statistical distribution or can be an average value estimated via history data.

For a signalized intersection, the moving time through the intersection from each direction is limited to the green time of that direction. From each direction, three possible turning movements of vehicles from link A-B – going straight, turning right and turning left – are restricted by the available space of the corresponding downstream links and the exit time of the previous vehicles in queue.

Note the modeling intersection logic can be extended to accommodate more complicated intersection control mechanisms anytime without any difficulty. For now, the applied traffic signal control is a pre-timed signal control in which a signal cycle (a combination of signal phases for different approaches of through vehicles) follows a fixed order of signal phases with fixed interval time lengths (Orcutt, 1993). A signal phase in this model includes three intervals: green time, yellow time and red time. Recall that "people" are assumed to not interrupt the traffic. Thus, current intersection control avoids the pedestrian signal phase. "People" are assumed "protected" from oncoming vehicular traffic to prevent crossing conflicts.

Figure 14 illustrates the possible turns at a four-leg single-lane intersection. Each coming approach to the intersection is reserved a separate signal phase. In other words, groups of "alike-color turns" move at the same time under the same signal phase. Also, the green time is the same for all approaches of a signalized intersection. Discussions on how to choose traffic signal green times can be found in Homburger, Hall, Loutzenheiser, and Reilly (1996). The DOE_EVAC model allows the users to specify which traffic signal green time they wish to incorporate.

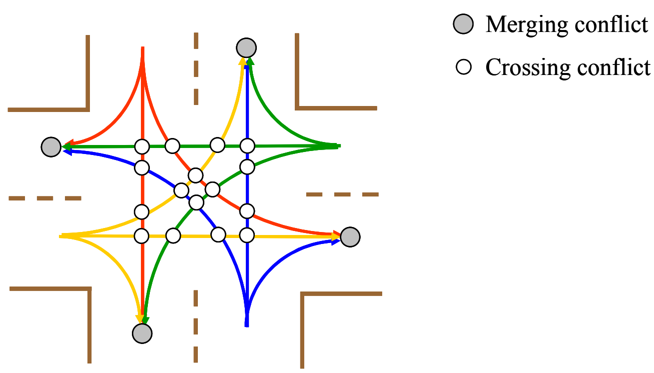


Figure 14: Four-Leg Single-Lane Intersection

For unsignalized intersections, priority is assigned for vehicles approaching earlier to the intersections. At unsignalized "T" intersections, vehicles from the secondary approach can be emitted into the primary approach if and only if there is sufficient space from the primary approach.

3.5 Arena Model – An Output of the DOE_EVAC Model

This section demonstrates how DOE_EVAC creates an Arena model for the example transportation network of Figure 7 and describes the simulation logic in Arena. It is included and detailed out for users for their understanding. However, note, users do not need to know the Arena and the DOE_EVAC's GUI language. The utilized Arena operands are summarized in Appendix C and the code to create the Arena model is in the *ArenaModel* class (see Appendix D from line 853 to line 2,740.)

The advantage of DOE_EVAC is that all data in the user supplied tables are imported into Arena under a variety of formats such as variables (listed in *Variable* module of *Basic Process* panel), expressions (in *Expression* module of *Advanced Process* panel) or as direct inputs. Thus, users can make changes to any of the Arena parameters directly within the Arena model to either update the data or test alternative scenarios. Current Arena parameters designated to allow designs of experiments are revealed in Table 9.

The difference between the Arena variable and the Arena expression formats is that variable format only accepts single numeric values as parameters while the expression format accepts all types of expressions including probability distributions (e.g., 2 + Process Time, or EXPO(2). Thus, if users would like to change the variable

Table 9: Parameters Designated for Designs of Experiments

Format	Parameter	Description
Variable	Public Transportation Size	Maximum number of people can be transported by public transportation vehicle at each transit center.
	Max Wait Time	Maximum time people have to wait at transit centers before the bus departs.
	Gap	Minimum gap between two vehicles in queue
	Vehicle Route Time	Vehicle moving time through intersection
	Green Time	Green time at intersection
	Warm Up Time	To initialize network traffic condition
Expression	Speed	Speed on link of *Vehicle* entities
	People Speed	Speed of *People* entities
Direct Input	Interarrival Time	Interarrival time at origin *Create* Module

parameters to a distribution form, they must define the tested parameter as an expression parameter in the source code (the syntax is in Equation 22 and the code is from line 2,580 to line 2,587) or in the Arena model (move parameters from Variable module to Expression module); and run the experiments directly in Arena.

expression = expressionModule (Expression Name, Value) (22)

Each node in the network is associated with a station in Arena. There are four types of stations: origin, destination, transit center, and intersection. A link is modeled as a queue buffer of the downstream node station. For example, in Figure 13, Station B has two queue buffers AB and EB; and the capacity of the buffers are the available space capacities of link AB and EB. Link BC is the queue buffer of Station C and link BD is the queue buffer of Station D. Each queue buffer has a variable to keep track of its available capacity.

Figure 15 – Figure 20 depicts the overall design structures for all types of nodes as well as traffic signal control at intersections. These structures will be described in the following sections. Note that the Arena module names end with the modeled node's ID.

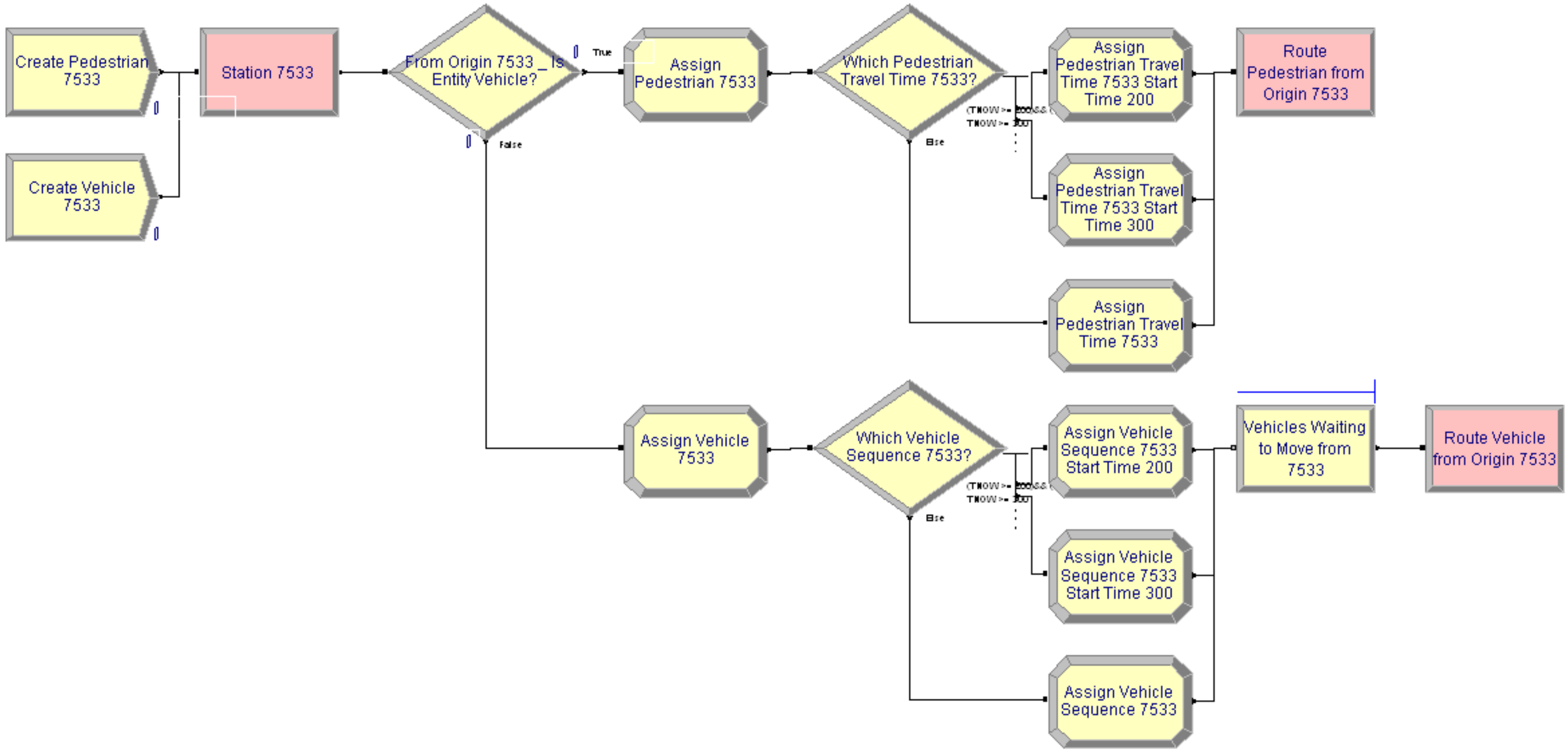


Figure 15: An Origin Station

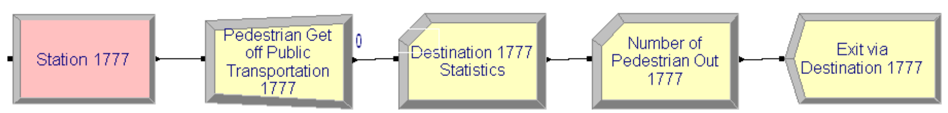


Figure 16: A Destination Station

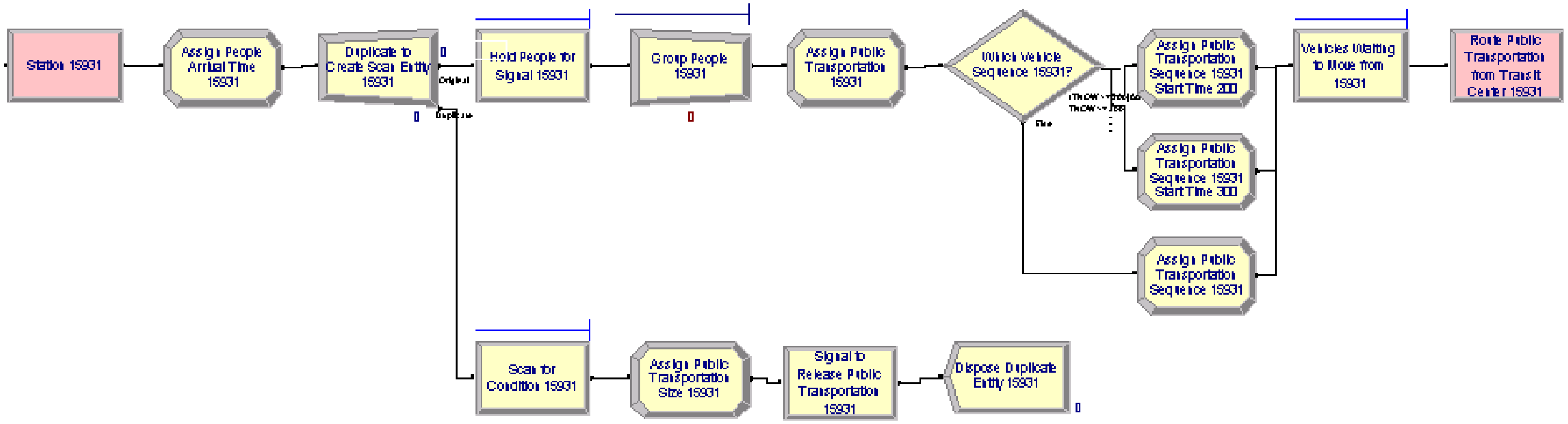


Figure 17: A Transit Center Station

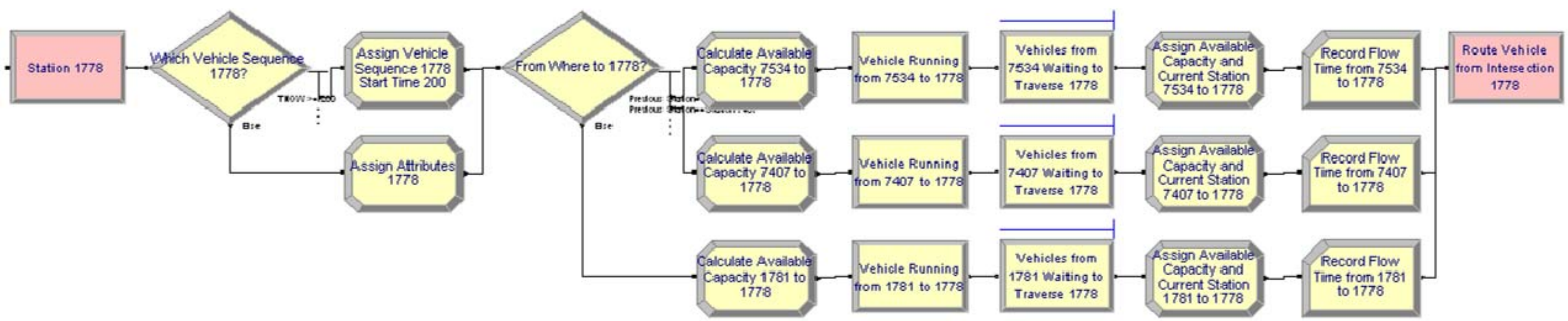


Figure 18: An Intersection Station - Incident on Incoming Link

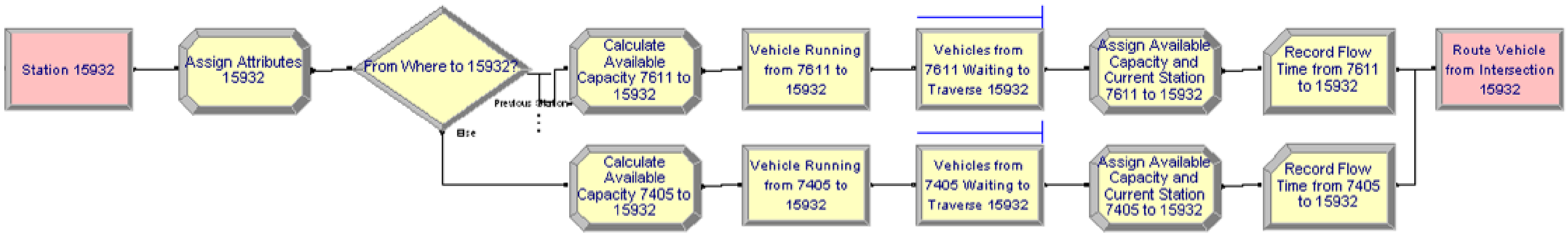


Figure 19: An Intersection Station - More than One Incoming Approach

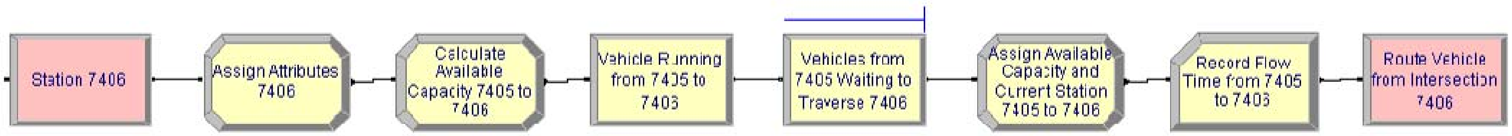


Figure 20: An Intersection Station - One Incoming Approach

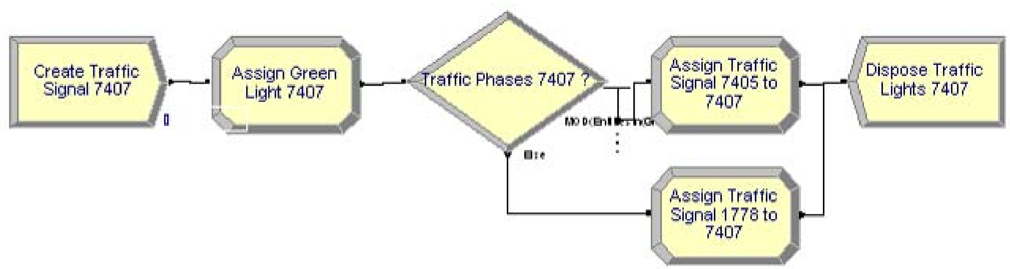


Figure 21: A Traffic Signal Control at Signalized Intersection

At each origin, two *Create* modules are generated: one for the vehicle and one for the people. The code to generate the Create modules is from line 2,456 to line 2,469. Figure 22 shows the Arena modules used to generate arrivals of the *People* and the *Vehicle* entities at origin node 7533. The *Max Arrivals* are the user-supplied number of people without vehicles in the “Create People” module and the user-supplied number of vehicles in the “Create Vehicle” module. Once the number of generated entities reaches *Max Arrivals*, the create process stops.

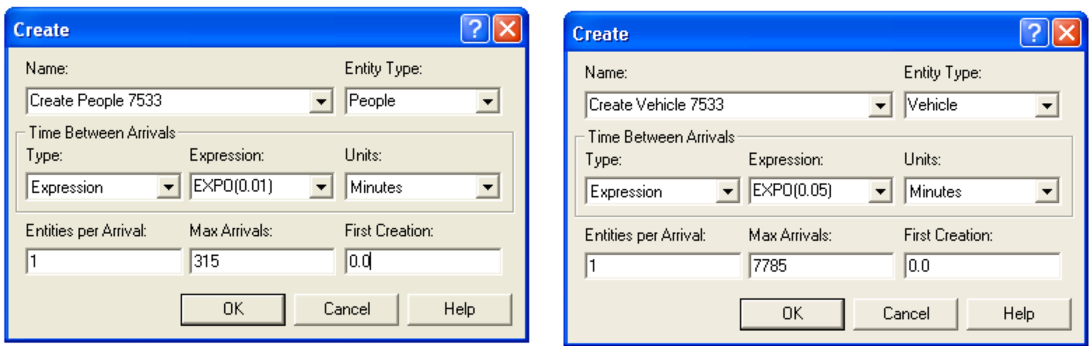


Figure 22: Arrivals

Currently, the percentages of people and vehicle types are randomly generated by the software in the form of cumulative discrete distributions (line 2,589 to line 2,611). Figure 23 shows that 48% of the *People* are *Pedestrian* while 52% are *Bicycle*; and 50% of the *Vehicle* are *Car*, 20% are *Truck*, and 30% are *SUV*. However, if other data is available to the users, the users can change them directly in the model. Note that data is imported into the program and numbered in Arena following their natural orders as created/arranged in the data files. For example, the order of *People* entity in the People file is *Pedestrian* and *Bicycle*, while their indexes in Arena are 1 and 2, respectively.

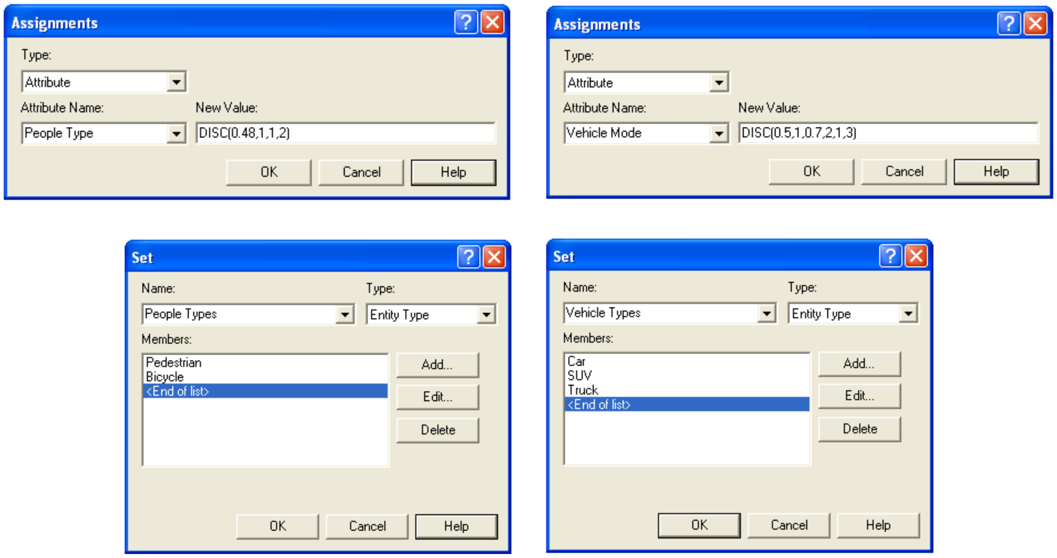


Figure 23: Assign Types to People and Vehicle Entities

More attributes are then assigned by the DOE_EVAC to the corresponding entities via Arena. Each *People* entity has an entity picture (attribute name: *Entity.Picture*), a closest transit center (*Transit Center*), and a travel time to the closest transit center (*People Travel Time*); while each *Vehicle* entity has an entity picture (*Entity.Picture*), a length (*Vehicle Length*), a destination (*Vehicle Destination*), and a sequence (*Entity.Sequence*) (from line 2,481 to line 2,543). Recall that the *Vehicle Destination* can be either specified by users (*DestDist* field in the *Nodes* table) or generated randomly by DOE_EVAC. Figure 24 shows that 35% of *Vehicle* entities go to destination 1777, 19% to destination 1782, 7% to destination 1776, and 39% to destination 1780.

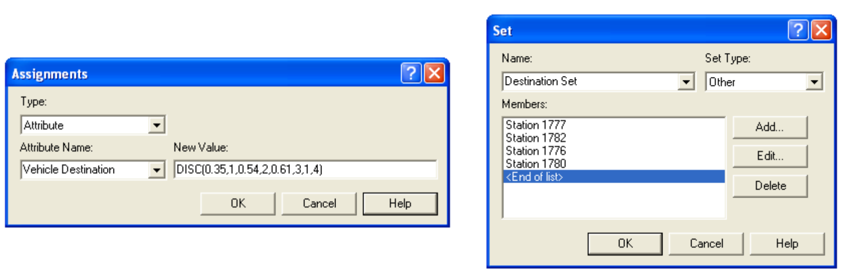


Figure 24: Vehicle Destination

DOE_EVAC keeps track of each *Vehicle* entity's path via Arena's *Sequence* module (from line 2,646 to line 2,685). Figure 25 displays the shortest path between origin 7533 and destination 1777 and reveals how the model routes vehicles to the next station. Whenever a *Vehicle* entity reaches a station (except destination station), the model searches for the entity's *Next Station* (*Entity.PlannedStation*) in the assigned sequence and sends the entity to the new station under allowable conditions. The conditions consist of (i) sufficient space in the queue buffer of the next station and (ii) a green traffic light in entity's moving direction for signalized stations. The entity is held in queue until those two conditions are satisfied (Figure 26).

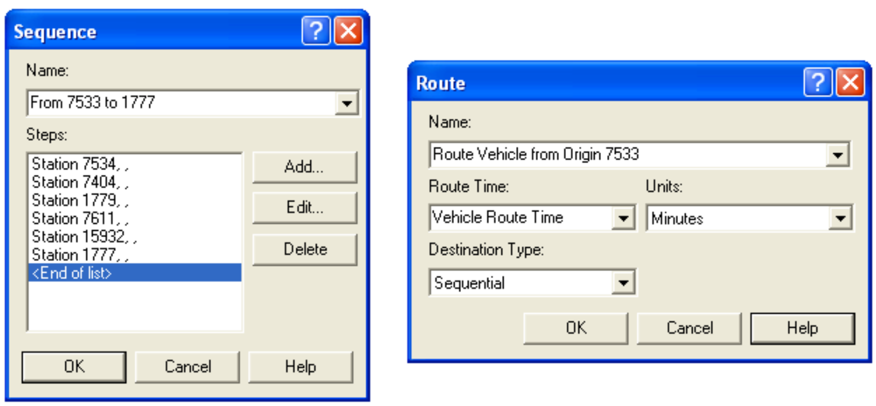


Figure 25: Vehicle Sequence and Route

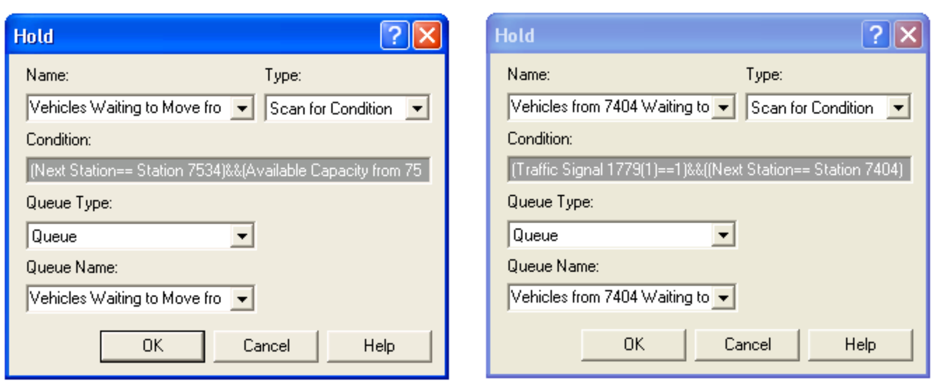


Figure 26: Vehicle Waiting to Move at Unsignalized and Signalized Stations

The *People* entities are then routed to the closest transit center (see line 2,710 to line 2,728 to see how to find closest transit center). At each transit center, the entity waits for a signal before it can be loaded onto a public transportation *Bus* entity. Each transit center has one unique signal and the value of the signal is the transit center ID. The signal is only released when the number of entities equals the public transportation capacity (*Public Transportation Size*) or when the last entity has to wait more than maximum allowable time (*Max Wait Time*). Syntax for the waiting condition at transit station 15931 is "*NQ (Hold People for Signal 15931.Queue) >= Public Transportation Size || (TNOW - People Arrival Time to Transit> Max Wait Time)*"

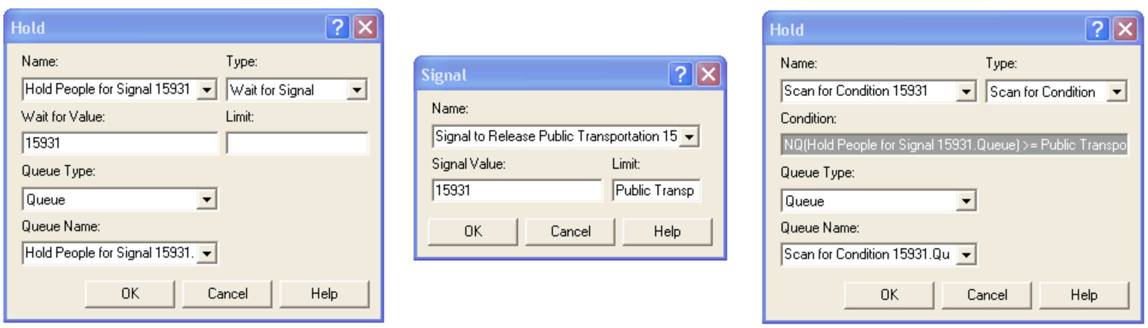


Figure 27: People Waiting for Public Transportation to Depart

Whenever an entity enters an intersection station, the available capacity of the entered queue buffer is recalculated:

$$\textit{Available Capacity from i to j} = \textit{Available Capacity from i to j} - \textit{Gap} - \textit{Vehicle Length} \quad (23)$$

The *Vehicle* entity continues moving to the end of the station with the moving (delay) time equal to the available capacity divided by the link speed and the numbers of lanes (Figure 28). It is because the vehicles are assumed to fill the link simultaneously across all lanes and front to back.

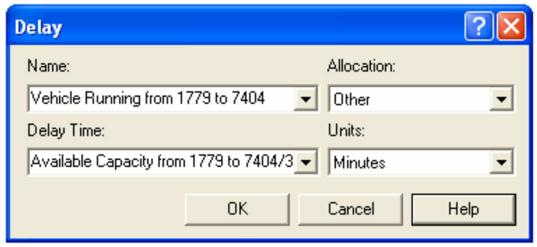


Figure 28: Vehicle Running on Link

Vehicle entities then wait for clear conditions to move to the next station (Figure 26). The traffic signal is generated for signalized intersections via the *Create* module (Figure 29). The signal phase length includes both green time and yellow time. Traffic signals at an intersection are stored in an array variable *Traffic Signal*, with each cell of the array representing the signal status of each incoming approach (from line 2,199 to line 2,359). For example, the three-leg intersection 1779 with only two incoming approaches (the third incoming approach starts from a destination 1782 and since no traffics can be generated from a destination, the approach is eliminated) has an array variable of size 2. Whenever one signal phase turns green (indicated by value 1) or yellow (0), all other signal phases become red (-1) (Figure 30).

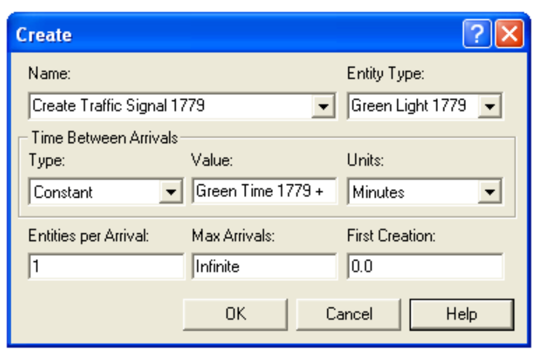


Figure 29: Create Traffic Signal

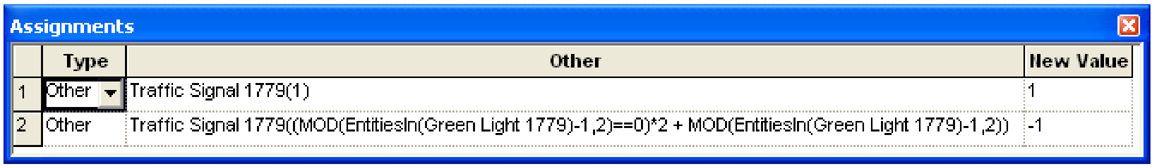


Figure 30: Manage Traffic Signals at Three-Leg Intersection

At destinations, the *People* entities leave the *Bus* (Figure 31). All statistics such as the number of people and the vehicles that left the system, total evacuation time, and so forth are recorded via *Record* modules (from line 1,379 to line 1,464).

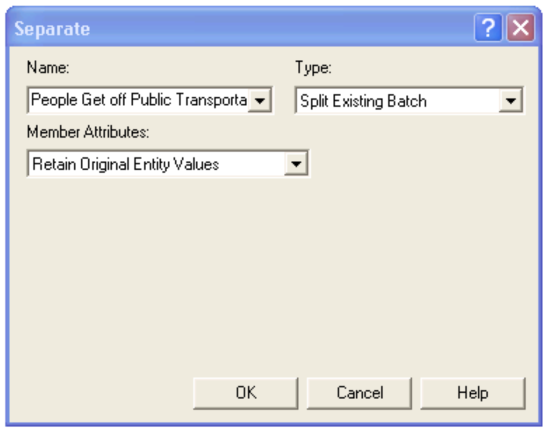


Figure 31: People Get off Public Transportation

Figure 32 presents the simulation run setup. The default terminating condition of evacuation model is when all people and vehicles egress the network. However, the users can terminate the simulation run at any time and view statistics such as how many people/vehicles are still in the network, how many public transportation vehicles have loaded, and so forth. The default number of replications is 20 (the minimum sample size for output data to be tested statistically).

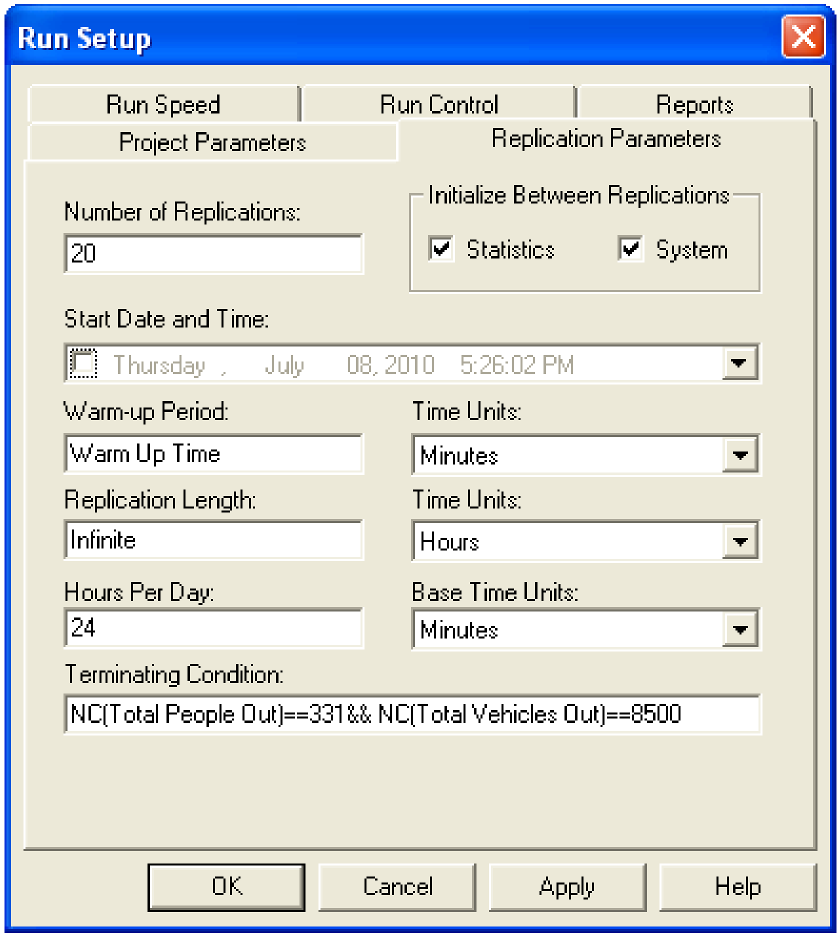


Figure 32: Default Simulation Run Setup

3.6 User Interruption

The DOE_EVAC model supports user interruptions to the simulation via Arena's Run Controller (Figure 33) so that changes of factors can be made to the input. For example, should the public transportation capacity change, the users can (1) temporarily suspend the simulation, (2) modify the capacity, and then, (3) continue the simulation run with the new capacity level. The Run Controller commands are found in Appendix C.

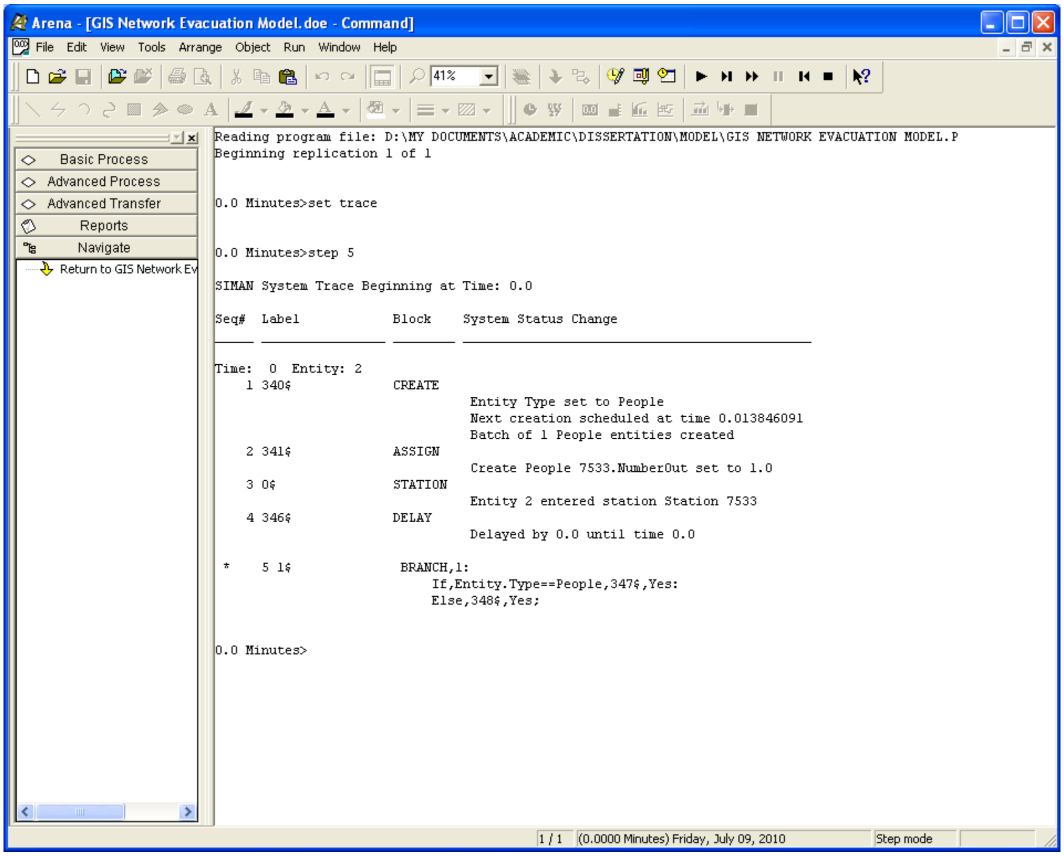


Figure 33: Arena's Run Controller

3.7 Output

The output data of the DOE_EVAC model are:

1. Total evacuation time: represented by *Total Evacuation Time*.
2. Total flow on each link (total number of vehicles): expressed as *Total Flow (Node) to (Node)*. For example, Total Flow 1779 to 7404 indicates the total flow on the link of which the *FromID* is 1779 and the *ToID* is 7404.
3. Average flow on each link (vehicles/time unit): expressed as *Average Flow (Node) to (Node)*, for example, Average Flow 1779 to 7404. The default time unit is minute, which is specified in Arena's Run Setup *Base*

Time Unit (see Figure 32). The user can change the base time unit in Run Setup as needed.

4. Average time in queue on each link (time unit): expressed as *Queue Name.Queue.Wait Time*, for example, Vehicles from 1779 Waiting to Traverse 7404.Queue.Wait Time.
5. Number of vehicles and number of people arrive at each destination: expressed as *Vehicles Out (Node)* and *People Out (Node)*, for example, Vehicles Out 1777 and People Out 1777.
6. Total number of vehicles and people exit the network: represented by *Total Vehicles Out* and *Total People Out*, respectively.

These output data are measures of effectiveness. The total evacuation time is the criteria to compare scenarios or evacuation plans. The flow and time in queue on links are used to determine link utilization and identify traffic congestion or bottlenecks on the network. The numbers of vehicles and people arriving at each destination can aid the emergency planners to prepare land facilities and services for evacuees. DOE_EVAC intentionally creates the Arena model that will generate the reports containing these output data at the end of the simulation runs. The DOE_EVAC output data can be found in the *User-Specified* and *Queues* reports. Figure 34 and Figure 35 illustrate a page of the *User-Specified* report and a page of *Queues* report, respectively that include DOE_EVAC output data. Other DOE_EVAC output data are found in Appendix E. The *Queues* report show the average, the maximum, and the minimum waiting time of each queue; and the half-width is the 95% confidence-interval half width of the observed values.

User Specified

Evacuation Model Replications: 1

Replication 1 Start Time: 0.00 Stop Time: 83.50 Time Units: Minutes

Counter

Count	Value
People Out 1776	155.00
People Out 1777	16.0000
People Out 1780	160.00
People Out 1782	0
Total Flow 15930 to 7611	1.0000
Total Flow 15931 to 1781	16.0000
Total Flow 15932 to 7611	0
Total Flow 1778 to 1781	203.00
Total Flow 1778 to 7407	2,435.00
Total Flow 1778 to 7534	0
Total Flow 1779 to 7404	0
Total Flow 1779 to 7611	1,546.00
Total Flow 1781 to 1778	8.0000
Total Flow 1781 to 7535	211.00
Total Flow 7404 to 1779	1,725.00
Total Flow 7404 to 7534	0
Total Flow 7405 to 15932	0
Total Flow 7405 to 7404	179.00
Total Flow 7405 to 7406	333.00
Total Flow 7405 to 7407	203.00
Total Flow 7407 to 1778	203.00
Total Flow 7533 to 7532	622.00
Total Flow 7533 to 7534	3,973.00
Total Flow 7533 to 7535	3,190.00
Total Flow 7534 to 1778	2,427.00
Total Flow 7534 to 7404	1,546.00
Total Flow 7535 to 1781	0
Total Flow 7611 to 15932	1,547.00
Total Flow 7611 to 1779	0
Total People Out	331.00
Total Vehicles Out	8,500.00
Vehicles Out 1776	2,576.00

Model Filename: D:\My Documents\ACADEMIC\Dissertation\GIS Network Evacuation Model Page 2 of 8

Figure 34: An Arena User Specified Report

Queues

Evacuation Model Replications: 1

Replication 1 Start Time: 0.00 Stop Time: 83.50 Time Units: Minutes

Vehicles from 1779 Waiting to Traverse 7611.Queue

Time	Average	Half Width	Minimum	Maximum
Waiting Time	0.4742	0.053175234	0	1.2255

Other	Average	Half Width	Minimum	Maximum
Number Waiting	8.7786	1.53459	0	55.0000

Vehicles from 1781 Waiting to Traverse 1778.Queue

Time	Average	Half Width	Minimum	Maximum
Waiting Time	0.4819	(Insufficient)	0.0987	1.1379

Other	Average	Half Width	Minimum	Maximum
Number Waiting	0.04617097	(Insufficient)	0	2.0000

Vehicles from 1781 Waiting to Traverse 7535.Queue

Time	Average	Half Width	Minimum	Maximum
Waiting Time	0.1382	(Insufficient)	0	0.6486

Other	Average	Half Width	Minimum	Maximum
Number Waiting	0.3493	(Insufficient)	0	23.0000

Vehicles from 7404 Waiting to Traverse 1779.Queue

Time	Average	Half Width	Minimum	Maximum
Waiting Time	0.1355	0.024820634	0	0.6977

Other	Average	Half Width	Minimum	Maximum
Number Waiting	2.8001	0.556010723	0	36.0000

Model Filename: D:\My Documents\ACADEMIC\Dissertation\GIS Network Evacuation Model Page 6 of 21

Figure 35: An Arena Queue Report

For the user's convenience, all output data of all replications of (1) and (2) are recorded into Arena output data files: *TotalEvacuationTime.dat*, and

Flow(Node)to(Node).dat (e.g., Flow1779to7404.dat). The *TotalEvacuationTime.dat* file contains *n Total Evacuation Time* for *n* simulation replications (one data point for each replication). The *Flow(Node)to(Node).dat* contains the vehicle count during the simulation run.

Note that the Arena output data files can only be opened via Arena's Output Analyzer. If the user would like access and manipulate the output data, the user must first export the Arena output files into usable files (e.g., .txt, .dax, etc.) via Arena Output Analyzer. The export function can be found via File → Data Files → Export; and the Output Analyzer export data window is shown below:

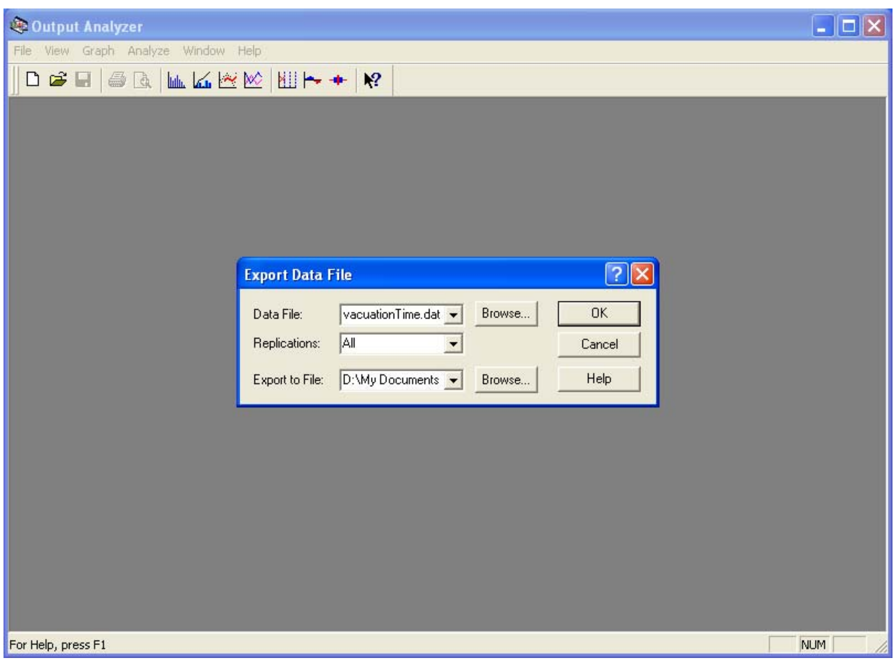


Figure 36: Data Exported via Arena Output Analyzer

If the user would like to obtain the vehicle count over each time unit period of the simulation run (e.g., each minute), DOE_EVAC can export the data of the usable *Flow(Node)to(Node)* files above to the desired data via *Export* menu item in *File* menu of DOE_EVAC.

CHAPTER 4

MODEL VALIDATION

4.1 Validation Methods

One of the most difficult tasks facing a simulation analyst is whether the developed simulation model is valid. Unlike other computer programs, a simulation model is never absolutely validated since it is only an approximation to a real-world system. Validation can only be judged "relative to those measures of performance that will actually be used for decision making" (Law, 2007).

Generally, a simulation model can be validated from three different perspectives – the modeler, the technical evaluator, and the ultimate user – via three questions (Pegden, Shannon, & Sadowski, 1995):

- "Does the model adequately represent the real-world system (conceptual validity)?
- Are the model-generated behavioral data characteristic of the real-world system's behavioral data (operational validity)?
- Does the simulation model's ultimate user have confidence in the model's results (believability)?"

To answer these questions, a variety of validation tests (see Pegden et al., 1995) can be performed on the basis of continuous interactions among perspective users during the model development process. They are classified into three major categories: reasonableness tests (e.g., continuity, consistency, degeneracy, and absurd conditions), model structure and data tests (e.g., face validity, parameters and relationships, structural and boundary verification, and sensitivity analysis), and model behavior tests (e.g., behavior comparison, symptom generation, behavior anomaly, and behavior prediction). In all of the above, the behavior comparison test is the most widely used test to study the model behavior in relation to the behavior of the referent system (Law, 2007); and it is applied in this research to compare the simulation output to the referent system output. The behavior comparison test is typically conducted via the confidence interval statistical procedure for two samples, which is described below.

Let “1” represent the real-world system and “2” be the corresponding simulation model, where $\overline{X}_i(n_i)$ and $S_i(n_i)$ are the mean and the standard deviation of a parameter of interest (e.g., total evacuation time) of system *i* (*i* = 1, 2), n_i is the size of the data samples *i*, and $\hat{f}$ is the estimated degrees of freedom. Then,

$$\overline{X}_i(n_i) = \frac{\sum_{j=1}^{n_i} X_{ij}}{n_i} \qquad \text{and} \qquad S_i^2(n_i) = \frac{\sum_{j=1}^{n_i}\left[X_{ij} - \overline{X}_i(n_i)\right]^2}{n_i - 1} \tag{24}$$

$$\hat{f} = \frac{\left[S_1^2(n_1)/n_1 + S_2^2(n_2)/n_2\right]^2}{\left[S_1^2(n_1)/n_1\right]^2/(n_1-1) + \left[S_2^2(n_2)/n_2\right]^2/(n_2-1)} \tag{25}$$

The 100(1-α)% two-sided confidence interval (C.I.) with α significance level for the difference between means is

$$\overline{X}_1(n_1) - \overline{X}_2(n_2) \pm t_{\hat{f},1-\alpha/2}\sqrt{\frac{S_1^2(n_1)}{n_1} + \frac{S_2^2(n_2)}{n_2}} \tag{26}$$

where $t_{\hat{f},1-\alpha/2}$ is the critical *t* value at $\hat{f}$ degrees of freedom.

If the confidence interval contains zero, the observed difference between the two systems is said to be not statistically significant at level α. Then, simulationist can conclude that the simulation model is mimicking the behavior of the parameter of interest and the model is therefore, credible (validated).

The simulationist can also conduct the behavior comparison test via an equivalent procedure – the hypothesis testing on the difference in means. The hypothesis test yields the same conclusions as those in the confidence interval statistical procedure (Montgomery & Runger, 2007). The null hypothesis for two samples in the hypothesis test is $H_0 : \mu_1 = \mu_2$. A *t*-statistic t_0 is used to test the hypothesis:

$$t_0 = \frac{\overline{X}_1(n_1) - \overline{X}_2(n_2)}{\sqrt{\frac{S_1^2(n_1)}{n_1} + \frac{S_2^2(n_2)}{n_2}}} \tag{27}$$

If $-t_{\hat{f},1-\alpha/2} < t_0 < t_{\hat{f},1-\alpha/2}$, the null hypothesis is not rejected and the difference between two systems is not statistically significant. Consequently, the model is validated.

In the case that only one observed data point (usually the deemed "true" mean) exists for the real-world observation (typically, the true standard deviation does no exist), a 100(1-α)% two-sided C.I. with α significance level is obtained for the simulation output as:

$$\overline{X}(n) \pm t_{n-1,1-\alpha/2} \sqrt{\frac{S^2(n)}{n}} \tag{28}$$

where *n* is the number of simulation replications, $t_{n-1,1-\alpha/2}$ is the critical t value at (*n-1*) degrees of freedom, and $\overline{X}(n)$ and $S(n)$ are the mean and the standard deviation of *n* replications of the parameter of interest. If the confidence interval contains the real-world's observed mean, the C.I. it is said to "cover" the true mean at level α and the simulation model is considered validated and representative of the real-world system.

Again, the simulationist can use the hypothesis test to conduct the behavior comparison test for this case. The null hypothesis is $H_0 : \mu = \mu_0$ and the *t*-statistic t_0 is:

$$t_0 = \frac{\overline{X}(n) - \mu_0}{S / \sqrt{n}} \tag{29}$$

If $-t_{n-1,1-\alpha/2} < t_0 < t_{n-1,1-\alpha/2}$, the null hypothesis is not rejected. The simulation output mean is not statistically significant different from the true mean; and thus, the model can be considered a representative of the real-world system.

4.2 Validation Methodology

Since the DOE_EVAC model will be used to simulate various types and sizes of networks, the basic logic utilized in DOE_EVAC is validated against a smaller real-world system. Note that a larger network is a cluster of multiple smaller networks and these sub-networks are connected to each other smoothly (e.g., a destination node of a sub-network A can be an origin node of a neighbor sub-network B). In addition, the required operational functions for the larger network are exactly the same as for its sub-networks (e.g., operations of nodes or intersection management). Thus, if the basic logic of DOE_EVAC can accurately simulate the logic of smaller networks, larger networks built on these validated smaller networks will also be valid. This approach is called piece-wise method (Kron, 1963; Chusanapiputt & Phoomvuthisarn, 2000).

The essential steps to perform the basic model validation logic for the DOE_EVAC model are listed as follows:

1. Collect data on the real-world system's parameters of interest and control variables (i.e., the total evacuation time, the traffic flows, the number of

vehicles arriving to destinations from each origin, the intersection green and yellow time)

2. Use the collected data (e.g., interarrival times at origin nodes) to identify the probability distributions for the DOE_EVAC model by following four steps (Law, 2007):
   a. Test the data for independence (e.g., scatter plot, correlation plot, and run tests)
   b. Hypothesize the distribution family using its descriptive statistics (e.g., summary statistics of mean, standard deviation) and its shape (graphical representations such as histogram and box plots)
   c. Estimate the parameters for the distributions of (b) (e.g., maximum-likelihood estimators)
   d. Test the fit (heuristic procedures such as P-P plot and Q-Q plot; goodness-of-fit tests such as Chi-Square test and Kolmogorov-Smirnov test)
3. Use the distributions of Step 2 as the input to calibrate the data files for the DOE_EVAC model
4. Run the DOE_EVAC model to generate the Arena simulation model
5. Run the simulation model for 20 replications and retrieve the Arena output data files
6. Calculate the confidence intervals to compare the simulation output parameters with the referent observed data

4.3 Validated Network, DOE_EVAC Results, and Comparison

No data on large-scale transportation evacuations can be found in the public domain. However, comparisons between the DOE_EVAC model's output and congested real-world traffic data can be used to investigate model behavior. The University of Oklahoma's campus corner (Figure 37) is chosen as the "real-world" system to validate the model's behavior. The traffic on West Boyd Street (W. Boyd), between South University Boulevard (S. University) and Asp Avenue (Asp) is investigated. The comparative parameters of interest are the total "evacuation" time (time between the first arrival to the network and the last vehicle exiting from the network), the traffic flows on each link, and the number of vehicles leaving the network at each destination.

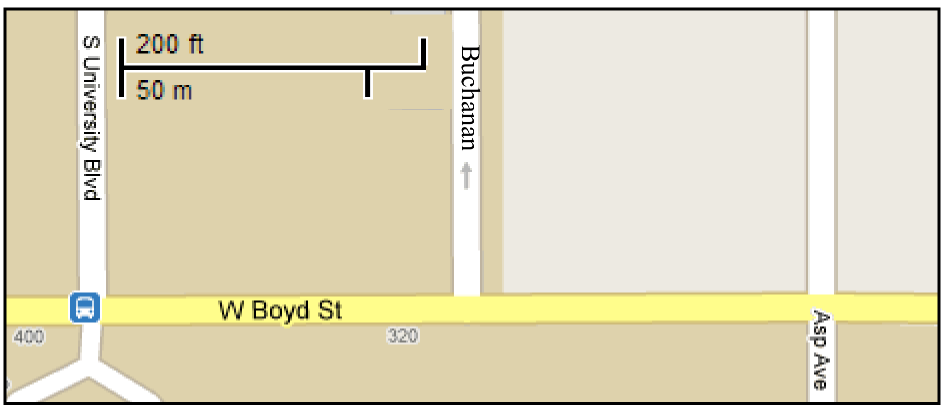


Figure 37: Map of the University of Oklahoma Campus Corner

For this system, the streets are represented or converted to the applicable network with nodes and links as shown in Figure 38. There are 7 origins (Nodes 1, 3, 5, 7, 9, 11, 13), 7 destinations (Nodes 2, 4, 6, 8, 10, 12, 14), 3 intersections (Nodes 15, 16, 17 in which nodes 15 and 16 are signalized), and 0 transit center. 4 links inside the network are investigated. The link's name consists of two nodes, for example link between node 15 and node 17 has name of "15-17". Other links act as dummy links since the vehicles were only counted when they reached the intersections. However, the dummy links are necessary since the model requires that a node can only be of one type (either an origin, a destination, an intersection, or a transit center). Note that since there are parking slots along W. Boyd in the direction from Asp to S. University, and the vehicles can come in and out of the parking slots, all vehicles that parked were considered as "leaving" the network via node 10 and all vehicles that went out of the parking slots were considered arriving to the network via node 11.

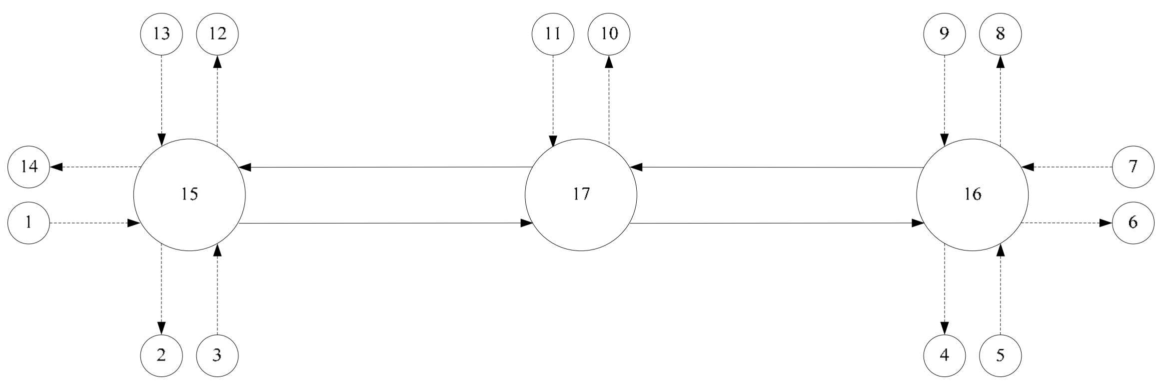


Figure 38: OU Campus Corner Network of Links and Nodes

The validation methodology was applied as follows:

<u>Step 1:</u>

Traffic on W. Boyd between S. University and Asp was video-monitored during peak hours from 4pm to 6pm on June 22nd, 2010 when traffic is congested. Pedestrian traffic was negligible and not disruptive to the traffic flow (as assumed in the model). Incoming traffic data were collected every minute at each origin. The vehicles arriving during the observing period were traced until they exited the network; and the time the last traced vehicle exited the network was attained.

Step 2 and 3:

The scatter plot of X_i versus X_{i+1} were first drawn to test the independence of the obtained data at each origin. Figure 39 shows the scatter plot of number of arrivals per minute at origin 7. The plot verifies the independence of data since there is no obvious pattern appearing in the plot. The results are the same for the number of arrivals per minute at all other origins (i.e., independent).

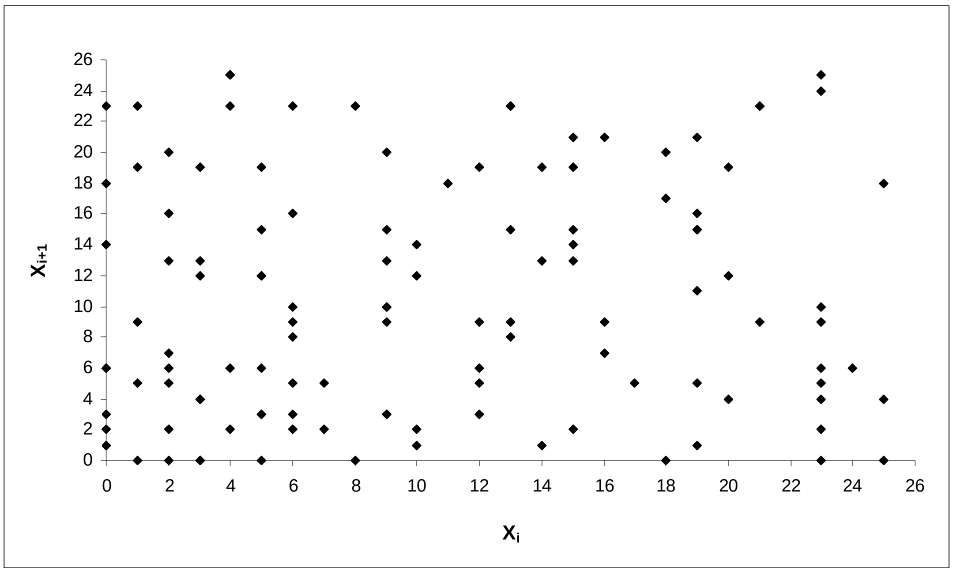


Figure 39: Scatter Plot of Number of Arrivals per Minute at Origin 7

The next steps are hypothesizing the family using shape, estimating the parameters, and testing the fit (see section 4.1). Here, the Arena Input Analyzer was used to fit the independent data into probability distributions. Refer to Arena's probability distribution notations in Table 8. Table 10 exhibits the fitted probability distributions of the number of arrivals per minute at each origin node. These arrival processes (number of arrivals) were then used to obtain the interarrival times at the origins (Table 10). It is well-known that if an arrival process follows a Poisson distribution with rate λ, its corresponding arrival times (time between arrivals) is exponential with mean 1/λ.

Table 10: Arrival Processes and Interarrival Times at Origin Nodes

Node	**Arrival Process** (min)	**Interarrival Time** (min)
1	POISSON(12.5)	EXPO(0.06)
3	POISSON(0.9)	EXPO(1.11)
5	POISSON(2.8)	EXPO(0.36)
7	POISSON(14.5)	EXPO(0.07)
9	POISSON(2.9)	EXPO(0.34)
11	POISSON(0.01)	EXPO(7)
13	POISSON(4.5)	EXPO(0.22)

Each vehicle entering the network was traced in order to determine the percentage of vehicles going to each destination. Table 11 reveals the percentage of traffic going from each origin to each destination and these values are used to compute the destination distribution from each origin in Table 12. The last column of Table 11 contains the total number of vehicles entering the network from each origin during the entire observation period. These values and the values of Table 12 will be used as the number of “vehicle” in the *Vehicle* column and the destination distribution in the *DestDist* column of the *Nodes* table (see Table 14), respectively.

Table 11: Traffic Percentages from Origin to Destination

To Node / From Node	**2**	**4**	**6**	**8**	**10**	**12**	**14**	**Total**
1	1.29%	9.03%	72.90%	1.94%	4.52%	10.32%	0.00%	1782
3	0.00%	11.11%	22.22%	0.00%	0.00%	44.44%	22.22%	103
5	0.00%	0.00%	25.00%	28.57%	0.00%	0.00%	46.43%	322
7	4.14%	7.59%	0.00%	4.14%	5.52%	2.07%	76.55%	1667
9	0.00%	6.90%	68.97%	0.00%	3.45%	3.45%	17.24%	333
11	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%	11
13	4.44%	4.44%	37.78%	0.00%	4.44%	0.00%	48.89%	517

Table 12: Destination Distributions at Origin Nodes

Node	Cumulative Discrete Distribution
1	DISC(0.01,1,0.1,2,0.83,3,0.85,4,0.9,5,1,6,1,7)
3	DISC(0,1,0.11,2,0.33,3,0.33,4,0.33,5,0.78,6,1,7)
5	DISC(0,1,0,2,0.25,3,0.54,4,0.54,5,0.54,6,1,7)
7	DISC(0.04,1,0.12,2,0.12,3,0.16,4,0.21,5,0.23,6,1,7)
9	DISC(0,1,0.07,2,0.76,3,0.76,4,0.79,5,0.83,6,1,7)
11	DISC(0,1,0,2,0,3,0,4,0,5,0,6,1,7)
13	DISC(0.04,1,0.09,2,0.47,3,0.47,4,0.51,5,0.51,6,1,7)

During the observation period, the network was congested and vehicles ran much slower at (e.g., 10mph to 20mph). Thus, the link speed of 15mph was applied for all links (see Table 15).

The signal phase for each direction on W. Boyd at both intersections 15 and 16 are 43 s including green and yellow time (as in the model). The green time is 40 s and the yellow time is 3 s. There is one signalized left turn (left turn that has an arrow signal) at each intersection: from W. Boyd to Asp (South) and from W. Boyd to S. University (North); and both left turn times are 10 s. These values are used as the *Green* and *Yellow* time in the *Nodes* table (see Table 14).

The means and the standard deviations of the link flows (vehicles per minute) are in Table 13. The flows were collected at the entrances of the links. The corresponding output values produced by DOE_EVAC will be compared to these link flows.

Table 13: Link Flows (Vehicles per Minute)

Link	15-17	17-16	16-17	17-15
Mean	16.10	15.20	14.80	13.90
Standard Deviation	5.04	4.76	3.33	3.70

The time the last traced vehicle went out of the network (versus entering at 4:00pm) is 2 hours 3.13 min (123.13 min).123.13 min represents the real-world system's observed or “true” total “evacuation” time; and its corresponding parameter in DOE_EVAC is *Total Evacuation Time*. The *Total Evacuation Time* is the parameter of interest produced by DOE_EVAC for comparing with the observed corresponding parameter, the true total “evacuation” time. If 123.13 min is contained within the 95% C.I. of the simulated *Total Evacuation Time*, DOE_EVAC will be statistically valid.

Table 14 - Table 16 contain the calibrated data for the validated network. The tables are ready to be imported in DOE_EVAC to generate the Arena evacuation model. Note that since pedestrian is negligible, the *People* table is left blank.

Table 14: DOE_EVAC *Nodes* Table of Validated Network

Dataview1 - OUNODES

ID	TYPE	GREEN	YELLOW	PEOTIME	VEHTIME	PEOPLE	VEHICLE	DESTDIST
1	0	0	0	0	EXPO(0.06)	0	1751	DISC(0.01,1,0.1,2,0.83,3,0.85,4,0.9,5,1,6,1,7)
2	1	0	0			--	--	
3	0	0	0	0	EXPO(1.11)	0	101	DISC(0,1,0.11,2,0.33,3,0.33,4,0.33,5,0.78,6,1,7)
4	1	0	0			--	--	
5	0	0	0	0	EXPO(0.36)	0	316	DISC(0,1,0,2,0.25,3,0.54,4,0.54,5,0.54,6,1,7)
6	1	0	0			--	--	
7	0	0	0	0	EXPO(0.07)	0	1638	DISC(0.04,1,0.12,2,0.12,3,0.16,4,0.21,5,0.23,6,1,7)
8	1	0	0			--	--	
9	0	0	0	0	EXPO(0.34)	0	327	DISC(0,1,0.07,2,0.76,3,0.76,4,0.79,5,0.83,6,1,7)
10	1	0	0			--	--	
11	0	0	0	0	EXPO(7)	0	11	DISC(0,1,0,2,0,3,0,4,0,5,0,6,1,7)
12	1	0	0			--	--	
13	0	0	0	0	EXPO(0.22)	0	508	DISC(0.04,1,0.09,2,0.47,3,0.47,4,0.51,5,0.51,6,1,7)
14	1	0	0			--	--	
15	3	0.65	0.08			--	--	
16	3	0.65	0.08			--	--	
17	3	0	0			--	--	

Table 15: DOE_EVAC *Links* Table of Validated Network

Dataview1 - OULINKS

ID	LENGTH	DIR	FROMID	TOID	ABLANES	BALANES	ABFLOWTIME	BAFLOWTIME	ABSPEED	BASPEED	FULLNAME
21438	0.01	1	15	14	2	0	0.04	0.00	15.00	0.00	W Boyd St
27621	0.01	1	15	2	1	0	0.04	0.00	15.00	0.00	Parrington Oval
46319	0.01	1	3	15	1	0	0.04	0.00	15.00	0.00	Parrington Oval
33784	0.06	2	17	15	2	2	0.24	0.24	15.00	15.00	W Boyd St
3009	0.05	2	16	17	2	2	0.20	0.20	15.00	15.00	W Boyd St
27665	0.01	1	7	16	2	0	0.04	0.00	15.00	0.00	W Boyd St
46362	0.01	1	9	16	1	0	0.04	0.00	15.00	0.00	Asp Ave
46333	0.01	1	11	17	1	0	0.04	0.00	15.00	0.00	Buchanan Ave
9208	0.01	1	13	15	1	0	0.04	0.00	15.00	0.00	S University Blvd
9211	0.01	1	16	4	1	0	0.04	0.00	15.00	0.00	Asp Ave
46367	0.01	1	1	15	2	0	0.04	0.00	15.00	0.00	W Boyd St
46364	0.01	1	16	6	2	0	0.04	0.00	15.00	0.00	W Boyd St
46365	0.01	1	16	8	1	0	0.04	0.00	15.00	0.00	Asp Ave
46366	0.01	1	17	10	1	0	0.04	0.00	15.00	0.00	Buchanan Ave
46368	0.01	1	5	16	1	0	0.04	0.00	15.00	0.00	Asp Ave
46369	0.01	1	15	12	1	0	0.04	0.00	15.00	0.00	S University Blvd

Table 16: DOE_EVAC *Vehicles* Table of Validated Network

Dataview1 - OUVEHS

TYPE	LENGTH
Car	13.50
SUV	16.40

Step 4:

DOE_EVAC was run for the validated network. Figure 40 and Figure 41 show the GIS map and the Arena model of the validated network generated by DOE_EVAC, respectively.

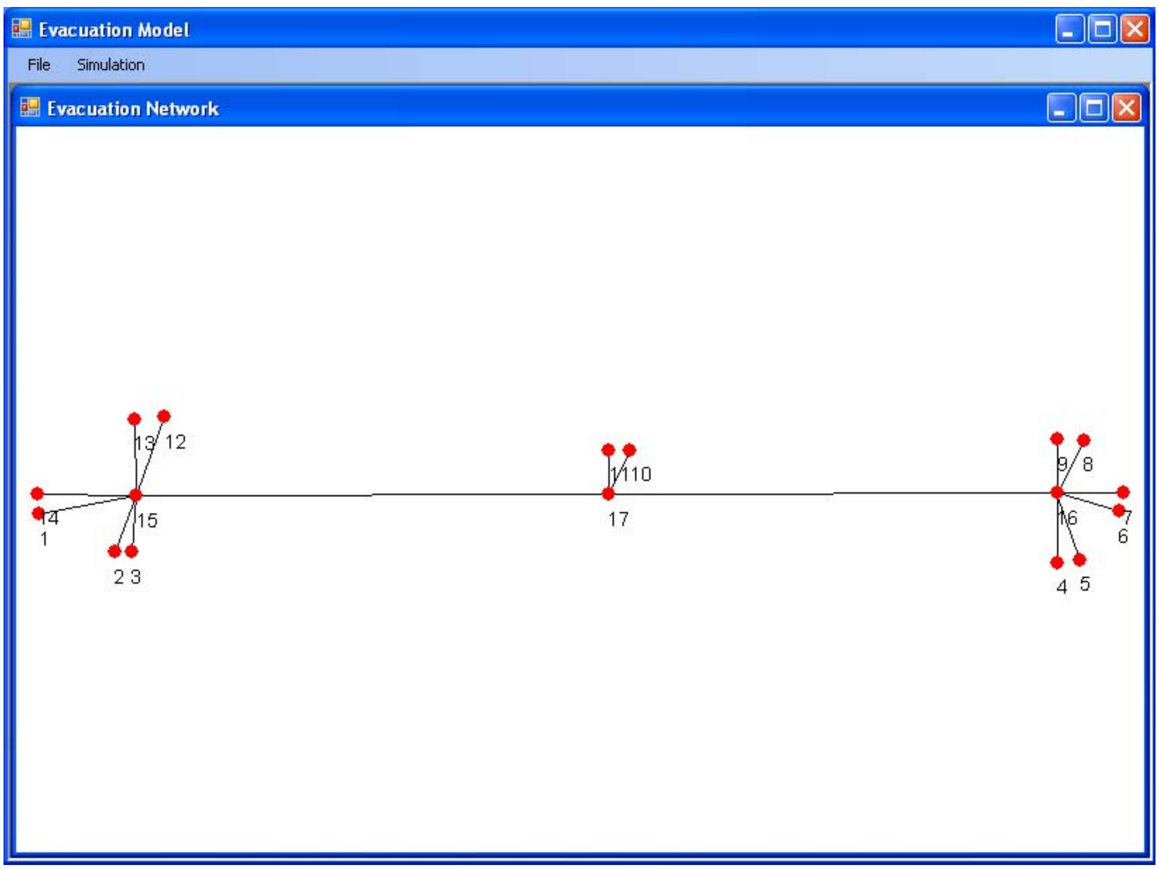


Figure 40: GIS Map of Validated Network

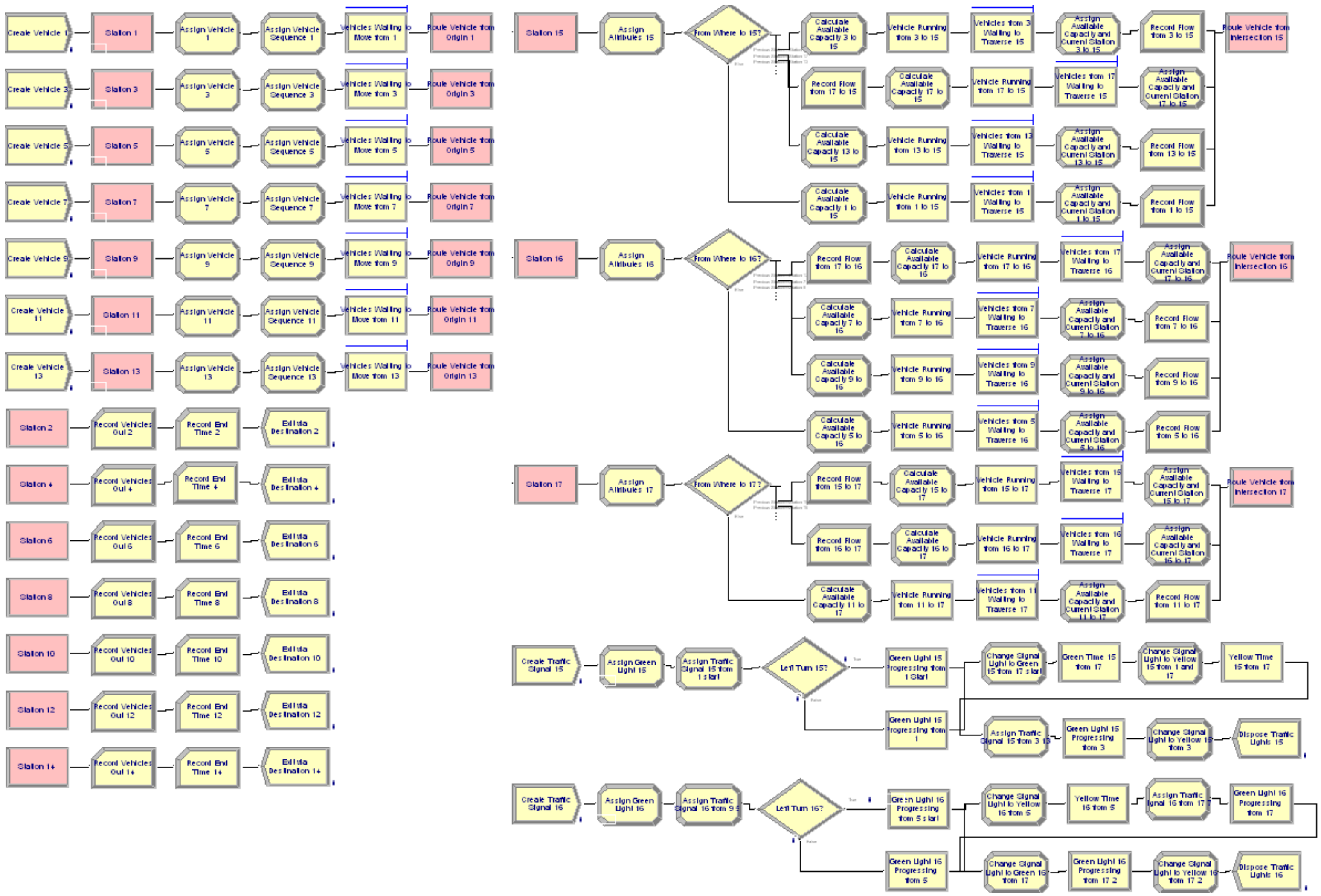


Figure 41: Arena Model of Validated Network

Step 5 and 6:

The Arena model was run for 20 replications. Table 17 reveals the *Total Evacuation Time* for each simulation run and the 95% C.I. for all 20 replications. The 95% C.I. was calculated by following Equation 28. It is obvious that the 95% C.I., which ranges from 118.62 to 123.94, covers the true total "evacuation" time of 123.13.

Table 17: Simulated *Total Evacuation Time*

Replication	***Total Evacuation Time***
1	126.94
2	128.29
3	118.49
4	123.03
5	119.85
6	126.29
7	115.28
8	116.29
9	120.03
10	116.73
11	118.73
12	120.29
13	117.59
14	136.73
15	122.29
16	120.73
17	123.73
18	114.19
19	113.42
20	126.74
Mean	**121.28**
95% C.I.	**(118.62, 123.94)**

Given the real world's observed (true) value for the mean link flow $\overline{X}(n)$ on link 15-17 as 16.10 vehicles/min and its standard deviation $S(n)$ as 5.04 vehicles/min (see Table 13), Table 18 presents the 95% C.I. of the difference between the real-world's observed value (true value) of average link flow on link 15-17 and the simulated link flow (20 replications). The 95% C.I. was calculated following Equation 26. 19 out of 20

(95%) of confidence intervals for the link 15-17 contain 0 ("True" in *Coverage* column). As expected, 95% of the observed difference between the real-world value of performance and the model output is not statistically significant.

Table 18: 95% C.I. of Difference between True and Simulated Flows of Link 15-17

j	Run	n	$\overline{X_j}(n_j)$	$S_j(n_j)$	$\overline{X}(n)-\overline{X_j}(n_j)$	Confidence Interval	Coverage
1	Replication 1	123	15.04	5.98	1.06	1.06 ± 1.40	True
2	Replication 2	113	16.40	6.05	-0.30	-0.30 ± 1.45	True
3	Replication 3	118	15.52	6.66	0.58	0.58 ± 1.52	True
4	Replication 4	114	16.18	4.84	-0.08	-0.08 ± 1.28	True
5	Replication 5	119	15.24	6.54	0.86	0.86 ± 1.50	True
6	Replication 6	107	17.37	4.99	-1.27	-1.27 ± 1.32	True
7	Replication 7	113	16.13	4.86	-0.03	-0.03 ± 1.28	True
8	Replication 8	109	16.67	4.10	-0.57	-0.57 ± 1.20	True
9	Replication 9	109	16.78	5.26	-0.68	-0.68 ± 1.35	True
10	Replication 10	116	15.83	6.31	0.27	0.27 ± 1.47	True
11	Replication 11	118	15.58	6.14	0.52	0.52 ± 1.44	True
12	Replication 12	115	15.75	5.40	0.35	0.35 ± 1.35	True
13	Replication 13	113	16.35	5.86	-0.25	-0.25 ± 1.42	True
14	Replication 14	136	13.38	7.46	2.72	2.72 ± 1.56	False
15	Replication 15	105	17.38	5.04	-1.28	-1.28 ± 1.33	True
16	Replication 16	120	15.53	7.25	0.57	0.57 ± 1.60	True
17	Replication 17	120	14.97	6.22	1.13	1.13 ± 1.45	True
18	Replication 18	113	16.42	5.05	-0.32	-0.32 ± 1.31	True
19	Replication 19	110	16.66	4.05	-0.56	-0.56 ± 1.19	True
20	Replication 20	119	15.32	5.62	0.78	0.78 ± 1.37	True

Table 19 shows the 95% C.I. of the difference between the real-world's observed (true) value of average link flows versus simulated link flows on link 17-16. The real-world's true values for link flow mean and standard deviation are 15.20 and 4.76 vehicles/min, respectively (see Table 13). Similar to that of the link 15-17, 19 out of 20 (95%) confidence intervals for the link 17-16 contain 0. Thus, 95% of the observed difference between the real-world's true value and the model output is not statistically significant.

Table 19: 95% C.I. of Difference between True and Simulated Flows of Link 17-16

j	Run	n	$\overline{X_j}(n_j)$	$S_j(n_j)$	$\overline{X}(n) - \overline{X_j}(n_j)$	Confidence Interval	Coverage
1	Replication 1	123	14.12	5.72	1.08	1.08 ± 1.34	True
2	Replication 2	113	15.35	5.72	-0.15	-0.15 ± 1.37	True
3	Replication 3	118	14.62	6.36	0.58	0.58 ± 1.44	True
4	Replication 4	114	15.29	4.68	-0.09	-0.09 ± 1.22	True
5	Replication 5	119	14.33	6.32	0.87	0.87 ± 1.43	True
6	Replication 6	106	16.33	4.78	-1.13	-1.13 ± 1.26	True
7	Replication 7	114	15.00	4.92	0.20	0.20 ± 1.25	True
8	Replication 8	110	15.49	4.21	-0.29	-0.29 ± 1.17	True
9	Replication 9	110	15.73	5.31	-0.53	-0.53 ± 1.32	True
10	Replication 10	116	14.97	6.05	0.23	0.23 ± 1.41	True
11	Replication 11	118	14.69	5.81	0.51	0.51 ± 1.36	True
12	Replication 12	115	14.79	5.12	0.41	0.41 ± 1.28	True
13	Replication 13	113	15.37	5.68	-0.17	-0.17 ± 1.36	True
14	Replication 14	136	12.61	7.03	2.59	2.59 ± 1.47	False
15	Replication 15	106	16.45	5.25	-1.25	-1.25 ± 1.33	True
16	Replication 16	121	14.50	6.96	0.70	0.70 ± 1.52	True
17	Replication 17	120	14.04	5.96	1.16	1.16 ± 1.38	True
18	Replication 18	113	15.36	4.85	-0.16	-0.16 ± 1.25	True
19	Replication 19	110	15.69	3.90	-0.49	-0.49 ± 1.13	True
20	Replication 20	119	14.42	5.38	0.78	0.78 ± 1.30	True

Table 21 and Table 22 reveal the 95% C.I. of the difference between the real-world true value of average link flow versus simulated link flow on link 16-17 and 17-15, respectively. The mean and standard deviation of the true values are in Table 13. The model generates the link flows accurately with 100% (20 out of 20) confidence intervals containing 0 for both links. Thus, 100% of the observed difference between real-world value and the model output is not statistically significant. The validity of the model on generating traffic flow is confirmed.

Table 20: 95% C.I. of Difference between True and Simulated Flows of Link 16-17

j	Run	n	$\overline{X_j}(n_j)$	$S_j(n_j)$	$\overline{X}(n)-\overline{X_j}(n_j)$	Confidence Interval	Coverage
1	Replication 1	123	13.68	5.53	1.12	1.12 ± 1.16	True
2	Replication 2	120	13.87	4.60	0.93	0.93 ± 1.03	True
3	Replication 3	114	14.42	4.34	0.38	0.38 ± 1.01	True
4	Replication 4	121	13.76	5.11	1.04	1.04 ± 1.10	True
5	Replication 5	117	14.08	3.72	0.72	0.72 ± 0.91	True
6	Replication 6	119	13.87	4.47	0.93	0.93 ± 1.01	True
7	Replication 7	116	14.29	4.13	0.51	0.51 ± 0.97	True
8	Replication 8	117	14.32	5.61	0.48	0.48 ± 1.19	True
9	Replication 9	119	14.00	5.10	0.80	0.80 ± 1.10	True
10	Replication 10	115	14.60	3.91	0.20	0.20 ± 0.94	True
11	Replication 11	119	13.98	4.55	0.82	0.82 ± 1.02	True
12	Replication 12	118	13.90	4.24	0.90	0.90 ± 0.98	True
13	Replication 13	118	14.14	4.29	0.66	0.66 ± 0.99	True
14	Replication 14	115	14.72	3.86	0.08	0.08 ± 0.93	True
15	Replication 15	123	13.80	5.56	1.00	1.00 ± 1.16	True
16	Replication 16	115	14.43	3.91	0.37	0.37 ± 0.94	True
17	Replication 17	115	14.64	4.18	0.16	0.16 ± 0.98	True
18	Replication 18	114	14.56	3.64	0.24	0.24 ± 0.90	True
19	Replication 19	114	14.38	4.28	0.42	0.42 ± 1.00	True
20	Replication 20	115	14.46	3.94	0.34	0.34 ± 0.94	True

Table 21: 95% C.I. of Difference between True and Simulated Flows of Link 17-15

j	Run	n	$\overline{X_j}(n_j)$	$S_j(n_j)$	$\overline{X}(n)-\overline{X_j}(n_j)$	Confidence Interval	Coverage
1	Replication 1	124	12.90	5.29	1.00	1.00 ± 1.15	True
2	Replication 2	123	12.85	4.88	1.05	1.05 ± 1.10	True
3	Replication 3	114	13.61	4.00	0.29	0.29 ± 1.00	True
4	Replication 4	122	12.84	4.99	1.06	1.06 ± 1.12	True
5	Replication 5	118	13.32	3.71	0.58	0.58 ± 0.95	True
6	Replication 6	123	12.82	4.86	1.08	1.08 ± 1.09	True
7	Replication 7	116	13.62	4.08	0.28	0.28 ± 1.00	True
8	Replication 8	117	13.68	5.22	0.22	0.22 ± 1.17	True
9	Replication 9	119	13.38	4.90	0.52	0.52 ± 1.11	True
10	Replication 10	115	13.72	3.47	0.18	0.18 ± 0.93	True
11	Replication 11	119	13.32	4.32	0.58	0.58 ± 1.03	True
12	Replication 12	121	12.93	4.34	0.97	0.97 ± 1.03	True
13	Replication 13	118	13.46	4.02	0.44	0.44 ± 0.99	True
14	Replication 14	115	14.07	3.64	-0.17	-0.17 ± 0.95	True
15	Replication 15	123	13.05	5.06	0.85	0.85 ± 1.12	True
16	Replication 16	115	13.70	4.00	0.20	0.20 ± 1.00	True
17	Replication 17	115	14.06	3.97	-0.16	-0.16 ± 0.99	True
18	Replication 18	114	13.75	3.21	0.15	0.15 ± 0.89	True
19	Replication 19	114	13.69	4.14	0.21	0.21 ± 1.02	True
20	Replication 20	115	13.73	3.65	0.17	0.17 ± 0.95	True

Table 22 displays the 95% C.I. (computed by following Equation 28) for the number of vehicles leaving each destination. It is obvious that all the confidence intervals contain the true value. Thus, the model simulates the numbers of vehicles leaving each destination correctly.

Table 22: 95% C.I. for Number of Vehicles Leaving Each Destination

Destination	True Value	Simulated Mean	95% C.I.
2	111	103	103 ± 10
4	339	347	347 ± 36
6	1796	1802	1802 ± 64
8	192	191	191 ± 21
10	203	201	201 ± 10
12	271	264	264 ± 42
14	1740	1743	1743 ± 57

All of the parameters of interest have passed their validity tests. In conclusion, the DOE_EVAC's basic model logic is statistically valid. Based on the validation of the sub-unit logic used to build network logic, the DOE_EVAC simulation models should accurately simulate larger networks.

CHAPTER 5

DOE_EVAC FOR DESIGNS OF EXPERIMENTS

Selecting which traffic parameters to control in evacuation planning and analysis is still a research area. Some traffic engineers and managers focus on how to control the green time and red time of traffic signals to allow the most vehicles to traverse through the intersections, but some concentrate on how to control evacuees' departure time in each zone to minimize traffic congestion. However, to scientifically investigate which input traffic parameters are important and how they effect the output measures of performance, designs of experiments must be applied to come up with the most reasonable conclusions. The DOE_EVAC model is conveniently designed for planners to apply designs of experiments analysis by making important traffic parameters readily accessible and easy to change. This chapter demonstrates how users can use those traffic parameters to perform designs of experiments and to draw statistically significant conclusions on the model output. However, this chapter is not meant to be exhaustive and is only included to show an example of how a design of experiments can be performed using DOE_EVAC.

5.1 Designs of Experiments in Simulation

Basically, "the input parameters and structural assumptions composing a model are called *factors*, and the output performance measures are called *responses*" (Law, 2007). The factors are changed decisively so that we may observe and identify the reasons for changes observed in the responses. The simulation is run at various values, or *levels*, of the factor.

Table 23 shows the design matrix for three tested factors. Each factor has two levels of interest (indicated as "–" and "+"). This type of design is a 2^k factorial design.

Table 23: Design Matrix for the 2^k Factorial Design

Factor Combination (Design Point)	**Factor**			**Response**
	1	**2**	**3**	
1	–	–	–	R_1
2	–	–	+	R_2
3	–	+	–	R_3
4	–	+	+	R_4
5	+	–	–	R_5
6	+	–	+	R_6
7	+	+	–	R_7
8	+	+	+	R_8

Two types of effect are obtained based on the responses: the main effect (measure the average change in the response due to change from "–" level to "+" level of the factor) and the interaction effect (measure the average change in the response due to change from "–" level to "+" level of a combination of factors). The main effect is computed as follow:

$$e_i = \frac{\sum R^{+i} - \sum R^{-i}}{2^{k-1}} \qquad (30)$$

where *k* is the total number of factors, *i* is the factor index ($i = 1, 2. \ldots k$), R^{+i} are the values of responses associating with "+" level of factor *i*, and R^{-i} are the values of responses associated with "–" level of factor *i*.

In order to calculate the interaction effect, one must first obtain the sign ("–" or "+") of crossed-responses of the factors of interest. Table 24 shows the sign of crossed-responses attained for evaluating two-factor interaction effect.

The interaction effect is then evaluated for two factors, three factors, and then all the way up to k factors:

$$e_{ij\ldots} = \frac{\sum R^{+(i \times j \times \ldots)} - \sum R^{-(i \times j \times \ldots)}}{2^{k-1}} \tag{31}$$

Table 24: Crossed-Response Sign of Factors for the Interaction Effect Computation

Factor Combination (Design Point)	Factor		Sign of Crossed-Response	Response
	1	2		
1	–	–	+	R_1
2	–	–	+	R_2
3	–	+	–	R_3
4	–	+	–	R_4
5	+	–	–	R_5
6	+	–	–	R_6
7	+	+	+	R_7
8	+	+	+	R_8

where k – total number of factors, *i, j…* – the factor index (*i, j…* = 1, 2. …*k and i* $\neq$ *j*) , $R^{+(i \times j \times \ldots)}$ – values of crossed-responses associating with "+" level of combination of factors *i, j…*, $R^{-(i \times j \times \ldots)}$ – values of crossed-responses associating with "–" level of combination of factors *i, j…*.

To find out whether the effects are statistically significant, the method of independent replications is applied for the expected response at each of the factor levels. Applying this method, multiple independent replications are generated for each factor combination. The multiple values of responses are then utilized to obtain multiple independent values of each effect (see Table 25).

A 100(1-α)% two-sided confidence interval must be generated for each effect with α significance level (same as Equation 26):

$$\overline{X}(n) \pm t_{n-1,1-\alpha/2}\sqrt{\frac{S^2(n)}{n}} \tag{32}$$

where n is the number of simulation replications, $t_{n-1,1-\alpha/2}$ is the critical t value at (n-1) degrees of freedom, and $\overline{X}(n)$ and $S(n)$ are the mean and the standard deviation of the investigated effect, respectively.

Table 25: Multiple Replication Responses and Effects

Factor Combination (Design Point)	**Factor**			**Responses**		
	1	**2**	**3**	**1**		**n**
1	–	–	–	R_1^1	…	R_1^n
2	–	–	+	R_2^1	…	R_2^n
3	–	+	–	R_3^1	…	R_3^n
4	–	+	+	R_4^1	…	R_4^n
5	+	–	–	R_5^1	…	R_5^n
6	+	–	+	R_6^1	…	R_6^n
7	+	+	–	R_7^1	…	R_7^n
8	+	+	+	R_8^1	…	R_8^n
	Effect of Factor 1			e_1^1	⋯	e_1^n
	Effect of Factor 2			e_2^1	⋯	e_2^n
	Effect of Factor 3			e_3^1	⋯	e_3^n
	Effect of Factor 1 & 2			e_{12}^1	⋯	e_{12}^n
	Effect of Factor 1 & 3			e_{13}^1	⋯	e_{13}^n
	Effect of Factor 2 & 3			e_{23}^1	⋯	e_{23}^n
	Effect of Factor 1, 2, & 3			e_{123}^1	⋯	e_{123}^n

If the confidence interval of a particular effect does not contain zero, the effect is statistically significant. In other words, a change in the factor(s) causing this effect absolutely generates a change in the output. For example, if the confidence interval of the main effect of factor 1 does not contain zero, this effect is statistically significant; and thus, a change in levels of factor 1 will significantly change the performance measures of the output. Otherwise, there is no statistical evidence that the effect is actually present (Law, 2007).

5.2 A DOE_EVAC Application – An Example of Design of Experiments

This section demonstrates how the DOE_EVAC model supports the performance of designs of experiments. Again, the parameters available for designs of experiments are in Table 9. The GIS network example in Figure 7 is utilized again to investigate the traffic measures. The map is redisplayed here for convenience.

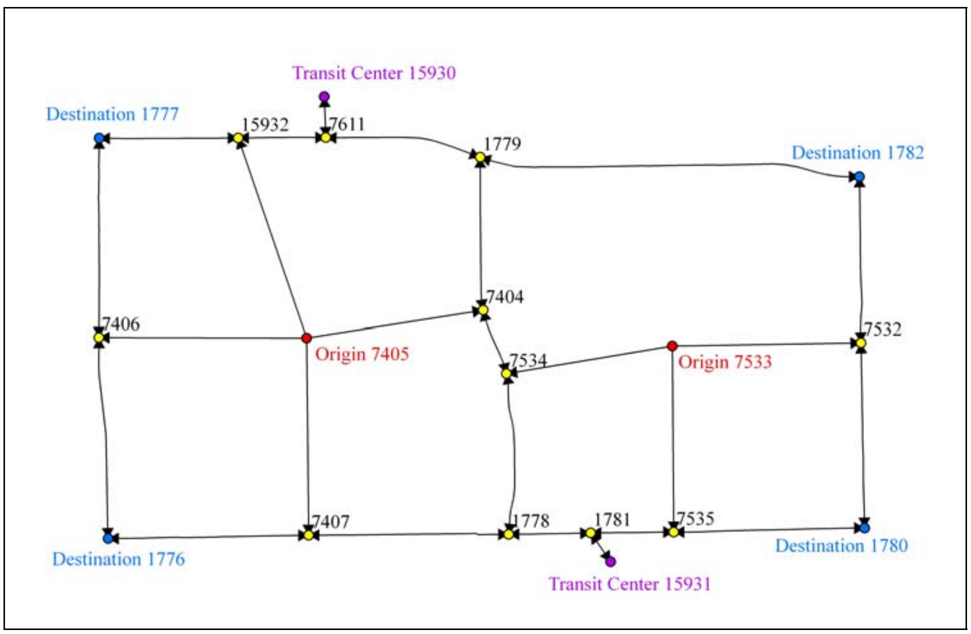


Figure 42: Example GIS Network

There are 2 origins, 2 transit centers, 4 destinations, 2 unsignalized intersections (node 7404 and 7534), and 7 signalized intersections. The total number of vehicles participating in the evacuation is 8,500 and the total number of people without vehicles is 331. A pilot run is invoked to obtain the pilot *Total Evacuation Time* of the network to roughly estimate the warm up time for the designs of experiments. The *Total Evacuation Time* of the pilot run is 1 hr 22 min (82 min).

The following factors and their levels are chosen for the designs of experiments:

1. Interarrival times at the origins: exponential distribution (–) versus interarrival time mean (+), which is a constant.Table 26 displays the interarrival time at the origins used for the designs of experiments demonstration.

Table 26: Designs of Experiments Interarrival Time at Origins

Origin		**Levels**	
		Exponential (–)	**Mean (+)**
7533	*People*	EXPO(0.025)	0.025
	Vehicles	EXPO(0.03)	0.03
7406	*People*	EXPO(0.04)	0.04
	Vehicles	EXPO(0.01)	0.01

2. Warm up period: empty and idle (–) versus double of the pilot run time (+), which is 0 min versus 164 min, respectively.
3. Green time management at intersection: 0.5 min (–) versus 1 min (+)

Note that the loading rate at each origin is a stochastic parameter. In designs of experiments, the tendency of using deterministic values as "+" and "-" values is extremely high since deterministic values are controllable by the analyst. However, there are situations that the values of factors follow probability distributions or time-dependent mathematical expressions. The developed model is advantageous in that it allows users to investigate both deterministic and stochastic factors. None of today's evacuation models have this flexibility.

The design matrix is in Table 27. 20 replications were run for each of 8 factor combinations (called scenarios). The output values of *Total Evacuation Time* were used as the response to compare among alternative scenarios.

Table 27: Designs of Experiments Design Matrix

Scenario	**Factor**		
	Interarrival Time (1)	**Warm Up Time (2)**	**Green Time (3)**
1	Exponential (–)	0 (–)	0.5 (–)
2	Exponential (–)	0 (–)	1 (+)
3	Exponential (–)	164 (+)	0.5 (–)
4	Exponential (–)	164 (+)	1 (+)
5	Mean (+)	0 (–)	0.5 (–)
6	Mean (+)	0 (–)	1 (+)
7	Mean (+)	164 (+)	0.5 (–)
8	Mean (+)	164 (+)	1 (+)

Table 28 reveals the 95% C.I. of the effects. As expected, all of the confidence intervals contain 0. It means the main effects and interaction effects among investigated factors are statistically significant. All three factors – the interarrival time, the simulation warm up time, and the intersection green time – affect the *Total Evacuation Time*; and they do interact with each other. It is suggested that the emergency planners must choose the traffic management strategies carefully in order to have a fast and safe evacuation.

Table 28: 95% C.I. of Effects

Effect	**Mean**	**95% C.I.**
Effect of Factor 1	-0.09	(-0.15, -0.02)
Effect of Factor 2	0.15	(0.15, 0.15)
Effect of Factor 3	0.43	(0.41, 0.45)
Effect of Factor 1 and 2	-0.15	(-0.15, -0.15)
Effect of Factor 1 and 3	0.13	(0.11, 0.14)
Effect of Factor 2 and 3	-0.15	(-0.15, -0.15)
Effect of Factor 1, 2, and 3	0.15	(0.15, 0.15)

CHAPTER 6

CONCLUSIONS AND FUTURE RESEARCH

6.1 Summary

In this research, a stochastic simulation model, DOE_EVAC, has been developed. The new model provides the ability to (i) effectively simulate alternative modes of transportation during evacuations, (ii) support designs of experiments, thus, provide users (e.g., emergency planners and traffic engineers) with means to investigate "what-if" scenarios with sound statistical analysis capabilities, and (iii) allow the users to build and execute these models without having to know complex simulation or coding language.

The DOE_EVAC model was developed using VB.Net and consists of the model's GUI, data accessing and processing, and Arena models and Arena outputs. DOE_EVAC model logic follows a mesoscopic simulation logic. DOE_EVAC has characteristics of microscopic models in terms of keeping track of individual vehicles, except DOE_EVAC does not include the lane-changing behavior and the acceleration or deceleration of vehicles. Other than that, it behaves similarly to that of a microscopic model while it still can simulate a large-scale transportation network as macroscopic models (i.e., it has lower computation and computer memories requirements).

The DOE_EVAC model is also a discrete-event simulation model in which each event occurs at a real-valued time point. The model skips inactive period and speeds up the simulation time, thus reduces waiting time to obtain results. This advantage enables users to anticipate emergency situations ahead of time and adjust the evacuation strategies effectively.

DOE_EVAC's credibility and validity have been confirmed via the behavior comparison validation test. DOE_EVAC is now ready to create quality stochastic simulation models to support designs of experiments on transportation evacuation planning.

6.2 Contributions

The DOE_EVAC model bridges the gaps of current simulation transportation evacuation modeling approaches by the ability to:

- Treat alternative vehicle modes differently based on their characteristics (lengths).
- Allow evacuees to select their own destinations, while allowing users to implement pre-defined evacuation routes.
- Warm up the system in order to provide realistic initial conditions of the traffic network (avoid empty-and idle initial conditions).
- Implement and analyze various traffic management strategies, for example intersection control (adjust green, red, and yellow time at intersections).
- Reroute traffic if critical infrastructure is damaged.
- Allow users to interrupt the simulation for the purpose of changing:
  - Entity attributes such as vehicle capacity.
  - Traffic management strategies.

DOE_EVAC is also capable of supporting designs of experiments and confidence interval generation by the ability to perform multiple simulation runs and obtain important traffic performance measures so that alternative plans/strategies can be analyzed to identify the "best" evacuation plan. No existing evacuation model provides users with this capability.

In addition, DOE_EVAC is user-friendly in developing transportation evacuation models. That is users do not need to know any specialized computer language or data structure in order to set up input data and run DOE_EVAC. Data is manipulated in table formats and there is no need to reformat data to run the model as in other evacuation models. Furthermore, the model has lots of flexibility in its input modeling so that the use of probability distributions and mathematical expressions can be incorporated. Finally, users have the ability to alter parameters anytime without having to reload the data files.

6.3 Future Research

Interests in transportation evacuation simulation have increased broadly over the last few decades. Even though DOE_EVAC has contributed significantly to this research and application area via its advanced features, there remain potential fields for future research:

1. Allowing decision makers to route traffic based on decision trees (Winston, 2003) generated via their own judgments. By applying this method, one will estimate the vehicle's utility value of improved travel time when changing from one route to another based on the vehicle's predefined utility function. The utility function is generated via the users' objectives. For example, the users can set the utility of improving more than 60 minutes of travel time as 1, the utility of improving 0 minute travel time as 0, and so on. The users then specify the route choice's behavioral rules, e.g. if the vehicle's utility value of

improved travel time is greater than 0.3, the vehicle will switch to the new route. Even though this approach is quite subjective, it will be supportive for the emergency planner to assess the traffic conditions and thus, to provide evacuees with traffic commands to safely speed up the evacuation process.

2. Investigating different incidents such as vehicles running out of gas (vehicular blocking).
3. Implementing pedestrian traffic signal at intersections.
4. Providing visual effects via traffic animation.
5. Creating an algorithm to effectively update the vehicle shortest paths during evacuations. Currently, the only time DOE_EVAC recalculates the vehicle shortest paths is when a link is not available due to an infrastructure failure. Even though this approach is acceptable for congested traffic during evacuations when the traffic flow times do not fluctuate greatly, a new algorithm is needed to find shortest paths under varying traffic flow times. Note that generating shortest paths based on current flow time is a NP-hard problem. Thus, updating the vehicle shortest paths too frequently is ineffective and unnecessary.

BIBLIOGRAPHY

Abkowitz, M. & Meyer, E. (1996). Technological advancements in hazardous materials evacuation planning. *Transportation Research Record, 1552*, 116-121.

Ames, D. P., Aburizaiza, A., Grover, D., Kadlec, J., Oberndorfer, N., Dunford, T., & Veluppillai, T. (2010). MapWindow GIS Open Source Software (Version 4.8) [Software]. Available from http://www.mapwindow.org/

Ben-Akiva, M. E. & Lerman, S. R. (1985). *Discrete choice analysis: Theory and application to travel demand*. Cambridge MA: MIT Press.

Britton, C. & Doake, J. (2005). *A student guide to object-oriented development*. Amsterdam: Elsevier Butterworth-Heinemann.

Bureau of Public Roads. (1964). *Traffic assignment manual for application with a large, high speed computer*. Washington, DC: Author.

Burghout, W. (2004). Hybrid microscopic-mesoscopic traffic simulation (Doctoral Dissertation, Royal Institute of Technology, Stockholm, Sweden, 2004).

Burghout, W., Koutsopoulos, H. N., & Andreasson, I. (2006). A discrete-event mesoscopic traffic simulation model for hybrid traffic assignment. *Proceedings of the 2006 IEEE Intelligent Transportation Systems Conference*, 1102-1107.

Caliper Corporation. (2005). *TransCAD: Transportation GIS software*. Newton, MA: Author.

Chusanapiputt, S. & Phoomvuthisarn, S. (2000). An embedded piecewise method in equivalent networks for fast decoupled load flow. *Proceedings of the 2000 Power Conference*, 769-773.

Del Castillo, J.M. & Benitez, F.G. (1995). On the functional form of the speed density relationship I: General theory. *Transportation Research B, 29*(5), 373-389.

Dijkstra, E. W. (1959). A note on two problems in connection with graphs. *Numeriche Mathematik*, 1, 269–271.

Drake, J.S., Schofer, J.L. & May, A.D. (1967). A statistical analysis of speed-density hypotheses. *Highway Research Record, 156*, 53-87.

Gazis, D.C., Herman, R. & Potts, R. (1959). Car-following-theory of steady-state traffic flow. *Operations Research, 7*, 499-505.

Greenberg, H., 1959. An analysis of traffic flow. *Operations Research*, *7*, 79-85.

Greenshields, B.D. (1935). A study in highway capacity. *Highway Research Board Proceedings*, *14*, 448-477.

Han, A. F. (1990). TEVACS: Decision support system for evacuation planning in Taiwan. *Journal of Transportation Engineering, 116*(6), 821-830.

Hiramatsu, T. (1983). Development of a network flow simulator (NESSY-IV) for analyses of mass evacuation in case of emergency. *Journal of Information Processing, 6*(1), 1-9.

Hobeika, A. G. & Jamei, B. (1985). MASSVAC: A model for calculating evacuation times under natural disasters. *Proceedings of the Conference on Computer Simulation in Emergency Planning, 5*(1), 23-28.

Hobeika, A. G. & Kim, C. (1998). Comparison of traffic assignments in evacuation modeling. *IEEE Transactions on Engineering Management, 45*(2), 192-198.

Hobeika, A. G., Kim, S., & Beckwith, R. E. (1994). A decision support system for developing evacuation plans around nuclear power stations. *Interfaces, 24*(5), 22-35.

Homburger, W. S, Hall, J. W., Loutzenheiser R. C., & Reilly, W. R. (1996). *Fundamentals of traffic engineering* (14^{th} ed.). Berkeley, CA: Institute of Transportation Studies, University of California, Berkeley.

Houston Galveston Area Council. (2009). *GIS data clearing house*. Available from http://www.h-gac.com/rds/gis/clearinghouse/default.aspx

Houston Transtar (2010). *Hurricane evacuation route map*. Retrieved from http://traffic.houstontranstar.org/weather/hurricane_evac.html.

Jaske, R. T. (1985). FEMA's computerized aids for accident assessment. *Proceedings of an International Symposium on Emergency Planning and Preparedness for Nuclear Facilities*, 181-205.

Jeannotte, K., Chandra, A., Alexiadis, V., & Skabardonis, A. (2004). *Traffic analysis toolbox volume II: Decision support methodology for selecting traffic analysis tools* (Report No. FHWA-HRT-04-039). Washington, DC: Federal Highway Administration.

Kelton, W. D., Sadowski, R. P, & Sturrock D. T. (2007). *Simulation with Arena* (4^{th} ed.). New York: McGraw-Hill.

Kron, G. (1963). *The piecewise solution of large-scale system*. London: MacDonald.

Kwon, E. & Pitt, S. (2005). Evaluation of emergency evacuation strategies for downtown event traffic using a dynamic network model. *Transportation Research Record, 1922,* 149-155.

Lahmar, M., Assavapokee, T., & Ardekani, S. A. (2006). A dynamic transportation planning support system for hurricane evacuation. *Proceedings of the IEEE Intelligent Transportation Systems Conference 2006*, 612-617.

Law, A. M. (2007). *Simulation modeling and analysis* (4^{th} ed.). New York, NY: McGnaw-Hill.

May, A. D. (1990). *Traffic flow fundamentals*. Englewood Cliffs, NJ: Prentice-Hall, Inc.

McLean, M. A., Moeller, M., Desrosiers, A., & Urbanik, T. (1983). CLEAR: A model for calculation of evacuation time estimates in emergency planning zones. *Proceedings of the Conference on Computer Simulation in Emergency Planning, 11*(2), 58-63.

Microsoft Corporation (2010). QuickGraph (Version 2010.6.8.16821) [Software]. Available from http://quickgraph.codeplex.com/

Montgomery, D. C. (2001). Design and Analysis of Experiments. New York, NY: John Wiley & Sons, Inc.

Montgomery, D.C. & Runger, G.C. (2007). *Applied statistics and probability for engineers* (4^{th} ed.). New York, NY: John Wiley & Sons, Inc.

Orcutt, F. L. (1993). *The traffic signal book*. Englewood Cliffs, NJ: Prentice Hall.

Pegden, C. D., Shannon, R. E., & Sadowski, R. P. (1995). *Introduction to simulation using SIMAN* (2^{nd} ed). New York, NY: McGraw-Hill.

Pidd, M., Silva, F. N., & Eglese, R. W. (1996). A simulation model for emergency evacuation. *European Journal of Operational Research*, *90*, 413-419.

Pham, H., Pittman, J., & Court, M. (2008). A review of simulation modeling methodologies for large-scale evacuations. *Summer Computer Simulation Conference 2008.* Edinburgh, Scotland. 16-19 June, 2008.

Rathi, A. K. (1994). *A microcomputer based traffic evacuation modeling system for emergency planning applications* (Contract No. DE-AC05-84OR21400). Washington, DC: U.S. Department of Energy.

Rathi, A. K. & Solanki, R. S. (1993). *Simulation of traffic flow during emergency evacuations: A microcomputer based modeling system* (Contract No. DE-AC05-84OR21400). Washington, DC: U.S. Department of Energy.

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

Sheffi, Y., Mahmassani, H, & Powell, W. B. (1982). A transportation network evacuation model. *Transportation Research Part A, 16*(3), 209-218.

Tufekci, S. & Kisko, T. M. (1991). Regional evacuation modeling system (REMS): A decision support system for emergency area evacuations. *Computers and Industrial Engineering, 21(1-4)*, 89-93.

Underwood, R.T. (1961). *Speed, volume and density relationships, quality and theory of traffic flow*. Newhaven, CT: Yale Bureau of Highway Traffic.

United States Census Bureau. (2009). *2009 TIGER/Line shapefiles* [Data file]. Available from http://www.census.gov/geo/www/tiger/

United States Department of Transportation (2010). Census Transportation Planning Product Packages [Data file]. Available from http://www.fhwa.dot.gov/ctpp/2000dataprod.htm

University of Oklahoma Campus Map (2010). [Graph illustration the University Campus Corner June 29, 2010]. University of Oklahoma Campus Corner. Retrieved from http://www.ou.edu/map/

Urbanik, T. II, Moeller, M. P., & Barnes, K. (1988). *Benchmark study of the I-DYNEV evacuation time estimate computer code* (NUREG/CR-4873 PNL-6171). Washington, DC: U.S. Nuclear Regulatory Commission.

Winston, W. (2003). *Operations research: Applications and algorithms* (4rd ed.). Belmont, CA: Duxbury Press.

Wolshon, B., Urbina, E., Wilmot, C., & Levitan, M. (2005). Review of policies and practices for hurricane evacuation I: Transportation planning, preparedness, and response. *Natural Hazards Review, 6*(3), 129-142.

Wu, S., Shuman, L., Bidanda, B., Kelly, M., Sochats, K., & Balahan, C. (2007). Embedding GIS in disaster simulation. *Environmental Systems Research Institute (ESRI) International User Conference Proceedings*. San Diego, CA: ESRI.

APPENDIX A

GLOSSARY

Term	Description
A-B flow/direction/lanes	Forward topological flow/direction/lanes between two nodes A and B of bidirectional road segment
B-A flow/direction/lanes	Backward topological flow/direction/lanes between two nodes A and B of bidirectional road segment
Census block	Smallest unit of geography used by US Census Bureau to tabulate population
Census tract	A unit of geography that coincides with city or town limits. Census tracts can be subdivided into census block groups or census blocks.
Congestion	Occurs when travel demand is much greater than the capacity of the link or node, i.e. the traffic is at a complete standstill.
Contraflow	Reverse the direction of some inbound lanes to direct evacuees out of risk area. Contraflow increases the outbound capacity and improves the traffic flow.
Density	Number of travel units (vehicles or passengers) that traverse a given facility in a unit distance
Designs of Experiments	Methods to conduct experiments to investigate and draw valid conclusions on how a system or a process works
Destination (D)	The place to which evacuees travel. It is usually a geographical point located outside of risk area.
Discrete-event simulation	Simulation type in which state variables of the system change at discrete or countable points in time.
Flow	Number of travel units that traverse a given facility in a unit time
Free flow speed	The speed when there are no constraints placed on a vehicle by other vehicles on the link
Geographic information system (GIS)	Present real-world objects (such as roads or land use) with digital data. GIS provides users with graphical presentation, spatial analysis, and geographic database management.
Link	Unidirectional road segment connecting nodes. Each link has a starting node and an ending node.
Node	The jointing of traffic streams such as intersections, on/off ramps, origins, and destinations.

Term	Description
O-D matrix	A table containing the travel demands (number of vehicles) from each Origin to each Destination of the network.
Origin (O)	Departure location of evacuees. It is usually the center of the geographical unit.
Signal phase	A group of intervals (green, red, and/or yellow) that is assigned to an independent traffic movement or combination of movements through a signalized intersection
Traffic Analysis Zone (TAZ)	A unit of geography specifically used in transportation planning. The size of a TAZ is usually less than 3000 people, but it can vary.

APPENDIX B

TRANSPORTATION PLANNING APPLIED IN EVACUATION

A classical urban transportation planning model includes four steps: trip generation, trip distribution, modal split, and traffic assignment. The results produced by this model are deterministic. Thus, application of urban transportation planning to simulation evacuation is extremely limited. The following sections represent the latest advanced urban transportation planning steps.

B.1 Trip Generation

The goal of trip generation is to estimate the number of trips deploying from origins (via trip production procedure) and the number of trips arriving to destinations (via trip attraction procedure) in each subarea of the transportation network, i.e. the traffic analysis zone (TAZ). Trip production procedures address the descriptions of the population distribution and socioeconomic characteristics, vehicle utilization, and land use across the origin nodes; while trip attraction procedures address the descriptions of the location and intensity of land use across the destination nodes. Some factors influencing trip generation include: personal characteristics (gender, age, income, and occupation), household characteristics (household size, vehicle availability, number of children and senior in the household, and total income), zonal characteristics (land use, residential density, and accessibility) and transportation network characteristics (level of service) (Caliper Corporation, 2005). Additional factors must be considered under evacuation conditions. For example, some people refuse to evacuate (to protect their properties) or some evacuate even though threats are not directly exposed to them.

Sources and approaches to obtain these data are enormous, but the most accurate and reliable data source is the (online) database of the U.S. Census Bureau (http://www.census.gov/). Data are now attainable via table and geographic information system (GIS) formats, which support graphical presentation, spatial analysis, and geographic database management. In addition, to providing a convenient analysis access for transportation specialists, the U.S. DOT maintains the Census Transportation Planning Package (CTPP) which associates census data with elaborate transportation analysis levels (from national to local level) at aggregate scale (http://www.fhwa.dot.gov/ctpp/). Other large data sources include the Public Use Microdata Sample (http://www.census.gov/acs/www/Products/PUMS/) in disaggregate scale and the National Personal/Household Transportation Survey (http://www.fhwa.dot.gov/policy/ohpi/nhts/index.htm). In case no data at local level are available, previous models of same or similar geographical area, or information provided by the National Cooperative Highway Research Program (NCHRP) Report 365, Travel Estimation Techniques for Urban Planning, can be applied (Barton-Aschman Associates & Cambridge Systematics, 1997).

The challenge is how to extract the obtained data into usable trip productions and trip attractions. Conventional urban transportation planning specifies three primary methodologies to estimate trip productions from provided data: cross-classification, logistics regression and discrete choice (Caliper Corporation, 2005).

- **Cross-classification methods**: Cross-classification methods separate populations into categories based on mixture of socioeconomic characteristics such as household size, number of available vehicles, income, occupation, and so forth. Each category associates with one trip rate, which is estimated based on history/survey data. The previously achieved average values of the classification parameters of each TAZ in the study area then are empirically compared to those in cross-classification categories to obtain trip production for corresponding TAZ.

- **Logistics regression methods**: History data are used to formulate a linear statistical relationship between population characteristics (independent variables) and number of possible trips (dependent variable). With y – dependent variable, x_i – independent variables, and a, b_i – variable coefficients, the regression model can be expressed as $y = a + \sum_i b_i x_i$.

  Population characteristics x_i of each TAZ then are plugged in the regression model to generate number of trips y.

- **Discrete choice methods**: Since users alternatively choose whether to make the trip or not, the binary logit of discrete choice methods can be employed to estimate average trip production. Binary logit methods assume that each alternative associates with a utility and the chosen is based on the importance of consequent utility. The probability that an individual user n decides to travel, i.e. choice 1, can be expressed as:

$$P_n(1) = \frac{1}{1 + e^{-\beta(x_{1n} - x_{0n})}}$$

  where β is the vector of coefficients estimated by the model, x_{1n} is the vector of explanatory variables in person n's utility of making the trip and x_{0n} is the vector of explanatory variables in person n's utility of not making the trip.

Note that $\beta(x_{1n} - x_{0n})$ is the relative utility function of making the trip; the higher the utility, the higher the probability that user will decide to travel. The disaggregate probabilities of individual users then are aggregated to derive the proportion of the population that chooses to travel.

In order to apply conventional urban trip generation methodologies into evacuation conditions, past empirical evidence must be retrieved via surveys or emergency management observations/judgments under similar estimation and planning purposes. If no history data exists, one must be able to reasonably estimate trip rates by making the most of present on-hand data. Southworth (1991) proposes a rational approach by first calculating the average population assigned to an origin node at the starting time of evacuation based on populations at home, school, work, and special facilities such as hospitals, correctional facilities, large retail centers, and recreational centers. The vehicle utilization and driver availability then are collaborated with populations to find the number of vehicles or trips that will be loaded onto the network.

In general, emergency planners can apply the same methodologies to estimate trip attractions. Since evacuees' destinations, such as their relative's houses, hotels, and so forth, are totally different from their daily trips of which collection data are available, it is difficult for the planners to predict evacuees' final destinations and number of trips coming towards each TAZ. However, by providing evacuation routes and establishing situ-shelters, planners can somehow portion populations, guide evacuees to designated destinations and reasonably estimate the final trip attraction for each TAZ.

B.2 Trip Distribution

The objective of trip distribution is to predict the spatial pattern of trips between origins and destinations. The most popular method to forecast the number of trips T_{ij} traveling from origin *i* to destination *j* is a gravity model (Sheffi, 1985):

- If the sum of trips produced by each origin is constrained to be equal forecasted total production for each origin:

$$T_{ij} = P_i \frac{A_j f(d_{ij})}{\sum_{all\ zone\ z} A_z f(d_{iz})}$$

- If the sum of trips attracted to each destination is constrained to be equal forecasted total attraction for each destination:

$$T_{ij} = A_j \frac{P_j f(d_{ij})}{\sum_{all\ zone\ z} P_z f(d_{zj})}$$

where P_i stands for the number of trips located at *i*, A_j are number of trips attracted to destination *j*, and $f(d_{ij})$ is the function of travel cost between origin *i* and destination *j*. The function of travel cost, also called friction factor function, $f(d_{ij})$ are best known under a negative exponential form $e^{-a.c_{ij}}$ of which *a* is a calibrated travel cost-decay parameter. It can be represented under matrix form, which contains travel cost for each *i, j* pair as well.

Again, evacuation planners can significantly impact evacuees' destination choices via well-publicized evacuation routes, shelters, and provided network guidance systems. The idea is that more and more personal route guidance systems such as GPS will tend to be used during the evacuation. Utilizing these systems for communicating to evacuees possible evacuation routes will not only influence evacuees' decisions, but also will generate a sequence of predictable traffic distributions as a result of that action.

B.3 Modal Split

Same as trip production, modal split or mode choice analysis estimates the number of trips emitting from origin nodes, except that modal split focuses on the separation of trips into different transportation modes. Typically, transportation modes are characterized by vehicle type, capacity, size, and speed. Despite driver's behaviors, different transportation modes produce dissimilar traffic pattern including the occupation of spaces, possible accelerating speeds, fuel consumption, and so forth. Consequently, people's choice of transportation modes will ultimately influence traffic conditions and total evacuation costs.

Data sources for mode choice are quite the same as those for trip production. Since mode choice of an evacuee is based on his/her ability to access a means of transportation, data such as vehicle occupancy, household sizes or other socio-economic characteristics are quite valuable.

During evacuation, there are two possible mode choices for evacuees. Obviously, people who own cars will tend to prefer to drive their own cars than to use public transportation for a good reason protecting their property. However, older cars are not always reliable; some people may choose to abandon their cars and take public transportation instead. People who have no or little access to personal transportation will have to use public/mass transportation. Hence, planners must estimate the vehicle availability across affected zones and ensure the arrangement of public transportation to support evacuees.

Methodologies such as cross-classification, logistics regression or discrete choice models can also be applied to extract data into utilizable mode trips. The most common methods for modal split are two choice models: multinomial logit (MNL) and nested logit (NL). The MNL model calculates the number of mode trips by finding the probability $P_n(i)$ that a person *n* will choose alternative mode *i* (Ben-Akiva & Lerman, 1985):

$$P_n(i) = prob(Y_n = i) = \frac{e^{V_{in}}}{\sum_{j \in C_n} e^{V_{jn}}}$$

where Y_n stands for the value of the response variable for individual *n*, C_n is the set of alternatives in person *n*'s choice set and V_{in} is the measurable component of the utility of alternative *i* for individual *n*. Since the utility is not always known with certainty, the utility is treated as random variable. An independently and identically distributed Gumbel error term ε_{in} is added to V_{in} to obtain the random utility U_{in}:

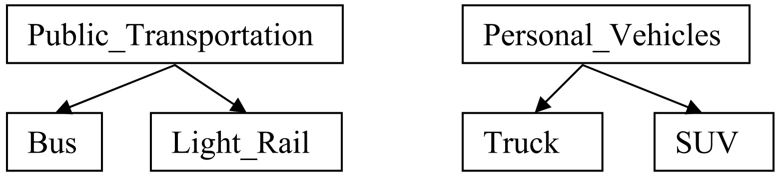
$$U_{in} = V_{in} + \varepsilon_{in}$$

The probability $P_n(i)$ is reformed as follow:

$$P_n(i) = prob(Y_n = i) = \frac{e^{\mu V_{in}}}{\sum_{j \in C_n} e^{\mu V_{jn}}}$$

where μ is a positive scale parameter of the Gumbel distribution of ε_{in}.

The NL model investigates the choice among alternatives that are categorized into different groups. The below figure illustrates an example of NL model structure. NL structure can contain multiple levels. Each group at the lowest level can be considered as a ML model.



The probabilities of the bottom elements are computed as the product of the conditional and marginal probabilities. For example,

$$P(Bus) = P(Bus \mid Public\_Transportation) * P(Public\_Transportation)$$

Note that the accessibility of evacuees to public transportation can require extra modes such as walk, bike, and drive. The planners have to take this fact into consideration during modeling process since usage of these modes can impact traffic conditions, for example walking evacuees with their belongings can pour into street and block flow of traffic.

B.4 Traffic Assignment

Established as a key element in traffic management, traffic assignment problem is defined as the allocation of the flows to transportation routes based on factors that impact route choice. Basically, the mission is to find the link flows given the transportation network, the link performance functions and the O-D trip rate matrix. The resulting flows are then used to evaluate the transportation network via a set of computed performance measures. The ultimate goal of traffic assignment is to stabilize the transportation system at an equilibrium point of travelers' moving decisions and levels of congestion.

Traffic assignment concept can be enclosed in some classifications:

- Stochastic/Deterministic: Stochastic approach involves using random processes for the travelers' route choice behavior while the deterministic approach does not.
- Static/Dynamic: Static approach deals with steady-state O-D matrix, which is the result of fixed link flows and the independence of link operation to connected links. In contrast, time-dependent O-D matrix is utilized in dynamic approach, which is more realistic and certainly more complex.
- Path-based/Link-based: Path-based algorithm allows travelers to select routes based on attributes of entire path. This process requires enumeration of all used paths and excessive computer cost. On the contrary, travelers choose the next link to travel on at each decision point in link-based models. However, link choice can be cyclic and travelers' myopic behavior can direct to impractical choices (Toledo, Koutsopoulos, Ben-Akiva, & Jha, 2005).
- Flow-based/Vehicle-based: Flow-based models update the network flows in fixed time interval. Vehicle-based models advance flows on a continuous time-line basis (Koohbanani, 2004).

The most advanced and popular traffic assignment methods are user equilibrium and system optimum. These methods will be briefly reviewed in following section. Descriptions of other traffic assignment methods such as all-or-nothing, STOCH or Dial's assignment, incremental assignment, and capacity restraint can be found in Sheffi (1985).

The User Equilibrium (UE) assumes that all network users have perfect traffic information and their behaviors are identical. Furthermore, it assumes that users choose routes via selfish tactic to minimize their own travel costs and they always make correct route choice decisions. The equilibrium criterion, proposed by Wardrop (1952), is satisfied when no individual users can unilaterally improve their travel time by using alternative paths. In other words, all utilized paths for users departing from the same origin to the same destination eventually generate the same minimum costs. UE problem can be solved by employing Frank-Wolfe's convex combination algorithm (Frank and Wolfe, 1956) or its improved version PARTAN (LeBlanc, Helgason, & Boyce, 1985).

The Stochastic User Equilibrium (SUE) is a generalization of the UE definition. It has the same philosophy as that of UE with respect to the optimized unique travel cost of each traveler. However, more enhanced and realistic than deterministic UE (which only exploits higher utilization routes and completely ignores lower utilization routes), SUE makes the most of both lower and higher utilization routes. In addition, SUE assumes that users have imperfect information about network paths and they perceive network attributes diversely. Equilibrium is reached when no individual user believes that he/she can improve travel time using alternative paths (Sheffi, 1985). SUE can be solved by applying the Method of Successive Averages (MSA) proposed by Daganzo and Sheffi (1977). This method guarantees a convergent solution.

While UE and SUE take into consideration the benefit of individual users by minimizing their travel costs, these methods do not necessarily optimize the total travel cost in the system and they are "nonmonotonicity with respect to the network's capacity" (Jahn, Möhring, Shulz, & Stier-Moses, 2005). Hence, although UE and SUE are the most preferred traffic assignment methods, one can alternatively opt to combine UE/SUE with System Optimum (SO). SO autonomously serves best to traffic managers since its target is to minimize total travel cost in the system. It is well-known that a few users will experience excessive travel costs under SO traffic patterns in order to obtain global optimum. In other words, if those users change routes to reduce their unilateral travel costs, the total travel cost definitely increases and the system goes far away from global optimal state. The question here is how much sacrifice does a user agree to make?

Jahn et al. (2005) propose a route guidance called Constrained System Optimum (CSO) that "adopt a system-optimum approach, but honor the individual needs by imposing additional constraints to ensure that drivers are assigned to "acceptable" paths only". CSO is in fact a compromise between UE and SO objectives. The idea is to enforce constraints on paths to bind the maximum travel cost for all users in the network. By restricting the unfairness - the ratio of the traverse time of the recommended path to that of the shortest path - to be smaller than a tolerance factor, CSO guarantees a close optimum to that of SO and also provides fairness (in terms of individual cost and limits) among network users as well.

The notion of exploring the reconciliation between UE and SO has also been employed by Zhenlong and Xiaohua (2008). By introducing the concept of satisfactory degree, the authors apply game theory to assess different balance levels between UE and SO objectives. The only concern about this modeling as well as Jahn et al.'s modeling (2005) is that only static traffic flows are utilized to analyze the network performance. Even though Sheffi et al. (1982) indicates that the traffic flows may reach steady state during heavy congestion conditions such as evacuation, not considering dynamics flow in the model diminishes its realistic representation of the real-world evacuation process.

Static traffic assignment models deal with constant link flows, link traverse time and fixed O-D matrix over the planning duration (Koohbanani, 2004). However, since static traffic assignment ignores the fact that link travel time is flow dependent and link flow is time dependent, it is inadequate for modeling real-time applications. Applying dynamic traffic assignment can prevail over this issue; and dynamic traffic assignment has been one of the main topics of current traffic assignment research since a precise formulation has not been established yet. Few examples of research efforts include studying time dependent traffic assignment and formulate dynamic UE and SO under congestion by Peeta and Mahmassani (1995), or developing a simulation-based dynamic traffic assignment model by applying MSA for time-dependent path flows and space-time queuing approach by Mahut (Florian, 2005), and so forth.

B.5 References

Barton-Aschman Associates & Cambridge Systematics. (1997). *Model validation and reasonableness checking manual*. Washington, DC: Federal Highway Administration.

Ben-Akiva, M. E. & Lerman, S. R. (1985). *Discrete choice analysis: Theory and application to travel demand*. Cambridge MA: MIT Press.

Caliper Corporation. (2005). *Travel demand modeling with TRANSCAD 4.8*. Newton, MA: Author.

Daganzo, C. & Sheffi, Y. (1977). On stochastic models of traffic assignment. *Transportation Science, 11*, 253-274

Florian, M. (2005). Application of a simulation-based dynamic traffic assignment model. In R. Kitamura & M. Kuwahara (Eds.) *Simulation approaches in transportation analysis: Recent advances and challenges*. New York: Springer.

Frank, M. & Wolfe. P. (1956). An algorithm for quadratic programming. *Naval Research Logistics Quarterl*, *3*, 95-110.

Jahn, O., Möhring, R. H., Schulz, A. S., & Stier-Moses N. E. (2005). System-optimal routing of traffic flows with user constraints in networks with congestion. *Operations Research, 53*(4), 600-616.

Koohbanani, M. J. (2004). Enhancements to Transportation Analysis and Simulation System (TRANSIMS). (Doctoral Dissertation, Virginia Polytechnic Institute and State University, 2004). Retrieved from http://scholar.lib.vt.edu/theses/available/etd-12142004-161543/.

LeBlanc, L. J., Helgason, R. V., & Boyce, D. E. (1985). Improved efficiency of the Frank-Wolfe algorithm for convex network programs. *Transportation Science*, *19*, 445-462.

Peeta, S. & Mahmassani, H. S. (1995). System optimal and user equilibrium time-dependent traffic assignment in congested networks. *Annals of Operations Research, 60*, 81-113.

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

Southworth, F. (1991). *Regional evacuation modeling: A state-of-the-art review* (ORNL/TM-11740). Oak Ridge, TN: Oak Ridge National Lab.

Toledo, T., Koutsopoulos H., Ben-Akiva, M., & Jha, M. (2005). Microscopic traffic simulation: Models and Application. In R. Kitamura & M. Kuwahara (Eds.) *Simulation approaches in transportation analysis: Recent advances and challenges*. New York: Springer.

Wardrop, J. G. (1952). Some theoretical aspects of road traffic research. *Proceedings of Institute of Civil Engineers II, 1*, 325-378.

Zhenlong, L. & Xiaohua, Z. (2008). Integrated-equilibrium routing of traffic flows with congestion. *Proceedings of the 17th International Federation of Automatic Control World Congress*, 16065-16070.

APPENDIX C

ARENA CONFIGURATIONS

C.1 Operands

Basic Process

Module	Operand Name	Prompt Text	Names of Module in Model
Assign	Name	Name	Assign Attributes
Assignments	Type	Type	Assign Available Capacity and Current Station
	VName	Variable Name	Assign Green Light
	Row	Row	Assign People
	Column	Column	Assign People Arrival Time
	AName	Attribute Name	Assign People Travel Time
	TypeName	Entity Type	Assign Public Transportation
	PicName	Entity Picture	Assign Public Transportation Batch Size
	OtherName	Other	Assign Public Transportation Sequence
	Value	New Value	Assign Traffic Signal
			Assign Vehicle
			Assign Vehicle Sequence
			Calculate Available Capacity
			Change Signal Light to Yellow
Batch	Name	Name	Group People
	Type	Type	
	Batch Size	Batch Size	
	Save Criterion	Save Criterion	
	Rule	Rule	
	Representative Entity Type	Representative Entity Type	
Create	Name	Name	Create People
	Entity Type	Entity Type	Create Traffic Signal
	Interarrival Type	Interarrival Type	Create Vehicle
	Expression	Expression	
	Units	Units	
	Batch Size	Entities per Arrival	
	Max Batches	Max Arrivals	
	First Create	First Create	
Decide	Name	Name	From Origin _ Is Entity Vehicle?
	Type	Type	From Where to Here?
	Percent True	Percent True	Traffic Phases?

Module	Operand Name	Prompt Text	Names of Module in Model
	If	If	
	VNamed	Named	
	Row	Row	
	Column	Column	
	ANamed	Named	
	TypeNamed	Named	
	Is	Is	
	Value	Value	
Conditions	N Percent True	Percent True	
	N If	If	
	N VNamed	Named	
	N Row	Row	
	N Column	Column	
	N ANamed	Named	
	N TypeNamed	Named	
	N Is	Is	
	N Value	Value	
Dispose	Name	Name	Dispose Duplicate Entity
	EntStats	Record Entity Statistics	Dispose Traffic Lights
			Exit via Destination
Record	Name	Name	Destination Statistics
	Type	Type	Number of People Out
	Attribute	Attribute Name	Number of Vehicles Out
	Value	Value	Record End Time
	Counter Name	Counter Name	Record Flow from Station to Station
	Tally Name	Tally Name	
Separate	Name	Name	Duplicate to Create Scan Entity
	Type	Type	People Get off Public Transportation
	Cost	Percent Cost to Duplicates	
	Number	# of Duplicates	
Set	Name	Name	People Types
	Type	Type	People Pictures
Member	Picture Name	Picture Name	Vehicle Types
	Entity Name	Entity Type	Vehicle Pictures
Variable	Name	Name	Available Capacity from Station to Station
	Rows	Rows	Gap
	Columns	Columns	Green Time
Initial Values	Initial Value	Initial Value	Max Wait Time
			Public Transportation Batch Size
			Public Transportation Size
			Traffic Signal

Module	Operand Name	Prompt Text	Names of Module in Model
			Vehicle Route Time Warm Up Time Yellow Time

Advanced Process

Module	Operand Name	Prompt Text	Names of Module in Model
Advanced Set	Name Type	Name Set Type	Destination Set Public Transportation Sequences Vehicle Sequences
Members	Queue Name Other Storage Name	Queue Name Other Storage Name	
Delay	Name ValueAdded DelayType Units	Name Allocation Delay Time Units	Green Light Progressing Vehicle Running from Station to Station
Expression	Name Dim1 Dim2 Data Type IO Point Usage Description	Name Rows Columns Data Type I/O Point Usage Description	Hour To Minutes Mile To Feet People Shortest Route Time People Speeds Speed from Station to Station Vehicle Lengths
Expression Values	Value	Expression Value	
Hold	Name Type Value Limit Condition QSG QSGInfinite QName QSet QMem QAttr QExp	Name Type Wait for Value Limit Condition Queue Type Queue Type Queue Name Set Name Set Index Attribute Expression	Hold People for Signal Scan for Condition Vehicles Waiting to Move Vehicles Waiting to Traverse
Signal	Name	Name	Signal to Release Public Transportation

Module	Operand Name	Prompt Text	Names of Module in Model
	Value	Signal Value	
	Limit	Limit	
Statistics	Name	Name	Simulation End Time
	Type	Type	
	Tally	Tally Name	
	Tally Output File	Tally Output File	
	Counter	Counter Name	
	CLimit	Limit	
	StartTime	Start Time	
	StartTimeUnits	Units	
	Duration	Duration Time	
	DurationUnits	Units	
	RepeatStatistics	Repeat Statistics	
	CInit	Initialization Option	
	CounterOutputFile	Counter Output File	
	DExp	Expression	
	DLabel	Report Label	
	DOutputFile	Output File	
	ValueState	Frequency Type	
	FValue	Expression	
	FRes	Resource Name	
	FLabel	Report Label	
	FOutputFile	Output File	
Categories	ValueRange	Constant or Range	
	Value1	Value	
	Value2	High Value	
	Category	Category Name	
	ExcInc	Category Option	

Advanced Transfer

Module	Operand Name	Prompt Text	Names of Module in Model
Route	Name	Name	Route People from Origin
	RouteTime	Route Time	Route Public Transportation from Transit Center
	Units	Units	Route Vehicle from Origin
	SG	Destination Type	Route Vehicle from Intersection

Module	Operand Name	Prompt Text	Names of Module in Model
	Station	Station Name	
Sequence	Name	Name	From Station to Station
Steps	Station	Station Name	
Station	Name Statn	Name Station Name	Station

C.2 Run Controller Commands

General Commands

ASSIGN	CLEAR	END
EVENT	GO	STEP
QUIT	SIGNAL	
SHOW		

Cancel Commands

CANCEL BREAK	CANCEL INTERCEPT
CANCEL TRACE BLOCKS	CANCEL TRACE CONDITIONS
CANCEL TRACE ENTITIES	CANCEL TRACE EXPRESSIONS
CANCEL TRACE FILE	CANCEL TRACE TIMES
CANCEL WATCH	

Set Commands

SET BREAK	SET INTERCEPT
SET MODEL	SET TRACE
SET TRACE BLOCKS	SET TRACE CONDITIONS
SET TRACE ENTITIES	SET TRACE EXPRESSIONS
SET TRACE FILE	SET TRACE TIMES
SET WATCH	

View Commands

VIEW	VIEW BREAK
VIEW CALENDAR	VIEW CONVEYORS
VIEW ENTITY	VIEW INTERCEPT
VIEW MODEL	VIEW QUEUE
VIEW SOURCE	VIEW TRACE
VIEW WATCH	

APPENDIX D

PROGRAMMING CODE

Main.vb

```
'The main application of the model: control GUI

Imports System

Public Class Main

    Private Sub mnuExit_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles mnuExit.Click
        Me.Close()
    End Sub

    Private Sub mnuImportData_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles mnuImportData.Click
        Dim open As New ImportDataForm
        open.MdiParent = Me
        open.Show()
    End Sub

    Private Sub mnuCreateModel_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles mnuCreateModel.Click
        Dim model As ArenaModel
        model = New ArenaModel
    End Sub

    Private Sub mnuDOE_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnuPAN.Click
        Dim p As New Process()
        p.StartInfo.FileName =
Environment.GetFolderPath(Environment.SpecialFolder.ProgramFiles) &
"\Rockwell Software\Arena 7.0\pan.exe"
        p.Start()
    End Sub

    Private Sub mnuFlow_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnuFlow.Click

    End Sub

    Private Sub openFile_FileOk(ByVal sender As System.Object, ByVal e As
System.ComponentModel.CancelEventArgs) Handles openFile.FileOk
        Dim flowFilePath As String
        With openFile
            .Reset()
            .Title = "Open File"
            .Filter = "Dbase Files (*.dbf)|*.dbf|Excel Files
(*.xls)|*.xls|Access Files (*.mdb)|*.mdb"
            .Multiselect = False
            If (.ShowDialog() = Windows.Forms.DialogResult.OK) Then
                flowFilePath = .FileName
            End If
        End With
    End Sub
End Class
```

GlobalData.vb

```
'Global Data
Imports QuickGraph

Module GlobalData
    Public Directory, IncidentsFileName, LinksFileName,
LinksShapeFileName As String
    Public NodesFileName, NodesShapeFileName, PeopleFileName,
VehiclesFileName As String
    Public Gap, MaxWaitTime, PublicTransportationLength As Double
    Public PeopleCount, VehicleCount, PublicTransportationCapacity As
Integer
    Public DestinationList, IntersectionList, OriginList, TransitList As
List(Of Node)
    Public IncidentList As List(Of Incident)
    Public IncidentStartTimeList As List(Of Double)
    Public LinkList As List(Of Link)
    Public PeopleList As List(Of People)
    Public VehicleList As List(Of Vehicle)
    Public Graph As IVertexAndEdgeListGraph(Of Integer, Edge(Of Integer))
    Public EdgeFlow, EdgeLength As Dictionary(Of String, Double)
End Module
```

Link.vb

```
'This class is to create and get values of a link

Public Class Link
    Dim _Length As Double
    Dim _FromNode, _ToNode, _Lanes As Integer
    Dim _Speed As String

    'Create an instance of a link
    Public Sub New(ByVal length As Double, ByVal fromNode As Integer,
ByVal toNode As Integer, ByVal lanes As Integer, ByVal speed As String)
        _Length = length
        _FromNode = fromNode
        _ToNode = toNode
        _Lanes = lanes
        _Speed = speed
    End Sub

    Public Function getLength() As Double
        Return _Length
    End Function

    'Return start node of link
    Public Function getStartNode() As Integer
        Return _FromNode
    End Function

    'Return number of AB lanes
    Public Function getLanes() As Integer
        Return _Lanes
    End Function

    'Return speed limit
    Public Function getSpeed() As String
        Return _Speed
```

```
    End Function

    'Return end node of link
    Public Function getEndNode() As Integer
        Return _ToNode
    End Function
End Class
```

Node.vb

```
' This class is to create and get values of a node

Public Class Node
    Dim _ID As Integer
    Dim _Green As Double
    Dim _Yellow As Double
    Dim _PeoTime As String
    Dim _VehTime As String
    Dim _Peo As Integer
    Dim _Veh As Integer
    Dim _DestDist As String
    Dim _IntersectionFrom As List(Of Link)
    Dim _IntersectionTo As List(Of Link)
    Dim _OriginTo As List(Of Link)
    Dim _TransitTo As List(Of Link)

    'Create an instance of an Origin node that has available destination
distribution
    Public Sub New(ByVal ID As Integer, ByVal peopleTime As String,
ByVal vehicleTime As String, ByVal people As Integer, ByVal vehicle As
Integer, ByVal destdist As String, ByVal toList As List(Of Link))
        _ID = ID
        _PeoTime = peopleTime
        _VehTime = vehicleTime
        _Peo = people
        _Veh = vehicle
        _OriginTo = toList
        _DestDist = destdist
    End Sub

    'Create an instance of an Origin node that does not have available
destination distribution
    Public Sub New(ByVal ID As Integer, ByVal peopleTime As String,
ByVal vehicleTime As String, ByVal people As Integer, ByVal vehicle As
Integer, ByVal toList As List(Of Link))
        _ID = ID
        _PeoTime = peopleTime
        _VehTime = vehicleTime
        _Peo = people
        _Veh = vehicle
        _OriginTo = toList
    End Sub

    'Create an instance of Destination node
    Public Sub New(ByVal ID As Integer)
        _ID = ID
    End Sub

    'Create an instance of Transit Center node
    Public Sub New(ByVal ID As Integer, ByVal toList As List(Of Link))
        _ID = ID
        _TransitTo = toList
```

```
    End Sub

    'Create an instance of an Intersection node
    Public Sub New(ByVal ID As Integer, ByVal greenTime As Double, ByVal
yellowTime As Double, ByVal fromList As List(Of Link), ByVal toList As
List(Of Link))
        _ID = ID
        _Green = greenTime
        _Yellow = yellowTime
        _IntersectionFrom = fromList
        _IntersectionTo = toList
    End Sub

    'Return node ID
    Public Function getID() As Integer
        Return _ID
    End Function

    'Return green time of Intersection node
    Public Function getGreen() As Double
        Return _Green
    End Function

    'Return yellow time of Intersection node
    Public Function getYellow() As Double
        Return _Yellow
    End Function

    'Return traffic loading rate at Origin node
    Public Function getPeopleTime() As String
        Return _PeoTime
    End Function

    'Return traffic loading rate at Origin node
    Public Function getVehicleTime() As String
        Return _VehTime
    End Function

    'Return number of People at the Origin node
    Public Function getPeople() As Integer
        Return _Peo
    End Function

    'Return number of vehicles at the Origin node
    Public Function getVehicles() As Integer
        Return _Veh
    End Function

    'Return destination distribution
    Public Function getDestinationDistribution() As String
        Return _DestDist
    End Function

    'Return list of next intersections connected to this origin
    Public Function getOriginTo() As List(Of Link)
        Return _OriginTo
    End Function

    'Add value to the next intersections connected to this origin
    Public Sub addOriginTo(ByVal toIntersection As Link)
        _OriginTo.Add(toIntersection)
    End Sub

```

```
    'Add value to the next intersections connected to this transit center
    Public Sub addTransitTo(ByVal toIntersection As Link)
        _TransitTo.Add(toIntersection)
    End Sub

    'Return list of next intersections connected to this transit center
    Public Function getTransitTo() As List(Of Link)
        Return _TransitTo
    End Function

    'Add value to the previous intersection list of this intersection
    Public Sub addFromIntersection(ByVal fromIntersection As Link)
        _IntersectionFrom.Add(fromIntersection)
    End Sub

    'Return list of previous intersections connected to this intersection
    Public Function getFromIntersections() As List(Of Link)
        Return _IntersectionFrom
    End Function

    'Add value to the next intersection list of this intersection
    Public Sub addToIntersection(ByVal toIntersection As Link)
        _IntersectionTo.Add(toIntersection)
    End Sub

    'Return list of next intersections connected to this intersection
    Public Function getToIntersections() As List(Of Link)
        Return _IntersectionTo
    End Function

End Class
```

Vehicle.vb

```
'This class is to create and get values of a vehicle

Public Class Vehicle
    Dim _Type As String
    Dim _Length As Double

    'Create an instance of a vehicle
    Public Sub New(ByVal type As String, ByVal length As Double)
        _Type = type
        _Length = length
    End Sub

    'Return vehicle type
    Public Function getVehicleType() As String
        Return _Type
    End Function

    'Return vehicle length
    Public Function getLength() As Double
        Return _Length
    End Function

End Class
```

People.vb

```
Public Class People
    Dim _Type As String
    Dim _Speed As Double

    'Create an instance of a vehicle
    Public Sub New(ByVal type As String, ByVal speed As Double)
        _Type = type
        _Speed = speed
    End Sub

    'Return vehicle type
    Public Function getPeopleType() As String
        Return _Type
    End Function

    'Return vehicle length
    Public Function getSpeed() As Double
        Return _Speed
    End Function
End Class
```

Incident.vb

```
Public Class Incident
    Dim _ID As Integer
    Dim _FromNode As Integer
    Dim _ToNode As Integer
    Dim _StartTime As Double
    Dim _CapPercent As Integer

    'Create an instance of an Incident
    Public Sub New(ByVal ID As Integer, ByVal fromNode As Integer, ByVal
toNode As Integer, ByVal startTime As Double, ByVal capacityPercentage
As Double)
        _ID = ID
        _FromNode = fromNode
        _ToNode = toNode
        _StartTime = startTime
        _CapPercent = capacityPercentage
    End Sub

    'Return incident ID
    Public Function getID() As Integer
        Return _ID
    End Function

    'Return incident start node
    Public Function getFromNode() As Integer
        Return _FromNode
    End Function

    'Return incident end node
    Public Function getToNode() As Integer
        Return _ToNode
    End Function

    'Return incident start time
    Public Function getStartTime() As Double
        Return _StartTime
```

```
    End Function

    'Return incident capacity percentage
    Public Function getCapacityPercentage() As Double
        Return _CapPercent
    End Function
End Class
```

ImportDataForm.vb

```
'User interface to import data

Imports QuickGraph

Public Class ImportDataForm
    Private Sub btnBrowseNodesFile_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles btnBrowseNodesFile.Click
        With openFile
            .Reset()
            .Title = "Open File"
            .Filter = "Dbase Files (*.dbf)|*.dbf|Excel Files
(*.xls)|*.xls|Access Files (*.mdb)|*.mdb"
            .Multiselect = False
            If (.ShowDialog() = Windows.Forms.DialogResult.OK) Then
                txtNodesFile.Text = .FileName
            End If
        End With
    End Sub

    Private Sub btnBrowseLinksFile_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles btnBrowseLinksFile.Click
        With openFile
            .Reset()
            .Title = "Open File"
            .Filter = "Dbase Files (*.dbf)|*.dbf|Excel Files
(*.xls)|*.xls|Access Files (*.mdb)|*.mdb"
            .Multiselect = False
            If (.ShowDialog() = Windows.Forms.DialogResult.OK) Then
                txtLinksFile.Text = .FileName
            End If
        End With
    End Sub

    Private Sub btnBrowseVehiclesFile_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
btnBrowseVehiclesFile.Click
        With openFile
            .Reset()
            .Title = "Open File"
            .Filter = "Dbase Files (*.dbf)|*.dbf|Excel Files
(*.xls)|*.xls|Access Files (*.mdb)|*.mdb"
            .Multiselect = False
            If (.ShowDialog() = Windows.Forms.DialogResult.OK) Then
                txtVehiclesFile.Text = .FileName
            End If
        End With
    End Sub

    Private Sub btnBrowsePeople_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles btnBrowsePeople.Click
        With openFile
```

```
            .Reset()
            .Title = "Open File"
            .Filter = "Dbase Files (*.dbf)|*.dbf|Excel Files (*.xls)|*.xls|Access Files (*.mdb)|*.mdb"
            .Multiselect = False
            If (.ShowDialog() = Windows.Forms.DialogResult.OK) Then
                txtPeopleFile.Text = .FileName
            End If
        End With
    End Sub

    Private Sub btnBrowseIncidentsFile_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnBrowseIncidentsFile.Click
        With openFile
            .Reset()
            .Title = "Open File"
            .Filter = "Dbase Files (*.dbf)|*.dbf|Excel Files (*.xls)|*.xls|Access Files (*.mdb)|*.mdb"
            .Multiselect = False
            If (.ShowDialog() = Windows.Forms.DialogResult.OK) Then
                txtIncidentsFile.Text = .FileName
            End If
        End With
    End Sub

    'Load data
    Private Sub btnLoad_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnLoad.Click
        Dim result As DialogResult
        Dim model As ArenaModel
        Try
            'Import Nodes, Links, Vehicles, People and Incidents files
            If String.IsNullOrEmpty(txtIncidentsFile.Text) Then
                ImportData.ReadFiles(txtNodesFile.Text, txtLinksFile.Text, txtVehiclesFile.Text, txtPeopleFile.Text)
                IncidentStartTimeList = New List(Of Double)
            Else
                ImportData.ReadFiles(txtNodesFile.Text, txtLinksFile.Text, txtVehiclesFile.Text, txtPeopleFile.Text)
                ImportData.ReadFiles(txtIncidentsFile.Text)
            End If

            ' Display GIS Map
            If System.IO.File.Exists(NodesShapeFileName) AndAlso System.IO.File.Exists(LinksShapeFileName) Then
                Dim gisMapForm As New GISMap()
                gisMapForm.MdiParent = Me.MdiParent
                gisMapForm.Dock() = DockStyle.Fill
                gisMapForm.Show()
            End If

            PublicTransportationCapacity = Integer.Parse(txtPublicTransportCapacity.Text)
            PublicTransportationLength = Double.Parse(txtPublicTransportationLength.Text)
            MaxWaitTime = Double.Parse(txtMaxWaitTime.Text)
            Gap = Double.Parse(txtGap.Text)
            Directory = System.IO.Path.GetDirectoryName(txtNodesFile.Text)

            result = MessageBox.Show(Me, "Would you like to create an Arena model?", "Create Arena Model", MessageBoxButtons.YesNo)
```

```
                If result = DialogResult.Yes Then
                    model = New ArenaModel
                    Main.mnuExportData.Enabled = True
                    Me.Close()
                Else
                    Main.mnuCreateModel.Enabled = True
                    Me.Close()
                End If
            Catch ex As Exception
                MessageBox.Show(Me, ex.Message)
                Console.WriteLine(ex.StackTrace)
            End Try
        End Sub

End Class
```

ImportData.vb

```
'This class is to import data into the model.

Imports System.Data
Imports System.String
Imports System.Collections.Generic
Imports QuickGraph
Imports QuickGraph.Algorithms

Public Class ImportData
    Const FILE_NAME_LENGTH As Integer = 8

    Public Shared Function ReadFiles(ByVal nodesFilePath As String,
ByVal linksFilePath As String, ByVal vehiclesFilePath As String, ByVal
peopleFilePath As String) As Boolean
        Const DIR_TWO_WAY As Integer = 2
        Const ORIGIN_TYPE As Integer = 0
        Const DESTINATION_TYPE As Integer = 1
        Const TRANSIT_TYPE As Integer = 2
        Dim nodesTable, linksTable, vehiclesTable, peopleTable, sql, key
As String
        Dim con As New Odbc.OdbcConnection
        Dim g As AdjacencyGraph(Of Integer, Edge(Of Integer))
        Dim edge As Edge(Of Integer)
        Dim length, abflow, baflow As Double
        Dim startNode, endNode As Integer
        Dim fromStation, toStation As Link
        Dim cmd As Odbc.OdbcCommand
        Dim reader As Odbc.OdbcDataReader

        'Create Graph
        Graph = New AdjacencyGraph(Of Integer, Edge(Of Integer))
        g = Graph
        EdgeLength = New Dictionary(Of String, Double)
        EdgeFlow = New Dictionary(Of String, Double)

        '---------------------------------------------NODES--------------
-----------------------------------
        'Establish connection to Nodes file
        connectToFile(nodesFilePath, NodesFileName, "nodes", con)

        'Nodes shapefile
        NodesShapeFileName =
System.IO.Path.ChangeExtension(NodesFileName, ".shp")
```

```

        'Import Nodes file and create Nodes table
        nodesTable =
System.IO.Path.GetFileNameWithoutExtension(NodesFileName)
        sql = "SELECT id, type, green, yellow, peotime, vehtime, people,
vehicle, destdist FROM " & nodesTable & ""

        'Add nodes into Graph
        cmd = con.CreateCommand()
        cmd.CommandType = CommandType.Text
        cmd.CommandText = sql
        reader = cmd.ExecuteReader()
        PeopleCount = 0
        VehicleCount = 0
        OriginList = New List(Of Node)
        DestinationList = New List(Of Node)
        TransitList = New List(Of Node)
        IntersectionList = New List(Of Node)
        While reader.Read()
            g.AddVertex(reader.GetInt32(0))
            If reader.GetInt32(1) = ORIGIN_TYPE Then
                OriginList.Add(New Node(reader.GetInt32(0),
reader.GetString(4), reader.GetString(5), reader.GetInt32(6),
reader.GetInt32(7), reader.Item(8).ToString(), New List(Of Link)))
                PeopleCount = PeopleCount + reader.GetInt32(6)
                VehicleCount = VehicleCount + reader.GetInt32(7)
            ElseIf reader.GetInt32(1) = DESTINATION_TYPE Then
                DestinationList.Add(New Node(reader.GetInt32(0)))
            ElseIf reader.GetInt32(1) = TRANSIT_TYPE Then
                TransitList.Add(New Node(reader.GetInt32(0), New List(Of
Link)))
            Else
                IntersectionList.Add(New Node(reader.GetInt32(0),
reader.GetDouble(2), reader.GetDouble(3), New List(Of Link), New List(Of
Link)))
            End If
        End While

        reader.Close()
        con.Close()

        '--------------------------------------------LINKS--------------
-----------------------------------
        'Establish connection to Links file
        connectToFile(linksFilePath, LinksFileName, "links", con)

        'Links shapefile
        LinksShapeFileName =
System.IO.Path.ChangeExtension(LinksFileName, ".shp")

        ' Import Links file and create Links table
        linksTable =
System.IO.Path.GetFileNameWithoutExtension(LinksFileName)
        sql = "SELECT Length, Dir, FromID, ToID, ABLanes, BALanes,
ABFlowTime, BAFlowTime, ABSpeed, BASpeed FROM " & linksTable & ""

        'Add links into Graph
        cmd = con.CreateCommand()
        cmd.CommandType = CommandType.Text
        cmd.CommandText = sql
        reader = cmd.ExecuteReader()
        LinkList = New List(Of Link)
        While reader.Read()
```

```
            length = reader.GetDouble(0) ' Initialize cost
            startNode = reader.GetInt32(2)
            endNode = reader.GetInt32(3)
            abflow = reader.GetDouble(6)
            baflow = reader.GetDouble(7)
            fromStation = New Link(length, startNode, endNode,
Integer.Parse(reader.GetDouble(4)), reader.GetString(8))
            toStation = New Link(length, endNode, startNode,
Integer.Parse(reader.GetDouble(5)), reader.GetString(9))

            edge = New Edge(Of Integer)(startNode, endNode)
            key = edge.ToString()
            ' If links are duplicate, get the smaller cost
            If (EdgeLength.ContainsKey(key)) Then
                If (EdgeLength(key) > length) Then
                    EdgeLength(key) = length
                End If
            Else
                EdgeLength.Add(key, length)
            End If

            If (EdgeFlow.ContainsKey(key)) Then
                If (EdgeFlow(key) > abflow) Then
                    EdgeFlow(key) = abflow
                End If
            Else
                EdgeFlow.Add(key, abflow)
            End If

            ' Insert edge
            If g.ContainsEdge(startNode, endNode) = False Then
                g.AddEdge(edge)
                LinkList.Add(fromStation)
            End If

            If (Integer.Parse(reader.GetDouble(1)) = DIR_TWO_WAY)
AndAlso startNode <> endNode Then
                ' Insert reverse edge
                edge = New Edge(Of Integer)(endNode, startNode)
                key = edge.ToString()
                If (EdgeLength.ContainsKey(key)) Then
                    If (EdgeLength(key) > length) Then
                        EdgeLength(key) = length
                    End If
                Else
                    EdgeLength.Add(key, length)
                End If

                If (EdgeFlow.ContainsKey(key)) Then
                    If (EdgeFlow(key) > baflow) Then
                        EdgeFlow(key) = baflow
                    End If
                Else
                    EdgeFlow.Add(key, baflow)
                End If

                If g.ContainsEdge(endNode, startNode) = False Then
                    g.AddEdge(edge)
                    LinkList.Add(toStation)
                End If
            End If

            'Add previous and next stations to each intersection
```

```
            For Each intersection In IntersectionList
                If intersection.getID() = fromStation.getStartNode()
Then
                    If isTransitCenter(toStation.getStartNode()) = False
Then
                        intersection.addToIntersection(toStation)
                    End If
                    If (Integer.Parse(reader.GetDouble(1)) =
DIR_TWO_WAY) And isDestination(toStation.getStartNode()) = False Then
                        intersection.addFromIntersection(toStation)
                    End If
                ElseIf intersection.getID() = toStation.getStartNode()
Then
                    If isDestination(fromStation.getStartNode()) = False
Then
                        intersection.addFromIntersection(fromStation)
                    End If
                    If (Integer.Parse(reader.GetDouble(1)) =
DIR_TWO_WAY) And isTransitCenter(fromStation.getStartNode()) = False
Then
                        intersection.addToIntersection(fromStation)
                    End If
                End If
            Next

            'Add next stations to each origin
            For Each origin In OriginList
                If origin.getID() = fromStation.getStartNode() Then
                    origin.addOriginTo(toStation)
                End If
            Next

            'Add next stations to each transit center
            For Each center In TransitList
                If center.getID() = fromStation.getStartNode() Then
                    center.addTransitTo(toStation)
                End If
            Next
        End While

        reader.Close()
        con.Close()

        '-----------------------------------------VEHICLES-----------
--------------------------------------
        'Establish connection to Vehicles file
        connectToFile(vehiclesFilePath, VehiclesFileName, "vehicles",
con)

        'Import Vehicles file and create Vehicles table
        vehiclesTable =
System.IO.Path.GetFileNameWithoutExtension(VehiclesFileName)
        sql = "SELECT type, length FROM " & vehiclesTable & ""

        'Add vehicles into Vehicle list
        cmd = con.CreateCommand()
        cmd.CommandType = CommandType.Text
        cmd.CommandText = sql
        reader = cmd.ExecuteReader()
        VehicleList = New List(Of Vehicle)
        While reader.Read()
            VehicleList.Add(New Vehicle(reader.GetString(0),
reader.GetDouble(1)))
```

```
        End While
        reader.Close()
        con.Close()

        '-------------------------------------------PEOPLE-------------
-----------------------------------
        'Establish connection to People file
        connectToFile(peopleFilePath, PeopleFileName, "people", con)

        'Import Vehicles file and create Vehicles table
        peopleTable =
System.IO.Path.GetFileNameWithoutExtension(PeopleFileName)
        sql = "SELECT type, speed FROM " & peopleTable & ""

        'Add vehicles into Vehicle list
        cmd = con.CreateCommand()
        cmd.CommandType = CommandType.Text
        cmd.CommandText = sql
        reader = cmd.ExecuteReader()
        PeopleList = New List(Of People)
        While reader.Read()
            PeopleList.Add(New People(reader.GetString(0),
reader.GetString(1)))
        End While
        reader.Close()
        con.Close()
    End Function

    '-------------------------------------------INCIDENTS--------------
----------------------------------
    Public Shared Function ReadFiles(ByVal incidentsFilePath As String)
As Boolean
        Dim incidentsTable, sql As String
        Dim con As New Odbc.OdbcConnection
        Dim cmd As Odbc.OdbcCommand
        Dim reader As Odbc.OdbcDataReader

        connectToFile(incidentsFilePath, IncidentsFileName, "incident",
con)

        'Import Incidents file and create Incidents table
        incidentsTable =
System.IO.Path.GetFileNameWithoutExtension(IncidentsFileName)
        sql = "SELECT id, fromnode, tonode, starttime, cappercent FROM "
& incidentsTable & ""

        'Add incidents into Incident List
        cmd = con.CreateCommand()
        cmd.CommandType = CommandType.Text
        cmd.CommandText = sql
        reader = cmd.ExecuteReader()
        IncidentList = New List(Of Incident)
        IncidentStartTimeList = New List(Of Double)
        While reader.Read()
            IncidentList.Add(New Incident(reader.GetInt32(0),
reader.GetInt32(1), reader.GetInt32(2), reader.GetDouble(3),
reader.GetDouble(4)))
        End While
        reader.Close()
        con.Close()

        'Incident start time list
        If String.IsNullOrEmpty(incidentsFilePath) = False And
```

```
IncidentList IsNot Nothing Then
            'Sort Incident list based on start time
            IncidentList.Sort(Function(incident1 As Incident, incident2
As Incident)
incident1.getStartTime().CompareTo(incident2.getStartTime()))
            For Each incidentEvent In IncidentList
                If
IncidentStartTimeList.Contains(incidentEvent.getStartTime()) = False
Then

IncidentStartTimeList.Add(incidentEvent.getStartTime())
                End If
            Next
        End If

        'Incident file must contain at least one incident
        If IncidentStartTimeList.ElementAt(0) <= 0 Then
            MessageBox.Show("Incident start time must be positive")
            Exit Function
        End If
    End Function

    'Check if a node is destination
    Public Shared Function isDestination(ByVal id As Integer) As Boolean
        For Each destination In DestinationList
            If destination.getID() = id Then
                Return True
            End If
        Next
    End Function

    'Check if a node is transit center
    Public Shared Function isTransitCenter(ByVal id As Integer) As
Boolean
        For Each transit In TransitList
            If transit.getID() = id Then
                Return True
            End If
        Next
    End Function

    Public Shared Sub connectToFile(ByVal filepath As String, ByRef
fileName As String, ByVal newFileName As String, ByRef con As
Odbc.OdbcConnection)
        Dim path, extension As String

        'Establish connection to file
        path = System.IO.Path.GetDirectoryName(filepath)
        extension = System.IO.Path.GetExtension(filepath)

        'Connection path based on file type/extension
        If extension.ToLower() = ".dbf" Then
            con.ConnectionString = "Driver={Microsoft dBASE Driver
(*.dbf)};DriverID=277;Dbq=" & path & ";"
        Else
            con.ConnectionString =
"PROVIDER=Microsoft.Jet.OLEDB.4.0;Data Source =" & path & ""
        End If

        con.Open()

        'Check file name length to be recognized by Microsoft dBASE
driver
```

```
        'If length > 8, copy to the new file name
        If System.IO.Path.GetFileNameWithoutExtension(filepath).Length >
FILE_NAME_LENGTH Then
            fileName = path & "\" & newFileName &
System.IO.Path.GetExtension(filepath)
            If System.IO.File.Exists(fileName) = False Then
                System.IO.File.Copy(filepath, fileName)
            Else
                System.IO.File.Delete(fileName)
                System.IO.File.Copy(filepath, fileName)
            End If
        Else
            fileName = filepath
        End If
    End Sub
End Class
```

ArenaModel.vb

```
'This class create evacuation model in Arena

Imports System.Collections.Generic
Imports QuickGraph
Imports QuickGraph.Algorithms

Public Class ArenaModel
    Public Shared model As Arena.Model
    Dim incidentGraph As AdjacencyGraph(Of Integer, Edge(Of Integer)) =
Graph
    Dim closestTransitCenter As Integer
    Dim shortestDistance As Double
    Dim nextStation As String = "Next Station"
    Dim transitStation As String = "Transit Station"
    Dim getPathByFlow, getPathByLength As TryFunc(Of Integer,
IEnumerable(Of Edge(Of Integer)))

    Public Sub New()
        Const X_INCREMENT As Integer = 750
        Const Y_INCREMENT As Integer = 500
        Const X_AFTER_DECIDE As Double = 1.25
        Const GREEN_SIGNAL As Integer = 1
        Const YELLOW_SIGNAL As Integer = 0
        Const RED_SIGNAL As Integer = -1
        Const CONVERT_MILE_TO_FEET As Integer = 5280
        Const HOUR_TO_MINUTES As Integer = 60
        Const PERCENTAGE As Double = 100

        Dim app As New Arena.Application()

        Dim assign, assignPeople, assignVehicle, assignPeople2,
assignVehicle2, assignIn, assignOut As Arena.Module
        Dim batch, createPeople, createVehicle, createSignal As
Arena.Module
        Dim decide, decideFromIntersections, decidePeople, decideSignal,
decideVehicle, delay, dispose, expression, hold, hold2 As Arena.Module
        Dim record, routePeople, routeTime, routeVehicle, scan,
separate, signal, station As Arena.Module
        Dim setPeopleType, setPeoplePicture, setPeopleSpeed,
setPeopleTransit As Arena.Module
        Dim setVehicleType, setVehiclePicture, setVehicleLength,
setDestination, statistic As Arena.Module
        Dim variable, variablePublicTransportationSize,
```

```
variablePublicTransportationWaitTime, variableSignal, vehicleGap As
Arena.Module
        Dim x, xSignal, xIntersection, xAssign, xAssign2, xAssignIn,
xScan, y, ySignal, yIntersection, people, startTimeCount, vehicles As
Integer
        Dim assignIndex, batchSizeIndex, fromIntersectionCount,
intersectionIncidentStartTimeCount, toIntersectionCount As Integer
        Dim nodeID, startNode, endNode As Integer
        Dim start, PeopleTypeDistList(PeopleList.Count - 1),
maxVehicleSize As Double
        Dim vehicleModeDistList(VehicleList.Count - 1),
vehicleDestinationDistList(DestinationList.Count - 1) As Double
        Dim PeopleTypeDist, vehicleDestinationDist, vehicleTypeDist,
condition, stopConditionVehicle As String
        Dim rand As New Random()
        Dim fromLink, toLink As Link
        Dim currentIncident As Incident
        Dim currentIncidentEdge As Edge(Of Integer)
        Dim incidentStartNodes As List(Of Integer)
        Dim intersectionIncidentStartTimeList As List(Of Double)
        Dim intersectionTime As Double = 0.01 'minute
        Dim replications As Integer = 20
        Dim warmUp As Double = 720

        model = app.Models.Add()

        '======================================= MODEL GLOBAL PARAMETERS
=======================================
        startTimeCount = IncidentStartTimeList.Count

        If PeopleCount > 0 Then
            'Generate sets of People mode, People picture and People
speed
            setPeopleType = model.Modules.Create("BasicProcess", "Set",
0, 0)
            setPeopleType.Data("Name") = "People Types"
            setPeopleType.Data("Type") = "Entity Type"
            setPeoplePicture = model.Modules.Create("BasicProcess",
"Set", 0, 0)
            setPeoplePicture.Data("Name") = "People Pictures"
            setPeoplePicture.Data("Type") = "Entity Picture"
            setPeopleSpeed = model.Modules.Create("AdvancedProcess",
"Expression", 0, 0)
            setPeopleSpeed.Data("Name") = "People Speeds"
            setPeopleSpeed.Data("Dim1") = PeopleList.Count
            For i = 0 To PeopleList.Count - 1
                setPeopleType.Data("Entity Name(" & i + 1 & ")") =
PeopleList.Item(i).getPeopleType()
                setPeoplePicture.Data("Picture Name(" & i + 1 & ")") =
"Picture.Man"
                setPeopleSpeed.Data("Value(" & i + 1 & ")") =
PeopleList.Item(i).getSpeed()
            Next
            setPeopleType.UpdateShapes()
            setPeoplePicture.UpdateShapes()
            setPeopleSpeed.UpdateShapes()
        End If

        'Generate sets of vehicle mode, vehicle picture and vehicle
length
        stopConditionVehicle = ""
        maxVehicleSize = 0
        setVehicleType = model.Modules.Create("BasicProcess", "Set", 0,
```

```
0)
        setVehicleType.Data("Name") = "Vehicle Types"
        setVehicleType.Data("Type") = "Entity Type"
        setVehiclePicture = model.Modules.Create("BasicProcess", "Set",
0, 0)
        setVehiclePicture.Data("Name") = "Vehicle Pictures"
        setVehiclePicture.Data("Type") = "Entity Picture"
        setVehicleLength = model.Modules.Create("AdvancedProcess",
"Expression", 0, 0)
        setVehicleLength.Data("Name") = "Vehicle Lengths"
        setVehicleLength.Data("Dim1") = VehicleList.Count
        For i = 0 To VehicleList.Count - 1
            setVehicleType.Data("Entity Name(" & i + 1 & ")") =
VehicleList.Item(i).getVehicleType()
            setVehiclePicture.Data("Picture Name(" & i + 1 & ")") =
"Picture.Truck"
            setVehicleLength.Data("Value(" & i + 1 & ")") =
VehicleList.Item(i).getLength()
            stopConditionVehicle = stopConditionVehicle & "EntitiesOut("
& setVehicleType.Data("Entity Name(" & i + 1 & ")") & ")+"
            If maxVehicleSize < setVehicleLength.Data("Value(" & i + 1 &
")") Then
                maxVehicleSize = setVehicleLength.Data("Value(" & i + 1
& ")")
            End If
        Next
        stopConditionVehicle =
stopConditionVehicle.Remove(stopConditionVehicle.Length - 1)
        stopConditionVehicle = stopConditionVehicle & "==" &
VehicleCount
        setVehicleType.UpdateShapes()
        setVehiclePicture.UpdateShapes()
        setVehicleLength.UpdateShapes()

        'Generate destination Set
        setDestination = model.Modules.Create("AdvancedProcess",
"Advanced Set", 0, 0)
        setDestination.Data("Name") = "Destination Set"
        setDestination.Data("Type") = "Other"
        For i = 0 To DestinationList.Count - 1
            setDestination.Data("Other(" & i + 1 & ")") = "Station " &
DestinationList.Item(i).getID()
        Next
        setDestination.UpdateShapes()

        If TransitList.Count > 0 Then
            'Generate public transportation capacity Variable
            variablePublicTransportationSize = variableModule("Public
Transportation Size", PublicTransportationCapacity)

            'Generate maximum allowable time for public transportation
to wait for People
            variablePublicTransportationWaitTime = variableModule("Max
Wait Time", MaxWaitTime)
        End If

        'Generate gap between vehicles
        vehicleGap = variableModule("Gap", Gap)

        'Generate vehicle's moving time through an intersection
        routeTime = variableModule("Vehicle Route Time",
intersectionTime)

```

```
        'Capacity (length), speed, and flow counter of links
        For i = 0 To LinkList.Count - 1
            startNode = LinkList.ElementAt(i).getStartNode()
            endNode = LinkList.ElementAt(i).getEndNode()
            variable = variableModule("Available Capacity from " &
startNode & " to " & endNode, Math.Max(LinkList.ElementAt(i).getLength()
* LinkList.ElementAt(i).getLanes() * CONVERT_MILE_TO_FEET,
maxVehicleSize + Gap))
            expression = expressionModule("Speed from " & startNode & "
to " & endNode, LinkList.ElementAt(i).getSpeed())
        Next

        'Expression CONVERT_MILE_TO_FEET, HOUR_TO_MINUTES
        expression = expressionModule("Mile To Feet",
CONVERT_MILE_TO_FEET)
        expression = expressionModule("Hour To Minutes",
HOUR_TO_MINUTES)

        'Create output file of total evacuation time
        statistic = model.Modules.Create("AdvancedProcess", "Statistic",
0, 0)
        statistic.Data("Name") = "Total Evacuation Time"
        statistic.Data("Type") = "Output"
        statistic.Data("DExp") = "TMAX(End Time)"
        statistic.Data("DOutputFile") = "TotalEvacuationTime.dat"
        statistic.UpdateShapes()

        '============================================= SETUP MODEL RUN
====================================
        'Setup warm up period
        variable = variableModule("Warm Up Time", warmUp)
        model.WarmUpPeriod = "Warm Up Time"
        model.WarmUpPeriodTimeUnits = smTimeUnits.smMinutes

        'Setup base time unit
        model.BaseTimeUnits = smTimeUnits.smMinutes

        'Setup termination condition
        If PeopleCount <> 0 AndAlso VehicleCount <> 0 Then
            model.TerminatingCondition = "NC(Total People Out)==" &
PeopleCount & "&& NC(Total Vehicles Out)==" & VehicleCount
        ElseIf PeopleCount = 0 Then
            model.TerminatingCondition = "NC(Total Vehicles Out)==" &
VehicleCount
        Else
            model.TerminatingCondition = "NC(Total People Out)==" &
PeopleCount
        End If

        'Setup number of replications
        model.NumberOfReplications = replications

        '=============================================== ORIGIN
===========================================
        y = 0

        For Each origin In OriginList
            x = 0
            nodeID = origin.getID()
            people = origin.getPeople()
            vehicles = origin.getVehicles()

            If people <> 0 Then
```

```
                'People type empirical discrete distribution
                PeopleTypeDist =
cumulativeDiscreteDistribution(PeopleTypeDistList)
                'Create People
                createPeople = createModule("Create People " & nodeID,
"People", x, y, origin.getPeople(), origin.getPeopleTime())
            End If

            If vehicles <> 0 Then
                'Vehicle mode empirical discrete distribution
                vehicleTypeDist =
cumulativeDiscreteDistribution(vehicleModeDistList)

                'Sequence sets from all origins to all destinations
                setVehicleSequences(nodeID, "", "Vehicle Sequences",
"From")

                'Vehicle destination empirical discrete distribution
                If
String.IsNullOrEmpty(origin.getDestinationDistribution()) = False Then
                    vehicleDestinationDist =
origin.getDestinationDistribution()
                Else
                    vehicleDestinationDist =
vehicleDestinationDiscreteDistribution()
                End If

                'Create Vehicle
                If people <> 0 Then
                    createVehicle = createModule("Create Vehicle " &
nodeID, "Vehicle", x, y + Y_INCREMENT, origin.getVehicles(),
origin.getVehicleTime())
                Else
                    createVehicle = createModule("Create Vehicle " &
nodeID, "Vehicle", x, y, origin.getVehicles(), origin.getVehicleTime())
                End If
            End If

            'Origin Station
            x += X_INCREMENT
            station = stationModule("Station " & nodeID, x, y)

            x += X_INCREMENT
            If people <> 0 And vehicles <> 0 Then
                'Decision Block to separate People and Vehicle to
different Station
                decide = model.Modules.Create("BasicProcess", "Decide",
x, y)
                decide.Data("Name") = "From Origin " & nodeID & " _ Is
Entity Vehicle?"
                decide.Data("Type") = "2-way by Condition"
                decide.Data("If") = "Entity Type"
                decide.Data("TypeNamed") = "People"
                decide.UpdateShapes()

                'Assign People attributes
                x += X_AFTER_DECIDE * X_INCREMENT
            End If

            xAssign = x

            If people <> 0 Then
                assignPeople = assignOriginPeopleModule(nodeID, x, y,
```

```
PeopleTypeDist)

                x += X_INCREMENT
                If startTimeCount > 0 Then
                    'Decide vehicle sequence based on current simulation
time
                    decidePeople = model.Modules.Create("BasicProcess",
"Decide", x, y)
                    decidePeople.Data("Name") = "Which People Travel
Time " & nodeID & "?"
                    decidePeople.Data("Type") = "N-way by Condition"
                    x += X_AFTER_DECIDE * X_INCREMENT
                    xAssign2 = x
                End If

                'Find shortest distance to closest transit center
                findClosestTransitCenter(nodeID)

                'Assign vehicle sequence
                assignPeople2 = model.Modules.Create("BasicProcess",
"Assign", x, y + Y_INCREMENT * startTimeCount)
                assignPeople2.Data("Name") = "Assign People Travel Time
" & nodeID
                assignPeople2.Data("Type(1)") = "Attribute"
                assignPeople2.Data("AName(1)") = "People Travel Time"
                assignPeople2.Data("Value(1)") = "People Shortest Route
Time " & nodeID & "(People Type)"
                assignPeople2.Data("Type(2)") = "Attribute"
                assignPeople2.Data("AName(2)") = transitStation
                assignPeople2.Data("Value(2)") = "Station " &
closestTransitCenter
                assignPeople2.UpdateShapes()

                'People Route Time
                setPeopleTransit =
model.Modules.Create("AdvancedProcess", "Expression", 0, 0)
                setPeopleTransit.Data("Name") = "People Shortest Route
Time " & nodeID
                setPeopleTransit.Data("Dim1") = PeopleList.Count
                For i = 1 To PeopleList.Count
                    setPeopleTransit.Data("Value(" & i & ")") =
shortestDistance & "/ People Speeds(" & i & ") * Hour to Minutes"
                Next
                setPeopleTransit.UpdateShapes()

                'Route People from Origin station
                routePeople = routePeopleModule("Route People from
Origin " & nodeID, x + X_INCREMENT, y, "People Travel Time",
transitStation)

                'If there are incidents, assign new People shortest
route time
                If startTimeCount > 0 Then
                    For i = 0 To startTimeCount - 1
                        start = IncidentStartTimeList.ElementAt(i)

                        'Decide vehicle sequence based on current
simulation time
                        decidePeople.Data("N Percent True(" & i + 1 &
")") = "50"
                        decidePeople.Data("N If(" & i + 1 & ")") =
"Expression"
                        If i = startTimeCount - 1 Then
```

```
                            decidePeople.Data("N Value(" & i + 1 & ")")
= "TNOW >= " & start
                        Else
                            decidePeople.Data("N Value(" & i + 1 & ")")
= "(TNOW >= " & start & ")&& (TNOW < " &
IncidentStartTimeList.ElementAtOrDefault(i + 1) & ")"
                        End If

                        'Assign vehicle sequence
                        x = xAssign2
                        assignPeople2 =
model.Modules.Create("BasicProcess", "Assign", x, y)
                        assignPeople2.Data("Name") = "Assign People
Travel Time " & nodeID & " Start Time " & start
                        assignPeople2.Data("Type(1)") = "Attribute"
                        assignPeople2.Data("AName(1)") = "People Travel
Time"
                        assignPeople2.Data("Value(1)") = "People
Shortest Route Time " & nodeID & " Start Time " & start & "(People
Type)"
                        assignPeople2.Data("Type(2)") = "Attribute"
                        assignPeople2.Data("AName(2)") = transitStation
                        assignPeople2.Data("Value(2)") = "People Closest
Transit Station " & nodeID & " Start Time " & start
                        assignPeople2.UpdateShapes()

                        model.Connections.Create(decidePeople,
assignPeople2)
                        model.Connections.Create(assignPeople2,
routePeople)
                        y += Y_INCREMENT
                    Next
                    decidePeople.UpdateShapes()
                End If

                model.Connections.Create(createPeople, station)
            End If

            If vehicles <> 0 Then
                'Assign Vehicle attributes
                If people <> 0 Then
                    y += Y_INCREMENT
                End If
                x = xAssign
                assignVehicle = assignOriginVehicleModule(nodeID, x, y,
vehicleTypeDist, vehicleDestinationDist)

                x += X_INCREMENT
                If startTimeCount > 0 Then
                    'Decide vehicle sequence based on current simulation
time
                    decideVehicle = model.Modules.Create("BasicProcess",
"Decide", x, y)
                    decideVehicle.Data("Name") = "Which Vehicle Sequence
" & nodeID & "?"
                    decideVehicle.Data("Type") = "N-way by Condition"
                    x += X_AFTER_DECIDE * X_INCREMENT
                    xAssign2 = x
                End If

                'Assign vehicle sequence
                assignVehicle2 = model.Modules.Create("BasicProcess",
"Assign", x, y + Y_INCREMENT * startTimeCount)
```

```
                assignVehicle2.Data("Name") = "Assign Vehicle Sequence " 
& nodeID
                assignVehicle2.Data("Type(1)") = "Attribute"
                assignVehicle2.Data("AName(1)") = "Entity.Sequence"
                assignVehicle2.Data("Value(1)") = "Vehicle Sequences " & 
nodeID & "(Vehicle Destination)"
                assignVehicle2.Data("Type(2)") = "Attribute"
                assignVehicle2.Data("AName(2)") = nextStation
                assignVehicle2.Data("Value(2)") = 
"Entity.PlannedStation"
                assignVehicle2.UpdateShapes()

                'VEHICLE WAITS FOR SIGNALS TO TRAVERSE ORIGIN
                x += X_INCREMENT
                hold = model.Modules.Create("AdvancedProcess", "Hold", 
x, y)
                hold.Data("Name") = "Vehicles Waiting to Move from " & 
origin.getID()
                hold.Data("Type") = "Scan for Condition"
                condition = ""
                For j = 0 To origin.getOriginTo.Count - 1
                    toLink = origin.getOriginTo.ElementAt(j)
                    If ImportData.isDestination(toLink.getStartNode()) 
Then
                        condition = condition & "(" & nextStation & "== 
Station " & toLink.getStartNode() & ")||"
                    Else
                        condition = condition & "(" & nextStation & "== 
Station " & toLink.getStartNode() & ")&&(Available Capacity from " & 
nodeID & " to " & toLink.getStartNode() & ">= Vehicle Length + " & 
vehicleGap.Data("Name") & ")||"
                    End If
                Next
                condition = condition.Remove(condition.Length - 2)
                hold.Data("Condition") = condition
                hold.UpdateShapes()

                'Route Vehicle from Origin station
                x += X_INCREMENT
                routeVehicle = routeVehicleModule("Route Vehicle from 
Origin " & nodeID, x, y, 0)
                xIntersection = x + X_INCREMENT

                If startTimeCount > 0 Then
                    For i = 0 To startTimeCount - 1
                        start = IncidentStartTimeList.ElementAt(i)

                        'Decide vehicle sequence based on current 
simulation time
                        decideVehicle.Data("N Percent True(" & i + 1 & 
")") = "50"
                        decideVehicle.Data("N If(" & i + 1 & ")") = 
"Expression"
                        If i = startTimeCount - 1 Then
                            decideVehicle.Data("N Value(" & i + 1 & ")") 
= "TNOW >= " & start
                        Else
                            decideVehicle.Data("N Value(" & i + 1 & ")") 
= "(TNOW >= " & start & ")&& (TNOW < " & 
IncidentStartTimeList.ElementAtOrDefault(i + 1) & ")"
                        End If

                        'Assign vehicle sequence
```

```
                        x = xAssign2
                        assignVehicle2 =
model.Modules.Create("BasicProcess", "Assign", x, y)
                        assignVehicle2.Data("Name") = "Assign Vehicle
Sequence " & nodeID & " Start Time " & start
                        assignVehicle2.Data("Type(1)") = "Attribute"
                        assignVehicle2.Data("AName(1)") =
"Entity.Sequence"
                        assignVehicle2.Data("Value(1)") = "Rerouted
Vehicle Sequences " & nodeID & " Start Time " & start & "(Vehicle
Destination)"
                        assignVehicle2.Data("Type(2)") = "Attribute"
                        assignVehicle2.Data("AName(2)") = nextStation
                        assignVehicle2.Data("Value(2)") =
"Entity.PlannedStation"
                        assignVehicle2.UpdateShapes()

                        model.Connections.Create(decideVehicle,
assignVehicle2)
                        model.Connections.Create(assignVehicle2, hold)
                        y += Y_INCREMENT
                    Next
                    decideVehicle.UpdateShapes()
                End If

                If people <> 0 AndAlso vehicles <> 0 Then
                    model.Connections.Create(decide, assignVehicle)
                End If
            End If
            y += Y_INCREMENT
        Next

        '============================================== DESTINATION
===========================================

        For Each destination In DestinationList
            x = 0
            nodeID = destination.getID()

            'Destination station
            station = stationModule("Station " & nodeID, x, y)

            'Record destination statistics
            x += X_INCREMENT
            record = model.Modules.Create("BasicProcess", "Record", x,
y)
            record.Data("Name") = "Destination " & nodeID & "
Statistics"
            record.Data("Type") = "Entity Statistics"
            record.UpdateShapes()

            If TransitList.Count <> 0 Then
                'Split entities
                x += X_INCREMENT
                separate = model.Modules.Create("BasicProcess",
"Separate", x, y)
                separate.Data("Name") = "People Get off Public
Transportation " & nodeID
                separate.Data("Type") = "Split Existing Batch"
                separate.Data("Member Attributes") = "Retain Original
Entity Values"
                separate.UpdateShapes()

```

```
                'Record People out for each destination
                x += X_INCREMENT
                record = model.Modules.Create("BasicProcess", "Record",
x, y)
                record.Data("Name") = "Number of People Out " & nodeID
                record.Data("Type") = "Count"
                condition = ""
                For i = 0 To PeopleList.Count - 1
                    condition = condition & "Entity.Type == " &
PeopleList.ElementAt(i).getPeopleType() & "||"
                Next
                condition = condition.Remove(condition.Length - 2)
                record.Data("Value") = condition
                record.Data("Counter Name") = "People Out " & nodeID
                record.UpdateShapes()

                'Record total number of People out
                x += X_INCREMENT
                record = model.Modules.Create("BasicProcess", "Record",
x, y)
                record.Data("Name") = "Total Number of People Out " &
nodeID
                record.Data("Type") = "Count"
                record.Data("Value") = condition
                record.Data("Counter Name") = "Total People Out"
                record.UpdateShapes()

            End If

            'Record Vehicle out for each destination
            x += X_INCREMENT
            record = model.Modules.Create("BasicProcess", "Record", x,
y)
            record.Data("Name") = "Number of Vehicles Out " & nodeID
            record.Data("Type") = "Count"
            condition = ""
            For i = 0 To VehicleList.Count - 1
                condition = condition & "Entity.Type == " &
VehicleList.ElementAt(i).getVehicleType() & "||"
            Next
            condition = condition.Remove(condition.Length - 2)
            record.Data("Value") = condition
            record.Data("Counter Name") = "Vehicles Out " & nodeID
            record.UpdateShapes()

            'Record Vehicle out
            x += X_INCREMENT
            record = model.Modules.Create("BasicProcess", "Record", x,
y)
            record.Data("Name") = "Total Number of Vehicles Out " &
nodeID
            record.Data("Type") = "Count"
            record.Data("Value") = condition
            record.Data("Counter Name") = "Total Vehicles Out"
            record.UpdateShapes()

            'Record end time
            x += X_INCREMENT
            record = model.Modules.Create("BasicProcess", "Record", x,
y)
            record.Data("Name") = "Record End Time " & nodeID
            record.Data("Type") = "Expression"
            record.Data("Value") = "TNOW"
```

```
            record.Data("Tally Name") = "End Time"
            record.UpdateShapes()

            'Dispose entities
            x += X_INCREMENT
            dispose = model.Modules.Create("BasicProcess", "Dispose", x,
y)
            dispose.Data("Name") = "Exit via Destination " & nodeID
            dispose.UpdateShapes()

            y += Y_INCREMENT
        Next

        '============================================== TRANSIT CENTER
==========================================
        If TransitList.Count > 0 Then
            batchSizeIndex = 0
            variable = model.Modules.Create("BasicProcess", "Variable",
0, 0)
            variable.Data("Name") = "Public Transportation Batch Size"
            variable.Data("Rows") = TransitList.Count
            For Each center In TransitList
                x = 0
                nodeID = center.getID()
                batchSizeIndex += 1

                'Generate public transportation capacity Variable
                variable.Data("Initial Value(" & batchSizeIndex & ")") =
PublicTransportationCapacity

                'Sequence sets from all transit centers to all
destinations
                setVehicleSequences(nodeID, "", "Public Transportation
Sequences", "From")

                'Vehicle destination empirical discrete distribution
                vehicleDestinationDist =
cumulativeDiscreteDistribution(vehicleDestinationDistList)

                'Transit Center station
                station = stationModule("Station " & nodeID, x, y)

                'Assign arrival time to transit center
                x += X_INCREMENT
                assign = model.Modules.Create("BasicProcess", "Assign",
x, y)
                assign.Data("Name") = "Assign People Arrival Time " &
nodeID
                assign.Data("Type") = "Attribute"
                assign.Data("AName") = "People Arrival Time to Transit"
                assign.Data("Value") = "TNOW"
                assign.UpdateShapes()

                'Create duplicate entities to scan the condition to
release public transportation
                x += X_INCREMENT
                separate = model.Modules.Create("BasicProcess",
"Separate", x, y)
                separate.Data("Name") = "Duplicate to Create Scan Entity
" & nodeID
                separate.Data("Cost") = "0"
                separate.UpdateShapes()

```

```
                'Hold People until the public transportation is filled
or until the last person is hold more than maximum allowable time
                x += X_INCREMENT
                xScan = x
                hold = model.Modules.Create("AdvancedProcess", "Hold",
x, y)
                hold.Data("Name") = "Hold People for Signal " & nodeID
                hold.Data("Type") = "Wait for Signal"
                hold.Data("Value") = nodeID
                hold.UpdateShapes()

                'Group People Module to load People on public
transportation
                x += X_INCREMENT
                batch = model.Modules.Create("BasicProcess", "Batch", x,
y)
                batch.Data("Name") = "Group People " & nodeID
                batch.Data("Type") = "Temporary"
                batch.Data("Batch Size") = variable.Data("Name") & "(" &
batchSizeIndex & ")"
                batch.UpdateShapes()

                'Assign public transportation attributes
                x += X_INCREMENT
                assignVehicle = model.Modules.Create("BasicProcess",
"Assign", x, y)
                assignVehicle.Data("Name") = "Assign Public
Transportation " & nodeID
                assignVehicle.Data("Type(1)") = "Entity Type"
                assignVehicle.Data("TypeName(1)") = "Bus"
                assignVehicle.Data("Type(2)") = "Entity Picture"
                assignVehicle.Data("PicName(2)") = "Picture.Van"
                assignVehicle.Data("Type(3)") = "Attribute"
                assignVehicle.Data("AName(3)") = "Vehicle Length"
                assignVehicle.Data("Value(3)") =
PublicTransportationLength
                assignVehicle.Data("Type(4)") = "Attribute"
                assignVehicle.Data("AName(4)") = "Vehicle Destination"
                assignVehicle.Data("Value(4)") = vehicleDestinationDist
                assignVehicle.Data("Type(5)") = "Attribute"
                assignVehicle.Data("AName(5)") = "Destination"
                assignVehicle.Data("Value(5)") = "Destination
Set(Vehicle Destination)"
                assignVehicle.Data("Type(6)") = "Attribute"
                assignVehicle.Data("AName(6)") = "Previous Station"
                assignVehicle.Data("Value(6)") = "Entity.Station"
                assignVehicle.UpdateShapes()

                x += X_INCREMENT
                If startTimeCount > 0 Then
                    'Decide vehicle sequence based on current simulation
time
                    decideVehicle = model.Modules.Create("BasicProcess",
"Decide", x, y)
                    decideVehicle.Data("Name") = "Which Vehicle Sequence
" & nodeID & "?"
                    decideVehicle.Data("Type") = "N-way by Condition"
                    x += X_AFTER_DECIDE * X_INCREMENT
                    xAssign2 = x
                End If

                'Assign vehicle sequence
                assignVehicle2 = model.Modules.Create("BasicProcess",
```

```
"Assign", x, y + Y_INCREMENT * startTimeCount)
                assignVehicle2.Data("Name") = "Assign Public
Transportation Sequence " & nodeID
                assignVehicle2.Data("Type(1)") = "Attribute"
                assignVehicle2.Data("AName(1)") = "Entity.Sequence"
                assignVehicle2.Data("Value(1)") = "Public Transportation
Sequences " & nodeID & "(Vehicle Destination)"
                assignVehicle2.Data("Type(2)") = "Attribute"
                assignVehicle2.Data("AName(2)") = nextStation
                assignVehicle2.Data("Value(2)") =
"Entity.PlannedStation"
                assignVehicle2.UpdateShapes()

                'VEHICLE WAITS FOR SIGNALS TO TRAVERSE ORIGIN
                x += X_INCREMENT
                hold2 = model.Modules.Create("AdvancedProcess", "Hold",
x, y)
                hold2.Data("Name") = "Vehicles Waiting to Move from " &
center.getID()
                hold2.Data("Type") = "Scan for Condition"
                condition = ""
                For j = 0 To center.getTransitTo.Count - 1
                    toLink = center.getTransitTo.ElementAt(j)
                    If ImportData.isDestination(toLink.getStartNode())
Then
                        condition = condition & "(" & nextStation & "==
Station " & toLink.getStartNode() & ")||"
                    Else
                        condition = condition & "(" & nextStation & "==
Station " & toLink.getStartNode() & ")&&(Available Capacity from " &
nodeID & " to " & toLink.getStartNode() & ">= Vehicle Length + " &
vehicleGap.Data("Name") & ")||"
                    End If
                Next
                condition = condition.Remove(condition.Length - 2)
                hold2.Data("Condition") = condition
                hold2.UpdateShapes()

                'Route Vehicle from Transit Center station
                x += X_INCREMENT
                routeVehicle = routeVehicleModule("Route Public
Transportation from Transit Center " & nodeID, x, y,
routeTime.Data("Name"))
                xIntersection = x + X_INCREMENT

                If startTimeCount > 0 Then
                    For i = 0 To startTimeCount - 1
                        start = IncidentStartTimeList.ElementAt(i)

                        'Decide vehicle sequence based on current
simulation time
                        decideVehicle.Data("N Percent True(" & i + 1 &
")") = "50"
                        decideVehicle.Data("N If(" & i + 1 & ")") =
"Expression"
                        If i = startTimeCount - 1 Then
                            decideVehicle.Data("N Value(" & i + 1 & ")")
= "TNOW >= " & start
                        Else
                            decideVehicle.Data("N Value(" & i + 1 & ")")
= "(TNOW >= " & start & ")&& (TNOW < " &
IncidentStartTimeList.ElementAtOrDefault(i + 1) & ")"
                        End If
```

```
                        'Assign vehicle sequence
                        x = xAssign2
                        assignVehicle2 =
model.Modules.Create("BasicProcess", "Assign", x, y)
                        assignVehicle2.Data("Name") = "Assign Public
Transportation Sequence " & nodeID & " Start Time " & start
                        assignVehicle2.Data("Type(1)") = "Attribute"
                        assignVehicle2.Data("AName(1)") =
"Entity.Sequence"
                        assignVehicle2.Data("Value(1)") = "Rerouted
Public Transportation Sequences " & nodeID & " Start Time " & start &
"(Vehicle Destination)"
                        assignVehicle2.Data("Type(2)") = "Attribute"
                        assignVehicle2.Data("AName(2)") = nextStation
                        assignVehicle2.Data("Value(2)") =
"Entity.PlannedStation"
                        assignVehicle2.UpdateShapes()

                        model.Connections.Create(decideVehicle,
assignVehicle2)
                        model.Connections.Create(assignVehicle2, hold2)
                        y += Y_INCREMENT
                    Next
                    decideVehicle.UpdateShapes()
                End If

                'Scan if number of waiting in Hold area exceeds the
public transportation capacity or if the last People has to wait more
than maximum allowable time
                x = xScan
                y = y + Y_INCREMENT
                scan = model.Modules.Create("AdvancedProcess", "Hold",
x, y)
                scan.Data("Name") = "Scan for Condition " & nodeID
                scan.Data("Type") = "Scan for Condition"
                scan.Data("Condition") = "NQ(" & hold.Data("Name") &
".Queue) >= " & variablePublicTransportationSize.Data("Name") & "  ||  (
TNOW - " & assign.Data("AName(1)") & "> " &
variablePublicTransportationWaitTime.Data("Name") & " )"
                scan.UpdateShapes()

                'Assign number of People can get onto one public
transportation
                x += X_INCREMENT
                assign = model.Modules.Create("BasicProcess", "Assign",
x, y)
                assign.Data("Name") = "Assign Public Transportation
Batch Size " & nodeID
                assign.Data("Type") = "Other"
                assign.Data("OtherName") = variable.Data("Name") & "(" &
batchSizeIndex & ")"
                assign.Data("Value") = "MN(NQ(" & hold.Data("Name") &
".Queue)," & variablePublicTransportationSize.Data("Name") & ")"
                assign.UpdateShapes()

                'Signal to release public transportation
                x += X_INCREMENT
                signal = model.Modules.Create("AdvancedProcess",
"Signal", x, y)
                signal.Data("Name") = "Signal to Release Public
Transportation " & nodeID
                signal.Data("Value") = nodeID
```

```
                signal.Data("Limit") = variable.Data("Name") & "(" &
batchSizeIndex & ")"
                signal.UpdateShapes()

                'Dispose duplicate entity
                x += X_INCREMENT
                dispose = model.Modules.Create("BasicProcess",
"Dispose", x, y)
                dispose.Data("Name") = "Dispose Duplicate Entity " &
nodeID
                dispose.UpdateShapes()

                model.Connections.Create(separate, scan)

                y += Y_INCREMENT
            Next
        End If


        '============================================== INTERSECTION
===========================================
        y = 0
        For Each intersection In IntersectionList
            x = xIntersection
            nodeID = intersection.getID()
            fromIntersectionCount =
intersection.getFromIntersections.Count
            toIntersectionCount = intersection.getToIntersections.Count

            'If this intersection is start node of incident(s), create a
list of incident start time involving this intersection
            If IncidentList IsNot Nothing Then
                intersectionIncidentStartTimeList = New List(Of Double)
                For Each incidentEvent In IncidentList
                    If incidentEvent.getFromNode() = nodeID AndAlso
intersectionIncidentStartTimeList.Contains(incidentEvent.getStartTime())
= False Then

intersectionIncidentStartTimeList.Add(incidentEvent.getStartTime())
                    End If
                Next
                intersectionIncidentStartTimeCount =
intersectionIncidentStartTimeList.Count
            Else
                intersectionIncidentStartTimeCount = 0
            End If

            'Coordinate of the next intersection station in Arena
            yIntersection = y + Y_INCREMENT *
Math.Max(intersectionIncidentStartTimeCount + 1, fromIntersectionCount)

            'Intersection station
            station = stationModule("Station " & nodeID, x, y)

            x += X_INCREMENT
            If intersectionIncidentStartTimeCount > 0 Then
                'Decide vehicle sequence based on current simulation
time
                decideVehicle = model.Modules.Create("BasicProcess",
"Decide", x, y)
                decideVehicle.Data("Name") = "Which Vehicle Sequence " &
nodeID & "?"
                decideVehicle.Data("Type") = "N-way by Condition"
```

```
                x += X_AFTER_DECIDE * X_INCREMENT
                xAssign = x
            End If

            'Obtain entity's next station
            assign = model.Modules.Create("BasicProcess", "Assign", x, y
+ Y_INCREMENT * intersectionIncidentStartTimeCount)
            assign.Data("Name") = "Assign Attributes " & nodeID
            assign.Data("Type(1)") = "Attribute"
            assign.Data("AName(1)") = "Arrival Time " & nodeID
            assign.Data("Value(1)") = "TNOW"
            assign.Data("Type(2)") = "Attribute"
            assign.Data("AName(2)") = nextStation
            assign.Data("Value(2)") = "Entity.PlannedStation"
            assign.UpdateShapes()

            x += X_INCREMENT
            If fromIntersectionCount > 1 Then
                'Decide which entities from which previous intersections
                decideFromIntersections =
model.Modules.Create("BasicProcess", "Decide", x, y)
                decideFromIntersections.Data("Name") = "From Where to "
& nodeID & "?"
                decideFromIntersections.Data("Type") = "N-way by
Condition"
                x += X_AFTER_DECIDE * X_INCREMENT
                xAssignIn = x
            End If

            'Compute available space/length to trigger the signal for
vehicle entering link
            fromLink =
intersection.getFromIntersections.ElementAt(fromIntersectionCount - 1)
            startNode = fromLink.getStartNode()
            assignIn = model.Modules.Create("BasicProcess", "Assign", x,
y + Y_INCREMENT * (fromIntersectionCount - 1))
            assignIn.Data("Name") = "Calculate Available Capacity " &
startNode & " to " & nodeID
            assignIn.Data("Type") = "Variable"
            assignIn.Data("VName") = "Available Capacity from " &
startNode & " to " & nodeID
            assignIn.Data("Value") = "Available Capacity from " &
startNode & " to " & nodeID & " - " & vehicleGap.Data("Name") & " -
Vehicle Length"
            assignIn.UpdateShapes()

            'Compute time needed for vehicle to reach downstream
            x += X_INCREMENT
            delay = model.Modules.Create("AdvancedProcess", "Delay", x,
y + Y_INCREMENT * (fromIntersectionCount - 1))
            delay.Data("Name") = "Vehicle Running from " & startNode & "
to " & nodeID
            delay.Data("DelayType") = "Available Capacity from " &
startNode & " to " & nodeID & "/ (Speed from " & fromLink.getStartNode()
& " to " & nodeID & "* Mile to Feet / Hour to Minutes *" &
fromLink.getLanes() & ")"
            delay.Data("Units") = "Minutes"
            delay.UpdateShapes()

            'VEHICLE WAITS FOR SIGNALS TO TRAVERSE INTERSECTION
            x += X_INCREMENT
            hold = model.Modules.Create("AdvancedProcess", "Hold", x, y
+ Y_INCREMENT * (fromIntersectionCount - 1))
```

```
            hold.Data("Name") = "Vehicles from " & startNode & " Waiting
to Traverse " & nodeID
            hold.Data("Type") = "Scan for Condition"
            condition = ""
            If intersection.getGreen() <> 0 AndAlso
fromIntersectionCount > 1 Then
                'Generate signal phases
                variableSignal = model.Modules.Create("BasicProcess",
"Variable", 0, 0)
                variableSignal.Data("Name") = "Traffic Signal " & nodeID
                variableSignal.Data("Rows") = fromIntersectionCount
                For j = 1 To fromIntersectionCount
                    variableSignal.Data("Initial Value(" & j & ")") =
RED_SIGNAL
                Next

                'Scan condition
                condition = "(" & variableSignal.Data("Name") & "(" &
fromIntersectionCount & ")==" & GREEN_SIGNAL & ")&&("
                For j = 0 To intersection.getToIntersections.Count - 1
                    toLink =
intersection.getToIntersections.ElementAt(j)
                    If ImportData.isDestination(toLink.getStartNode())
Then
                        condition = condition & "(" & nextStation & "==
Station " & toLink.getStartNode() & ")||"
                    Else
                        condition = condition & "(" & nextStation & "==
Station " & toLink.getStartNode() & ")&&(Available Capacity from " &
nodeID & " to " & toLink.getStartNode() & ">= Vehicle Length + " &
vehicleGap.Data("Name") & ")||"
                    End If
                Next
                condition = condition.Remove(condition.Length - 2)
                condition = condition & ")"
            Else
                For j = 0 To intersection.getToIntersections.Count - 1
                    toLink =
intersection.getToIntersections.ElementAt(j)
                    If ImportData.isDestination(toLink.getStartNode())
Then
                        condition = condition & "(" & nextStation & "==
Station " & toLink.getStartNode() & ")||"
                    Else
                        condition = condition & "(" & nextStation & "==
Station " & toLink.getStartNode() & ")&&(Available Capacity from " &
nodeID & " to " & toLink.getStartNode() & ">= Vehicle Length + " &
vehicleGap.Data("Name") & ")||"
                    End If
                Next
                condition = condition.Remove(condition.Length - 2)
            End If
            hold.Data("Condition") = condition
            hold.UpdateShapes()

            'Recalculate available space/length when vehicle leaves link
            x += X_INCREMENT
            assignOut = model.Modules.Create("BasicProcess", "Assign",
x, y + Y_INCREMENT * (fromIntersectionCount - 1))
            assignOut.Data("Name") = "Assign Available Capacity and
Current Station " & startNode & " to " & nodeID
            assignOut.Data("Type(1)") = "Variable"
            assignOut.Data("VName(1)") = "Available Capacity from " &
```

```
startNode & " to " & nodeID
            assignOut.Data("Value(1)") = "Available Capacity from " &
startNode & " to " & nodeID & " + " & vehicleGap.Data("Name") & " +
Vehicle Length"
            assignOut.Data("Type(2)") = "Attribute"
            assignOut.Data("AName(2)") = "Previous Station"
            assignOut.Data("Value(2)") = "Entity.Station"
            assignOut.UpdateShapes()

            'Record Total Flow on each link
            x += X_INCREMENT
            record = model.Modules.Create("BasicProcess", "Record", x, y
+ Y_INCREMENT * (fromIntersectionCount - 1))
            record.Data("Name") = "Total Flow " & startNode & " to " &
nodeID
            record.Data("Type") = "Count"
            record.Data("Value") = "1"
            record.UpdateShapes()

            'Create output file of flow counter at each link
            statistic = model.Modules.Create("AdvancedProcess",
"Statistic", 0, 0)
            statistic.Data("Name") = "Flow " & startNode & " to " &
nodeID
            statistic.Data("Type") = "Counter"
            statistic.Data("Counter") = "Total Flow " & startNode & " to
" & nodeID
            statistic.Data("CounterOutputFile") = "Flow" & startNode &
"to" & nodeID & ".dat"
            statistic.UpdateShapes()

            'Create output file of average flow on each link
            statistic = model.Modules.Create("AdvancedProcess",
"Statistic", 0, 0)
            statistic.Data("Name") = "Average Flow " & startNode & " to
" & nodeID
            statistic.Data("Type") = "Output"
            statistic.Data("DExp") = "NC(Total Flow " & startNode & " to
" & nodeID & ")/TMAX(End Time)"
            statistic.UpdateShapes()

            'Route Vehicle out of Intersection
            x += X_INCREMENT
            routeVehicle = routeVehicleModule("Route Vehicle from
Intersection " & nodeID, x, y, routeTime.Data("Name"))
            xSignal = x + X_INCREMENT
            ySignal = y

            'Create output file of queue time at each link
            'statistic = model.Modules.Create("AdvancedProcess",
"Statistic", 0, 0)
            'statistic.Data("Name") = "Queue " & startNode & " to " &
nodeID
            'statistic.Data("Type") = "Time-Persistent"
            'statistic.Data("DExp") = "TAVG(Vehicles from " & startNode
& " Waiting to Traverse " & nodeID & ".Queue.WaitingTime)"
            'statistic.Data("CounterOutputFile") = "Queue" & startNode &
"to" & nodeID & ".dat"
            'statistic.UpdateShapes()

            If intersectionIncidentStartTimeCount > 0 Then
                For i = 0 To intersectionIncidentStartTimeCount - 1
                    start =
```

```
intersectionIncidentStartTimeList.ElementAt(i)
                    'Decide vehicle sequence based on current simulation
time
                    decideVehicle.Data("N Percent True(" & i + 1 & ")")
= "50"
                    decideVehicle.Data("N If(" & i + 1 & ")") =
"Expression"
                    If i = intersectionIncidentStartTimeCount - 1 Then
                        decideVehicle.Data("N Value(" & i + 1 & ")") =
"TNOW >= " & start
                    Else
                        decideVehicle.Data("N Value(" & i + 1 & ")") =
"(TNOW >= " & start & ")&& (TNOW < " &
intersectionIncidentStartTimeList.ElementAtOrDefault(i + 1) & ")"
                    End If

                    'Assign vehicle sequence
                    x = xAssign
                    assign = model.Modules.Create("BasicProcess",
"Assign", x, y)
                    assign.Data("Name") = "Assign Vehicle Sequence " &
nodeID & " Start Time " & start
                    assign.Data("Type(1)") = "Attribute"
                    assign.Data("AName(1)") = "Arrival Time " & nodeID
                    assign.Data("Value(1)") = "TNOW"
                    assign.Data("Type(2)") = "Attribute"
                    assign.Data("AName(2)") = "Entity.Jobstep"
                    assign.Data("Value(2)") = "0"
                    assign.Data("Type(3)") = "Attribute"
                    assign.Data("AName(3)") = "Entity.Sequence"
                    assign.Data("Value(3)") = "Rerouted Vehicle
Sequences " & nodeID & " Start Time " & start & "(Vehicle Destination)"
                    assign.Data("Type(4)") = "Attribute"
                    assign.Data("AName(4)") = nextStation
                    assign.Data("Value(4)") = "Entity.PlannedStation"
                    assignIndex = 4
                    For j = 0 To IncidentList.Count - 1
                        currentIncident = IncidentList.ElementAt(j)
                        If currentIncident.getStartTime() = start
AndAlso currentIncident.getFromNode() = nodeID AndAlso
currentIncident.getCapacityPercentage <> 0 Then
                            assign.Data("Type(" & assignIndex & ")") =
"Variable"
                            assign.Data("VName(" & assignIndex & ")") =
"Available Capacity from " & nodeID & " to " &
currentIncident.getToNode()
                            assign.Data("Value(" & assignIndex & ")") =
"Available Capacity from " & nodeID & " to " &
currentIncident.getToNode() & " * " &
(currentIncident.getCapacityPercentage() / PERCENTAGE)
                            assignIndex += 1
                        End If
                    Next
                    assign.UpdateShapes()

                    model.Connections.Create(decideVehicle, assign)
                    model.Connections.Create(assign,
decideFromIntersections)
                    y += Y_INCREMENT
                Next
                decideVehicle.UpdateShapes()
            End If

```

```
            y = ySignal
            If fromIntersectionCount > 1 Then
                For i = 0 To fromIntersectionCount - 2
                    fromLink =
intersection.getFromIntersections.ElementAt(i)
                    startNode = fromLink.getStartNode()
                    decideFromIntersections.Data("N Percent True(" & i +
1 & ")") = "50"
                    decideFromIntersections.Data("N If(" & i + 1 & ")")
= "Attribute"
                    decideFromIntersections.Data("N ANamed(" & i + 1 &
")") = "Previous Station"
                    decideFromIntersections.Data("N Is(" & i + 1 & ")")
= "=="
                    decideFromIntersections.Data("N Value(" & i + 1 &
")") = "Station " & startNode

                    'Compute available space/length to trigger the
signal for vehicle entering link
                    x = xAssignIn
                    assignIn = model.Modules.Create("BasicProcess",
"Assign", x, y)
                    assignIn.Data("Name") = "Calculate Available
Capacity " & startNode & " to " & nodeID
                    assignIn.Data("Type") = "Variable"
                    assignIn.Data("VName") = "Available Capacity from "
& startNode & " to " & nodeID
                    assignIn.Data("Value") = "Available Capacity from "
& startNode & " to " & nodeID & " - " & vehicleGap.Data("Name") & " -
Vehicle Length"
                    assignIn.UpdateShapes()

                    'Compute time needed for vehicle to reach downstream
                    x += X_INCREMENT
                    delay = model.Modules.Create("AdvancedProcess",
"Delay", x, y)
                    delay.Data("Name") = "Vehicle Running from " &
startNode & " to " & nodeID
                    delay.Data("DelayType") = "Available Capacity from "
& startNode & " to " & nodeID & "/ (Speed from " &
fromLink.getStartNode() & " to " & nodeID & "* Mile to Feet / Hour to
Minutes *" & fromLink.getLanes() & ")"
                    delay.Data("Units") = "Minutes"
                    delay.UpdateShapes()

                    'VEHICLE WAITS FOR SIGNALS TO TRAVERSE INTERSECTION
                    x += X_INCREMENT
                    hold = model.Modules.Create("AdvancedProcess",
"Hold", x, y)
                    hold.Data("Name") = "Vehicles from " & startNode & "
Waiting to Traverse " & nodeID
                    hold.Data("Type") = "Scan for Condition"
                    condition = ""
                    If intersection.getGreen() <> 0 Then
                        'Scan condition
                        condition = "(" & variableSignal.Data("Name") &
"(" & i + 1 & ")==" & GREEN_SIGNAL & ")&&("
                        For j = 0 To
intersection.getToIntersections.Count - 1
                            toLink =
intersection.getToIntersections.ElementAt(j)
                            If
ImportData.isDestination(toLink.getStartNode()) Then
```

```
                                condition = condition & "(" &
nextStation & "== Station " & toLink.getStartNode() & ")||"
                            Else
                                condition = condition & "(" &
nextStation & "== Station " & toLink.getStartNode() & ")&&(Available
Capacity from " & nodeID & " to " & toLink.getStartNode() & ">= Vehicle
Length + " & vehicleGap.Data("Name") & ")||"
                            End If
                        Next
                        condition = condition.Remove(condition.Length -
2)
                        condition = condition & ")"
                    Else
                        For j = 0 To
intersection.getToIntersections.Count - 1
                            toLink =
intersection.getToIntersections.ElementAt(j)
                            If
ImportData.isDestination(toLink.getStartNode()) Then
                                condition = condition & "(" &
nextStation & "== Station " & toLink.getStartNode() & ")||"
                            Else
                                condition = condition & "(" &
nextStation & "== Station " & toLink.getStartNode() & ")&&(Available
Capacity from " & nodeID & " to " & toLink.getStartNode() & ">= Vehicle
Length + " & vehicleGap.Data("Name") & ")||"
                            End If
                        Next
                        condition = condition.Remove(condition.Length -
2)
                    End If
                    hold.Data("Condition") = condition
                    hold.UpdateShapes()

                    'Recalculate available space/length when vehicle
leaves link
                    x += X_INCREMENT
                    assignOut = model.Modules.Create("BasicProcess",
"Assign", x, y)
                    assignOut.Data("Name") = "Assign Available Capacity
and Current Station " & startNode & " to " & nodeID
                    assignOut.Data("Type(1)") = "Variable"
                    assignOut.Data("VName(1)") = "Available Capacity
from " & startNode & " to " & nodeID
                    assignOut.Data("Value(1)") = "Available Capacity
from " & startNode & " to " & nodeID & " + " & vehicleGap.Data("Name") &
" + Vehicle Length"
                    assignOut.Data("Type(2)") = "Attribute"
                    assignOut.Data("AName(2)") = "Previous Station"
                    assignOut.Data("Value(2)") = "Entity.Station"
                    assignOut.UpdateShapes()

                    'Record time in station
                    x += X_INCREMENT
                    record = model.Modules.Create("BasicProcess",
"Record", x, y)
                    record.Data("Name") = "Total Flow " & startNode & "
to " & nodeID
                    record.Data("Type") = "Count"
                    record.Data("Value") = "1"
                    record.UpdateShapes()

                    'Create output file of flow counter at each link
```

```
                    statistic = model.Modules.Create("AdvancedProcess",
"Statistic", 0, 0)
                    statistic.Data("Name") = "Flow " & startNode & " to
" & nodeID
                    statistic.Data("Type") = "Counter"
                    statistic.Data("Counter") = "Total Flow " &
startNode & " to " & nodeID
                    statistic.Data("CounterOutputFile") = "Flow" &
startNode & "to" & nodeID & ".dat"
                    statistic.UpdateShapes()

                    'Create output file of average flow on each link
                    statistic = model.Modules.Create("AdvancedProcess",
"Statistic", 0, 0)
                    statistic.Data("Name") = "Average Flow " & startNode
& " to " & nodeID
                    statistic.Data("Type") = "Output"
                    statistic.Data("DExp") = "NC(Total Flow " &
startNode & " to " & nodeID & ")/TMAX(End Time)"
                    statistic.UpdateShapes()

                    model.Connections.Create(decideFromIntersections,
assignIn)
                    model.Connections.Create(record, routeVehicle)

                    y += Y_INCREMENT

                    'Create output file of queue time at each link
                    'statistic = model.Modules.Create("AdvancedProcess",
"Statistic", 0, 0)
                    'statistic.Data("Name") = "Queue " & startNode & "
to " & nodeID
                    'statistic.Data("Type") = "Time-Persistent"
                    'statistic.Data("DExp") = "TAVG(Vehicles from " &
startNode & " Waiting to Traverse " & nodeID & ".Queue.WaitingTime)"
                    'statistic.Data("CounterOutputFile") = "Queue" &
startNode & "to" & nodeID & ".dat"
                    'statistic.UpdateShapes()
                Next
                decideFromIntersections.UpdateShapes()

            End If

            If intersection.getGreen() <> 0 And fromIntersectionCount >
1 Then

                variable = variableModule("Green Time " & nodeID,
intersection.getGreen())
                variable = variableModule("Yellow Time " & nodeID,
intersection.getYellow())

                'Create traffic signal
                createSignal = model.Modules.Create("BasicProcess",
"Create", xSignal, ySignal)
                createSignal.Data("Name") = "Create Traffic Signal " &
nodeID
                createSignal.Data("Entity Type") = "Green Light " &
nodeID
                createSignal.Data("Interarrival Type") = "Expression"
                createSignal.Data("Expression") = "Green Time " & nodeID
& " + Yellow Time " & nodeID
                createSignal.Data("Units") = "Minutes"
                createSignal.UpdateShapes()
```

```

                'Assign Entity Green Light picture
                xSignal += X_INCREMENT
                assignIn = model.Modules.Create("BasicProcess",
"Assign", xSignal, ySignal)
                assignIn.Data("Name") = "Assign Green Light " & nodeID
                assignIn.Data("Type") = "Entity Picture"
                assignIn.Data("PicName") = "Picture.Green Ball"
                assignIn.UpdateShapes()

                'Separate signal phases
                xSignal += X_INCREMENT
                decideSignal = model.Modules.Create("BasicProcess",
"Decide", xSignal, ySignal)
                decideSignal.Data("Name") = "Traffic Phases " & nodeID &
" ?"
                decideSignal.Data("Type") = "N-way by Condition"

                'Assign signal phases
                xSignal += X_AFTER_DECIDE * X_INCREMENT
                xAssign = xSignal
                assign = model.Modules.Create("BasicProcess", "Assign",
xSignal, ySignal + Y_INCREMENT * (fromIntersectionCount - 1))
                assign.Data("Name") = "Assign Traffic Signal " & nodeID
& " from " &
intersection.getFromIntersections.ElementAt(fromIntersectionCount -
1).getStartNode()
                assign.Data("Type(1)") = "Other"
                assign.Data("OtherName(1)") =
variableSignal.Data("Name") & "(" & fromIntersectionCount & ")"
                assign.Data("Value(1)") = GREEN_SIGNAL
                assign.Data("Type(2)") = "Other"
                assign.Data("OtherName(2)") =
variableSignal.Data("Name") & "(MOD(EntitiesIn(" &
createSignal.Data("Entity Type") & ")-1," & fromIntersectionCount & "))"
                assign.Data("Value(2)") = RED_SIGNAL
                assign.UpdateShapes()

                'Pass through green time
                xSignal += X_INCREMENT
                delay = model.Modules.Create("AdvancedProcess", "Delay",
xSignal, ySignal + Y_INCREMENT * (fromIntersectionCount - 1))
                delay.Data("Name") = "Green Light " & nodeID & "
Progressing from " &
intersection.getFromIntersections.ElementAt(fromIntersectionCount -
1).getStartNode()
                delay.Data("DelayType") = "Green Time " & nodeID
                delay.Data("Units") = "Minutes"
                delay.UpdateShapes()

                'Change signal light to yellow
                xSignal += X_INCREMENT
                assignOut = model.Modules.Create("BasicProcess",
"Assign", xSignal, ySignal + Y_INCREMENT * (fromIntersectionCount - 1))
                assignOut.Data("Name") = "Change Signal Light to Yellow
" & nodeID & " from " &
intersection.getFromIntersections.ElementAt(fromIntersectionCount -
1).getStartNode()
                assignOut.Data("Type(1)") = "Other"
                assignOut.Data("OtherName(1)") =
variableSignal.Data("Name") & "(" & fromIntersectionCount & ")"
                assignOut.Data("Value(1)") = YELLOW_SIGNAL
                assignOut.Data("Type(2)") = "Entity Picture"
```

```
                 assignOut.Data("PicName(2)") = "Picture.Yellow Ball"
                 assignOut.UpdateShapes()

                 'Dispose green light entities
                 xSignal += X_INCREMENT
                 dispose = model.Modules.Create("BasicProcess",
"Dispose", xSignal, ySignal)
                 dispose.Data("Name") = "Dispose Traffic Lights " &
nodeID
                 dispose.UpdateShapes()

                 For i = 0 To fromIntersectionCount - 2
                     decideSignal.Data("N Percent True(" & i + 1 & ")") =
"50"
                     decideSignal.Data("N If(" & i + 1 & ")") =
"Expression"
                     decideSignal.Data("N Value(" & i + 1 & ")") =
"MOD(EntitiesIn(" & createSignal.Data("Entity Type") & ")," &
fromIntersectionCount & ")==" & i + 1

                     'Assign signal phases
                     xSignal = xAssign
                     assign = model.Modules.Create("BasicProcess",
"Assign", xSignal, ySignal)
                     assign.Data("Name") = "Assign Traffic Signal " &
nodeID & " from " &
intersection.getFromIntersections.ElementAt(i).getStartNode()
                     assign.Data("Type(1)") = "Other"
                     assign.Data("OtherName(1)") =
variableSignal.Data("Name") & "(" & i + 1 & ")"
                     assign.Data("Value(1)") = GREEN_SIGNAL
                     assign.Data("Type(2)") = "Other"
                     assign.Data("OtherName(2)") =
variableSignal.Data("Name") & "((MOD(EntitiesIn(" &
createSignal.Data("Entity Type") & ")-1," & fromIntersectionCount &
")==0)*" & fromIntersectionCount & " + MOD(EntitiesIn(" &
createSignal.Data("Entity Type") & ")-1," & fromIntersectionCount & "))"
                     assign.Data("Value(2)") = RED_SIGNAL
                     assign.UpdateShapes()

                     'Pass through green time
                     xSignal += X_INCREMENT
                     delay = model.Modules.Create("AdvancedProcess",
"Delay", xSignal, ySignal)
                     delay.Data("Name") = "Green Light " & nodeID & "
Progressing from " &
intersection.getFromIntersections.ElementAt(i).getStartNode()
                     delay.Data("DelayType") = "Green Time " & nodeID
                     delay.Data("Units") = "Minutes"
                     delay.UpdateShapes()

                     'Change signal light to yellow
                     xSignal += X_INCREMENT
                     assignOut = model.Modules.Create("BasicProcess",
"Assign", xSignal, ySignal)
                     assignOut.Data("Name") = "Change Signal Light to
Yellow " & nodeID & " from " &
intersection.getFromIntersections.ElementAt(i).getStartNode()
                     assignOut.Data("Type(1)") = "Other"
                     assignOut.Data("OtherName(1)") =
variableSignal.Data("Name") & "(" & i + 1 & ")"
                     assignOut.Data("Value(1)") = YELLOW_SIGNAL
                     assignOut.Data("Type(2)") = "Entity Picture"
```

```
                    assignOut.Data("PicName(2)") = "Picture.Yellow Ball"
                    assignOut.UpdateShapes()

                    model.Connections.Create(decideSignal, assign)
                    model.Connections.Create(assignOut, dispose)

                    ySignal += Y_INCREMENT
                Next
                decideSignal.UpdateShapes()
            End If

            'Locate the coordinate of next intersection station module
in Arena
            y = yIntersection
        Next

        '==================================================== INCIDENT
=================================================
        If startTimeCount > 0 Then
            'Process group of incidents having same start time
            For Each startTime In IncidentStartTimeList
                incidentStartNodes = New List(Of Integer)
                For i = 0 To IncidentList.Count - 1
                    currentIncident = IncidentList.ElementAt(i)
                    'Modify the graph and/or flow of incident link
                    If currentIncident.getStartTime() = startTime Then
                        If
incidentStartNodes.Contains(currentIncident.getFromNode()) = False Then

incidentStartNodes.Add(currentIncident.getFromNode())
                        End If

                        'Find incident link on graph
                        currentIncidentEdge =
incidentGraph.Edges.FirstOrDefault(Function(qe As QuickGraph.Edge(Of
Integer)) qe.Source = currentIncident.getFromNode() AndAlso qe.Target =
currentIncident.getToNode())
                        If (currentIncident.getCapacityPercentage()) = 0
AndAlso (currentIncidentEdge IsNot Nothing) Then

incidentGraph.RemoveEdge(currentIncidentEdge)

EdgeFlow.Remove(currentIncidentEdge.ToString())

EdgeLength.Remove(currentIncidentEdge.ToString())
                        Else

EdgeFlow.Item(currentIncidentEdge.ToString()) =
EdgeFlow.Item(currentIncidentEdge.ToString()) /
(currentIncident.getCapacityPercentage() / PERCENTAGE)

                        End If
                    End If
                Next

                'New sequence sets from incident start node to all
destinations
                For Each incidentStartNode In incidentStartNodes
                    setVehicleSequences(incidentStartNode, "Start Time "
& startTime, "Rerouted Vehicle Sequences", "Rerouted from")
                Next

                For Each origin In OriginList
```

```
                    nodeID = origin.getID()

                    'New vehicle sequence sets from all origins to all
destinations
                    If incidentStartNodes.Contains(nodeID) = False Then
                        setVehicleSequences(nodeID, "Start Time " &
startTime, "Rerouted Vehicle Sequences", "Rerouted from")
                    End If

                    'Find closest Transit Center
                    findClosestTransitCenter(nodeID)

                    'Generate People route time to Transit Center under
incidents
                    setPeopleTransit =
model.Modules.Create("AdvancedProcess", "Expression", 0, 0)
                    setPeopleTransit.Data("Name") = "People Shortest
Route Time " & nodeID & " Start Time " & startTime
                    setPeopleTransit.Data("Dim1") = PeopleList.Count
                    For i = 1 To PeopleList.Count
                        setPeopleTransit.Data("Value(" & i & ")") =
shortestDistance & "/ People Speeds(" & i & ") * Hour to Minutes"
                    Next
                    setPeopleTransit.UpdateShapes()

                    'Generate closest Transit Center
                    setPeopleTransit =
model.Modules.Create("AdvancedProcess", "Expression", 0, 0)
                    setPeopleTransit.Data("Name") = "People Closest
Transit Station " & nodeID & " Start Time " & startTime
                    setPeopleTransit.Data("Value") = "Station " &
closestTransitCenter
                    setPeopleTransit.UpdateShapes()
                Next

                'New sequence sets from all transit centers to all
destinations
                For Each center In TransitList
                    nodeID = center.getID()
                    If incidentStartNodes.Contains(nodeID) = False Then
                        setVehicleSequences(nodeID, "Start Time " &
startTime, "Rerouted Public Transportation Sequences", "Rerouted from")
                    End If
                Next
            Next
        End If
    End Sub

    'Create Module
    Private Function createModule(ByVal name As String, ByVal type As
String, ByVal x As Integer, ByVal y As Integer, ByVal max As Double,
ByVal rate As String) As Arena.Module
        createModule = model.Modules.Create("BasicProcess", "Create", x,
y)
        createModule.Data("Name") = name
        createModule.Data("Entity Type") = type
        createModule.Data("Interarrival Type") = "Expression"
        createModule.Data("Expression") = rate
        createModule.Data("Units") = "Minutes"
        createModule.Data("Max Batches") = max
        createModule.UpdateShapes()
    End Function

```

```
    'Station Module
    Private Function stationModule(ByVal name As String, ByVal x As
Integer, ByVal y As Integer) As Arena.Module
        stationModule = model.Modules.Create("AdvancedTransfer",
"Station", x, y)
        stationModule.Data("Name") = name
        stationModule.Data("Statn") = name
        stationModule.UpdateShapes()
    End Function

    'Assign People attributes Module
    Private Function assignOriginPeopleModule(ByVal nodeID As Integer,
ByVal x As Integer, ByVal y As Integer, ByVal distribution As String) As
Arena.Module
        assignOriginPeopleModule = model.Modules.Create("BasicProcess",
"Assign", x, y)
        assignOriginPeopleModule.Data("Name") = "Assign People " &
nodeID
        assignOriginPeopleModule.Data("Type(1)") = "Attribute"
        assignOriginPeopleModule.Data("AName(1)") = "People Type"
        assignOriginPeopleModule.Data("Value(1)") = distribution
        assignOriginPeopleModule.Data("Type(2)") = "Attribute"
        assignOriginPeopleModule.Data("AName(2)") = "Entity.Type"
        assignOriginPeopleModule.Data("Value(2)") = "People Types(" &
assignOriginPeopleModule.Data("AName(1)") & ")"
        assignOriginPeopleModule.Data("Type(3)") = "Attribute"
        assignOriginPeopleModule.Data("AName(3)") = "Entity.Picture"
        assignOriginPeopleModule.Data("Value(3)") = "People Pictures(" &
assignOriginPeopleModule.Data("AName(1)") & ")"
        assignOriginPeopleModule.UpdateShapes()
    End Function

    'Assign vehicle attributes Module
    Private Function assignOriginVehicleModule(ByVal nodeID As Integer,
ByVal x As Integer, ByVal y As Integer, ByVal modeDistribution As
String, ByVal destinationDistribution As String) As Arena.Module
        assignOriginVehicleModule = model.Modules.Create("BasicProcess",
"Assign", x, y)
        assignOriginVehicleModule.Data("Name") = "Assign Vehicle " &
nodeID
        assignOriginVehicleModule.Data("Type(1)") = "Attribute"
        assignOriginVehicleModule.Data("AName(1)") = "Vehicle Mode"
        assignOriginVehicleModule.Data("Value(1)") = modeDistribution
        assignOriginVehicleModule.Data("Type(2)") = "Attribute"
        assignOriginVehicleModule.Data("AName(2)") = "Entity.Type"
        assignOriginVehicleModule.Data("Value(2)") = "Vehicle
Types(Vehicle Mode)"
        assignOriginVehicleModule.Data("Type(3)") = "Attribute"
        assignOriginVehicleModule.Data("AName(3)") = "Vehicle Length"
        assignOriginVehicleModule.Data("Value(3)") = "Vehicle
Lengths(Vehicle Mode)"
        assignOriginVehicleModule.Data("Type(4)") = "Attribute"
        assignOriginVehicleModule.Data("AName(4)") = "Entity.Picture"
        assignOriginVehicleModule.Data("Value(4)") = "Vehicle
Pictures(Vehicle Mode)"
        assignOriginVehicleModule.Data("Type(5)") = "Attribute"
        assignOriginVehicleModule.Data("AName(5)") = "Vehicle
Destination"
        assignOriginVehicleModule.Data("Value(5)") =
destinationDistribution
        assignOriginVehicleModule.Data("Type(6)") = "Attribute"
        assignOriginVehicleModule.Data("AName(6)") = "Destination"
        assignOriginVehicleModule.Data("Value(6)") = "Destination
```

```
Set(Vehicle Destination)"
        assignOriginVehicleModule.Data("Type(7)") = "Attribute"
        assignOriginVehicleModule.Data("AName(7)") = "Previous Station"
        assignOriginVehicleModule.Data("Value(7)") = "Entity.Station"
        assignOriginVehicleModule.Data("Type(8)") = "Attribute"
        assignOriginVehicleModule.Data("AName(8)") = "Arrival Time " &
nodeID
        assignOriginVehicleModule.Data("Value(8)") = "TNOW"
        assignOriginVehicleModule.UpdateShapes()
    End Function

    'Route Module
    Private Function routeVehicleModule(ByVal name As String, ByVal x As
Integer, ByVal y As Integer, ByVal routeTime As String) As Arena.Module
        routeVehicleModule = model.Modules.Create("AdvancedTransfer",
"Route", x, y)
        routeVehicleModule.Data("Name") = name
        routeVehicleModule.Data("RouteTime") = routeTime
        routeVehicleModule.Data("Units") = "Minutes"
        routeVehicleModule.Data("SG") = "Sequential"
        routeVehicleModule.UpdateShapes()
    End Function

    'Route People Module
    Private Function routePeopleModule(ByVal name As String, ByVal x As
Integer, ByVal y As Integer, ByVal routeTime As String, ByVal
stationName As String) As Arena.Module
        routePeopleModule = model.Modules.Create("AdvancedTransfer",
"Route", x, y)
        routePeopleModule.Data("Name") = name
        routePeopleModule.Data("RouteTime") = routeTime
        routePeopleModule.Data("Units") = "Minutes"
        routePeopleModule.Data("SG") = "Attribute"
        routePeopleModule.Data("Attr") = stationName
        routePeopleModule.UpdateShapes()
    End Function

    'Single value Variable Module
    Private Function variableModule(ByVal name As String, ByVal
initialValue As Double) As Arena.Module
        variableModule = model.Modules.Create("BasicProcess",
"Variable", 0, 0)
        variableModule.Data("Name") = name
        variableModule.Data("Initial Value") = initialValue
    End Function

    'Single value Expression Module
    Private Function expressionModule(ByVal name As String, ByVal value
As Double) As Arena.Module
        expressionModule = model.Modules.Create("AdvancedProcess",
"Expression", 0, 0)
        expressionModule.Data("Name") = name
        expressionModule.Data("Value") = value
    End Function

    'Create empirical discrete distribution
    Private Function cumulativeDiscreteDistribution(ByVal typeList() As
Double) As String
        Dim total, cumulative As Double
        Dim rand As New Random
        total = 0
        cumulative = 0
        'Assign a random probability number for each vehicle
```

```
        For i = 0 To typeList.Length - 1
            typeList(i) = Math.Round(rand.NextDouble(), 2)
            total = total + typeList(i)
        Next
        cumulativeDiscreteDistribution = "DISC("
        'Write discrete DISC() function that can be used in Arena
        For i = 0 To typeList.Length - 2
            cumulative = cumulative + Math.Floor(typeList(i) / total *
100) / 100
            cumulativeDiscreteDistribution =
cumulativeDiscreteDistribution & cumulative & "," & i + 1 & ","
        Next
        cumulativeDiscreteDistribution = cumulativeDiscreteDistribution
& "1," & typeList.Length & ")"
    End Function

    'Create empirical discrete distribution of destination. If there is
no path between origin and one destination,
    'probability that vehicle is sent to that destination is 0.
    Private Function vehicleDestinationDiscreteDistribution() As String
        Dim probabilityList(DestinationList.Count - 1) As Double
        Dim total, cumulative As Double
        Dim rand As New Random
        Dim pathByFlow As IEnumerable(Of Edge(Of Integer))
        total = 0
        cumulative = 0
        'Assign a random probability number for each vehicle
        For i = 0 To probabilityList.Length - 1
            If getPathByFlow(DestinationList.ElementAt(i).getID(),
pathByFlow) Then
                probabilityList(i) = Math.Round(rand.NextDouble(), 2)
            Else
                probabilityList(i) = 0
            End If
            total = total + probabilityList(i)
        Next
        vehicleDestinationDiscreteDistribution = "DISC("
        'Write discrete DISC() function that can be used in Arena
        For i = 0 To probabilityList.Length - 2
            cumulative = cumulative + Math.Floor(probabilityList(i) /
total * 100) / 100
            vehicleDestinationDiscreteDistribution =
vehicleDestinationDiscreteDistribution & cumulative & "," & i + 1 & ","
        Next
        vehicleDestinationDiscreteDistribution =
vehicleDestinationDiscreteDistribution & "1," & probabilityList.Length &
")"
    End Function

    'Create vehicle sequences
    Private Sub setVehicleSequences(ByVal nodeID As Integer, ByVal time
As String, ByVal sequenceSetName As String, ByVal sequenceName As
String)
        Dim setSequence, sequence As Arena.Module
        Dim path As List(Of Integer)
        Dim destination As Integer
        Dim pathByFlow As IEnumerable(Of Edge(Of Integer))
        getPathByFlow = findShortestPath(Graph, AddressOf getEdgeFlow,
nodeID)

        setSequence = model.Modules.Create("AdvancedProcess", "Advanced
Set", 0, 0)
        setSequence.Data("Name") = sequenceSetName & " " & nodeID & " "
```

```
& time
        setSequence.Data("Type") = "Other"
        For i = 0 To DestinationList.Count - 1
            path = New List(Of Integer)
            destination = DestinationList.Item(i).getID()
            setSequence.Data("Other(" & i + 1 & ")") = sequenceName & "
" & nodeID & " to " & destination & " " & time
            sequence = model.Modules.Create("AdvancedTransfer",
"Sequence", 0, 0)
            sequence.Data("Name") = setSequence.Data("Other(" & i + 1 &
")")
            If getPathByFlow(destination, pathByFlow) Then
                For Each edge In pathByFlow
                    path.Add(edge.Target)
                Next
            Else
                path.Add(destination)
            End If
            For j = 0 To path.Count - 1
                sequence.Data("Station(" & j + 1 & ")") = "Station " &
path(j)
            Next
            sequence.UpdateShapes()
        Next
        setSequence.UpdateShapes()
    End Sub

    'Find shortest path
    Private Function findShortestPath(ByVal g As
IVertexAndEdgeListGraph(Of Integer, Edge(Of Integer)), ByVal edgeCost As
Func(Of Edge(Of Integer), Double), ByVal source As Integer) As
TryFunc(Of Integer, IEnumerable(Of Edge(Of Integer)))
        Return g.ShortestPathsDijkstra(edgeCost, source)
    End Function

    'Find shortest distance between two nodes
    Public Function findShortestDistance(ByVal pathByLength As
IEnumerable(Of Edge(Of Integer))) As Double
        Try
            findShortestDistance = 0
            For Each edge In pathByLength
                findShortestDistance += EdgeLength(edge.ToString())
            Next
            Return findShortestDistance
        Catch ex As Exception
            Console.WriteLine(ex.StackTrace)
            MessageBox.Show(ex.Message)
        End Try
    End Function

    'Find closest Transit Center
    Private Function findClosestTransitCenter(ByVal nodeID As Integer)
As Boolean
        Dim pathByLength As IEnumerable(Of Edge(Of Integer))
        Dim nextShortestDistance As Double
        getPathByLength = findShortestPath(Graph, AddressOf
getEdgeLength, nodeID)
        shortestDistance = 999999
        For Each center In TransitList
            If getPathByLength(center.getID(), pathByLength) Then
                nextShortestDistance =
findShortestDistance(pathByLength)
                If shortestDistance > nextShortestDistance Then
```

```
                        shortestDistance = nextShortestDistance
                        closestTransitCenter = center.getID()
                    End If
                End If
            Next
        End Function

        'Get length value of each link
        Private Function getEdgeLength(ByVal e As Edge(Of Integer)) As
Double
            Return EdgeLength(e.ToString())
        End Function

        'Get flow value of each link
        Private Function getEdgeFlow(ByVal e As Edge(Of Integer)) As Double
            Return EdgeFlow(e.ToString())
        End Function
End Class
```

GISMap.vb

```
Imports MapWinGIS

Public Class GISMap

    Dim axMap As AxMapWinGIS.AxMap
    Private Sub GISMap_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        Dim shp As MapWinGIS.Shapefile
        Dim linksLayerHandle As Integer
        Dim nodesLayerHandle As Integer
        Dim label As String
        Dim label_x, label_y As Double

        'Add ActiveX Control
        axMap = New AxMapWinGIS.AxMap()
        Me.Controls.Add(axMap)
        axMap.Dock = DockStyle.Fill
        axMap.RemoveAllLayers()

        Try
            shp = New MapWinGIS.Shapefile()
            shp.Open(LinksShapeFileName)
            linksLayerHandle = axMap.AddLayer(shp, True)

            shp = New MapWinGIS.Shapefile()
            shp.Open(NodesShapeFileName)
            nodesLayerHandle = axMap.AddLayer(shp, True)

            axMap.ZoomToMaxExtents()
            axMap.set_ShapeLayerPointType(nodesLayerHandle,
tkPointType.ptCircle)
            axMap.set_ShapeLayerPointColor(nodesLayerHandle,
System.Drawing.ColorTranslator.ToOle(System.Drawing.Color.Red))
            axMap.set_ShapeLayerLineColor(linksLayerHandle,
System.Drawing.ColorTranslator.ToOle(System.Drawing.Color.Black))
            axMap.set_ShapeLayerPointSize(nodesLayerHandle, 10)

            For i As Integer = 0 To shp.NumShapes - 1
                label = shp.CellValue(0, i).ToString()
                label_x = shp.QuickExtents(i).xMin
                label_y = shp.QuickExtents(i).yMin
```

```
                axMap.AddLabel(nodesLayerHandle, label,
System.Drawing.ColorTranslator.ToOle(System.Drawing.Color.Black),
label_x, label_y, MapWinGIS.tkHJustification.hjRight)
            Next
        Catch ex As Exception
            MessageBox.Show(Me, ex.Message, "GIS",
MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Try

    End Sub
End Class
```

Counter.vb

```
'Count the traffic flow per time unit. Time unit has lower bound and
upper bound, e.g., [0, 1)

Public Class Counter
    Implements IComparable

    Public LowerBound As Double

    Public UpperBound As Double
    Public Count As Integer = 0

    Public Sub New()

    End Sub

    Public Sub New(ByVal LowerBound As Double, ByVal UpperBound As
Double)
        Me.LowerBound = LowerBound
        Me.UpperBound = UpperBound
    End Sub

    Public Function IsInRange(ByVal value As Double) As Boolean
        Return (value >= LowerBound) AndAlso (value < UpperBound)
    End Function

    Public Shared Function GetCounter(ByVal value As Double) As Counter
        Dim cnt As New Counter()
        cnt.LowerBound = Math.Floor(value)
        cnt.UpperBound = cnt.LowerBound + 1
        cnt.Count = 0
        Return cnt
    End Function

    Public Shared Function GetCounter(ByVal counterList As List(Of
Counter), ByVal value As Double) As Counter
        For Each c As Counter In counterList
            If c.IsInRange(value) Then
                Return c
            End If
        Next
        Return Nothing
    End Function

    Public Function CompareTo(ByVal obj As Object) As Integer
Implements System.IComparable.CompareTo
        Dim c2 As Counter = obj
        If (Me.LowerBound = c2.LowerBound AndAlso Me.UpperBound =
c2.UpperBound) Then
```

```
                Return 0
            ElseIf (Me.LowerBound < c2.LowerBound) Then
                Return -1
            Else
                Return 1
            End If
        End Function
End Class
```

ExportDataForm.vb

```
'User interface to import data

Public Class ExportDataForm
    Private Sub btnBrowseRepInput_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles btnBrowseRepInput.Click
        openInput.Filter = "All Files|*.*"
        If (openInput.ShowDialog() = Windows.Forms.DialogResult.OK)
Then
            txtRepInput.Text = openInput.FileName
        End If
    End Sub

    Private Sub btnBrowseRepOutput_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles btnBrowseRepOutput.Click
        saveOutput.Filter = "All Files|*.*"
        If (saveOutput.ShowDialog() = Windows.Forms.DialogResult.OK)
Then
            txtRepOutput.Text = saveOutput.FileName
        End If
    End Sub

    Private Sub btnRunRepTest_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles btnRunRepTest.Click
        Dim repList As List(Of Replication) =
ReplicationHelper.ReadFromFile(txtRepInput.Text)
        ReplicationHelper.WriteToTextFile(repList, txtRepOutput.Text)
        MessageBox.Show(Me, "File " & txtRepInput.Text & " is exported
successfully to " & txtRepOutput.Text)
    End Sub
End Class
```

Replication.vb

```
'Manage flow data for each simulation replication

Imports System.Text

Public Class Replication
    Public ReplicationNumber As Integer
    Public Data As New List(Of Double)

    Public Function InsertData(ByVal dataLine As String) As Boolean
        If (dataLine.StartsWith("-")) Then
            ' End of Replication
            ' Parse the replication number
            ReplicationNumber = Math.Abs(Double.Parse(dataLine.Split("
")(0)))
            Return False
```

```
        End If
        ' Extract the first value
        Dim value As String = dataLine.Split(" ")(0)
        ' Insert that value into the list
        Me.Data.Add(Double.Parse(value))
        ' Return true to indicate that this is not the end of
Replication
        Return True
    End Function

    Public Function GetHistogram(ByVal maxValue As Double) As List(Of
Counter)
        maxValue = Data.Max()
        Dim hist As New List(Of Counter)
        For i As Integer = 0 To Math.Floor(maxValue)
            hist.Add(New Counter(i, i + 1))
        Next
        If (Data IsNot Nothing) Then
            For Each value As Double In Data
                Dim c As Counter = Counter.GetCounter(hist, value)
                If (c Is Nothing) Then
                    c = Counter.GetCounter(value)
                    c.Count = 1
                    hist.Add(c)
                Else
                    c.Count += 1
                End If
            Next
        End If
        hist.Sort()
        Return hist
    End Function

    Public Function GetHistogramAsFormattedString(ByVal maxValue As
Double)
        Dim s As New StringBuilder()
        ' Histogram
        Dim hist As List(Of Counter) = GetHistogram(maxValue)
        ' Replication Number
        s.Append("Rep").Append(ReplicationNumber)
        ' Append histogram info
        For Each c As Counter In hist
            s.Append(vbTab).Append(c.Count)
        Next
        Return s.ToString()
    End Function
End Class
```

ReplicationHelper.vb

```
'Read and write traffic flow per time unit

Imports System.IO

Public Class ReplicationHelper
    Public Shared Function ReadFromFile(ByVal filePath As String) As
List(Of Replication)
        Dim repList As New List(Of Replication)
        Using sr As New StreamReader(filePath)
            Dim line As String
            Dim lineCount As Long = 0
```

```
            Dim rep As New Replication()
            ' Read lines from the file until the end of
            ' the file is reached.
            Do
                line = sr.ReadLine()
                lineCount += 1
                ' Ignore first 5 lines
                If Not (line Is Nothing Or lineCount < 5) Then
                    ' Insert into current replication
                    If Not (rep.InsertData(line)) Then
                        ' End of current replication
                        ' insert into the list
                        ' and create a new replication
                        repList.Add(rep)
                        rep = New Replication()
                    End If
                End If
            Loop Until line Is Nothing
        End Using
        Return repList
    End Function

    Public Shared Sub WriteToTextFile(ByVal repList As List(Of
Replication), ByVal filePath As String)
        Using outfile As New StreamWriter(filePath)
            ' Maximum value
            Dim maxValue As Double = Double.MinValue
            For Each rep As Replication In repList
                If (maxValue < rep.Data.Max()) Then
                    maxValue = rep.Data.Max()
                End If
            Next
            ' Header
            'outfile.Write("Rep#")
            'For i As Integer = 0 To Math.Floor(maxValue)
            'outfile.Write(vbTab)
            'outfile.Write("[" & i & ", " & (i + 1) & ")")
            'Next
            For Each rep As Replication In repList
                outfile.WriteLine()

outfile.Write(rep.GetHistogramAsFormattedString(maxValue))
            Next
        End Using
    End Sub
End Class
```

APPENDIX E

OUTPUT REPORTS

User Specified

Evacuation Model Replications: 1

Replication 1 Start Time: 0.00 Stop Time: 83.50 Time Units: Minutes

Counter

Count	Value
People Out 1776	155.00
People Out 1777	16.0000
People Out 1780	160.00
People Out 1782	0
Total Flow 15930 to 7611	1.0000
Total Flow 15931 to 1781	16.0000
Total Flow 15932 to 7611	0
Total Flow 1778 to 1781	203.00
Total Flow 1778 to 7407	2,435.00
Total Flow 1778 to 7534	0
Total Flow 1779 to 7404	0
Total Flow 1779 to 7611	1,546.00
Total Flow 1781 to 1778	8.0000
Total Flow 1781 to 7535	211.00
Total Flow 7404 to 1779	1,725.00
Total Flow 7404 to 7534	0
Total Flow 7405 to 15932	0
Total Flow 7405 to 7404	179.00
Total Flow 7405 to 7406	333.00
Total Flow 7405 to 7407	203.00
Total Flow 7407 to 1778	203.00
Total Flow 7533 to 7532	622.00
Total Flow 7533 to 7534	3,973.00
Total Flow 7533 to 7535	3,190.00
Total Flow 7534 to 1778	2,427.00
Total Flow 7534 to 7404	1,546.00
Total Flow 7535 to 1781	0
Total Flow 7611 to 15932	1,547.00
Total Flow 7611 to 1779	0
Total People Out	331.00
Total Vehicles Out	8,500.00
Vehicles Out 1776	2,576.00

User Specified

Evacuation Model Replications: 1

Replication 1 Start Time: 0.00 Stop Time: 83.50 Time Units: Minutes

Counter

Vehicles Out 1777	1,730.00
Vehicles Out 1780	3,393.00
Vehicles Out 1782	801.00

Output

Output	Value
Average Flow 15930 to 7611	0.01197546
Average Flow 15931 to 1781	0.1916
Average Flow 15932 to 7611	0
Average Flow 1778 to 1781	2.4310
Average Flow 1778 to 7407	29.1602
Average Flow 1778 to 7534	0
Average Flow 1779 to 7404	0
Average Flow 1779 to 7611	18.5141
Average Flow 1781 to 1778	0.0958
Average Flow 1781 to 7535	2.5268
Average Flow 7404 to 1779	20.6577
Average Flow 7404 to 7534	0
Average Flow 7405 to 15932	0
Average Flow 7405 to 7404	2.1436
Average Flow 7405 to 7406	3.9878
Average Flow 7405 to 7407	2.4310
Average Flow 7407 to 1778	2.4310
Average Flow 7533 to 7532	7.4487
Average Flow 7533 to 7534	47.5785
Average Flow 7533 to 7535	38.2017
Average Flow 7534 to 1778	29.0644
Average Flow 7534 to 7404	18.5141
Average Flow 7535 to 1781	0
Average Flow 7611 to 15932	18.5260
Average Flow 7611 to 1779	0
Total Evacuation Time	83.5041

Queues

Evacuation Model

Replications: 1

Replication 1 Start Time: 0.00 Stop Time: 83.50 Time Units: Minutes

Queue Detail Summary

Time

	Waiting Time
Group People 15930.Queue	0.00
Group People 15931.Queue	0.00
Hold People for Signal 15930.Queue	57.07
Hold People for Signal 15931.Queue	2.20
Scan for Condition 15930.Queue	60.00
Scan for Condition 15931.Queue	55.94
Vehicles from 15930 Waiting to Traverse 7611.Queue	1.26
Vehicles from 15931 Waiting to Traverse 1781.Queue	0.61
Vehicles from 1778 Waiting to Traverse 1781.Queue	0.50
Vehicles from 1778 Waiting to Traverse 7407.Queue	0.21
Vehicles from 1779 Waiting to Traverse 7611.Queue	0.47
Vehicles from 1781 Waiting to Traverse 1778.Queue	0.48
Vehicles from 1781 Waiting to Traverse 7535.Queue	0.14
Vehicles from 7404 Waiting to Traverse 1779.Queue	0.14
Vehicles from 7405 Waiting to Traverse 7404.Queue	0.00
Vehicles from 7405 Waiting to Traverse 7406.Queue	0.00
Vehicles from 7405 Waiting to Traverse 7407.Queue	0.19
Vehicles from 7407 Waiting to Traverse 1778.Queue	0.39
Vehicles from 7533 Waiting to Traverse 7532.Queue	0.00
Vehicles from 7533 Waiting to Traverse 7534.Queue	0.00
Vehicles from 7533 Waiting to Traverse 7535.Queue	0.20
Vehicles from 7534 Waiting to Traverse 1778.Queue	0.34
Vehicles from 7534 Waiting to Traverse 7404.Queue	0.00
Vehicles from 7611 Waiting to Traverse 15932.Queue	0.17
Vehicles Waiting to Move from 15930.Queue	0.00
Vehicles Waiting to Move from 15931.Queue	0.00
Vehicles Waiting to Move from 7405.Queue	0.00
Vehicles Waiting to Move from 7533.Queue	0.00

Queues

Evacuation Model

Replications: 1

Replication 1

Start Time: 0.00 Stop Time: 83.50 Time Units: Minutes

Other

	Number Waiting
Group People 15930.Queue	0.00
Group People 15931.Queue	0.00
Hold People for Signal 15930.Queue	10.93
Hold People for Signal 15931.Queue	8.29
Scan for Condition 15930.Queue	11.40
Scan for Condition 15931.Queue	204.05
Vehicles from 15930 Waiting to Traverse 7611.Queue	0.02
Vehicles from 15931 Waiting to Traverse 1781.Queue	0.12
Vehicles from 15932 Waiting to Traverse 7611.Queue	0.00
Vehicles from 1778 Waiting to Traverse 1781.Queue	1.20
Vehicles from 1778 Waiting to Traverse 7407.Queue	6.11
Vehicles from 1778 Waiting to Traverse 7534.Queue	0.00
Vehicles from 1779 Waiting to Traverse 7404.Queue	0.00
Vehicles from 1779 Waiting to Traverse 7611.Queue	8.78
Vehicles from 1781 Waiting to Traverse 1778.Queue	0.05
Vehicles from 1781 Waiting to Traverse 7535.Queue	0.35
Vehicles from 7404 Waiting to Traverse 1779.Queue	2.80
Vehicles from 7404 Waiting to Traverse 7534.Queue	0.00
Vehicles from 7405 Waiting to Traverse 15932.Queue	0.00
Vehicles from 7405 Waiting to Traverse 7404.Queue	0.00
Vehicles from 7405 Waiting to Traverse 7406.Queue	0.00
Vehicles from 7405 Waiting to Traverse 7407.Queue	0.45
Vehicles from 7407 Waiting to Traverse 1778.Queue	0.94
Vehicles from 7533 Waiting to Traverse 7532.Queue	0.00
Vehicles from 7533 Waiting to Traverse 7534.Queue	0.01
Vehicles from 7533 Waiting to Traverse 7535.Queue	7.55
Vehicles from 7534 Waiting to Traverse 1778.Queue	9.84
Vehicles from 7534 Waiting to Traverse 7404.Queue	0.00
Vehicles from 7535 Waiting to Traverse 1781.Queue	0.00
Vehicles from 7611 Waiting to Traverse 15932.Queue	3.17
Vehicles from 7611 Waiting to Traverse 1779.Queue	0.00
Vehicles Waiting to Move from 15930.Queue	0.00
Vehicles Waiting to Move from 15931.Queue	0.00
Vehicles Waiting to Move from 7405.Queue	0.00
Vehicles Waiting to Move from 7533.Queue	0.00