SIMULATION LAB

DEVELOPMENT OF TRAFFIC SIMULATION LABORATORY FOR DESIGN PLANNING AND TRAFFIC OPERATIONS (PHASE II)

FINAL REPORT

Prepared by

Paul Telega and Panos Michalopoulos
Department of Civil Engineering
University of Minnesota
Minneapolis, MN 55455

June 2000

Submitted to

Minnesota Department of Transportation
Office of Research Administration
200 Ford Building, 117 University Avenue
St. Paul, MN 55155

ACKNOWLEDGEMENTS

This research was supported by the Minnesota Department of Transportation (Mn/DOT) and the Intelligent Transportation Systems (ITS) Institute in the Center for Transportation Studies (CTS) at the University of Minnesota. The authors would also like to thank the following individuals for their contributions to this document.
James Aswegan — *Minnesota Department of Transportation*
Lowell A. Benson — *Center for Transportation Studies*
Ron Dahl — Mn/DOT s*Traffic Management Center*
John Hourdakis — *University of Minnesota Department of Civil Engineering*
Dr. Eil Kwon —*Center for Transportation Studies*

Table of Contents

Table of Figures

EXECUTIVE SUMMARY

A key element in improving traffic operations and performing effective real-time traffic management is using simulation to assess the effectiveness of various alternatives prior to implementation. The research conducted here is Phase II of a three phased project with the ultimate goal of creating and running traffic simulation experiments in real-time. In the first phase, a set of well-known freeway simulators was evaluated. Major difficulties were a lack of real data, and the time consuming effort required to prepare data for each simulator. Phase I found that developing an integrated traffic analysis environment, where data processing, simulation and output analysis can be automated as efficiently as possible was of critical importance in improving traffic management and operations.

In the second phase, the development of an Automated Simulations Tool (AST) was of critical importance. Phase II was partitioned into four major tasks: development of a Geometry Data Container (GDC), Creation of a partial Twin Cities Freeway Geometry, development of an AST, and specification of a real-time AST framework. Each part of this phase was essentially prescribed by the Phase I results.

The GDC would be the design and implementation of a common geometry database that could be shared among different simulators. Initially, only a micro simulator would be implemented, but later other simulators could be added.

Creation of a partial Twin Cities Freeway Geometry would be the base level common geometry that is entered. All the detail needed for a micro simulation is entered including freeway sections, detector locations and types, ramp meters, and other fine

details. This work needs to be done only once with our design. Any subnetwork of the original entered network can be selected with a mouse and saved as a new network.

Development of the AST will allow traffic engineers to select a freeway and a time period for simulation and then essentially run a simulation without any direct manipulation of data. Traffic engineers will not need to know anything about the data formats of either geometry or flow data in order to run a simulation.

Finally, from what we have learned in Phase II we can specify what needs to be done for a real-time implementation of the AST: Phase III. By real-time we mean using the detector data as soon as it is available from Mn/DOT for simulation.

1 Introduction

1.1 *Problem Statement*

The key element in improving traffic operations and performing real-time management is the ability to assess the effectiveness of various alternatives prior to implementation. Simulation methods have long been recognized as the most effective tool for such analysis, and various simulators have been developed by different agencies for analyzing freeway and/or arterial networks. However, simulation has not yet become a suitable tool for practical application. One reason for this is the extensive manual labor required to input the different kinds of traffic data into most simulation programs, which points to the need for a simulation tool that provides automatic access to large amounts of traffic data. The purpose of this research in Phase II, therefore, is to develop an *Automated Simulation Tool* (AST) with automatic access to both traffic geometry and traffic measurement data. The AST will be part of a *Laboratory Environment for TRaffic ANalysis* (LETRAN), which will provide easy and efficient access to various kinds of traffic data for use in simulation, control, incident detection, and other types of traffic analysis applications to be deployed in a next-generation traffic management center.

1.2 Research Objectives

Our major objective is to develop viable, intelligent, and automated traffic flow simulation systems. These systems can then be deployed in the next-generation traffic management center, and can be enabling advanced research as part of the ITS Laboratory infrastructure.

The ultimate objective of this research (in collaboration with other projects) is to develop a *Laboratory Environment for TRaffic ANalysis* (LETRAN) as part of the CTS ITS Lab. Such an environment would provide an integrated platform for traffic data management, model development and deployment, and the investigation of next-generation traffic management applications. The specific goal of this research is to develop an *Automated Simulation Tool* (AST) as an important part of this environment. This goal can be subdivided into two parts. The first is the creation of a *Geometry Data Container* (GDC) which provides a centralized and extensible location to store and manage traffic geometry data to be used by simulators and other lab analysis tools. The second is the development of the AST, which uses both the GDC and the *Traffic Data Management System* (TDMS) created in the I-394 Lab project for automated creation of simulation experiments.

In Phase II, we divided the project into four major objectives:

- Development of the Geometry Data Container (GDC)
- Creation of the I35W Freeway Geometry
- Development of an Automated Simulation Tool (AST)
- Specification of a Real-Time AST Framework

1.3 Background/ History/ Past Work

The full Simulation Lab Project is designed to be executed in three major phases:

- Phase I: Freeway simulator testing and evaluation
- Phase II: Development of the Automated Simulation Tool (AST)
- Phase III: Integration of Simulation Lab and I-394 Lab.

Phase I of this research evaluated a set of well-known freeway simulators, including AIMSUN, FREFLO, FREQ, FRESIM, INTEGRATION, and KRONOS. Standard test cases were developed and run on different simulators using real data collected by the Minnesota Department of Transportation (Mn/DOT). This evaluation process has already highlighted the difficulties of using current simulation software in a practical fashion, primarily due to the large amount of effort required to create traffic geometries and input traffic data. This work has therefore pointed to a need for the development of an automated traffic simulator where manual data input is minimized and automated input of data to the simulator is maximized.

An additional concept for improving simulation which was explored in Phase I was the idea of a *Common User-Interface* (CUI). A CUI would provide a single interface through which a user could create geometries, input traffic data, and run multiple simulation tools. A prototype CUI was developed with the assistance of Dr. Jaime Barcelo and researchers form the Universitat Politecnica De Catalunya from Barcelona, Spain which uses the GETRAM interface to create simulation input files not only for AIMSUN (for which it was created), but also for the traffic simulator developed at the

University of Minnesota (KRONOS). This process has helped to determine that the development of a CUI for even a small number of simulators is not a feasible task, primarily because of the vastly diverse modeling methodologies and approaches that different simulators possess. Therefore, a more feasible and equally useful goal was chosen: the development of an *Automated Simulation Tool* (AST) for making traffic simulation faster, easier, and more accessible to traffic engineers and managers.

In Phase II, which is discussed in the paper, we developed and implemented an Automated Simulation Tool that minimizes the tedious data entry, which is required for microscopic simulation. Traffic data from the Minnesota Department of Transportation can be automatically prepared for use in simulation.

In Phase III, the work that was done in Phase II will be extended so that real-time traffic data can be used in simulation and integrate this with the I-394 Lab and TRACLAB.

1.4 Work Summary

In completing this project, the following major work was completed:

- Creation of a traffic geometry database system
- Creation of a Twin Cities freeway geometry (partial)
- Development of the automated simulation tool

1.5 Report Organization

This report is organized essentially in order of major task.

2 Geometry Data Container

2.1 Objectives

The fundamental objectives of designing this Geometric Data Container (GDC) are to create:

- Centralized site for storing a network
- One-time creation of data for each network
- Allow data extensibility in the GDC
- Generic access protocol
- Data manipulation ability

2.1.1 Centralized site for storing a network and One-time creation of data for each network

A centralized site for storing all the major networks that are studied will exist at CTS in the ITS Laboratory. Only a one-time entry of each major network will be necessary. It will eliminate duplication of effort in the entering of and the updating of such networks. Researchers can then take advantage of already existing networks and use them in their work. Extra efforts will be made to keep the central database for each network up to date.

2.1.2 Allow data extensibility in the GDC

One major feature of database systems is that they allow augmentation of an existing database without disrupting current applications. That is, additional data fields and/or tables can be added to the current database to accommodate additional applications. In our case, additional fields or tables would contain information that allow other simulators (when suitably modified) to use the same database. This means that other simulators (once modified) can potentially use the same central geometry database that is created by our traffic editor for simulation and select only the information that they need in order to build the simulation model.

2.1.3 Generic access protocol

A generic access protocol to the database was selected called Open DataBase Connectivity (ODBC). ODBC allows any database system that supports the ODBC protocol to be used for information retrieval. This means that we do not need to program database commands for a specific database. All of our programs will be able to use any database that is ODBC compliant. Currently, Microsoft ACCESS, ORACLE, SYBASE, and most other major database systems have ODBC drivers. So by writing our programs using ODBC, we will have the capability to access many different database systems without reprogramming. For deployment in the field, this means that our system can more easily adapt to a commercial database that an engineering group already has without costly reprogramming.

2.1.4 Data manipulation ability

Choosing a database also shifts the burden of data management to the database system. That is, with a real database system all we need to do in an application is make store and retrieve requests of the database. The database system will then execute these commands for us. We don t need to worry about actually writing these to separate files or exactly where the data is. Not only will the database execute commands from the TSS, but any other program can also send commands to the database to retrieve or update information. The ASCII network file version will not allow data manipulation to be done without writing specific programs. For example, another program or the database tools could be used to query the I35W database for the number, type, and age of the detectors.

2.2 GIs Considerations

The Geographical Information Systems (GIS) that were reviewed for this project were not able to provide enough detail for microscopic simulation. This review of GISs was completed in Phase I. The following paragraphs describe some of our reasons for not selecting a GIS at this time.

2.2.1 Capability (lack of)

GISs organize information into layers. For example, a map could be constructed that has a pavement layer, a soil layer, and a hydrology layer. Each of these layers is separate and do not interact with the other layers. In the highway systems that we design and work with a roadway will pass over another roadway and then further down it will then pass under the same roadway. No GIS reviewed can handle this type of situation. In this case, it cannot be determined in which layer the freeway passing under and over another freeway belongs. This is a fundamental problem for GIS systems that model highways.

GISs are based on geometric shapes such as lines, arcs, and polygons, but not intuitive objects such as a directed roadway with three lanes. It may be possible to modify a GIS to handle this situation, but it would be cumbersome and non-intuitive to use. Generally, trying to adapt a program to do something that it was not intended to do will usually result in frustration and an inadequate representation of the problem itself.

2.2.2 Ease of use

Most GIS s reviewed were not simple to use. They required special editors that were not what a traffic engineer would be familiar with, and they provided many more types of modeling features most of which were not needed for traffic modeling. These extra modeling features could lead to confusion and frustration.

2.2.3 Cost of software

Finally, the cost of GIS specific software was very high when deployed to the field. Commercial pricing could be as high as $50,000 or higher. In addition, special training would be necessary to use it properly.

2.2.4 Uniformity and availability of G.I.S.

Most GIS s had no standard way of storing data. Each GIS had its own method of dealing with saved data making it difficult to interface with other programs. In addition, it was not entirely clear that any of the GISs provided an application programming interface (API) that could be used to get at the geometric features that it managed.

2.3 Design of the GDC

The Geometry Data Container designed here will be implemented in a relational database. The database design required analysis of the original structure of the information written to the ASCII file system, and the development of a set of tables and relationships between the tables. The relationships between the tables are defined by how information is used in the original ASCII file design. The GDC contains the geometry, detector, metering, and variable message sign information.

Changing the underlying structure of the geometry to create the GDC will have no visual or operational impact on the user of the traffic simulation system. The format of the geometry data is essentially hidden from the user of the system. It makes no difference if this information is stored in separate files, a relational database, or an object database because the application programming interface (API) actually interacts with the information directly not the user. In figure 2-1, the GDC in the lower left part is the new feature being added with major additions and modifications to the API.

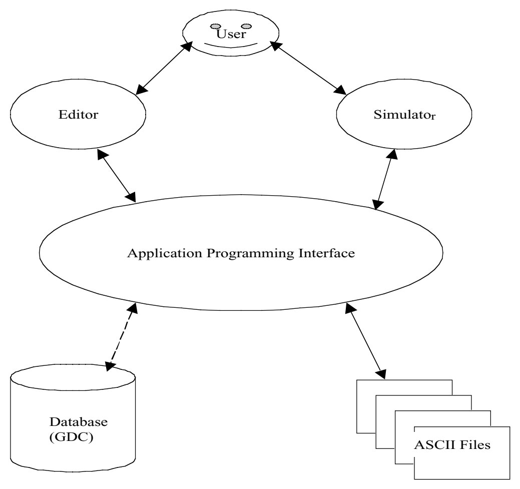


Figure 2-1: Traffic editor and simulator with an API connecting it to the geometry.

2.3.1 Relational database model chosen

Originally our design was to include an object database, but the complexity of implementing completely object oriented database was too much. The TSS data that was stored was not directly object oriented, but it behaved somewhat like an object oriented system. Unfortunately, the API in the TSS was not object oriented and would be very difficult to convert. Therefore, since a relational database would provide many of the features we wanted, was much less expensive, and the learning curve was much less, we decided to use a relational database system.

2.3.2 ODBC/SQL chosen to access the database

Once the database was chosen, we had to make another decision on how to communicate with it. We again had two choices: pick a particular database and use its proprietary commands, or choose a generic protocol that most databases would be able to communicate with.

In the first case, we would have to design functions for a particular database. If we decided to switch to a different database later, it would cause many of the functions to be redesigned for a different database. This would be very costly.

In the second case, a generic protocol such as Open Database Connectivity (ODBC) with embedded Structured Query Language (SQL) would allow most functions designed to work with many different database systems with little reprogramming. There is some small cost because generic protocols many run somewhat slower than native database functions, but the flexibility of using other databases far outweighs this cost.

Finally since the SQL commands are embedded within the ODBC, additional functions can be easily written by a non-expert to search the database for specific patterns or other information of interest to the engineer. Without a database and ODBC/SQL these types of searches require custom programming which can be very expensive.

In order to build a database from the original format of the geometry data, we had to do an analysis of the current ASCII Network system of files and define a mapping that would carry the information from the files to a database. The ASCII Network was composed of eleven files: network, blocks, texts, vehicle classes, global messages, sections, nodes, controllers, centroids, roguis, and routes. Each of these files contains specific information on some part of the geometry.

The simplest file is Blocks. It contains a list of blocks including block points, which provide a visual background for the user. Each block could represent a building, while each block point is defines the actual outline of the block. All this information exists in one file. In mapping this to a database, two tables are defined: blocks, and blockpoints. In figure 2-2, a simple block is drawn in the traffic editor and its representation in ASCII Network format is in figure 2-3. Finally, figure 2-4 represents the GDC representation in two tables.

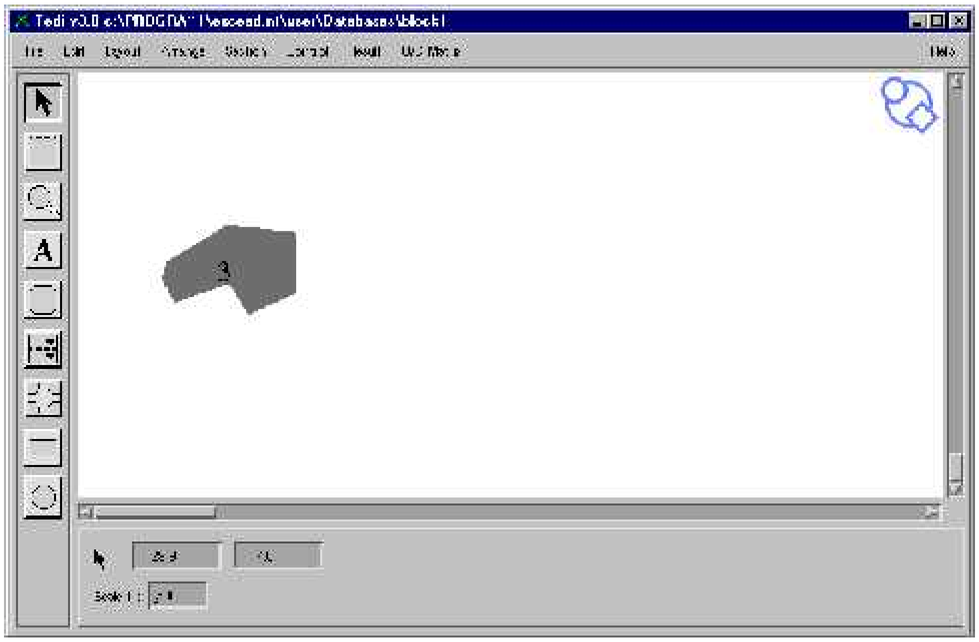
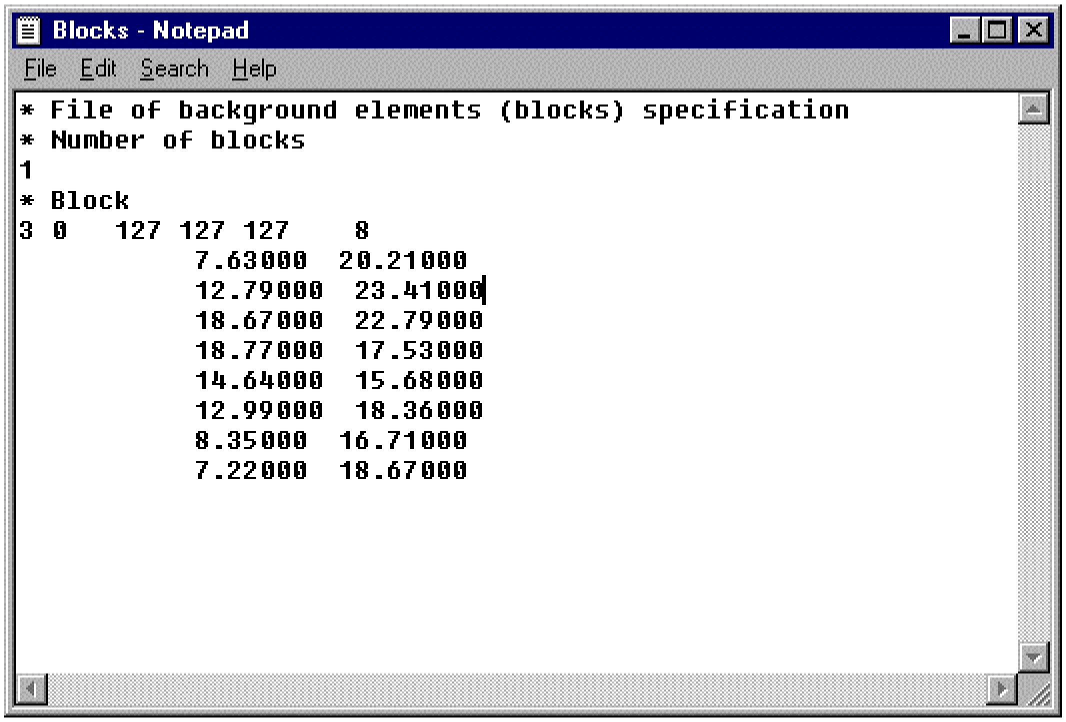


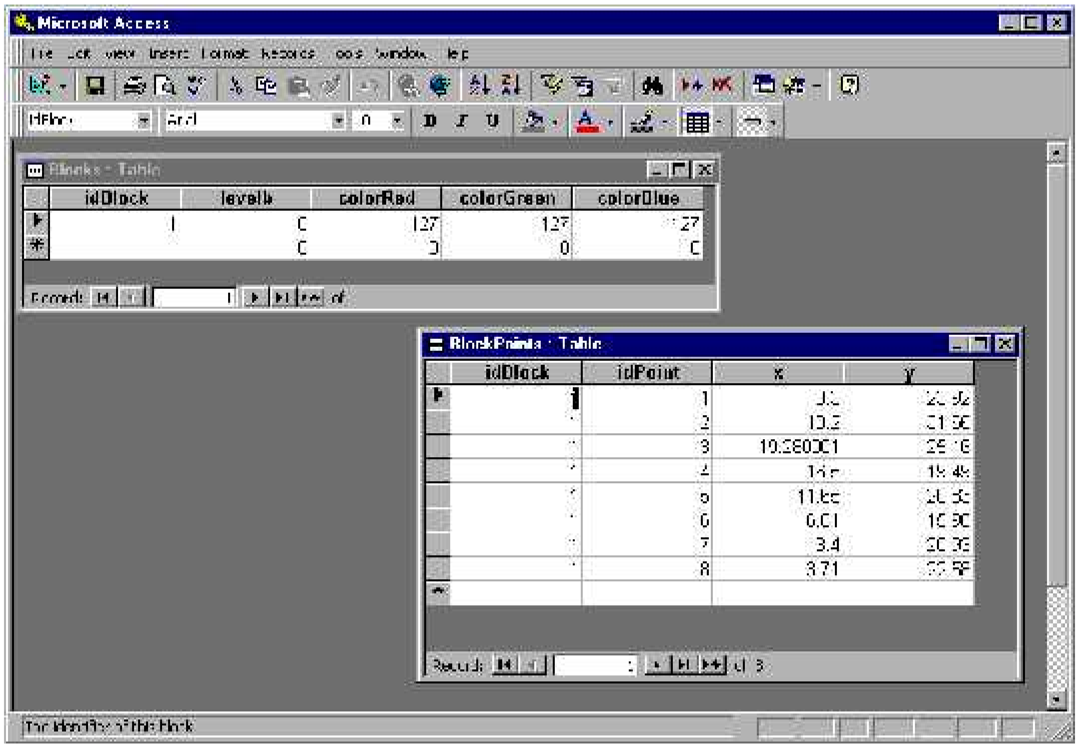
Figure 2-2 : Simple block drawn in the traffic editor


Blocks - Notepad

File Edit Search Help

```
* File of background elements (blocks) specification
* Number of blocks
1
* Block
3 0   127 127 127    8
           7.63000  20.21000
           12.79000  23.41000
           18.67000  22.79000
           18.77000  17.53000
           14.64000  15.68000
           12.99000  18.36000
           8.35000  16.71000
           7.22000  18.67000
```

Figure 2-3: Representation of the block in Figure 2-2 in ASCII Network form



Blocks : Table

idBlock	levelN	colorRed	colorGreen	colorBlue
1	0	127	127	127
	0	0	0	0

BlockPoints : Table

idBlock	idPoint	x	y
1	1	[illegible]	[illegible]
1	2	10.2	[illegible]
1	3	10.280001	[illegible]
1	4	[illegible]	19.49
1	5	[illegible]	[illegible]
1	6	6.01	[illegible]
1	7	3.4	[illegible]
1	8	3.71	[illegible]

Figure 2-4: Representation of the block in Figure 2-2 in database form.

The most complex file is Sections. This file contains all the information about sections including meterings, detectors, turnings, rights, capacity, speed, elevation, and other geometry information. Detailed information on the table SECTIONS and other tables can be found in Appendix A.

2.3.3 Relationships between tables

It is not enough just to define tables. To use a relational database efficiently, relationships between the tables must be defined. Usually defining these relationships occurs naturally from the way that the tables were derived.

For example, in the Blocks example there is a natural relationship between blocks and blockpoints. Each block that is displayed to the user is a record in the blocks table while outline of the block itself is a set of points contained in the table blockpoints with a field linked to the block table.

Figure 2-5 is a picture with all the database tables defined and all the relationships between them. The lines between tables indicate a relationship between tables. Each end of a relationship line has either a 1 or an ¥ indicating the type of relationship: 1-1, 1-many, or many-many. These relationships are described in detail in Appendix A.

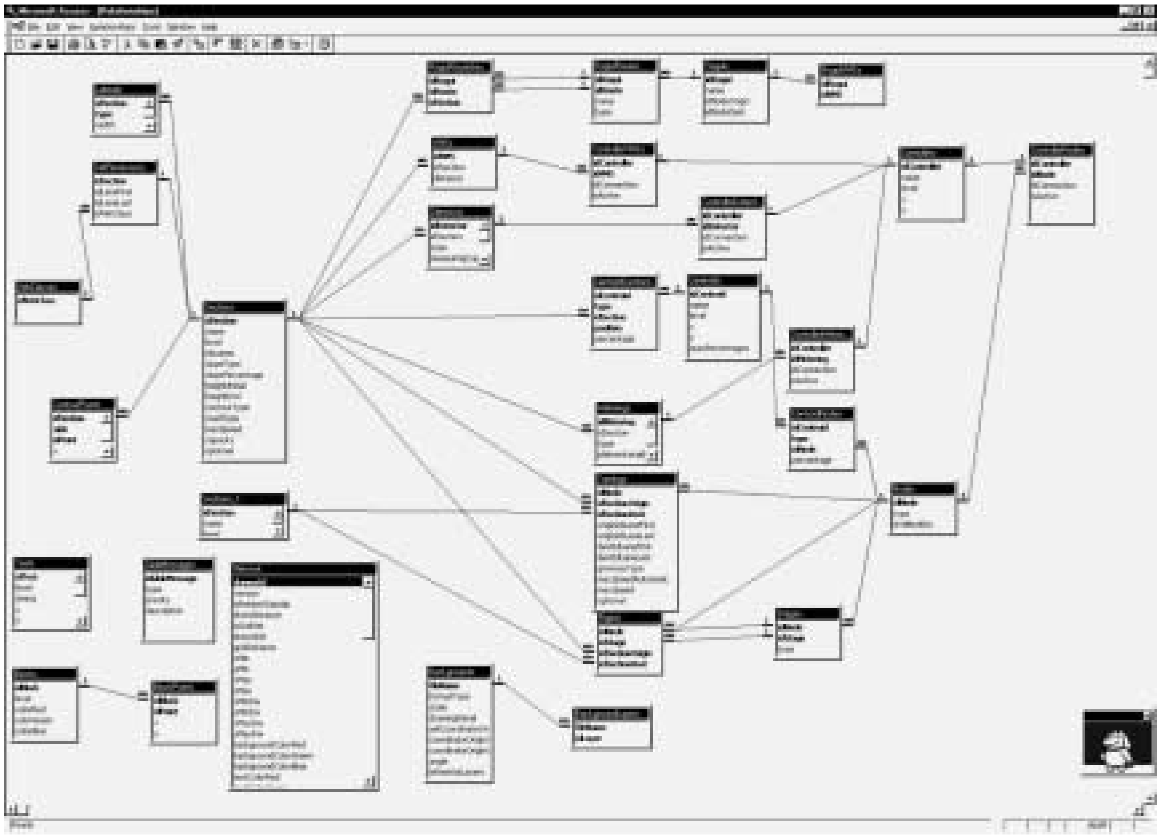


Figure 2-5: Relational database with tables and relationships

2.4 Implementation

This design of a database required the introduction of approximately 105 low level functions that will interact with the database during implementation. The functions are grouped into classes, which are similar to the ASCII network version of these functions. All the functions will be defined and implemented in the API of the system. They will be transparent to any user of the system. All the details of the function definitions are contained in Appendix B. In this section we give a basic overview of the functions and what they do.

2.4.1 GDC Prototype Implementation

The geometry data container (GDC) is implementation of the original geometry system done in ASCII Files, but redesigned to make it functional in a relational database format. This redesign of functions in the input/output portion of the traffic simulation system (TSS) is not a trivial rewriting of code. Our initial design of the database includes more than 105 completely new functions at the lowest level. Each function accesses the relational database system for either retrieval, updating, or saving of geometry or geometry-like information. These functions can be grouped into functional categories that operate on specific parts of the geometry. The functional classes correspond to their analogs in the original ASCII version of the geometry and are defined as follows:

• Blocks:	blocks help in visualizing the area being simulated
• Texts:	text used to identify objects being viewed

• Vehicle Classes:	vehicle class definitions such as HOV->TRUCKS+BUSES
• Global Messages:	global messages for drivers
• Sections:	geometry information on sections of roadways
• Nodes:	nodes link sections to form networks
• Controllers:	controllers link meters and detectors to nodes/sections
• Centroids:	used in origin/destination and route selection
• Roguis:	used in origin/destination and route selection
• Routes:	used in origin/destination and route selection
• Network:	general network characteristics

In order to explain the implementation, it is necessary to understand not only the overall structure of the classes listed above, but the individual functions that need to be implemented for the relational database to function. In the following, we will explain how each category is used in the implementation and a definition and explanation of each function that needs to be implemented. More than 105 new functions had to be designed and implemented to make the GDC fully operational.

2.4.2 Blocks

Blocks are background objects (polygons) used to help visualize the traffic network. Blocks are polygons that can have shape and color. For example, blocks can be used visualize physical features such as rivers, buildings, and other like scenes. In figure 1, a block is drawn using high level functions that ultimately call the low level Block functions described below.

Implementing Blocks in the relational database requires defining eight new functions. In order to understand the implementation, it is necessary to explain each function, its parameters, and what action that it should perform on the database Appendix B defines these functions in detail.

2.4.3 Texts

Texts are background texts, which are used to identify parts of the traffic network. Texts can be used to identify scenes such as rivers, buildings, other scenes, as well as parts of the traffic network. They are not used directly in simulation.

2.4.4 Vehicle Classes

Vehicle Classes are groupings of types of vehicles. For example, one might define an HOV1 (High Occupancy Vehicle) class with taxis and buses, and another, HOV2, with trucks and large trucks. Now lanes could be reserved in sections just for HOV1 and HOV2 type vehicles and no others.

2.4.5 Global Messages

Global Messages are messages that are visible on either variable message signs, or changeable message signs that are used on modern freeways. Each section can contain a variable message sign with the potential messages and potential actions by drivers once the message sign is activated. For example, if freeway traffic is heavy then the sign may be set to read Congestion Ahead, Use Alternate Routes.   Once the sign is activated then a potential action would be for 10% of the drivers to take the next exit.

2.4.6 Sections

Sections contain the basic geometry of the network. A network itself is composed of a set of sections linked together by nodes. The sections themselves contain information about the physical characteristics of each section in the network such as number of lanes, lane length, lane width, capacity, maximum speed, type of roadway, slope, and contour.

2.4.7 Nodes

Nodes are the links that form an actual network. Nodes link the sections together to form networks. In general, there are two type of nodes: junctions, and junctures. Junctions are links between sections that have driver selectable turnings, and junctures are links between sections that have no driver selectable turnings no intersections.

2.4.8 Controllers

Controllers connect detectors and meters for traffic counting and controlled entry. They provide a way to implement controls plans for a network. Any information that is received from detectors or sent to meterings and variable message signs is essentially routed through a controller.

2.4.9 Centroids, Roguis, and Routes

Centroids, roguis, and routes are used by the TSS system to help determine traffic flow given origin/destination matrices.

2.4.10 Network

Network contains the basic information about the network such as type, the default values for different types of structures. It also includes indices for background files that can be traced over to produce a network.

2.4.11 ODBC Functions

ODBC functions represent a new set of functions that perform all the storing/retrieval to and from the relational database. Any of the other functions defined above must call these functions in order to communicate with the database.

2.5 I35WI94 Twin City Freeway Geometry

In order to test our tools, we chose a forty mile section of I35W from I694 to I494 and the intersection of I35W and I94 for about three miles. Figure 2-6 shows the zoomed out view of this segment of I35W in the traffic editor. This set of sections should be enough geometry to sufficiently test the simulation modeling, and the tools that we are building. In addition, building this section of the Twin Cities traffic network should give us a good way to measure the time and effort that it takes to enter other traffic networks of similar complexity.

The software that we have chosen has a traffic network editor that allows visual input of the network geometry. This greatly simplifies the entry, and the accuracy of the data entered. The general plan of attack for entering a new network is: get a background image of the network, trace over the sections in the network, add additional detail, add detectors and controllers, and save the resulting network. Finally, we discuss some of the problems that we encountered and how we solved them. Each of these tasks is quite a bit more complex than it seems. For example, entering a cloverleaf took much more time to complete than entering a straight ramp. The following sections describe the effort in entering a section of freeway.

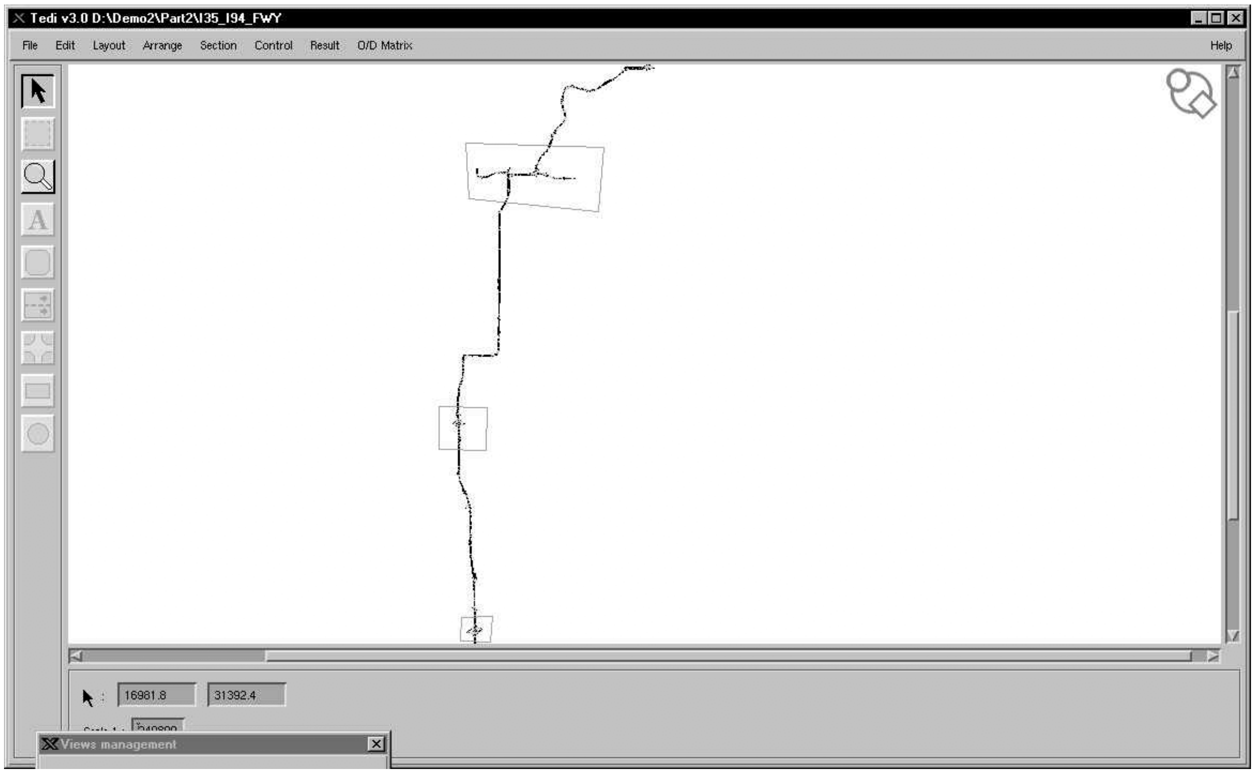


Figure 2-6: I35W from I694—I494 view from within the Traffic Editor

2.5.1 Background Image of the Network

A background image of the I35W/I94 freeway was provided to our research group by MN/DOT. The image was done in Microstation, a CAD program that Mn/DOT uses for keeping track of the freeway system. The image contained much more information than we needed. The image, figure 2-7, that we received contained the entire network of state aid roads for the Twin Cities. The size of the file was 54MB and took thirty minutes to load on a Silicon Graphics Workstation using the traffic network editor. Because the file contained more than what we needed and because the loading time was so significant we edited out essentially all the details except for the freeway sections of interest.

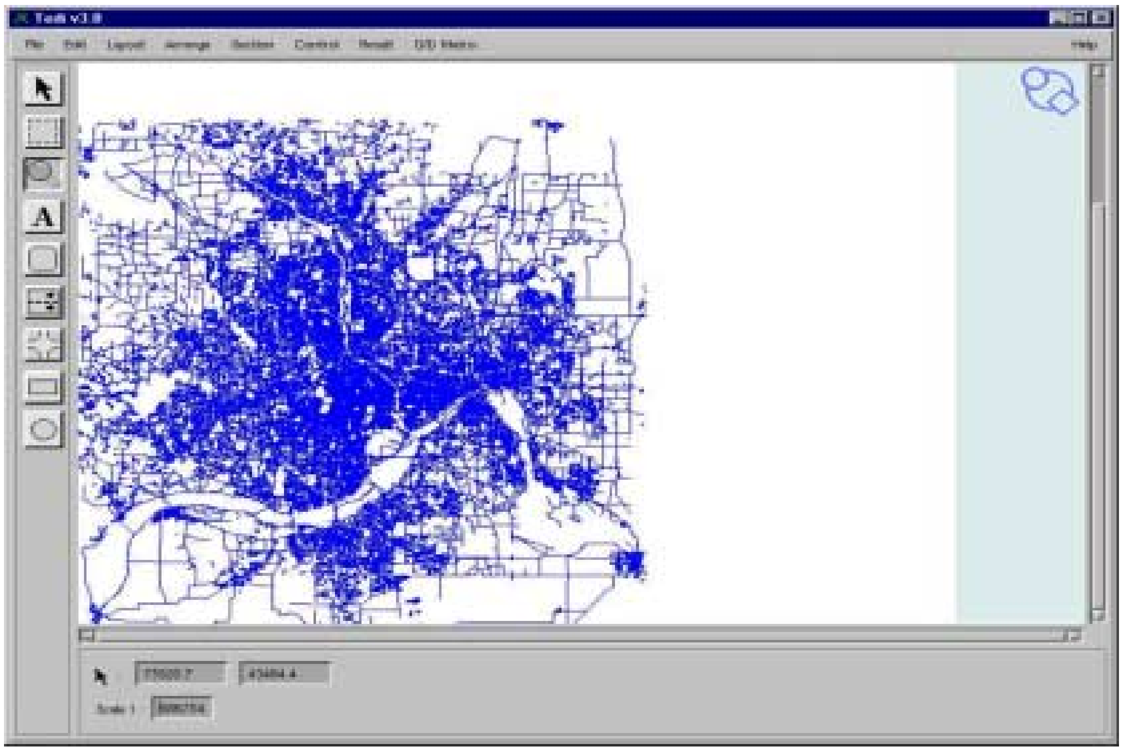


Figure 2-7: Background map of Twin Cities roadways (GIS view)

This new file was much smaller, about 3MB and took less than one minute to load. The background file is only needed for the initial entry of the geometry data explained in the next step. In figure 2-8, the difference in the number of details is visible. The background image only provides a centerline drawing of the freeway. Additional detail, such as number of lanes, turnings, detectors, and controllers, is provided by the ASB files, which were also provided by Mn/DOT. This background image and the ASB files are used to essentially trace out a geometry for the network.

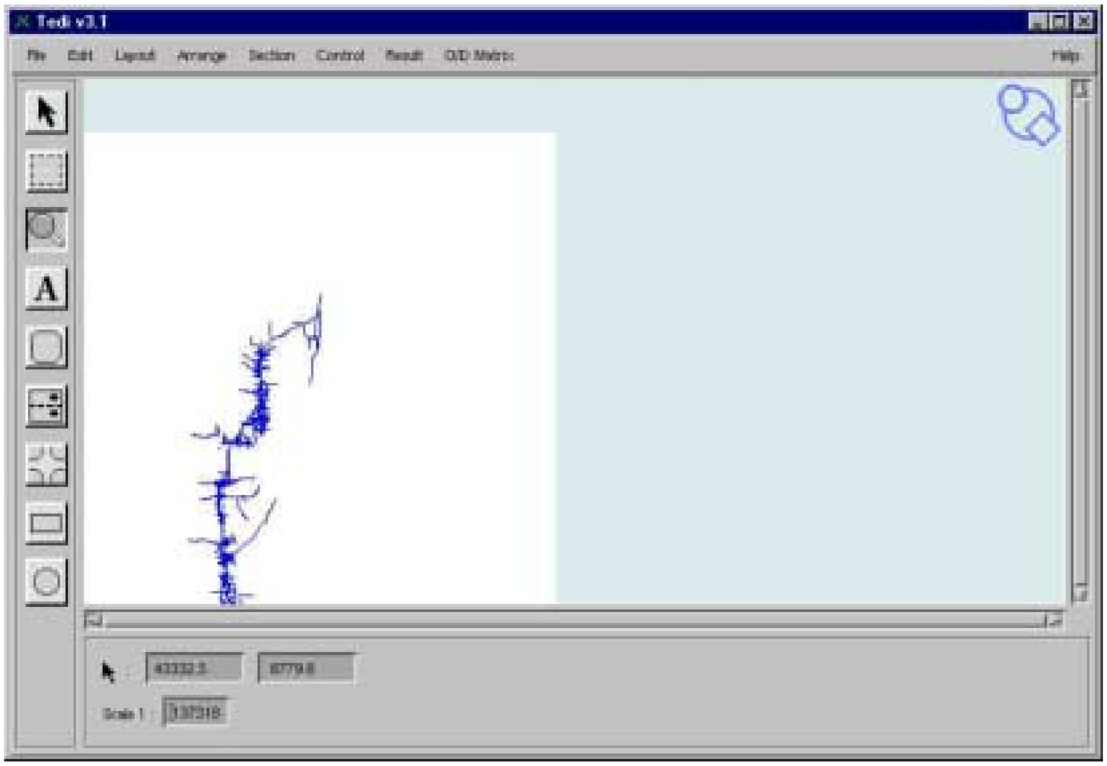
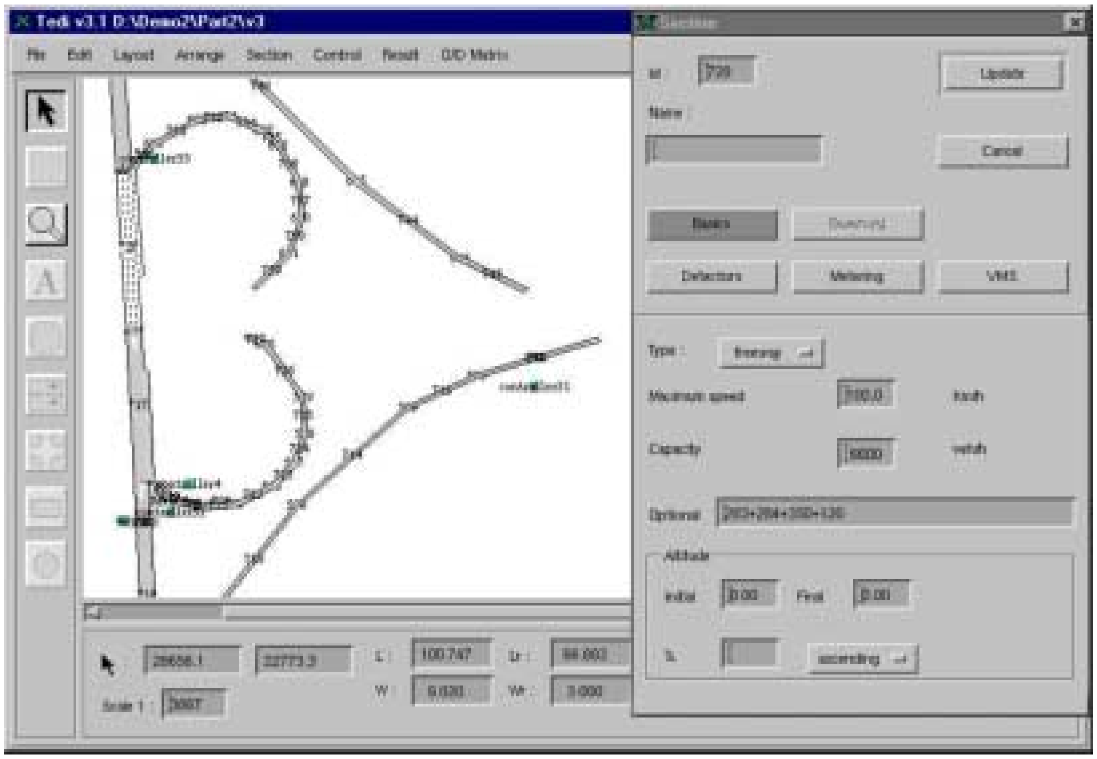


Figure 2-8: Simplified Map with much less detail (essentially I35W)

2.5.2 Trace over the Sections in the Network

Using the traffic network editor, the background image is opened and registered. Now we just start from the top and begin drawing freeway sections over the centerline image and adjust the detail to what the ASB files describe. For example, a section with 4 lanes , no turnings, no detectors, and no controllers. This drawing of sections is very tedious and time consuming, but it is visually accurate, and only needs to be done once. Drawing it with other systems would require the entering of coordinates for each section this would even be more effort.

Actually, drawing the sections only becomes the most difficult with cloverleaves because these roadways have very sharp turning curves. The system that we are using has only straight sections for modeling networks, but sections can be angled where they meet other sections to produce curves. The result is that a single curve in a cloverleaf may be composed of 10-20 sections, and a full cloverleaf may contain 40-80 sections. Figure 2-9 shows a partial cloverleaf that was entered with the traffic editor, the numbers on the curves are the section identifiers.



Id: 720
Name:
Type: freeway
Maximum speed 100.0 km/h
Capacity 6000 veh/h
Optional 203+204+200+120
Altitude
Initial 0.00 Final 0.00
%: ascending
L: 100.747 Lr: 98.003
W: 9.020 Wr: 3.000
29656.1 22773.3
Scale 1: 3887

Figure 2-9: Cloverleaf view from within the traffic editor

Freeway sections need to be entered for each direction vehicles are to travel in because each section in the network can only provide for traffic flow in one direction.

2.5.3 Add additional detail

Once the basic geometry of the freeway is entered, the other details of the network geometry are added. This information is available from the ASB files such as section capacity, maximum speed, type of roadway, level, and slope. All these parameters help to determine the actual flow in a section based purely on the geometry. Although the network appears visually, all the detail parameters used here are entered numerically. They provide information to the simulator so that it can calculate acceleration, deceleration, and speed for vehicles in a section. In addition, they also can be used to help calculate pollution caused by the vehicles used during simulation.

2.5.4 Add meters, detectors, and controllers

Now that all the geometry has been entered, meters, detectors, and controllers can be added. Again these devices are attached to each section in the geometry. Meters are classified by type (green-time, delay, and flow) and are usually connected to a controller. Detectors are also classified by type (inductance loop, pneumatic tube, coaxial cable, tape switch, and queue-length). In our example, we used green time for meters, and inductance loops for detectors. Detectors also had to have the correct identification so that when volume information is requested from one of the Mn/DOT traffic volume databases the correct detector volume information is transferred. This correspondence will be important when using the tool to automatically create the simulation states for a simulation. Mn/DOT provided the detector location file so that the correct detector identifiers were correct.

2.5.5 Save the resulting network

Once all the geometry information, along with metering and detector information is entered, the next step is to name and save the network. Actually, after entering several sections it's a good idea to save the work. There were times when entering section data that the system became inoperable and all the work was lost since the last save. So, as in using all software it is prudent to save often and keep multiple versions so that major system crashes do not cause loss of much data and effort.

2.5.6 Problems and solutions

Generally all the problems that we had could be classified into two types: underestimating of the scope of work, and deciding how to model freeway characteristics that are not overtly in the model.

Our original intent was to model the entire Twin Cities freeway network because with a graphical editor this would be fairly fast we thought. Entering the freeways with all the detail that we have took about three hours per mile of freeway. See figure 2-4 for how complex some freeways can be.

Some freeway sections merged from four lanes into three lanes. This particular characteristic was not directly available in the model, but our solution was to model two sections: one with four lanes and the other with three lanes with an intersection connecting them. The only additional detail that had to be done was to determine the correct turnings for vehicles moving from one section to the other.

The simulation system that we use has very many primitive structures that can be used to create complex systems, but this requires a good knowledge of the system in order to come up with a solution.

3 Sensor Automation Development

3.1 Objectives

The primary objective is to develop a program that takes as input network geometry, traffic volume data, and user specified simulation times to generate the simulation input files. That is, reduce the amount of time it takes to prepare data for a simulation. In the phase I report, it was found that input file preparation could take anywhere from one to twenty days, but in most cases it took only one day. The AST tool described here can reduce that day to a few minutes. This time saving should help make simulation a process that can be used more often and with much less effort.

3.2 What we need

In order to automate the simulation, the following need to be developed:

- Determine user input for geometry selection (point and click)
- Modify traffic editor to save subnetworks
- Modify/augment geometry database to include flows (traffic volumes)
- Define user input for traffic volume selection
- Design an algorithm to add traffic volume to each entrance section for each simulation state.

3.2.1 Determine user input for geometry selection (point and click) and modify traffic editor to save subnetworks.

It is assumed that the geometry network is either the complete network to be simulated or a superset of the network to be simulated. If the entire network is to be simulated then no modification of the geometry database is necessary.

On the other hand, if the network to be simulated is a subset of the current network then the user of the traffic editor must somehow select it. How is this to be accomplished? Well, the original version of the traffic editor allowed users of it to make certain views of the whole network. These views showed only the area selected, not the whole network. Unfortunately, the views still held all the information in memory and did not give up any space. Views are generally created by using a mouse to draw a polygon around the subnetwork of interest. We modified the traffic editor so that a view could be created and then saved as a complete network. In addition, the memory that was not in the saved view was released making the size of the geometry smaller.

Using the view for selection of a subnetwork and saving this view as a new network we now have the complete subnetwork (network) to work with. In our design, the AST tool works with the entire geometry presented to it. Therefore, it is necessary to save only that part of the network that is needed for simulation.

In the selection of a subnetwork, it does not matter whether the network is a relational database or an ASCII network. The AST tool uses the Application Programming Interface (API) to access the geometry. The API itself actually makes the connection to the network. Figure 3-1 shows essentially how the communication to the geometry is made. If you look closely, it does not matter what form the geometry is stored in. The detail of whether the true underlying data are stored in a database or an ASCII network is hidden from the user. There is no need for the user to understand all the details just to do a simulation. A normal user of this system will work at the level of a traffic engineer and work with objects that he will understand, not fine details that make the program work.

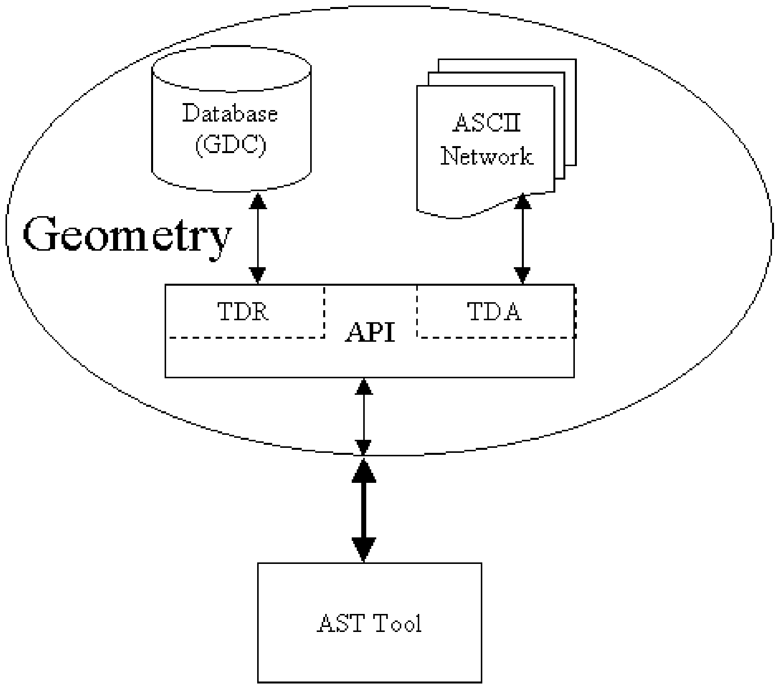


Figure 3-1: View of the AST accessing the network geometry

3.2.2   Modify/augment geometry database to include flows.

In order to simulate the network saved in the previous step, the geometry database had to be augmented because the saved network did not preserve all the detector flow information. That is, some sections may be entrance (boundary) sections, but they may not be associated with a detector. Since detectors essentially give the flows for this model, no detector information would be provided to these sections and the simulation states could not be constructed.

A simple solution to this problem was to determine by hand which upstream detector(s) provided flow information for each section. This information was then added to the optional input field for each section header. In entering the flows, there are two cases that could exist: only entrance detectors, mixed entrance and exit detectors. Entrance detectors are handled by encoding the detectors with plus signs between them. Mixed detectors are encoded with plus or minus signs depending on whether the detector is at an entrance or exit. Entering this extra information is just a small additional task that can be completed at the same time the detectors are input into the geometry.

3.2.3 Define user input for traffic volume selection

Provide an input menu for the user to make selections. The selections should include the traffic database, the date and a range of times for traffic volume data. An Open DataBase Connectivity (ODBC) compliant database is used to store traffic volume data collected by Mn/DOT. The traffic volume data to be retrieved will be selected from this database by detector number, date, and time. The detectors do not discriminate among the different types of vehicles on the freeway: only a simple vehicle count is given. Although the simulator being used can handle several different vehicle modalities (for example, car, truck, bus), currently only a simple vehicle count is used.

3.2.4 Design an algorithm to add traffic volume to each entrance section for each simulation state

The selection of traffic volumes for each entrance section in the selected network is no trivial task. Both the traffic editor and the micro simulator have a very good application programming interface (API) connecting them to the geometry data. The API provides the programmer with access to any element that exists in the geometry.

There are API functions that determine which sections are entrance sections to the selected network. Each time an entrance section is found, then a special field encoded in the section is read. The information read lists the detectors that determine the flow for this section. If more than one detector is listed then there must be a + or a - between them indicating whether the value given by the detector is to be added or subtracted from the flow. A subtraction would occur if there were an exit ramp in this section.

Once the entrance sections and the flows are determined, the simulation states can be defined. Simulation states are the time slices that provide new input for each time period in the simulation. Producing these simulation states is the main purpose of the AST program.

In generating simulation states, for each time slice (for example, 5 minute period) the special encoded field is read from each entrance section (for example, 5+8-10), the flows are calculated from the detectors listed here, and the net flow is written to this simulation state section. These steps are repeated for each entrance section in the network. Now a simulation state file can be saved for this time slice. Next repeat the same process for the next time slice, and then repeat until all the simulation states have been computed. All these files created are stored in the same directory. Collectively, they are called the simulation states. The file extension on all the files created is .stt , indicating system states. These files are exactly what the micro simulator needs to perform a simulation. Without the AST, this complete process would need to be done manually via the traffic editor interface.

3.3 Putting it all together

Combining the geometry, the traffic volume data, and user specified simulation times a sequence of simulation state files is generated. These files will be input for the microsimulator for the traffic network. Figure 3-2, depicts the architecture of the AST tool.

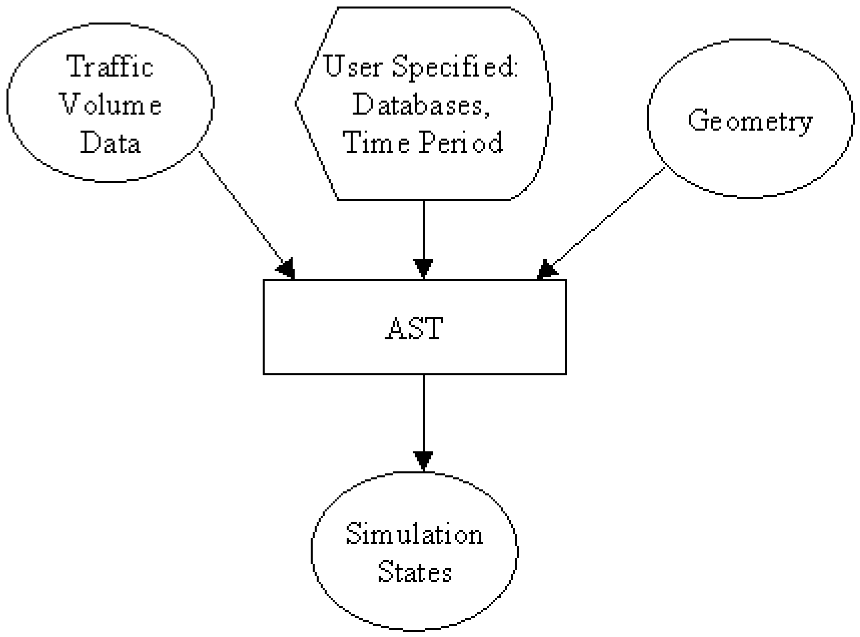


Figure 3-2: The AST with inputs and the AST generated simulation states

3.4 Making a Simulation Run

Finally all the information that is needed for simulation is ready. The simulation states have been generated (and traffic volumes), the geometry is set, a control plan has been defined. The next section will describe how to use the AST tool and then run an actual simulation.

4 Sensor Data Automation Use

In using the Automated Simulation Tool (AST), essentially all manual data entry is eliminated. The AST takes as input the traffic geometry, a user specified time period, and the traffic volumes to generate the simulation states (files) needed for simulation. Figure 4-1 gives a dataflow view of this process. All the real work collecting detector data, and entering a geometry has already been completed. The only real input required is the time period for the simulation states to be generated.

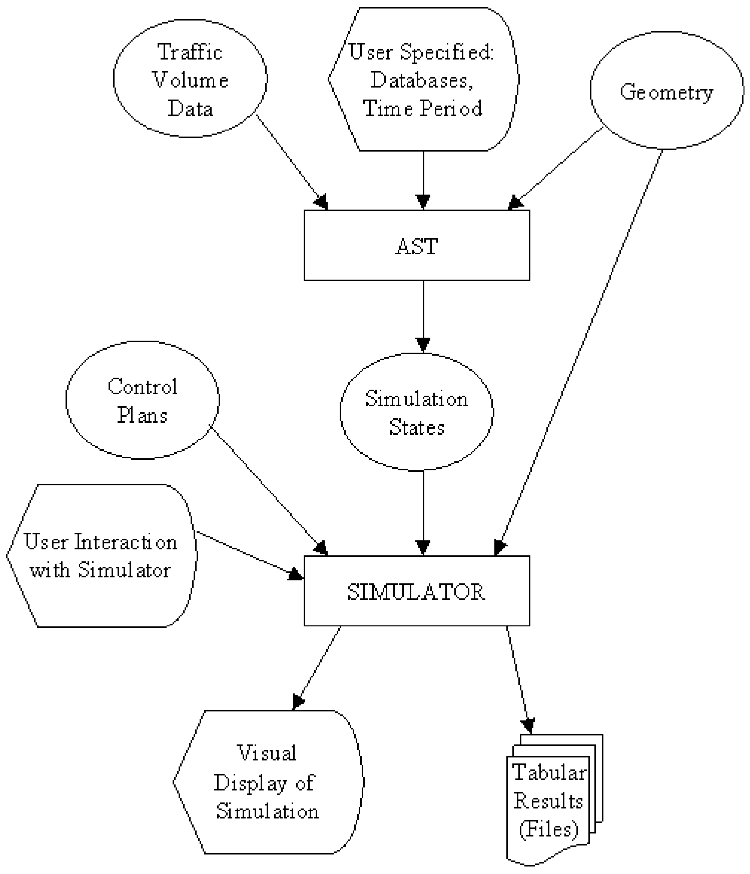

Traffic Volume Data
User Specified: Databases, Time Period
Geometry
AST
Control Plans
Simulation States
User Interaction with Simulator
SIMULATOR
Visual Display of Simulation
Tabular Results (Files)

Figure 4-1 Overview of the AST, the Simulator, their inputs, and the results

4.1 Objectives

In general, our objectives were to automate the simulation process so that a simulation could be performed within minutes of the selection of an existing geometry.

- Visually (with a mouse) select the (sub)network to be simulated
- Eliminate the need for any manual entry of traffic network geometry (other than a one time entry)
- Automatically retrieve traffic volume data for the selected network from a traffic volume database
- Automatically generate the necessary simulation states using the selected network and the traffic volumes

4.1.1 Visually (with a mouse) select the network to be simulated.

Once the complete network is entered, we may be interested in simulating just a portion or a subnetwork of the original network. The traffic network editor was modified so that one could use a mouse to outline only that part of the network of interest and save it as a new network. This new network then has only the subnetwork selected, none of the other information is saved relating to parts of the original network. This is precisely why we encoded flows in the optional field in the previous section so flow information would not be lost by selecting a subnetwork.

Figure 4-2 shows an original network with one area highlighted (rectangle around). This rectangle represents the subnetwork that we have defined. Figure 4-3, shows the view after the subnetwork is saved as a new network. Notice that the optional field contains 120+283+284+350. This list indicates which detectors are needed to generate the correct flow for section 717.

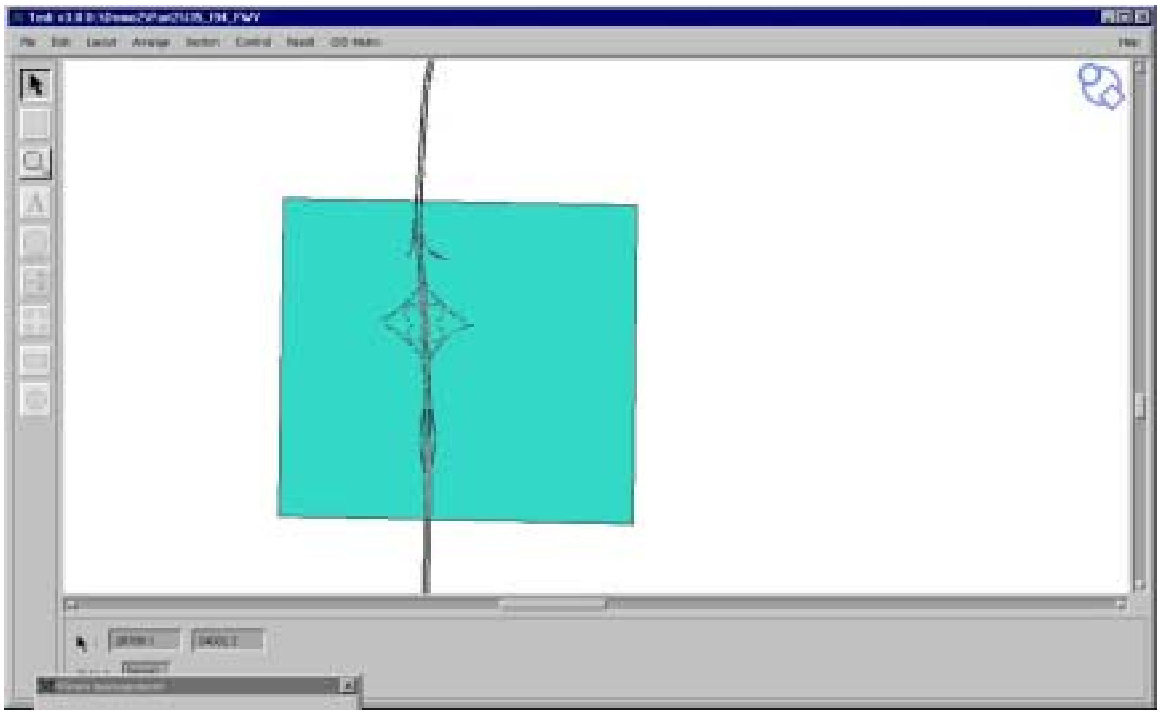


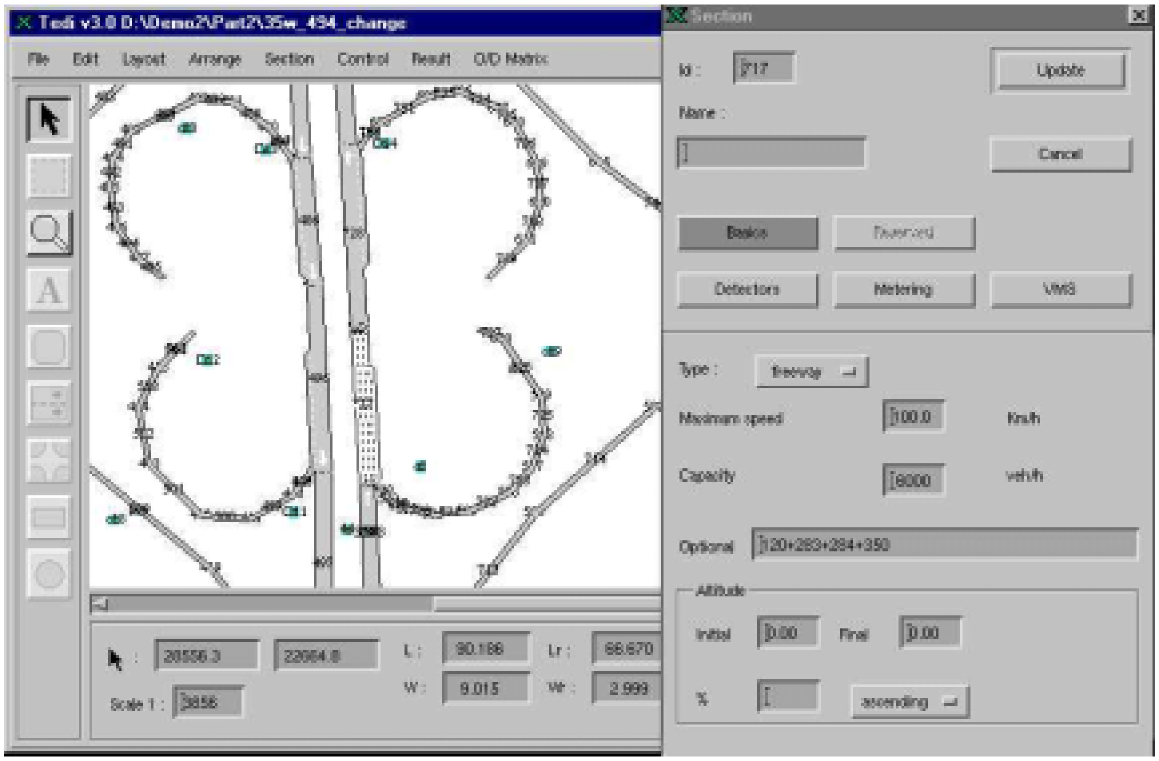
Figure 4-2 I35W at I494, subnetwork selected in gray

4.1.2 Eliminate the need for any manual entry of traffic network geometry (other than a one time entry).

One of the most time consuming activities is entering the traffic network itself. In the previous section, we have already created a traffic network that is part of the Twin Cities freeways so that the only additional pieces of information that are needed for the geometry are the list of entrance sections and their flows. We can encode this into the original geometry by determining which detectors give the flow for each section. These additions need to be done only once for each geometry.

In figure 4-4, the detectors in the first section are numbered 8, 9, and 10. Notice that the optional field to the right of the section contains 8+9+10. This is the additional information that is encoded in section one indicating which detectors determine the flow for it. It implies that the total flow for this section is completely determined by summing the flows for detectors 8, 9, and 10.

If we extend this concept to the next section (section two), then the optional field here will be 8+9+10+11. Detector eleven is included because there is an entrance ramp with detector eleven on it. So by extending this idea to all sections, we can select any subnetwork and be able to determine the flow for each section by viewing the optional field.


Tedi v3.0 D:\Demo2\Part2\35w_494_change

Section

Id : 717

Name :

Type : freeway

Maximum speed 100.0 Km/h

Capacity 8000 veh/h

Optional 120+283+284+350

Altitude

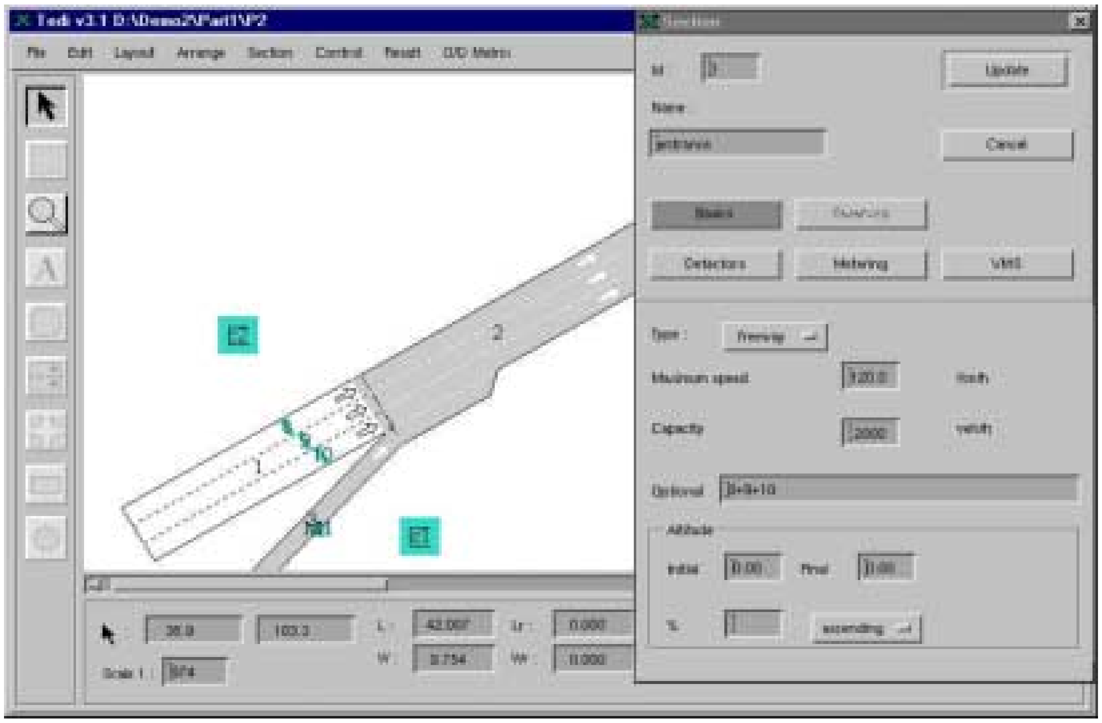
Initial 0.00 Final 0.00

% ascending

28556.3 22064.8 L : 90.188 Lr : 66.670

Scale 1 : 3856 W : 9.015 Wr : 2.999

Figure 4-3: Section 717 at I35W-I494 cloverleaf. Section 717 has no detectors in it.


Tedi v3.1 D:\Demo2\Part1\P2

Section

Id 3

Name entrance

Type : Freeway

Maximum speed 120.0 Km/h

Capacity 2000 veh/h

Optional 8+9+10

Altitude

Initial 0.00 Final 0.00

% ascending

36.9 103.3 L : 42.007 Lr : 0.000

Scale 1 : 974 W : 3.734 Wr : 0.000

Figure 4-4: Freeway section with detectors, meters, and controllers

4.1.3 Automatically retrieve traffic volume data for the selected network from the traffic volumes.

Our automated tool is designed to gather all the traffic volumes for a specified time period for the set of detectors generating entrance flows for the network to be simulated. The entrance flows are easily determined by making a special call to an API function and reading a special optional field which was encoded with a list of detectors that determine the flow for that section (see figure 4-4).

4.1.4 Automatically generate the necessary simulation states using the selected network and the traffic volumes.

For each simulation state, detector information is provided from the traffic volume database. The data may need to be adjusted to reflect vehicles per hour flows rather than sampling period flows. The algorithm described in section 3 describes how the simulation states are generated. For the user, generation of the simulation states is completely automatic and should not have to worry about any details.

4.2 Specification of data requirements for simulation

In order to do an automated simulation the following components are needed:

- Geometry network selected and saved
- Traffic volume database
- User selection network, date, and time periods to simulate

- Control plan for the selected geometry
- Simulation states generated by the AST tool

4.2.1 Geometry selected and saved

Simulation of a particular geometry can be done quickly and easily by opening a previously stored network with the traffic network editor, selecting an area (subnetwork) for simulation, and then saving this selected area as a new network. If the entire network is to be used then no subnetwork selection is necessary just use the entire network.

4.2.2 Traffic volume database

The traffic volume database for the time period that simulation is to be done for must be available. Usually Mn/DOT can provide this information. To use the data it must be imported into an ACCESS (or other ODBC compliant database) with the following columns as described in Appendix D. The next step is to register the database with the ODBC32 Administration program in Windows NT/95. The name of the database should be LOOPDATA. This name can be changed later, but the current program will look for a database named LOOPDATA.

Once the database is registered then it becomes available for any ODBC compliant program to retrieve and store data in. That is, any of the simulation tools that we designed can make use of it.

4.2.3 User selection network, date, and time periods to simulate

The user of the AST program can select any existing network (previously saved) for the geometry, and the date, start, and end times for detector data. Any detector data that cannot be found is defaulted to some preset value and a message is written to the user s screen stating that no detector data was found. In our experience, detector data is not always available for all the detectors in a particular network so substituting some data helps to complete the simulation. In the future, this addition of missing data can be made more intelligent, but a quick fix is just to substitute in a fixed value.

4.2.4 Control plan for the selected geometry

The microsimulator that is being used must have a control plan to control the rate at which vehicles enter the freeway. Otherwise, any vehicle on a ramp enters the freeway at the average speed without any waiting. This effect is not desirable and therefore a control plan must be entered.

4.2.5 Simulation states generated by the AST tool

The simulation states (time slices) for each time period for each section are generated by the AST tool with the information gathered from the user, the geometry and the detector values. Each state generated is written to a file and contains information on volumes for the entrance sections. For example, if the traffic engineer wants to simulate the traffic on the network from 0600 to 0900 with 5-minute data then there will be 12*3 or 36 state files generated for this simulation.

In our simulation, all the simulation state files are saved in a folder (directory) called a results container. The results container itself can have any name, but it makes sense to choose a meaningful one. Later when a simulation network is being loaded a particular results container will need to be specified in order to run the simulation.

5 Framework for Phase III work

In this phase of the project, we automated the geometry entry and generation of simulation states. In the next phase, we will automate the control plans and the results output by the simulator.

- Automate control plans
- One time geometry entry: including all ramps, detectors, etc.
- Automatic selection of detector data from a traffic database that is ODBC compliant
- Automatic generation of simulation states (more fully integrated)
- Generic/Integrated geometry for both a macro/micro simulator
- Real-time access to Mn/DOT detector data for simulation

5.1 Automation of control plans

Currently control plans are essentially fixed in this version of the simulator. That is, once a network is opened and a control plan is selected then no other control plan can be opened for this network. Our plans call for automating control plans so that multiple control plans can be used during any one simulation. In addition, we would implement Mn/DOTs control plan so that it could be used in the field.

Automated control plans will allow the simulator to dynamically modify the ramp control for each simulation state. This process will allow the traffic engineer to build algorithms based on current traffic flow to control ramp meters, not on metering based on historical data or specific time periods during the day. The development of automated control plans will be an important step on the way to real-time traffic management for freeways.

5.2 Automation of results presentation

Once the simulation is completed, it would be good to automatically see the results of the simulation visually. Not just simple X-Y graphics, but some kind of visualization that makes traffic congestion easily standout. Currently, the simulation data can be saved in spreadsheet style files that can later be used by other programs to generate graphs and perform statistical analyses. None of these graphic views or analyses are integrated with the simulator. Our goal will be to integrate these features within one graphical user interface (GUI) so that a traffic engineer could run a simulation and immediately view the results with just a few menu selections. Some research will need to be done to define exactly what type of visualization would be helpful to traffic engineers.

5.3 Integrating another simulator with the GDC

Integrating a macroscopic simulator so that it can make use of the GDC. Currently only a microscopic simulator works with the GDC. Maintaining only one GDC (geometry) will eliminate the cost of generating a special geometry for a macro simulator. In addition, more effort can be put into keeping the GDC up to date allowing more researchers to actually work on traffic simulation and analysis not geometry or data entry.

The ITS Laboratory at the University of Minnesota is continuing to develop the macro simulator: KRONOS. Since this development is being done in the lab, we have access to the developers and the source code, so that GDC geometry can be augmented with additional information to generate macro simulator geometries from the macro geometry currently in the GDC.

5.4 Real-time access to Mn/DOT detector data for simulation

Finally, real-time access to Mn/DOT detector data from their data feed for real-time simulation. In another project (Activation of the I394 Lab), a prototype was built that received real-time data from Mn/DOT and stored it in a file system.

6 Conclusions and Recommendations

6.1 Problems that we encountered

6.1.1 Problems with the GDC

- Debugging the system
- Determining which functions to test first
- Finding internal bugs
- Defining a test suite to determine if things are done correctly
- Real-time and deleting an object on the screen will permanently delete it in the database
- Database cannot translate from relational to ascii.

6.1.1.1 Debugging the system

The system that we designed was difficult to debug because we did not foresee the system as large and as complex at it was. We had multiple layers of software to develop and understand. In figure 6-1, the API has at least four levels of software. All the names in figure with an asterisk (*) were new software that had to be implemented. At the TDR level there were more than 105 new functions to design and implement. The TDG functions all had to be modified to make the correct calls to the TDR functions. In addition, the TDI functions on the right side of the figure were not available for use by the TDR function on the left side. TDI functions provide the capability to create and modify internal API objects. They are utility functions.

Since these TDI utility functions did not exist for many TDR functions that we designed, retrieving objects from the GDC was much more difficult. We had to write some of our own translation functions, and debug them without knowing very much about the internal structure of objects in the API. In addition, at the lowest level ODBC was also a problem to work with because the special handles required to use it were difficult to add to the API. In some cases, the macro definitions of the API conflicted with the macro definitions of Microsoft windows. This conflict resulted in errors during compilation.

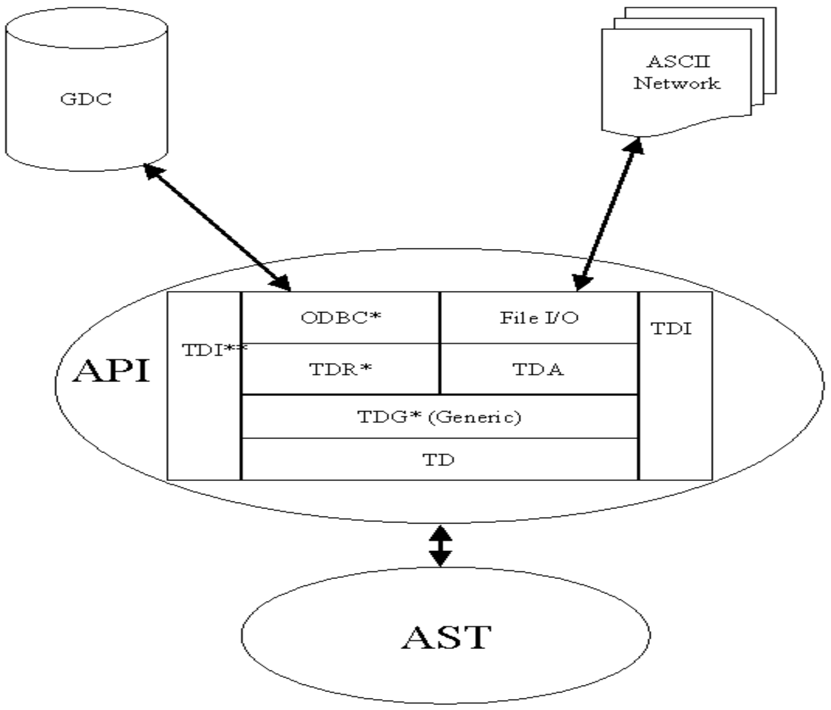


Figure 6-1: Software architecture from the AST to the GDC/ASCII Network

6.1.1.2 Determining which functions to test first

Since there were so many functions to design and test, the most logical step was to start with the network object and functions. Unfortunately, this network object while being one of the more important was not so simple and had many parts to implement before being able to see any results. Of course, nothing else would work without the network object first working. Most of this was done by trial and error because we had no knowledge of the internal working of this environment. In fact, a variable in the ODBC database was named level, which conflicted with some function calls. This was a very difficult error to find, but easy to correct. The name was changed from level to levelb.

Once the network object was finished, the block object was next. Unfortunately, the same problem was encountered here with the name level as in the network object. Basically, there are conflicts in names between what ODBC will accept in calls to the database.

6.1.1.3 Finding internal bugs

In addition to general debugging problems, the software that we were using had additional internal bugs related to the modifications made to call the new TDR functions. These types of errors required much contact with the developers of the simulation suite in order solve these problems. In some cases, solutions were not found.

6.1.1.4 Defining a test suite to determine if things are done correctly

Since the system we were developing was so complex and so many changes were made to the software, it was difficult to come up with a simple plan to test the pieces of the system we were building. Initially, we thought that we could do end to end testing. That is, use the traffic network editor to make a drawing, which is saved as a database, and also as an ASCII network.

6.1.1.5 Real-time and deleting an object on the screen will permanently delete it in the database

An unintended side effect of the implementation of the GDC was the dynamic nature of an object delete. For example, if an object was open in the GDC and being viewed with the network traffic editor, then any change to that object would immediately be a directed change to the GDC. Since the GDC was changed in this operation even closing the network traffic editor without saving it will not preserve or bring back the original object. This could be a significant problem for any user of the system. With the current design this problem cannot be easily fixed. This is also inconsistent with the ASCII network implementation of the system. A user of the system should be able to perform the same operations with the same results independent of the whether the system is to be saved in a GDC or an ASCII network. This feature is a design fault and will be fixed in the next phase of the project.

6.1.1.6 Implementation cannot translate from GCD to ASCII network.

The design of the GDC system would allow automatic translation from GDC to ASCII network and vice versa. Unfortunately, the implementation of the GDC could only translate from GDC to ASCII network, not ASCII network to GDC. Since the software is very complex, it was not possible to completely test the design before implementation. Consequently, there were some internal errors and mismatches that should have been caught earlier, but were not. Part of this problem is related to the number of layers that had to be modified in the software. Refer to figure 6-1 to see the layers of software that were modified.

6.1.2 Problems with the Automated Simulation Tool

Although we were successful in building the AST, we did encounter problems along the way. The following is a list of the major problems that we encountered in designing the tool.

- Many detectors have missing data or do not work
- Sometimes the system fails to find a detector that exists
- Not all flows were computed accurately if turnings exist

6.1.2.1 Many detectors have missing data or do not work

In running the AST to generate the system flows, many times the detector information was missing or invalid. In order to correct for potential missing detectors, we automatically entered a default flow for any missing or invalid detector. Substituting a default value was a good idea, but a default value for a detector cannot be generalized across the whole network. The solution is to get complete and reliable data from the detectors or systems that translate detector information. This is not a job for the AST to complete.

6.1.2.2 Sometimes the system fails to find a detector that exists

In several of our test runs of the AST, some detectors could not be found in the database even though they existed. This was not a programming error on our part; it was an error in the software connection to the traffic volume database. The underlying ODBC drivers had an error that caused some calls to the traffic volume database to not find a detector. This problem was corrected with new drivers for the ACCESS database.

6.1.2.3 Not all flows were computed accurately if turnings exist

When we compute the flows in a network, our algorithm specifies which detectors to select from the traffic volume database and where to put the flows. Unfortunately if a vehicle has choice of a turning in a section then the simulator assumes that turns are of equal probability. In reality, this assumption may not be true. It may be that given a choice of going left or right that a driver more often will choose left. In this case, our flows would not be calculated correctly. A solution to this problem is to compute the flows from the detectors in the left and right turn areas and then adjust the turning probabilities to reflect the actual flows.

6.2 Things that we learned

- Adding a database capability to an application that is not currently using a database can be a very complex project. even more so if the database must be integrated internally into the application. For example, just replacing the input files with database calls is much simpler than modifying the application to use the database as a real-time updating tool.

- Problems in the HW/SW interface may not be apparent until implementation.
- Even though training costs $$, time lost trying to figure out software can be even more costly in $$ than a training class. Training on externally purchased software should be a must.
- Coordination on projects where details are not completely worked out can be a difficult problem to solve. As many details as possible must be nailed down before the project starts.
- Designing a new software system requires thorough planning and diagrams with all the interfaces nailed down before work is started.
- Test plans should be done at design time so that components can be tested
- Choosing ODBC protocol will pay off. Many other software products now have this protocol available.
- GDC implementation does not completely work. Some functions work, but a redesign of the connection to the database needs to be done

6.3 What we would do different, if we had to do it again

Do a more thorough design of the system. Define all the interface components, all the functions that need to be designed, all the levels of abstraction that must be considered, and ensure that the system design is consistent and workable. In our case, we did not do a complete detailed system design. We thought we did, but it was not detailed enough. We did not know what the interfaces would really look like; we did not have a complete detailed paper design to follow. What we missed is that we did not have complete knowledge of the geometry software, we did not know ODBC, we did not know the process for building DLLs in the visual C++ environment, we did not know ACCESS, and the geometry translation from a set of ASCII files to a relational database was not a direct simple translation. All these factors combined together made this task overwhelming. In addition, the database design had a few errors that were difficult to find because of lack of knowledge of SQL and ODBC.

6.4 Major accomplishments

- GDC was designed. On the top the overall GDC looked like a good design, but implementation was not as easy. I think that too many things were changed to make a successful project without checkpoints to ensure that the software would eventually work.
- AST was designed and Implemented
- AST can generate simulation states with flow data from a selected network
- Some portions of the GDC are working, but they have no effect on whether the AST is working or not.
- I35W geometry with all ramps, meterings, and detectors was entered (it took approximately 3 hours/mile of freeway to enter the data).
- Currently an ITS lab user can take any contiguous portion of I35W and do a simulation with either current TMC data (relatively new data, not real-time), and in less than 30 minutes. This is a major step forward in being able to simulate so quickly.

6.5 Conclusions

The completion of the AST essentially completes most of the objectives of this project. The other objectives included partial entry of the Twin Cities Freeway network, and the design and implementation of the GDC. Although the GDC was designed and partly implemented, there were many problems both in the design and implementation of a complete GDC. Portions of the GDC worked, but the complete implementation with all 100+ functions was not completed. Fortunately, a redesign and a new implementation of the GDC have overcome many of these problems. The process of design and implementation of this project and all the problems that we had gave us great insight in how to improve our methodology for completion of the next project phase.

A major result of this project is that any traffic engineer can now select a network for simulation, determine a time period for simulation, and run the simulation without any manual entry of data or re-entry of geometry. This process alone reduces the amount of time that it takes to prepare data for a simulation run, and it also minimizes errors in data entry. Of course, the network must already be in the GDC (either in ASCII or database format), and the detector information must be available.