Turning Movement Estimation in Real Time (TMERT)

Peter T. Martin

Department of Civil Engineering
The University of Utah

California PATH Research Report
UCB-ITS-PRR-95-29

This work was performed as part of the California PATH Program of the University of California, in cooperation with the State of California Business, Transportation, and Housing Agency, Department of Transportation; and the United States Department of Transportation, Federal Highway Administration.

Issue Date: September 1995

ISSN 1055-1425

Turning Movement Estimation in Real Time (TMERT)

Abstract

The dramatic increase in the processing and communicating power of computers, and the recent development of vehicle sensors and detectors, offers exciting opportunities for real time traffic monitoring, management and control. Conventional transportation planning models assume that travel patterns are tangible, stable and predictable, and that movement demands are directly related to the distribution and intensity of land uses. These do not lend themselves to on-line traffic forecasting. This report describes the development a new model that can monitor system performance and derive management and control strategies in real time. The concept of the TMERT (Turning Movement Estimation in Real Time) model, originated in the United Kingdom. It offers a method of estimating turning movement flows from link detected flows at small recurrent intervals, in real time.

California cities need better control and congestion management systems that operate in real time. The development of this model contributes to the management and control of congested urban traffic. It operates from real time detector data and represents the stage of traffic control that operates strategically and above the local level of adaptive signal control. It will contribute, therefore, to the design of advanced traffic management through the provision of more information.

The research reported here shows how the model has been refined with improved data input and analysis of output. The model is shown to be transferable through estimation of turning movements from a real Californian City.

Keywords: real-time, detectors, intelligent transportation systems, adaptive signal control

Executive Summary

Modern adaptive traffic signal control systems rely on a continuous supply of remotely detected traffic data to optimize signal settings. Signal co-ordination is achieved by minimizing a performance measure which is a function of such parameters as delay, journey time or vehicle stops. As a by-product of traffic demand responsive control, a wealth of traffic data is available, on-line for other uses.

Since the signal settings control flow capacity in urban traffic networks, the capacity along any link can be measured in real time. If the turning movement flows at junctions could also be identified, then the spare capacity of routes could be derived in real time. This information would enable alternative re-routing strategies to be defined and implemented in real time to both alleviate congestion and divert traffic following incidents.

It has been shown (Martin and Bell 1991) that algorithms such as NETFLO (Kennington and Helgason 1980) which solve minimum cost problems, can be applied to the modeling of traffic flow. The input to the NETFLO (NETwork FLOw) algorithm included link unit costs, upper and lower bounds on link flows, demand at sources, supply at sinks, and constraints which were governed by the observed link flows. The output consisted of a set of turning movement flows which satisfied the demand and observed link flow constraints and minimized some notion of cost which was embodied in the form of an objective function. Employing Monte Carlo simulation techniques to model traffic flow in a small network representation of a part of Middlesborough (North East England), with a simple set of upper bounds, lower bounds, link costs and detector link flows, the output turning movement flows were compared with those anticipated. The investigation concluded that before significant progress could be made in demonstrating the application of NETFLO, a comprehensive set of real data was required.

Modeling of another English network (Martin and Bell 1992a) showed that the method bore promise. This research applies and develops the model through data drawn from a Californian City. The model uses actual data collected from a real network in the city of San Luis Obispo in Southern

California in February 1994. The NETFLO model is used as the tool to achieve accurate correlation between observed and estimated turning movement flows by constraining the model in a way that represents the state of the control system at the time the link flows were measured, and in such a way that it can be applied to different flow conditions. In this way, the solution derived is system driven, and not user driven. A formal network notation will be presented which will enable any urban street network to be modeled.

In the first section, the role the TMERT model can play in Traffic Control is defined through placing it in its broader context. The development of Adaptive Signal Control systems from early isolated signal control to the demand responsive systems of today is traced. A review of routing models shows how the TMERT model's development relates to other work in the field. The section concludes with an outline of the Research Plan as proposed. The second section details the data collection procedure and sets out how turning movement surveys were devised and implemented. In section 3, the theoretical basis of the model is presented with a detailed description of the Mathematical concepts which support the model. The software development is discussed in section 4 which concludes with the validation of the new code through the testing on the English data. In section 5, the theoretical basis for model improvement is detailed. The results of the improved model are presented in section 6. The report concludes with a critical appraisal of the effectiveness of the research and demonstrates that all the objectives for the one year project have been met.

List of Figures

List of Tables

Table of Contents

Section 1 Introduction

1.1 The Context of the TMERT Model

The supply of urban road facilities can be enlarged through increased urban highway capacity. More roads is one way. Increasing the efficiency of the existing urban highway infrastructure is another. In the four decades since the Second World War, urban road building has been substantial. There is little room left for new urban roads. So the challenge of increasing the supply of road facilities lies with improving the capacity of the existing road network. These measures have become known as traffic management.

Traffic signals at intersections control flow by sharing time between competing streams. When signal installations are co-ordinated, the efficiency of the traffic system is substantially improved. Selby and Powell (1985) reported on the improvements in Southampton after the introduction of a signal co-ordination system. They showed that journey times were reduced by 18% in the morning and 26% in the evenings while delays were reduced by 39% and 48% respectively. Such coordinated systems are referred to Area Traffic Control (ATC).

There are two types of ATC system. Fixed-Time Systems are a set of pre-designed plans while Demand-Responsive Systems adjust dynamically to changing traffic patterns, either through traffic-responsive plan selection or fully traffic-responsive control.

Regular daily changes of traffic characteristics are accommodated by a series of plans. A set of pre-defined plans incorporates a series of co-ordinated signal settings which are called by a command from a central control point at fixed times throughout week days. Settings are fixed within a plan which serves to accommodate a known traffic demand. A traffic signal plan is implemented for a variety of different traffic conditions.

A study tool known as TRANSYT (TRAffic Network StudY Tool, Robertson, 1969) developed by the Transport and Road Research Laboratory (TRL) is now in its ninth version, and has been used world-wide. It calculates co-ordinated fixed-time signal settings by modeling traffic behavior

using histograms to represent arrival and departure patterns of traffic. The tool determines a performance index which is a measure of queues, average and random delays and stops on weighted links. Signal settings are defined and then tested in its model to determine the minimum performance index. The program rigorously checks data input and displays flow profiles graphically.

Special event plans provide for known exceptional changes in traffic patterns such as major sports events or Christmas shopping. Overnight, signals can be made to revert to vehicle actuated operation. Optimal plans are often constrained by policy. In Nottingham, for example, a maximum cycle time of 60 seconds is imposed to minimize delay to pedestrians. When implemented, adjustments have to be made to the settings defined by TRANSYT for difficulties such as queues, or blocking-back. Adjustments are sometimes necessary for road safety reasons. The process of adjustment has become known as “tweaking”.

Bell and Gault (1982) showed that there is an optimum number of plans. For most cities, three or four plans are adequate. Additional plans add to maintenance costs. Furthermore, plan changes bring delay as traffic adjusts to new flow conditions. Frequently, it is better to switch plan change time-tables than introduce extra plans.

Signal plans age because of general changes in traffic behavior. Link flows change when vehicles re-route in response to alterations to the street network. Flow distribution can change following the introduction of a new optimized plan. A route which has been relieved by a new plan can then become attractive to traffic. Aging has been shown to contribute 3% per annum to the deterioration of the efficiency of a set of plans (Bell and Bretherton, 1986).

By the mid 1970's, many large cities had fixed time ATC systems. While these systems improved the efficiency of traffic flow, as they aged, they deteriorated. The limitations of fixed time ATC were recognized in the late 1960's and early 1970's with a substantial research effort pursuing the development of demand responsive traffic control systems. However

Holroyd and Robertson (1973) suggested a means of enabling fixed time ATC to respond to traffic changes, so that plans were automatically updated.

Early systems which respond to detected flow changes in real time were shown to be difficult to develop, Humphrey and Wong (1976) and Rach (1976). These systems failed for three main reasons. Short term changes in traffic conditions could not be predicted, traffic was disrupted when changing plans, and hardware was limited. Computers were too slow and traffic detectors unreliable. As microprocessors became faster and cheaper and the reliability of detectors improved, the design of fully automated dynamically responsive systems matured.

1.2 Adaptive Control Systems

World-wide, many different systems have been installed. In the USA, demand responsive traffic control is in its infancy with automated plan selection providing a degree of self adjustment. The OPAC system (Optimizing Policies for Adaptive Control), see Gartner (1982) and (1983) is based on dynamic programming. There is no central control, no common cycling and while OPAC can be implemented across a network, it was developed for isolated intersection control. In France, the city of Toulous has the PRODYN system, see Henry and Farges (1989), which is also based on dynamic programming. In Tokyo Japan, a master computer determines control strategy with 20 subordinate computers setting signal timings. A review of responsive ATC in Japan is provided by Koshi (1988).

Only two systems have been installed extensively worldwide: SCOOT (Split, Cycle and Offset Optimization Technique), Hunt, Robertson, et al (1981) from the UK and SCATS (Sydney Co-ordinated Adaptive Traffic System) from Australia (Luk et al, 1982).

SCATS optimizes by balancing the degree of saturation on strategic links. Offsets for all links in the network are pre-defined, based on historic data. A set of offsets is selected from a library according to time of day, day of the week etc. with detectors sited just over the stop line. The network is sub-divided into autonomous sub-systems. Each installation has its own local controller. Cycle by cycle, a strategic control algorithm selects appropriated

combinations of green splits, offsets and cycle times for each sub-system and a set of offsets to apply between sub-systems. Single intersection sub-systems are designated critical. The optimization algorithm seeks to balance the degree of saturation with the flows on strategically defined approaches. The combination of splits, offsets and cycle times is selected on a voting system whereby each installation advocates its own preference. Three out of four votes for the same combination in successive cycles will determine its selection. The traffic engineer, on installing the system, defines the split plans, offset plans, cycle times, strategic detector locations and the voting rational.

SCOOT was introduced to the UK in the early 1980's. The detectors monitor flow which enables the SCOOT system to model traffic flow to make systematic adjustments to signal settings, in real time. It can be described as "an on-line TRANSYT". The signal timings, namely green splits and offsets, are adjusted by small amounts (typically four seconds) each cycle in an optimum way. The signal plans are continuously adjusted in response to demand, and are designed to minimize vehicle delay and stops. Signal plans evolve by continuously responding to changing demands. They do not age. Adjacent junctions are grouped into sub-areas which have a common cycle time. At any instant, the cycle time, green durations and offsets between signals are held in computer storeage. The traffic model uses information from vehicle detectors on the approaches to each junction to predict the total delay and stops expected for the current signal timings. The SCOOT model then establishes whether any adjustments to these timings can reduce delay and stops further. In this manner, frequent small alterations adapt the signals to short term fluctuations in the traffic demand. Longer term trends are satisfied by the accumulation of small changes over several minutes, so there are no large disruptive alterations in timings.

SCOOT is quite different from SCATS. SCOOT is a system which has a sound theoretical basis, having developed from the area wide fixed time approach established through TRANSYT, while SCATS lacks such a theoretical basis. SCOOT detectors are sited upstream of stop lines which means that the model has to predict platoon dispersion, while SCATS with detectors just after the stop line, has no need to implement a platoon

dispersion model. This means that SCATS optimized splits better than SCOOT which in turn is better at queue prediction. A continuous range of offsets is available to SCOOT while SCATS must select offsets from a discrete library. Double cycling is permissible at all intersections in SCOOT while SCATS permits double cycling only on minor junctions. Technological developments in vehicle detection through video detectors would enhance SCOOT's platoon dispersion prediction and its split optimizing, while SCATS would be able to determine queue length and would be better able to follow platoon progression. The overall development of signal control from isolated junction control to adaptive or dynamic responsive ATC is summarized in figure 1.1.

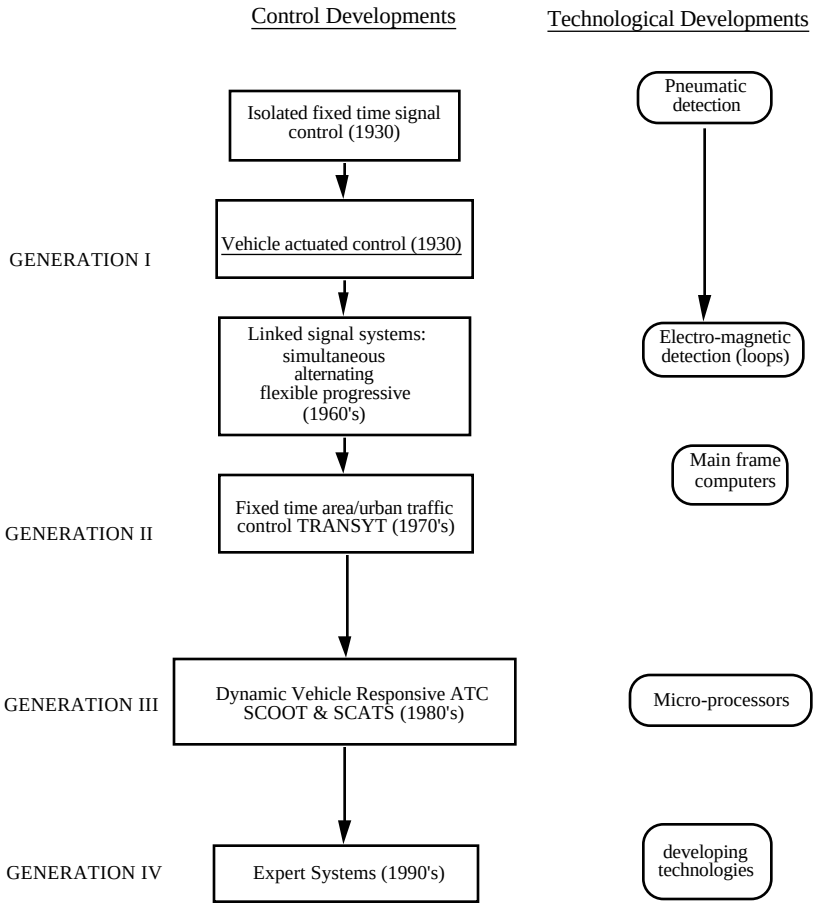


Figure 1.1 Four Generations of Signal Control

1.3 Deficiencies in Current ATC

Despite advances in urban control, however, both fixed-time and demand responsive systems are limited to managing traffic in largely under-saturated conditions and seek to minimize vehicle delay and stops. Accordingly, provided over-saturation does not occur too often or persist for periods of no longer than about five or ten minutes during the peak period, fixed time ATC systems can usually cope. Although SCOOT has a congestion detection algorithm which initiates shifts in signal split and offset to alleviate congestion, it tends only to postpone the onset of congestion and does not cater for system overload.

All adaptive signal control systems rely on extensive quantities of real-time detection of link flows. All are sensitive to detector failure and all collect vast quantities of information which are not fully exploited.

When networks become congested, ATC sub-area boundaries can lose their significance as traffic redistribution occurs. Operator intervention relying on data from Closed Circuit Television cameras (CCTV), allows changes in offset which can relieve the congestion problem in the view of the CCTV, but often creates problems elsewhere in the network. Dynamically responsive ATC systems cannot deal with "incidents" such as sudden changes due to accidents or break-downs and current systems are unable to model traffic conditions across an entire city. Furthermore, ATC systems cannot automatically recognize the onset or growth congestion over a network and even if they could, current systems are incapable of assembling remedial strategies which can respond in real time to the specific problem detected. They operate tactically. The next, and fourth generation of traffic control will also need to act strategically.

Euler (1988) identified the social, political and economic impedimenta to the development of the fourth generation of traffic control. Among the technical impedimenta, he identified the need to develop reliable models to predict Origin and Destination Matrices (O-Ds), assign traffic, simulate traffic flow and make sensible controlled decisions.

One of the most critical parameters in identifying the on-set of congestion and its alleviation is the routes drivers take through an urban traffic network. Shifts in traffic routes occur for many different reasons: following changes in

land use and implementation of new traffic management schemes, in response to the build-up of recurrent congestion and incident, and because drivers choice of route can vary from day to day. Routes can be identified by the flow along links and the turning movement flows at junctions. Automatic vehicle detectors supply link flows.

In designing a fourth generation of signal control, to monitor and control congestion, it is necessary to identify shifts in traffic patterns in relation to transient changes in traffic conditions. In other words, it is necessary to define turning movement flows from detector flows on-line. By establishing algorithms and calibrating methods which will predict changes in traffic patterns, remedial congestion control strategies can be defined in real time. Also such algorithms can be further developed to define alternative routing strategies which can be implemented to improve traffic flow following an incident.

Traffic routes are described by O-D matrices, on a macro scale. On a micro scale, traffic routes are defined by the flow along links and the turning movement flows at junctions. Automatic vehicle detectors already provide a measure of link flows. Automatic traffic counting is well advanced, Bell and Martin (1990), McDonald et al (1987) and Davies et al (1982) with much current research investigating data base manipulation of flows (Bell and Kerridge, 1992).

Turning movement flows can be derived from a variety of transportation planning models. A series of models are discussed. Using link flows, some predict turning movement flows directly, while others predict O-D trip matrices. An O-D matrix estimating model can supply turning movement flows in real time, providing it is quick enough to generate the O-D matrices with time left to assign trips to provide turning movement flows. The models are described as static and dynamic. Static models consider flows as a "snapshot" by modeling conditions from a single view or aggregate view of flows.

1.4 A Review of Routing Models

The principle of entropy for O-D matrix estimation is introduced. Some flow prediction models which serve isolated intersections are discussed. A model which predicts route choice and therefore an O-D matrix from a perceived cost of travel follows and several enhancements of the entropy maximizing matrix estimation method are appraised. The review of so called static models concludes with a discussion of how the transportation community is attempting to standardize matrix estimation procedures.

The review of dynamic methods for route prediction, which make use of the rhythmic nature of traffic flow data, begins with a description of a covariance analysis of detector flows. The estimation of turning flows at an intersection through a recursive algorithm is introduced. The development of the method from a single intersection to a network is traced.

Conventional methods of studying route patterns in urban areas are by traffic assignment techniques. These require knowledge of O-D matrices which are expensive to define. However, even with up-to-date traffic data, assignment techniques cannot provide enough accuracy to control traffic in real time. Their use by traffic engineers is limited to giving an indication of shifts in traffic that may be caused by a major change in traffic or network characteristics. Rarely do absolute flows from assignment agree with those measured on the street. Also, traffic assignment techniques are exhaustive on computer time and in their present form are unlikely to have application for the prediction of alternative routing strategies on-line.

The assignment process, which requires a matrix of origins and destinations, defines the routes taken which then provides an estimate of the link volumes. Matrix Estimation can be interpreted as the inverse of the assignment process. Instead of an O-D matrix being used to estimate link flows, link flows are used to provide an O-D matrix. On a larger scale, the process seeks to provide an O-D matrix which provides movement relationships between zones. On a smaller scale, the objective is to derive a schedule of O-D matrices within a cordon, and thereby infer turning movement flows. A set of observed traffic counts would give rise to a variety of O-D matrices. The task is to select the right O-D matrix, called "the estimate".

1.4.1 Static Methods

Willumsen (1981) provided a comprehensive review of simplified transport models which predict routing information from traffic counts. He identified three approaches: gravity models as developed by Robillard (1975), equilibrium assignment approach as presented by Van Vliet and Dow (1979) and Nguyen (1977), and entropy maximizing models such as those proposed by Murchland (1977) and Van Zuylen and Willumsen (1980).

Wilson (1970) introduced the notion of entropy as a transport modeling tool in his seminal work "Entropy in urban and regional modeling". In classical physics, entropy is a measure of the unavailability of a systems thermal energy for conversion into mechanical work. The state of a gaseous system is fully specified when the co-ordinates and velocity of each particle in the gas at any time are known. The state of a transport system is specified by the number of people in a city, with a number of work places, leisure places, and any other destination to which to travel. Entropy seeks to explain how people behave. So the gaseous co-ordinates are analogous to origins and destinations, while molecular velocity is analogous a set of trips. The O-D matrix, therefore, describes the state of the system.

The states of a transport system are arranged in a three tier hierarchy: Location, Distribution and Assignment. The number of states within each tier increases with decreasing tier rank. At the top, Location states are trip generators (origins) and trip attractors (destinations), and their associated cost or penalty of travel. For the middle tier, there are many distribution states which pair groups of generators with groups of attractors. These pairings are represented by an origin and destination matrix. The bottom tier is the many assignments of individual trip makers for each element of the origin and destination matrix. Entropy is maximized by identifying the O-D matrix as the one associated with the largest possible number of states.

Willumsen (1982) applied entropy principles to transportation planning with the introduction of the Maximum Entropy Matrix Estimation (MEME or ME2) model. Using independent flow paths, it produces better estimates of the O-D matrix than “all-or-nothing”, equilibrium, or heuristic methods. The results demonstrated that the method of estimating the proportion selecting each path still needs improvement. The ME2 model does not require a full set of counts and is geared to making maximum use of the information

available. The matrix generated reproduces observed counts when loaded onto a network. However, the model relies on a prior O-D matrix. The resultant matrix is highly dependent on the starting solution. This means that it is a useful tool for updating O-D matrices but not very good at creating them.

Van Zuylen (1981) and Bell (1983) modified the Willumsen model with a log linear relationship which needs prior information of turning movement flows. The model produces confidence intervals for fitting turning volumes. A Newton Raphson method was applied. The problem presented by this method is that it cannot model large networks because of the exhaustive computer time needed for program execution.

Considering flows at a single junction only, Jeffreys and Norman (1977) presented a method of deriving turning movement flows from flows using the Elementary Rooks Tour. They gave conditions under which feasible turning flows can be calculated and proposed various methods for generating these turning flow patterns. These were usually based on a linear programming approach which solves the equations subject to some linear constraint, such as maximizing the smallest flow. Jeffreys and Norman felt that such solutions were not necessarily the most likely and therefore that a "statistical" solution was more desirable. The methods may help in selecting one set of turning flows from all other possibilities but they failed to recommend any particular method, nor provide a solution.

Van Zuylen (1979) introduced a means of estimating turning flows for isolated intersections where "in" and "out" flows are known. The algorithm, which is both simple and quickly converging, is based on the principle that the turning flows should be compatible with the flows into and out of the junction. From the set of feasible solutions, the most probable solution is defined, therefore, as the solution that uses the minimum amount of information. The weaknesses are that the method cannot accommodate noisy data, and solutions fail to converge when routes and flows are inconsistent.

Fisk and Boyce (1983) proposed a network equilibrium approach for estimating a trip matrix using link count data. The algorithm combines distribution and assignment models in which observed link flow data serve to

furnish an estimate for the sum of the integrals of the link cost functions, which can be described as cost perception parameters.

Bell (1983) presented a program called ODELV (Origin and Destination Estimation from Link Volumes) which produced estimates for the elements of an O-D matrix from link volumes. It includes an explicit treatment of errors in observed link counts. Like ME2, the model requires as input, a prior estimate of an O-D matrix, and a matrix of route choice preferences which have been derived from a traffic assignment model. The model resembles entropy maximization and information minimization models. Variances were derived from logarithms of the fitted values with the parameters incorporating the prior information, treated as random variables. Assuming that traffic streams are measured without error, for an isolated junction, an O-D matrix was derived from flows on the entrance and exit as column and row totals.

Hauer and Shin (1981) proposed a simple two stage manual method. In the first stage, the known flow sums are set onto a matrix, making the last row and last column flow totals, inserting summation values. There is a notation of a simple system whereby all movements can be described on a matrix and all cells which remain empty are assumed contiguous. In the second stage, the shortest row is selected. Row sums are apportioned to the cells in that row in proportion to the column sums. The estimates are subtracted from column sums and the procedure is repeated until all estimates have been derived. The arithmetic is checked to ensure all column and row sums are satisfied. This approach is useful for the analysis of an isolated junction, but cannot be applied to a network.

Echenique and Williams (1982) described the methodology to develop, test and implement a suite of programs to process a freight flow survey for Merseyside, again relying on prior information. The purpose of the study was to take the file of information collected at the roadside interviews, giving the origins and destinations of the sample of lorries interviewed, and then to produce from this an estimated expanded matrix of freight flows between all zone pairs in Merseyside. Having produced an expanded partial matrix, a gravity-type model was calibrated for this partial matrix and then used to produce estimates for the flows between the unobserved zone pairs in a

consistent fashion. They showed that where 27% of the zone pairs are missing, the estimated trips suffer little in accuracy from those estimated using the full matrix.

Mountain (1983) developed an algorithm which estimates turning flows off-line, by taking junction flows and estimates of turning proportions to derive improved estimates of turning proportions. Turning proportions were estimated from both historical data and a sample of turning surveys. The methods cannot be applied on-line as they are insensitive to changes in flow patterns. Their merit is in reducing the surveying necessary for transportation planning, providing estimates of reasonable accuracy. They provide a cheaper method of obtaining data than the more conventional techniques. The application of these methods, however is with large scale transportation planning, and not traffic control.

Robertson (1984) presented a program called MODCOST (Modifying Origin and Destination COsts to Simulate Trips) which simulates driver choice by a random number generator. Hyper-links accommodate perceived costs of factors related to housing and employment while real links represent time costs. The method estimates O-D trips by using observed flows to calibrate a simulation of choices of individuals who are assumed to act to minimize their perceived costs. These are behavioral assumptions. The method rejects the abstract gravitational and entropy maximizing principles and needs no initial estimate of an O-D trip table. The model combines trip generation, trip distribution and trip assignment within a unified random utility model framework, and assigns individuals to trips simulating the choices of several thousand individuals. Trips are all assumed to flow from a hypothetical source to a hypothetical sink. Route choice is available on a one way network. The method uses a random utility model developed by Williams (1976) and Sheffi and Daganzo (1979). A least cost algorithm finds routes which are cheapest throughout.

Irving et al (1986) outlined the experience of practitioners of matrix updating. They represent an alternative procedure employing maximum likelihood with based weights which offers a logically appealing procedure for estimating trip-end growths. With count data supplemented by a small amount of road-side interview data, the resulting matrix retains the deterrence to travel of the

prior matrix, which is something not done in ME2. The method offers the scope to add trip-end data if required and forces the process to match a hierarchy of count sites.

By the late 1980's, the ME2 model had been tested with a variety of experimental modifications. Deficiencies in the information maximization and maximum entropy models were reported by Maher (1987). He argued that when estimating trips from O-D pairs observed few times relative to those which have been observed many times, a bias in the trip estimation matrices was created. He compared the performance of maximum entropy and information minimization. Each method was tested with artificially generated data. He showed that for uniform overall growth, maximum entropy gives positive correlation for the number of counts and for uniform overall growth information minimization gives unbiased results. For an unequal growth, however, for factors which are randomly drawn from the same distribution, information minimization gives negative correlations for the number of counts.

Spiess (1987) introduced another model which improved existing information, for estimating an O-D matrix from an observed sample matrix, when volumes on a subset of the links of the network or the total generation and attraction of the zones are known. Borrowing from the ME2 philosophy, a maximum likelihood model estimates the means of the independent Poisson distributed elements of the observed sample matrix.

Fisk (1988) combined maximum entropy trip matrix estimation with user optimal assignment solving three problems simultaneously. He incorporated the equilibrium conditions as a constraint in the ME2 model by adopting inequality developed by Smith (1979). Oh (1989) proposed a method for estimating trip matrices by solving entropy maximization and equilibrium assignment simultaneously without using route choice proportions. Stark (1989) addressed how the effectiveness of ME2 can be improved by incorporating a mechanism to assess the accuracy of the changes it has affected.

Maher (1983) and later Timms (1990) addressed the use of prior information for the modeling of revised estimates by outlining a procedure, using

Bayesian statistics to subjectively assess the faith held in the quality of the estimate. The procedure seeks to enable the planner to incorporate his degree of skepticism into the estimation process. The modeling framework facilitates the planner in quantitatively scoring the value of his prior knowledge. Two extremes of skeptics are proposed: a "fundamentalist authoritarian" and a "dedicated follower of fashion".

The appeal of entropy maximizing and information minimizing models is their generality, their capacity to make full use of the information contained in the observed flows and their flexibility to use other information. However, gravity models are not realistic for small urban areas. The Entropy Maximization technique (ME2) has found a number of practical applications, particularly for small-town traffic models and has been incorporated in traffic assignment models such as SATURN (Simulation and Assignment of Traffic in Urban Road Networks), see Hall et al (1980).

Transport practitioners had become so familiar with the estimation and improvement of O-D matrices, that by the late 1980's, the first attempts at standardizing matrix estimating procedures appeared. Cascetta and Nguyen (1988) proposed a unified framework for estimating or updating O-D matrices from traffic counts and Logie and Hynd (1989), working under contract to the Dutch Ministry of Transport, proposed a framework for estimating O-D matrices from flows. Stark (1989), addressing the effectiveness of the ME2 method, provided guide-lines on the testing of model reliability. This work indicates a trend towards standardizing the various methods for estimating O-D matrices.

1.4.2 Dynamic Methods

Static models ignore the rhythmic nature of flow data that traffic detectors can supply. When considering their use on-line, static methods suffer other disadvantages. They demand considerable computational effort, the solutions give approximations to real values of turning proportions or O-D matrices and it is difficult to give reliable statements about the quality of estimates.

Wright (1974) estimated journey time distribution by measuring the flows at the beginning and end of an uninterrupted link. His method relies on the

propagation of variations in density from one observation point to the next. The theoretical basis of the method is a covariance analysis. The technique was developed by Jarrett and Wright (1990) in estimating O-D matrices, traffic flows from the random variability in automatic detector counts.

Cremer and Keller (1981) showed how turning movement flows from detector flows, at an intersection, can be estimated by making use of the changing flows in time. The model works recursively by adjusting with flow changes provided such changes occur gradually in practice. The estimation algorithm relies on regular intervals. For each new interval, the change in link in-flows and link out-flows generates and improves estimates of turning proportions. Incremental flow changes are incorporated in a feedback structure. The algorithm uses not only the sum of the accumulated traffic volumes over a longer period, but also evaluates the time sequence of the counted traffic volumes and thus received more information about the process. This model relates to a single junction.

Ploss and Keller (1986) presented a dynamic algorithm which links the causal dependencies of the volume profiles with entropy maximization distribution models. The estimation procedure was compared with the models presented by Cremer and Keller (1981). By introducing travel times between the counting sites, time delays which occur in the progression of the traffic through the network, are taken into consideration. Cremer and Keller (1987) showed that short period exit flows depend by causal relationships, upon the time variable sequences of entrance flow volumes. Unique bias free estimates were derived without a priori information using four methods: Least Square using cross correlation matrices after Cremer and Keller (1981), Constrained Optimization, Recursive Estimation and Kalman Filtering. Bell et al (1991) showed how these dynamic methods can be extended from complex intersections to networks. Currently, dynamic methods are unable to accommodate a city area, or an area the size of a typical urban traffic control sub-area. The method must be extended to more complex groups of intersections and then on to networks where travel time is greater and where the causal dependency of exit flows on entry flows is less direct.

1.5 Summary

The aim of static models was to use detector flows to update O-D matrices over a transportation study area. These models all stemmed from Newtonian analogies and led to the key contribution of Van Zuylen and Willumsen (1980) with models such as ME2. These rely on a prior estimate of the matrix. The dynamic models, pioneered by Cremer and Keller (1981) introduced a dynamic approach which rely on a time series relationship. Today, we see a development of models which are emerging from the single junction treatment to the network approach. The work continues, and is closest in both its performance and scope to the kind of model that an expert system will need.

Unfortunately, predicted flows from static models rarely agree with those measured on the street. Transportation models are both slow and exhaustive on computer time, and in their present form are unlikely to have application for monitoring and control on-line. They were developed as off-line planning tools. The development of dynamic models has potential to address traffic control and monitoring.

The report details the progress made on a one year project which has tested and developed a new model, Turning Movement Estimation in Real Time (TMERT). The proposal set out to collect a meaningful set of data with which to test the ability of the model to estimate turning movements on a Californian network. The scope of the project extended to improving the software so that data input and output would be easier. Collection and testing of the data from the English research would demonstrate that the renewed code was providing the same results as the FORTRAN original. Finally testing the data on a Californian network would demonstrate the repeatability of the model. Each of these objectives has been met. The report discusses the elements of the project, objective by objective. The research planned for the first year of the project is outlined below in figure 1.2.

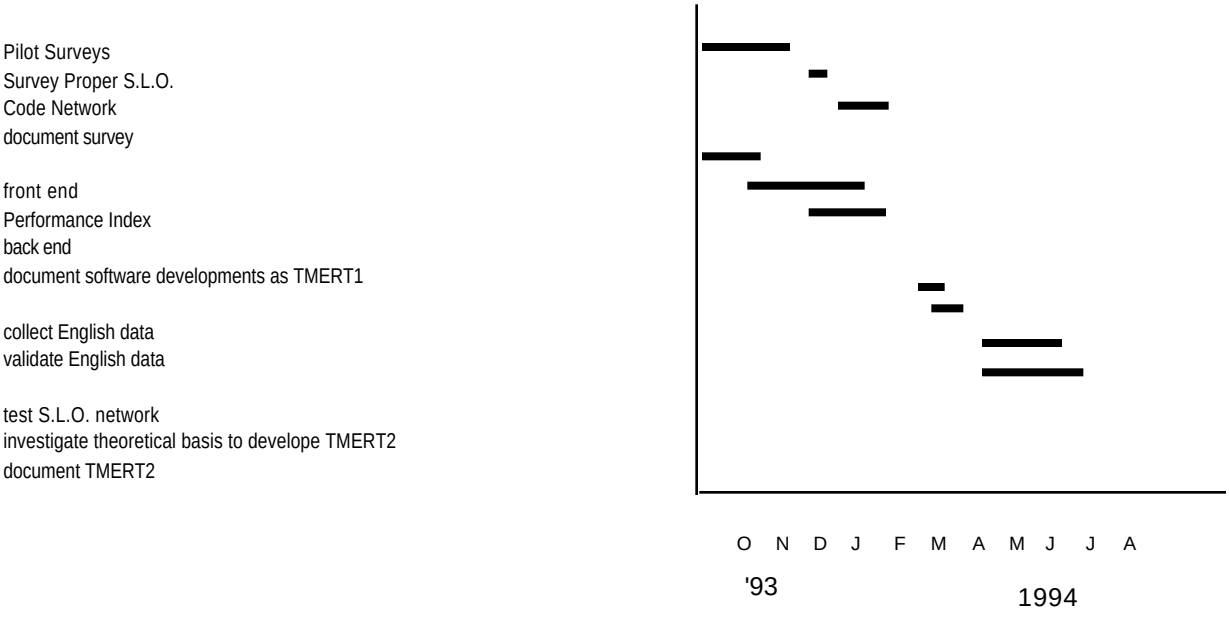


Figure 1.2 Research Plan

Section 2 Turning Movement Survey, San Luis Obispo

This section describes how a network was selected and surveyed to supply the real-time data necessary for the testing and development of the TMERT model. The methodology for the survey is traced through several Pilot Surveys. The management and organization of the Survey Proper concludes the section.

The survey process was developed over the course of several months . Pilot 2 survey was 3 separate surveys performed in early October of '93. Pilot 3 survey was 2 separate surveys performed in early February '94. Pilots 2 and 3 were transformed into laboratory exercises and performed by students of an introductory transportation class in Civil Engineering at California Polytechnic State University, San Luis Obispo (Cal Poly, SLO). The actual data collection survey occurred on March 3rd, 1994 and utilized 38 surveyors and 3 supervisors. The majority of these surveyors were students that had performed the survey as a laboratory exercise and were familiar with the survey process. The schedule of the three pilot and data collection surveys is shown below:

Table 2.1 Survey Schedule

Survey	Activity	Date
Pilot 1	Walk Through	September '93
Pilot 2 survey	3 Student Labs	October '93
Pilot 3 survey	2 Student Labs	February '94
Data Collection	Data Collection	March 3, 1994

2.1 Defining the Network

San Luis Obispo has a population of approximately 50,000 people and is located on the central coast of California half way between Los Angeles and San Francisco. Figure 2.1 shows the analyzed networks location within San Luis Obispo, CA. This area was chosen because it encompasses a freeway, arterials as well as feeder and secondary streets.

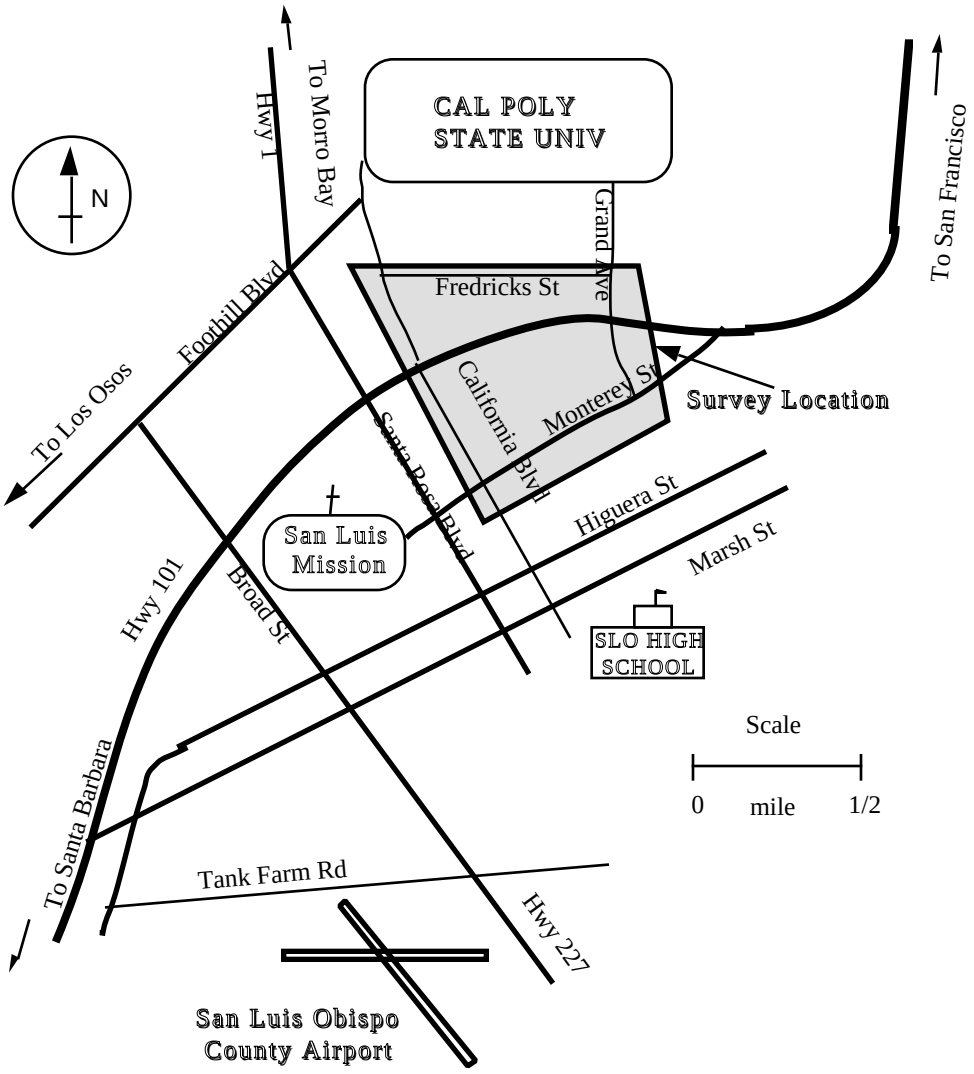


Figure 2.1: Network Location

Figure 2.2 shows the surveyed area network used for modeling. This is referred to as the San Luis Obispo Network or the SLO Network.

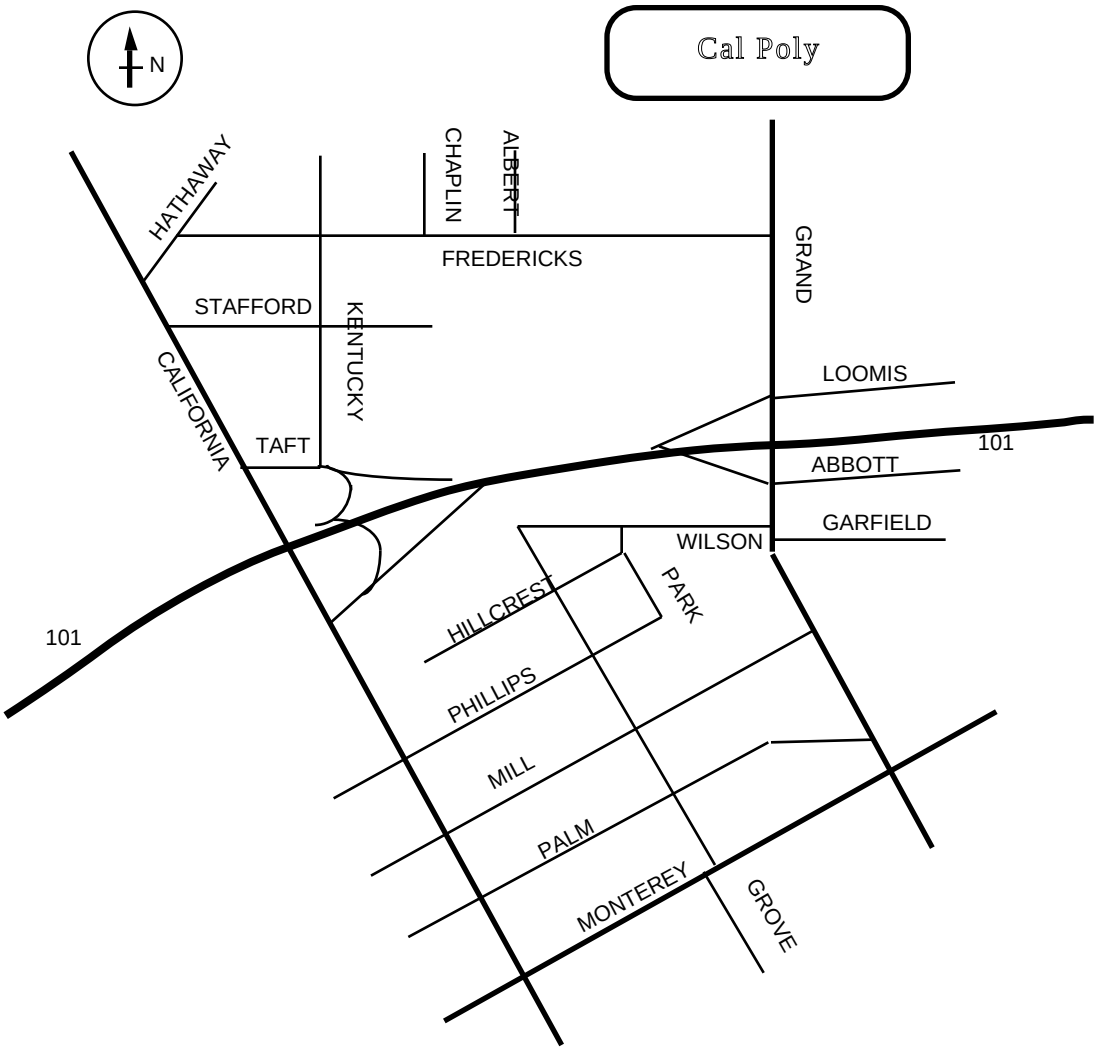


Figure 2.2: San Luis Obispo Network

2.2 Pilot 1 Survey

Pilot 1 survey was a "walk-through" to define the network location and obtain a preliminary assessment as to how many surveyors would be needed at each location. Pilot 1 allowed for a viewing of the geometry of each intersection so that unique data collection sheets could be developed which included the allowed turning movements for each intersection. Each intersection within the network was assigned a unique letter or combination of letters designation as shown in Figure 2.3.

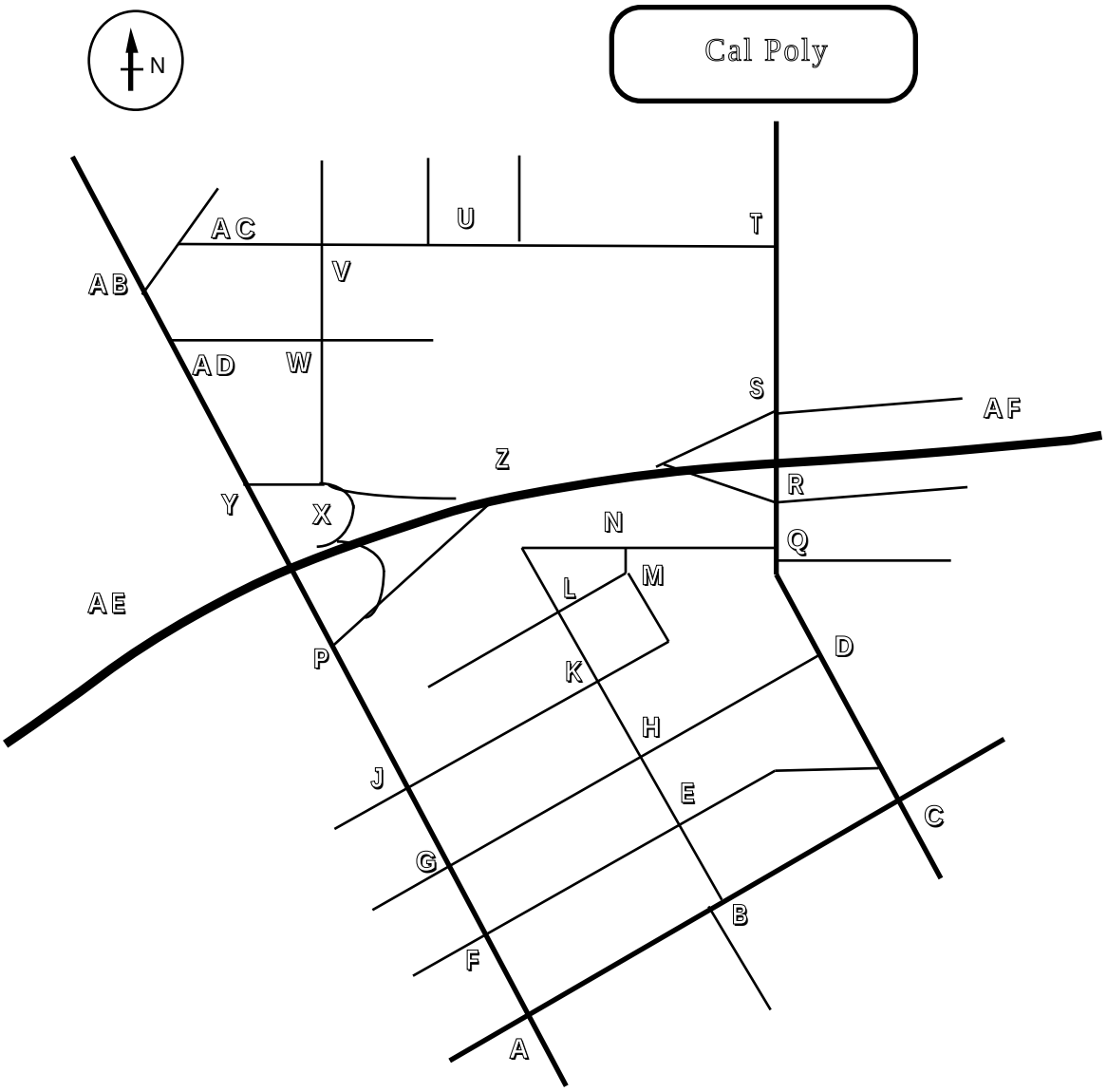


Figure 2.3: Intersection Designation

2.3 Network Characteristics

Intersection geometry, journey times, stop sign locations and orientation, and signalized timing throughout the network were recorded. These data provided flow capacity. The upper bound capacity flows for the internal links and the turning movements were found using the Highway Capacity Manual (1985) and Highway Capacity Software (1992). The stop sign and signalized intersection locations are shown below.

Table 2.2 Stop Sign Location Information

Intersection	way-stop	Direction of Stop
B	2	N-S Grove
CP	1	ER Palm
D	1	E Mill Street
E	2	N-S Grove
F	2	E-W Palm
H	2	N-S Grove
J	2	E-W Phillips
K	2	E-W Phillips
L, M, N		NONE
Q	2	E-W Garfield/Wilson
R	2	E-W 101 Off/Abbott
S	2	E-W 101 On/Loomis
T	1	E Frederick's
U	2	S Albert / S Chaplin
V	2	N-S Kentucky
W	2	E-W Stafford
X	1	S Kentucky
Y	1	W Taft
AB	1	W Hathway
AC	1	W Frederick's
AD	1	W Stafford
Signalized Intersections		
A	4	4-Way Intersections
C	4	4-Way Intersections
G	4	4-Way Intersections

Journey times were measured to ensure that the modeling interval would not be smaller than the journey time necessary to transverse the network as this would be attempting to model traffic activity which is changing faster than the interval permits. The measurements were made by driving across the network by car. The mean of three journeys for each of the alternative trips across the network provided a set of journey times. The results of the journey time surveys are summarized in below.

Table 2.3: Journey Time Observations December 17, 1993

Route	Trial	Time of Day (min:sec)	Journey Time (min:sec)	Avg. Time
AB-A	1	15:57	1:07	
	2	16:02	1:08	1:12
	3	16:08	1:21	
A-AB	1	15:27	1:00	
	2	15:32	1:24	1:13
	3	15:41	1:15	
T-C	1	15:30	0:44	
	2	15:39	0:44	0:44
	3	15:44	0:44	
C-T	1	15:54	0:35	
	2	15:59	0:42	0:38
	3	16:05	0:38	
AC-T	1	15:29	1:11	
	2	15:34	1:13	1:12
	3	15:43	1:14	
T-AC	1	15:55	1:02	
	2	16:00	1:00	1:01
	3	16:05	1:03	
A-C	1	15:54	0:34	
	2	15:59	0:35	0:31
	3	16:04	0:26	
C-A	1	15:31	0:33	
	2	15:40	0:34	0:45
	3	15:45	1:10	
T-Q-N-L-B-A	1	13:08	2:06	
	2	13:15	1:59	2:08
	3	13:22	2:20	
A-B-L-N-Q-T	1	13:11	2:19	
	2	13:19	2:05	2:07
	3	13:25	1:58	
AB-J-K-B-C	1	12:45	2:23	
	2	12:53	2:01	2:08
	3	12:59	2:02	
C-B-K-J-AB	1	12:50	1:52	
	2	12:56	1:52	1:51
	3	13:02	1:51	

Of the 3 signalized intersections located within the network, only the intersection of Mill Street and California Boulevard is operated on a fixed time plan. Two others, located at the intersections of California Boulevard and Monterey Street and Monterey Street and Grand Avenue, are vehicle actuated. Large attractors or generators are referred to as sources or sinks.

During the pilot 1 survey the location and size of sources or sinks existed within the network were noted.

2.3.1 Booking Sheets

The data collection sheets for the survey were referred to as "Booking Sheets". These booking sheets were developed to collect the turning movements of each intersection. The primary purpose of the pilot surveys was to ensure that the booking sheets were properly designed and clear to understand. The data collection sheets were intersection specific. They included the configuration of the intersection and only the movements to be recorded for that surveyor. Some intersections could be accommodated by one surveyor, others needed two or more. For single person intersections, the turning movements included all possible legal movements. For dual person intersections, the turning movements would be split between the two surveyors. By not including all the turning movements for the intersection, but only the ones desired to be recorded by the specific surveyor, the possibility of recording the wrong movements was reduced.

A North arrow was placed on the booking sheet for orientation of the surveyor to the intersection. Street names were provided to prevent the surveyors from recording turning movements with the data collection sheet rotated 90 degrees but the north arrow was provided such that the surveyors were not recording turning movements with the booking sheet rotated 180 degrees.

Each booking sheet represented 5 minutes of the data collection. Counting was recorded by tick marks. One tick mark represented one vehicle making one turning movement. Observers were encouraged to apply the five bar gate method. For each turning movement, a box was provided so that each group of tick marks had its own box. This made the counting of the tick marks easier. It also allowed the next step in the data analysis process, the data entry, more efficient because it allowed the individuals in-putting the data to identify the corresponding counts and movement on the data collection sheet.

Figure 2.4 shows the different aspects of the data collection sheets described above for intersection of California Boulevard and Mill Street.

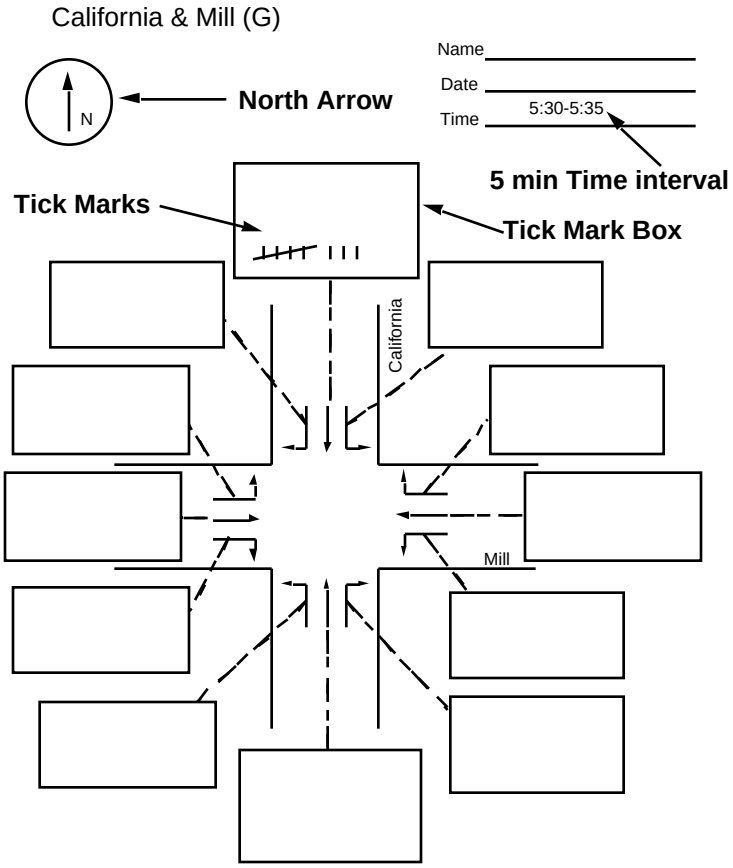


Figure 2.4: Booking Sheet

2.4 Pilot 2 Survey

Pilot 2 survey was the preliminary group of surveys in which the preliminary surveying techniques were implemented by surveyors in the field. The survey, which was developed into a laboratory exercise for was performed by 3 different class sections of undergraduate civil engineering students enrolled in the Fundamentals of Transportation class at Cal Poly on 3 different days during early October of 1993. The three pilot 2 surveys - 2a, 2b, 2c were independent of each other. However, there were not enough students to provide full coverage of the entire network so each survey covered a separate portion of the network.

A briefing of the survey techniques and explanation of the booking sheets was given before the survey. The students were told before they left the

briefing room to perform the survey that they would be asked 5 questions when they returned. The questions asked were:

- Did you rotate the booking sheets to record the movements and if so, how? (i.e. clockwise, counterclockwise, or upside down).
- Were you able to accurately count the movements? Does the intersection need another person or was it excessively slow?
- Could you orient yourself adequately?
- Were the booking sheets clear as to which movements you were asked to record?
- Did you notice any sources or sinks in your area?

From these questions, several lessons were learned from the three Pilot 2 surveys. These lessons resulted in changes to the data collection sheets, collection methods, and surveyor location. Surveyors indicated the locations of intersections that contained heavy flow.

The booking sheets used for the pilot 2 survey were hand drawn intersections and included only the turning movements to be recorded. The surveyors were responsible for drawing their own tick mark boxes for each movement. This led to the problem of not everyone putting the tick boxes in the same location which obstructed the smooth process of data entry. The combination of hand drawing and the tick boxes being located in a variety of places, made entering the observed turning movement flows into spreadsheets both difficult and time consuming.

Although each booking sheet had a north arrow to help the surveyors orient themselves, many surveyors failed to identify north. Since all of the surveyors were students at Cal Poly and familiar with the Universities relative location, an arrow indicating the direction to Cal Poly provided an additional orienting device.

2.5 Data entry problems

The majority of the data entry problems resulted from poor organization of the booking sheets. As the surveyors returned, they were asked how they held the booking sheets. It was noted if they rotated them 90 degrees clockwise, 90 degrees counterclockwise, or 180 degrees, so that the booking sheets orientation could be changed.

2.6 Revised Turning Movement Notation

The turning movements were labeled sequentially in a clockwise direction starting at movement "a" to movement "n", with "i" and "l" being eliminated due to their likeness with the number 1. This key allowed data entry to ensure that each movement was consistently identified (i.e. "f" was always the left turn from the West bound approach and "h" was always the through movement of the North bound approach). Figure 2.5 shows how the turning movements were labeled.

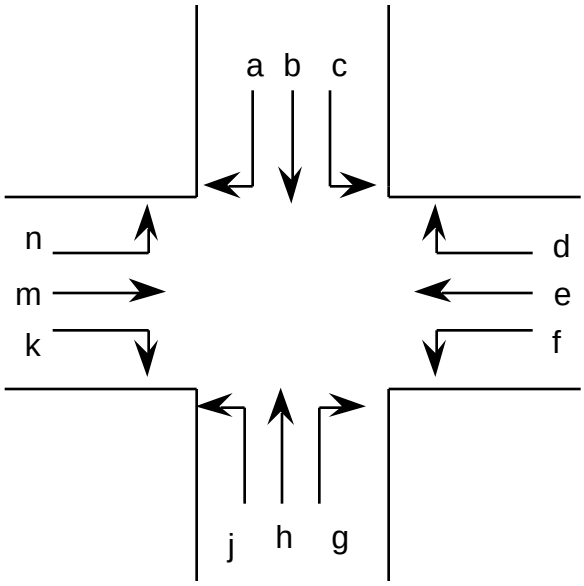


Figure 2.5: Turning Movement Labels

2.7 Improvements for Pilot 3 survey

The pilot 3 surveys tested the Pilot 2 generated improvements. To ensure that the students were familiar with the entire network instead of just their intersection, maps of the network were provided which included the street names, North direction, Cal Poly's location as well as the letter designations for each intersection. One change in the designation of the intersection was

made. Location U was divided into the two “T” intersections of UC (for U at Chaplin St.) and UA (for U at Albert St.). This eased data entry. Figure 2.6 shows the revised intersection regime.

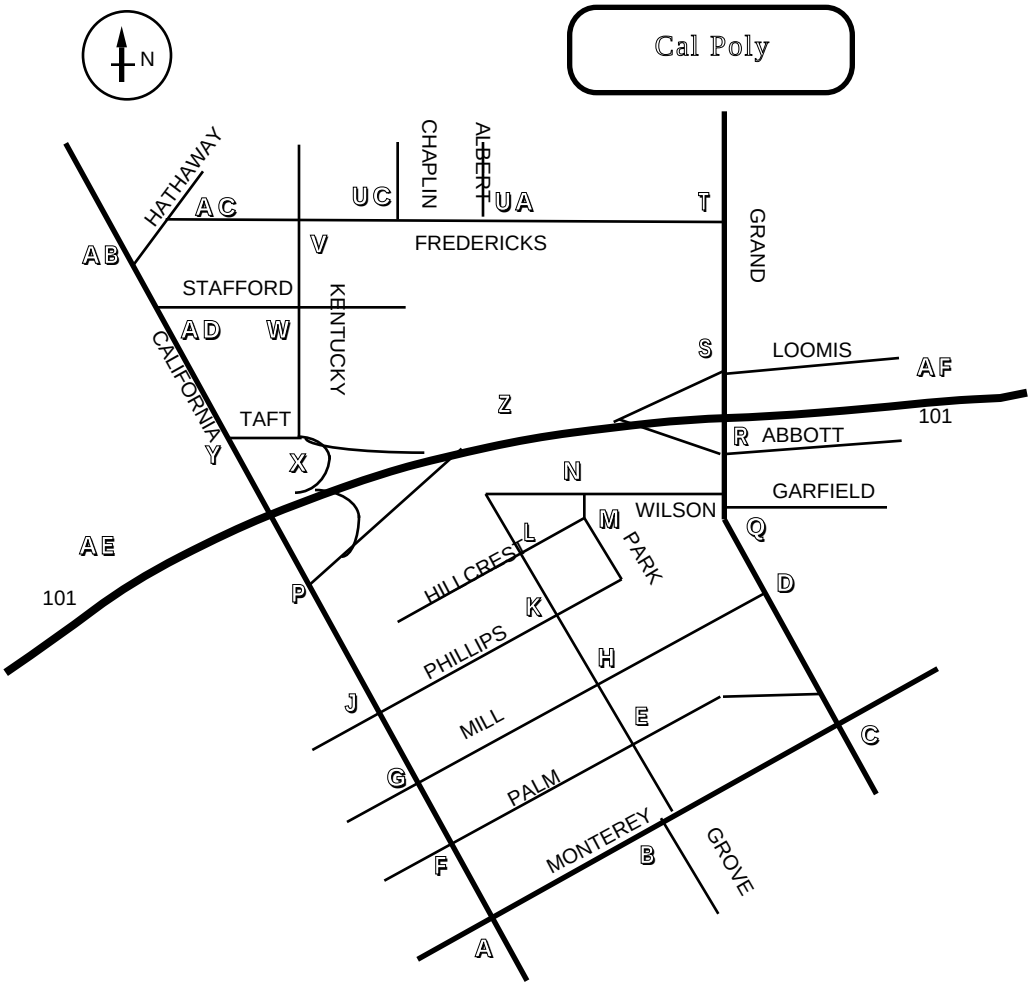


Figure 2.6: Revised Network

To eliminate some of the clarity problems associated with the hand drawn booking sheets, computer generated booking sheets were drawn. The sheets were customized for each intersection and contained the north arrow, the direction to Cal Poly, and pre-drawn tick mark boxes. Each bore a precise location for the surveyors to stand at the intersection. This spot was designated by a circled "X". Figure 2.7 shows an example of an improved booking sheet.

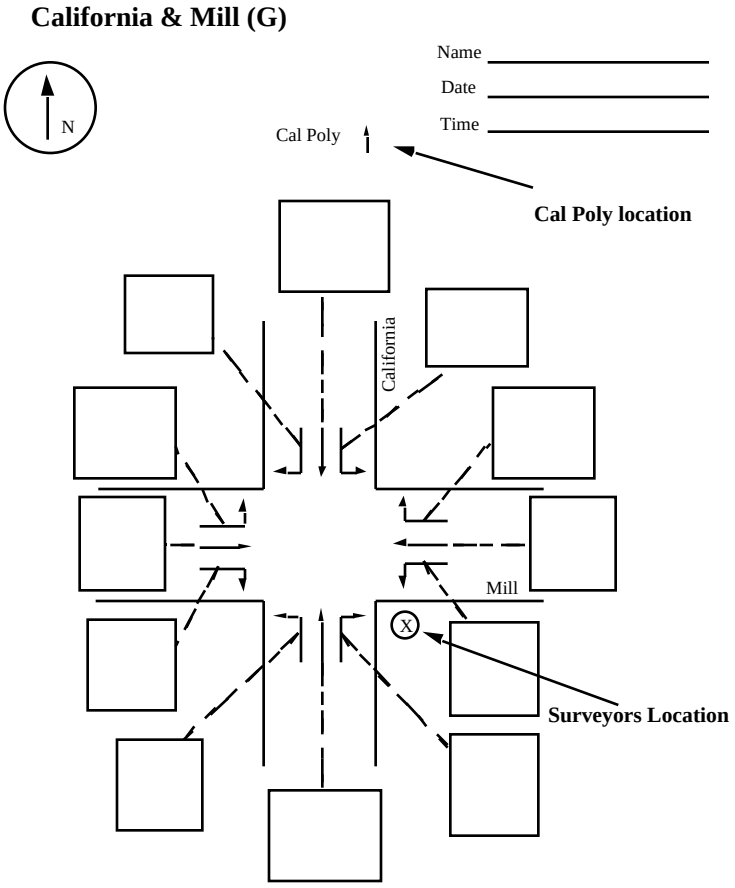


Figure 2.7: Computer Drawn Booking Sheet for Intersection G

The direction of how the booking sheet was held varied by surveyor. So the notion of attempting to orient each sheet to suit each surveyor was abandoned. North was drawn so that it always pointed upward. Therefore, during data entry it was not necessary to determine which way to rotate the data sheets so that north was at the top. This meant that reliability of data entry is deemed more important than customizing sheets for each individual's orientation preference.

Since the tick mark boxes were already provided, their location was constant and the need to determine which turning movements correlated to which tick mark boxes was no longer necessary. Instead of the data entry personnel summing the tick marks, the surveyors, after the survey was completed, summed the tick marks in each turning movement box, putting the numerical value next to the box. This reduced the time taken over the data input process. Random checks of observers arithmetic reassured their reliability.

The turning movement labels (a - n) were found to be confusing and added an unnecessary complexity to the interpretation of the data. Figure 2.8 shows the modified labels with the approach direction and the movement (i.e. "WR" is a right turn from the west bound approach and "ST" is the through movement of the south bound approach).

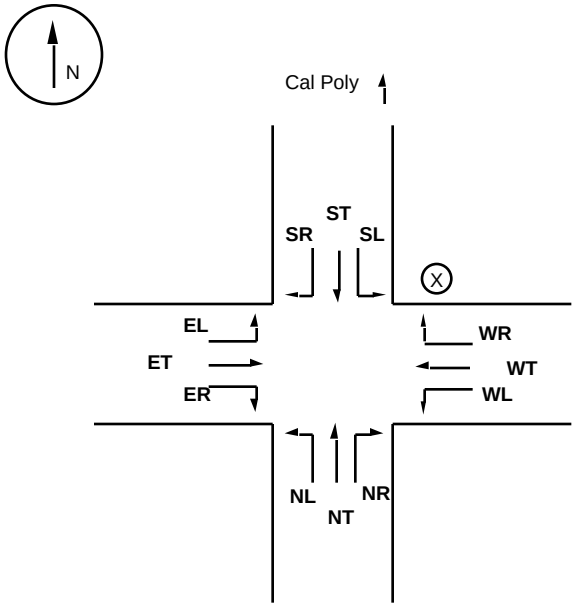


Figure 2.8: Turning Movement Key

2.8 Student Surveyors

From the pilot surveys the names of those students that were interested in taking part in the actual survey were identified. It was desired to obtain as many students as possible due to their prior experience in the survey, however, Student reliability varied. It was not clear as to how many of the students that said they were going to participate in the survey would actually show up. To try and avoid this problem, two surveyors were employed as alternates in the event of "no shows". In case more than two student did not show, a contingency list of low priority locations was established in order to reduce the network size if needed. In order of elimination, the intersections were AB, AC, AD, V, W. Alternatively, flexibility for over-provision was

provided by identifying low priority peripheral intersections. There included the intersections in order of descending priority as Intersections B, J, F, G, Z, AE, AF.

The surveyors were predominately selected from the students that had participated in the pilot surveys. For the survey proper, 36 surveyors and two alternates were selected. On the survey proper, two of the surveyors did not show, but two unexpected student did, so the desired total of 38 surveyors was still achieved. Supervisors were necessary to ensure that the surveyors understood their duties and correctly recorded the turning movements. The supervisors were also available for any questions that the surveyors raised or if temporary relief was required.

2.9 Survey Proper

Before each survey began, a briefing was given to explain the layout of the network, its location, the booking sheets, the method of counting, and to synchronize watches so that the common disaggregating survey interval of five minutes would be consistent. Administrative chores such as the completion of time cards, tax forms, and insurance waivers completed the briefing. The surveyors were instructed to measure turning movements from 4:00 PM to 6:30 PM, to return to the briefing room and count their tick marks for each box, and write the numerical equivalent next to each box. Table 2.4 is the schedule of the survey proper.

Table 2.4: Survey Schedule

Time	Activity
3:00 - 3:45	briefing and form filling-PIF, Timecards, Waivers
3:45 - 4:00	Campus to survey site
4:00 - 6:30	survey
6:30 - 6:45	survey site to campus
6:45 - 7:00	data return & de-brief

Although the survey was planned to run from 4:00 - 6:30 PM, the briefing session overran and therefore many of the surveyors did not get to their locations until 4:05 or even 4:10. However, the original intentions was to use only the data from 4:30 to 6:30. This half hour of "warm-up" time would allow the surveyors to become comfortable with their intersections and the

counting procedure. It also allowed time for the supervisors to visit each surveyor and ensure that they were recording properly.

Appendix C is a record of all turning movement observations recorded by arc label and observation interval.

2.10 Summary

Section 2 has explained the method of collecting the validation data for the model. The process used to attempt to acquire as error free data as possible has been explained as has the improvement of the survey method as the pilot surveys were completed, and how the survey was organized and the method of obtaining the surveyors. Having collected and stored a reliable data set, the research proceeded with the coding of the network.

Section 3 The Theoretical Basis of the Model

Section 3 describes how a network composed of streets links connected by intersections into a network of nodes connected by arcs.

3.1 Model Philosophy

Conventional transportation models which predict flows seek optimality by maximizing or minimizing an objective function which usually has a behavioral basis. An objective function can take the form of the sum of a series of costed links. Costs reflect some form of physical travel characteristic such as journey time, distance or other perceived journey cost. An equilibrium model, for example, would suggest that drivers re-route when links approach capacity, minimizing journey length or travel time. Usually, there is a notion of cost implicit in the objective function which is minimized for optimality.

TMERT has no behavioral basis, because its purpose is to infer traffic movements from a limited set of detected link flows. A weight function is introduced which serves as a model controlling mechanism rather than a cost function. Some weights deter, whilst others encourage and no cost is minimized. The model does not seek to imitate drivers route choice. In principle, it seeks to estimate the state of the traffic system (i.e. derive turning movement flows and unknown flows) from geometric and traffic data. For this reason, the objective function plays a subordinate role. It is a mechanism for identifying the one solution from the many possible solutions and has no physical meaning. Since each flow inference is associated with a set of constraints, the aim here is to establish a constraint regime, so that the minimized objective function is associated with a particular solution, whereby unknown flows are reliably inferred. The model can be described as a traffic flow curve fitting device.

It stands apart from other models in two ways. Firstly, its route logic is node oriented. It moves through the network node by node, seeking to satisfy continuity by balancing in-flows with out-flows. The flow on consecutive links are effectively defined independently. Secondly, it has been structured

to draw on real-time detector flows which is the hallmark of an adaptive signal control system.

3.2 A Linear Program

The network is modeled as a series of nodes connected by arcs which represent homogeneous stretches of road between interruptions such as junctions and car parks. Network entry and exit points are represented as external arcs where vehicles are simulated to enter and leave as Requirement Flows. Kirchoff's Law for the flow of electricity states that the sum of electrical flows entering a node must equal the sum of flows leaving that node, as shown below:

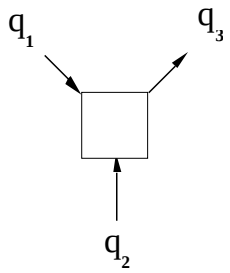


Figure 3.1 Kirchoff's Law

$$q_1 + q_2 = q_3$$

The same principle can be applied to traffic flow once links are represented as arcs and the intersection of links are represented as nodes. Often more than three arcs meet at a node.

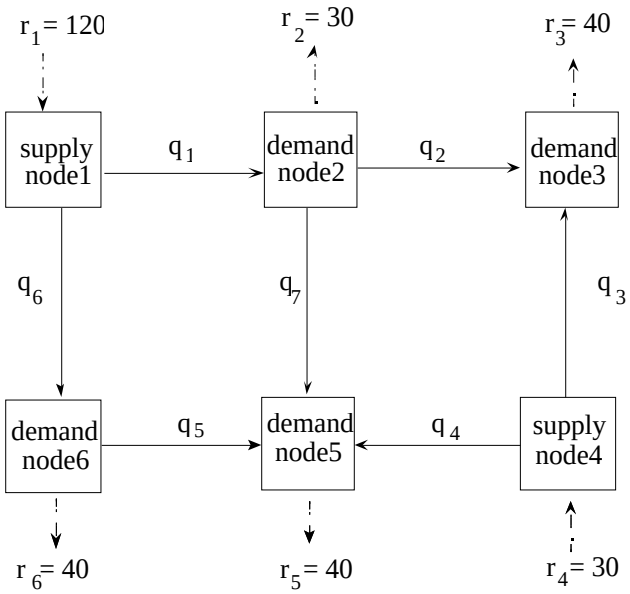


Figure 3.2 The Six Node Network

Consider a Six node, seven arc, six external arc network shown in Figure 3.2. The nodes are loaded by a series of external node loads shown as r. Traffic flow into and out of the network can be modeled as a set of requirement flows, r, which enter the network as a positive value at a supply node and leave the network as a negative value at a demand node. Kirchoffs Law connotes nodal continuity which sets up the flow continuity equations:

$$q_1 + q_6 = r_1 \quad \text{(node 1)}$$
$$-q_1 + q_2 + q_7 = -r_2 \quad \text{(node 2)}$$
$$-q_2 - q_3 = -r_3 \quad \text{(node 3)}$$
$$q_3 + q_4 = r_4 \quad \text{(node 4)}$$
$$-q_4 - q_5 - q_7 = -r_5 \quad \text{(node 5)}$$
$$q_5 - q_6 = -r_6 \quad \text{(node 6)}$$

In reality, traffic flows cannot increase beyond a certain flow level. Signal controlled junctions impose a constraint. Accordingly, each arc can be constrained by a capacity value, shown as u. The capacity constraints supply a further six equations for each node:

$$q_1 \leq u_1 \quad \text{(node 1)}$$
$$\text{"} \qquad \text{"}$$
$$q_6 \leq u_6 \quad \text{(node 6)}$$

Each arc can be constrained by a flow minimum value, shown as v. Flow minima provides a further six equations:

$q_1 \geq v_1$ _____(node 1)

" "

$q_6 \geq v_6$ _____(node 6)

The cost or attractiveness of flow along an arc can be modeled by providing each arc with a weight value shown as w. Then the sum of the product of each flow with its weight provides an objective function F which is minimized:

$$F= q_1w_1 + q_2w_2 + q_3w_3 + q_4w_4 + q_5w_5 + q_6w_6 + q_7w_7$$

or more generally, minimize $F = \Sigma q_i w_i$

Substituting the given requirement flows:

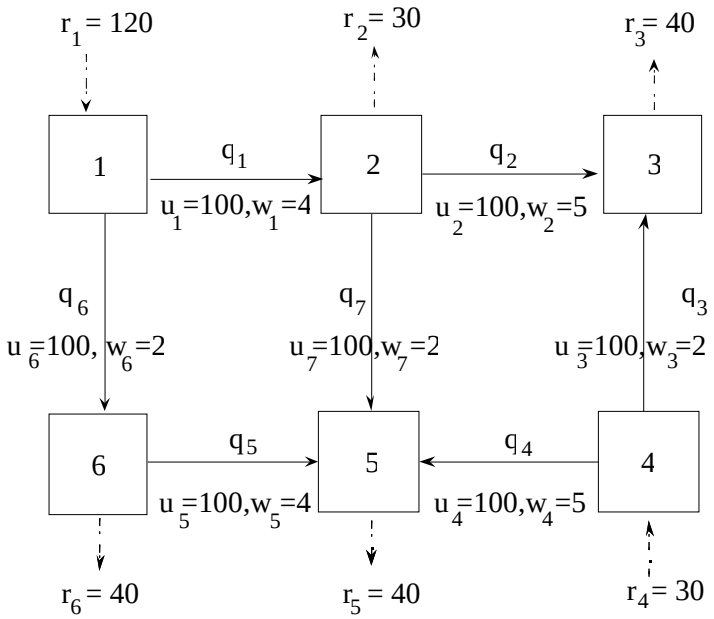


Figure 3.3 Six Node Network Constrained

$q_1 + q_6 = 120$ _____(1)

$-q_1 + q_2 + q_7 = -30$ _____(2)

$-q_2 - q_3 = -40$ _____(3)

$q_3 + q_4 = +30$ _____(4)

$-q_4 - q_5 - q_7 = -40$ _____(5)

$q_5 - q_6 = -40$ _____(6)

Setting arc capacity constraints to 100, provides capacity equations:

$$q_1 \leq 100 \quad (7)$$
$$q_2 \leq 100 \quad (8)$$
$$q_3 \leq 100 \quad (9)$$
$$q_4 \leq 100 \quad (10)$$
$$q_5 \leq 100 \quad (11)$$
$$q_6 \leq 100 \quad (12)$$
$$q_7 \leq 100 \quad (13)$$

The inequalities of equations (7) to (13) are addressed by introducing slack variables (q_{s1} to q_{s7}) which remove the in-equalities:

$$q_1 + q_{s1} = 100 \quad (7a)$$
$$q_2 + q_{s2} = 100 \quad (8a)$$
$$q_3 + q_{s3} = 100 \quad (9a)$$
$$q_4 + q_{s4} = 100 \quad (10a)$$
$$q_5 + q_{s5} = 100 \quad (11a)$$
$$q_6 + q_{s6} = 100 \quad (12a)$$
$$q_7 + q_{s7} = 100 \quad (13a)$$

The application of arc weights as defined above in Figure 3.3 to predicted flows provides an objective function F:

$$F= 4q_1 + 5q_2 + 2q_3 + 5q_4 + 4q_5 + 2q_6 + 2q_7$$

These equations may be solved by the "Simplex Method" (Danzig et al, 1951), which solves the series of inequality equations. If a set of solutions to the series of equations exists, then such solutions are said to be feasible. The one solution which results from the optimization of a weighted flow function, or objective function, is known as the optimum solution. The Six Node Network generates 13 equations and 14 unknowns. Setting a number of variables n, to zero, such that:

$$n = \text{number of unknowns - number of equations,}$$

provides a basic solution which is a basic feasible solution providing each solution q has a positive value. The objective function is evaluated by applying the weights to the solution variables. The next variable is selected and set to zero, and the process repeated. When the objective function cannot

be reduced with further starting values, then the optimum feasible solution has been reached. For the six node network, the optimum feasible solution is given by:

$$\mathbf{q} = \begin{bmatrix} 40 \\ 10 \\ 30 \\ 0 \\ 40 \\ 80 \\ 0 \end{bmatrix}$$

where **q** is the vector of solution flows. The slack variables represent the unused capacity on each arc. They are given by:

$$\mathbf{q_k} = \begin{bmatrix} 60 \\ 90 \\ 70 \\ 100 \\ 60 \\ 20 \\ 100 \end{bmatrix}$$

where $\mathbf{q_k}$ is the vector of solution slack variables.

The optimal feasible solution objective function has a value of 590. It has no numerical importance, it simply identifies the optimal solution. Its numerical value has no use as a system deterrence measure and does not represent cost in the conventional route choice manner.

3.3 A Network form of a Linear Program

A set of linear equations accompanied by an objective function constitutes a linear program. When such a program is set up to model a series of variables, such as flow, there are three components: a system, a problem, and a solution. Here, the system is the network represented by a series of nodes connected by arcs with a structure defined by the geography of the town or

city which it represents. Each arc accommodates a traffic commodity or flow. (The term “arc” is used in relation to a network, while the use of “link” is kept for the description of a road). The signal timings quantify the control environment, i.e. the capacities of both nodes and arcs. The problem is to infer a comprehensive set of turning flows from a sparse set of measured detector data which correspond to the measured link flows. The solution is the feasible solution which satisfies the constraints of the signal control and detector flows such that turning movement flows are inferred to match the actual turning flows with acceptable accuracy.

An Integer Linear Program forms the structure of a network model. An Integer model enables the passage of individual vehicles to be modeled. The method applies a single commodity network to a multi-commodity problem, which appears prima facie as inappropriate. Here traffic movements are modeled devoid of a behavioral influence. Detector flow information supplies the constraint to a flow minimization problem. The objective is not, however, to minimize flow; it is rather to influence the optimization to converge onto that particular solution which most closely resembles the observed flows.

The flows at the intersections are left turns, right turns and straight on maneuvers. These are termed turning movements. The model supplies a set of these turning movements, that are consistent with the link flows that are supplied through detectors. Simple imposition of a set of limited link detector flows as constraints cannot provide feasible solutions. Flexibility is provided by the incremental substitution of theoretical arcs, or "error" arcs, which are attached to each link represented by a detector arc. The linear programming algorithm provides flow estimates by the imposition of detector constraining flows on a sparse number of links. The program, “NETFLO”, which is written in FORTRAN is the Simplex Method (see Danzig et al, 1951) that is fast enough to permit “on-line” application. At each iteration, sparse matrices hold the minimum of information in a more compact form, and therefore the procedure requires less computer storage. As a result larger network problems can be solved.

The network is modeled as a series of nodes connected by arcs. Each link flow and turning movement is represented by its own unique uni-directional arc.

The minimizing of an objective function which is influenced by weights is a special form of linear program known as the minimum cost flow problem. The need to solve the equations quickly in real time, leads to the choice of a network approach to their solution. The first data structure for a network approach to the solution of a linear program was suggested by Johnson (1966). The first implementations were by Srinivasan and Thompson (1973) and Glover et al (1974). The first computer coding of the network approach, which was reported by Glover et al (1977) was shown to be 100 times faster than a general linear programming code. The program NETFLO (NETwork FLOw).

NETFLO is the Simplex Method in matrix form which serves to streamline the original method considerably. Its computational speed provides its potential for on-line application. At each iteration, sparse matrices hold the minimum of information:

1. The coefficients of the non-basic variables in the objective function.
2. The coefficients of the entering basic variable in the other linear equations.
3. The right hand side of the linear equations.

As these essential data only are carried, in a compact form, the procedure requires less computer storage and therefore has the potential to solve much larger network problems. An algebraic procedure, the algorithm moves at each iteration from the current basic feasible solution to a better adjacent feasible solution by choosing both an entering basic variable and a leaving basic variable. Then, the system of linear equations are solved using gaussian elimination. When the current solution cannot be improved, it is deemed optimal and the algorithm stops. Mathematically, the problem reduces to minimizing an objective function F, which when written in matrix notation takes the form:

$$F = \mathbf{w}\,\bar{\mathbf{q}}$$

The node continuity equations, when written in matrix notation take the form:

$$\mathbf{A}\,\bar{\mathbf{q}} = \bar{\mathbf{r}}$$

subject to the capacity constraints:

$$\mathbf{0} \le \bar{\mathbf{v}} \le \bar{\mathbf{q}} \le \bar{\mathbf{u}}$$

where,

A is an [i x j] node-arc incidence matrix, which defines the network structure, where the nodes are numbered from 1 to i, and arcs are numbered from 1 to j

w is a [1 x j] vector of unit weights associated with each arc - this denotes an arc weighting which can be given any set of values

$\bar{\mathbf{r}}$ is an [i x 1] vector of flows which enter and leave the network at the periphery, known as the requirement vector - internal nodes are given zero values

$\bar{\mathbf{v}}$ is a [j x 1] vector of arc lower bounds - this denotes minimum arc flow

$\bar{\mathbf{u}}$ is a [j x 1] vector of arc upper bounds or arc capacities

$\bar{\mathbf{q}}$ is a [j x 1] vector of arc flow - the decision variable, i.e. the unknown quantity

The node-arc incidence matrix, **A** can be likened to a circuit diagram or pipe network drawing. The unit weight component, **w** can be loosely equated to electrical resistance measured in ohms or pipe roughness measured in millimeters. The analogy, however is tenuous because the traffic model applies **w** as a deterrent device. The requirement vector $\bar{\mathbf{r}}$ can be compared to the potential difference, measured in volts or hydraulically as head measured in units of length. The minimum flow vector $\bar{\mathbf{v}}$ will in most cases

represent zero flow, in traffic terms. Some water supply systems rely on a flow minima for pipe-works to self cleanse. The vector $\bar{\mathbf{u}}$ can represent the saturation flow at a non-signalized junction or reflect the maximum possible departures during a green time, or stop-line capacity. The hydraulic equivalent is a combination of pipe diameter and gradient. The arc flow vector $\bar{\mathbf{q}}$ denotes the output traffic volume in number of vehicles for a given period. The electrical equivalent is current measured in coulombs, the hydraulic equivalent is water volume measured, say, in liters for a known flow interval.

The node arc incidence matrix **A** identifies the direction of travel of traffic at each node. The element A_{ij} of the array takes a value +1 if arc j is directed away from node i, and a value of -1 if arc j is directed towards node i. The value 0 is applied if arc j and node i do not meet.

The node-incidence matrix for the Six Node Network is given by:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 \end{bmatrix}$$

The requirement vector $\bar{\mathbf{r}}$ specifies the traffic flow into and out of a network. If for node i, $r > 0$, the node i is a supply node where traffic flows into the network, and supply is equal to r. If for node i, $r < 0$ the node i is a demand node and traffic flows out of the network. Internal nodes or turning points at a junction have $r = 0$, which are known as trans-shipment points. Therefore, for the Six Node Network example, these matrices are given by:

$$\mathbf{w} = \begin{bmatrix} 4 & 5 & 2 & 5 & 4 & 2 & 2 \end{bmatrix}$$

$$\bar{\mathbf{r}} = \begin{bmatrix} +120 \\ -30 \\ -40 \\ +30 \\ -40 \\ -40 \end{bmatrix} \qquad \bar{\mathbf{v}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \bar{\mathbf{u}} = \begin{bmatrix} 100 \\ 100 \\ 100 \\ 100 \\ 100 \\ 100 \\ 100 \end{bmatrix}$$

The decision variable vector of unknowns is given by: $\bar{\mathbf{q}} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \\ q_6 \\ q_7 \end{bmatrix}$

In order to transform the lower bound vector, $\bar{\mathbf{v}}$, to a zero vector, four new vectors are introduced. The vectors **r, u, q**, and α are defined by:

$$\mathbf{r} = \bar{\mathbf{r}} - \mathbf{A}\,\bar{\mathbf{v}}$$

$$\mathbf{u} = \bar{\mathbf{u}} - \bar{\mathbf{v}}$$

$$\mathbf{q} = \bar{\mathbf{q}} - \bar{\mathbf{v}}$$

$$\alpha = \mathbf{w}\,\bar{\mathbf{v}}$$

As the lower bounds are reduced to zero, then the objective function reduces to:

$$F = \mathbf{wq} + \alpha$$

such that $\mathbf{Aq} = \mathbf{r}$

and $\mathbf{0} \geq \mathbf{q} \geq \mathbf{u}$

The NETFLO model applies a heuristic procedure by quickly finding paths through the network which satisfy node continuity. Spanning trees are

supplemented by artificial arcs. A spanning tree is a subsidiary network which is set up to contain all the nodes but a reduced number of arcs to eliminate loops. Part of a spanning tree is formed to satisfy the induced supply and induced demand. For each supply node s, an artificial node y is set up, connected by an artificial arc (s,y) of infinite weight and capacity, and flow t_{sy} satisfying node continuity. For each demand node (d), where flow leaves the network, an artificial node, p is set up connected by an artificial arc (p,d) of infinite weight and capacity, and flow t_{pd} satisfying node continuity. The use of spanning trees which are supplemented by artificial arcs where necessary, makes the algorithm quick. The artificial nodes are illustrated below.

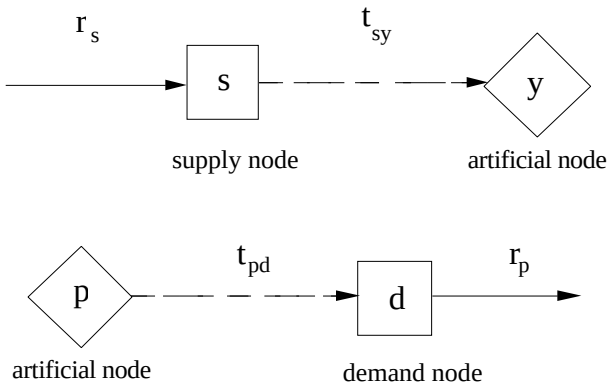


Figure 3.4 Artificial Nodes

Since the artificial arcs are given such high weights, the optimum solution is dominated by a set of flows whereby all artificial arcs are assigned zero flow. This approach is similar to the "Big-M" method, see Hillier and Lieberman (1990), whereby the objective function is supplemented with an additional term M, which denoting a very large positive number carries an overwhelming penalty. The objective function now becomes:

$$F = \mathbf{wq} + \alpha + M\mathbf{t}$$

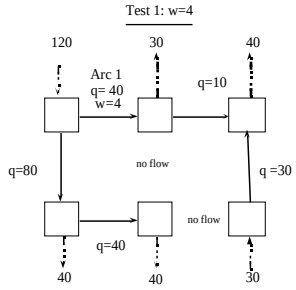
where $\mathbf{t}$ is an [m x 1] vector of artificial arc flows, where m is the number of artificial arcs generated by the particular network. The spanning trees are completed with the addition of more artificial arcs and basis exchanges are performed to achieve optimality.

The data input is in two parts. First, the Node Precedence Data specifies the relationship between each node. NETFLO requires this information in the form of an [8 x i] matrix, which gives the number of times each node appears as a downstream node. Second, the Arc Oriented Data show how nodes are connected, define weight loadings and give upper and lower bound flow limits.

Weights have a strong influence on the flow solution, providing a control mechanism for flow prediction. Consider the effect of a weight variation for the first arc of the Six Node Network. For a weight reduction from 4 to 1, the new optimum solution shows that flow is diverted along the central arc. The weights and solutions are given in Table 3.1, and represented in Figure 3.5.

Table 3.1 Weight Test

Arc	Test 1		Test 2	
	Weight w	Flow q	Weight w	Flow q
1	4	40	1	80
2	5	10	5	10
3	2	30	2	30
4	5	0	5	0
5	4	40	4	0
6	2	80	2	40
7	2	0	2	40



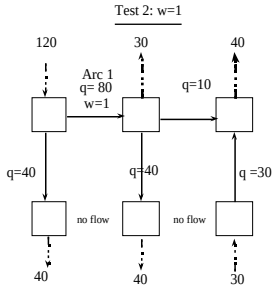


Figure 3.5 Six Node Network, Weight Change Test

Since the flows entering the network, and the network structure itself are fixed, the remaining parameters of capacity, flow minima and weight are the manipulative devices. The combination of capacities, flow minima and weights which are defined by the control system parameters can be described as the Constraint Regime. The emerging model input and output specification are summarized in Figure 3.6.

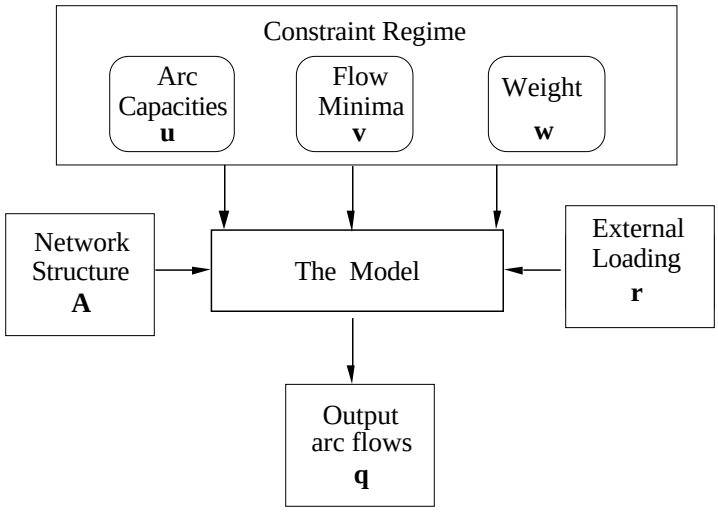


Figure 3.6 Model Input and Output

3.4 Defining Nodes & Arcs

Supply and demand nodes are described as Cordon Nodes. Those internal nodes with zero requirement flow are described as Trans-shipment Nodes.

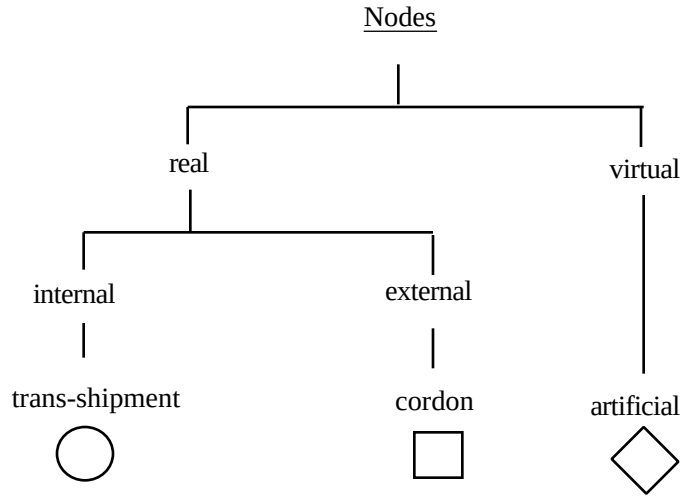


Figure 3.7 Node Notation

Four types of arc are defined. An Internal Arc transmits flows from any node type to any other node type. The External Arcs represent entry and exit flows, and are not afforded a label. An Internal Detector Arc models the site of a SCOOT detector, within the network. An External Arc is a virtual arc which signifies the entry or exit flow into or out of the cordon. The model does not assign flow along External Arcs because one end of the arc is free.

Two types of real node are defined. A Cordon Node marks an entry or exit point to the network. Each is associated with an External Arc and is connected to any number of Internal arcs and/or Detector Arcs. Trans-shipment Nodes are not connected to External arcs. They have any number of inflow arcs (Internal or Detector), and any number of outflow arcs (Internal or Detector). The various combinations of nodes and arcs are illustrated below.

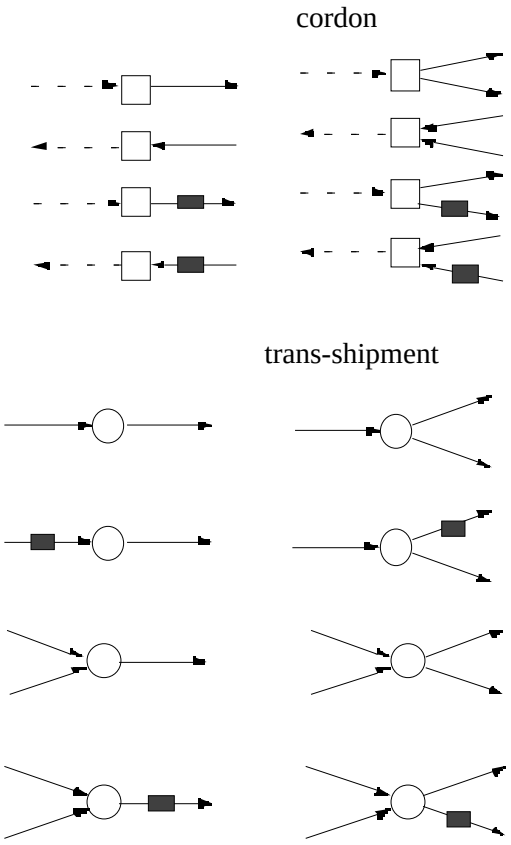


Figure 3.8 Permissible Node Arc Connections

Each turning movement at an intersection is represented as its own unique arc. Arcs present represent the turning movements available at an intersection and banned turning movements are described by their absence from the network. Figure 3.9 shows how a 4 way intersection is modeled.

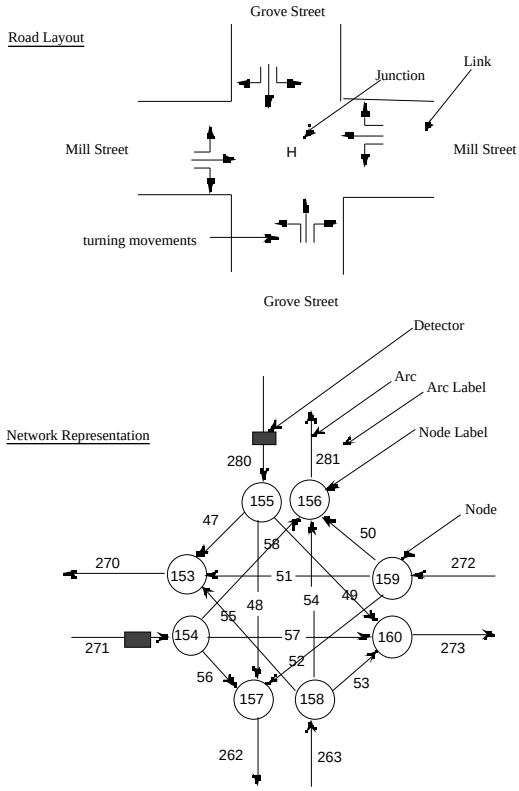


Figure 3.9 A Single Junction Network Representation

3.5 Network characteristics

The network was built in stages with increasing complexity. Appendix D contains representations of the various stages of network development. The final configuration, the one used for analysis of the network, is shown in Figure 3.10

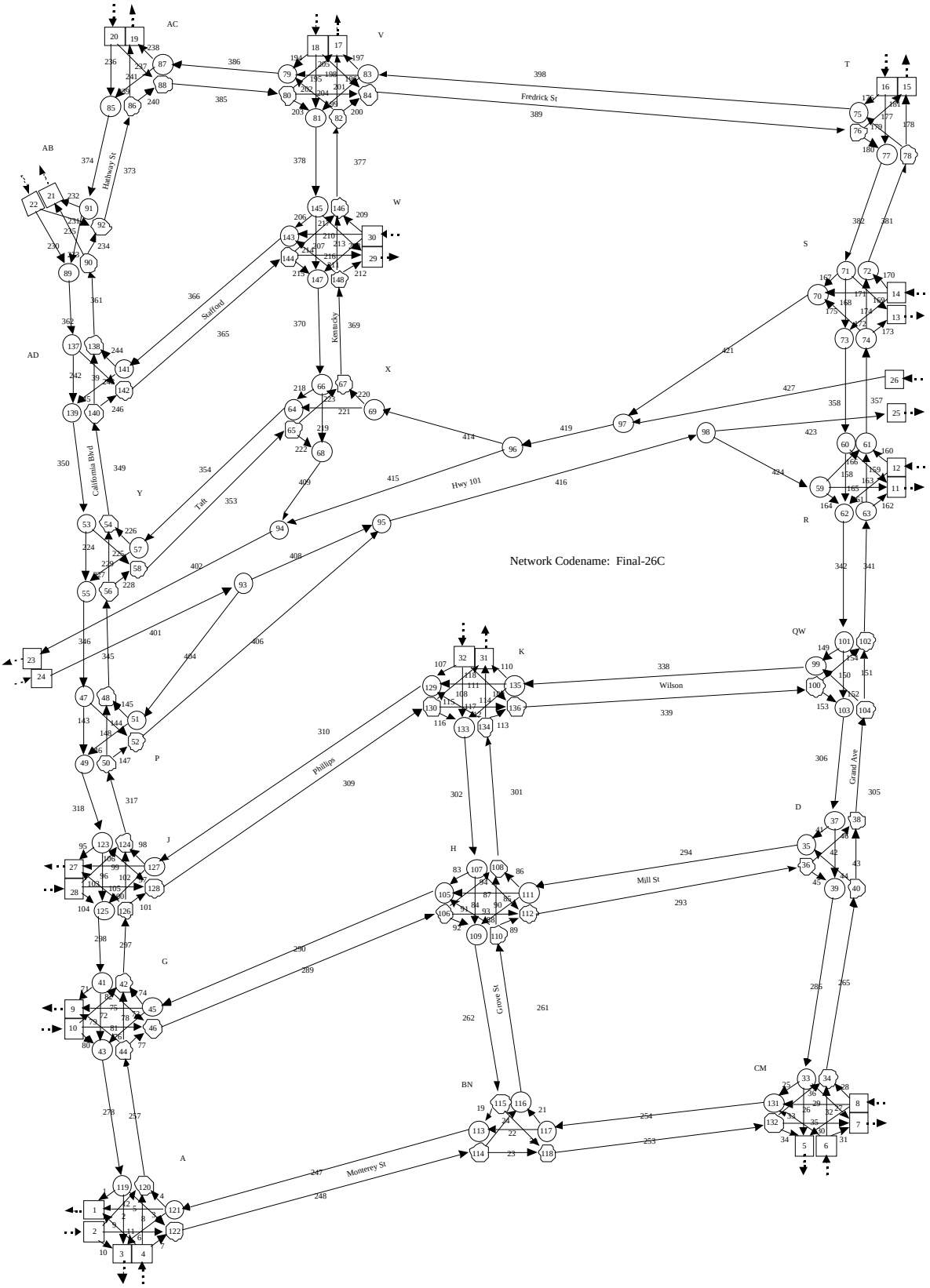


Figure 3.10: Final Network Configuration

Section 4 Software Development

4.1 Code Development

Software development began by analyzing the original FORTRAN code of NETFLO. The FORTRAN code was written in a pre-structured or "spaghetti" FORTRAN manner, so it was thought that flowcharting the code may show some of the algorithmic properties, that were not readily apparent. Flowcharting the code showed that the code contained no complex data structures and that there were no complex flow control constructs. The possibility of moving code to a traditional spreadsheet format, such as that offered by Microsoft Excel looked enticing. Excel's macro language (version 4.0) supports a subset of the traditional Pascal type languages and it was the case that this subset would be sufficient.

Since the first goal was to replicate the original "Notttingham" code, it was essential to maintain the integrity of the original FORTRAN code. So the FORTRAN code was translated into structured "Excel" code. In converting from FORTRAN to Excel there were few difficulties in the actual languages. Excel's support for arrays was one major weakness (especially arrays with variable lower bounds).

Before the laborious process of setting down the Excel code it was necessary to break the 1200 line algorithm into logical subroutines, to make a structured piece of code. Logical subroutines were imposed and redundant code was eliminated.

The code was tested. There were major bugs that related to array access. Repeated attempts to overcome these difficulties failed. So, the original FORTRAN code was compiled and executed step by step simultaneously with the Excel code. Several tests checking at various breakpoints determined the location of the bugs. The "XOR approximation function" proved the source of the bug. Its purpose is to return a value greater than zero if the XOR(a, b) is greater than zero, or return less than zero if XOR(a, b) is less than zero, and zero if otherwise. Unfortunately it was returning

zero on several occasions when it should not have. Here is an extract of the defective code:

```
XOR(arg_1, arg_2)

=RESULT(1)

temp = arg_1 * arg_2

=IF(temp = 0, temp = arg_1 + arg_2)

=RETURN(temp)
```

This is how the code was written in the original FORTRAN code. The first bug is that the code fails to approximate the XOR function properly. When arg_1 equals arg_2 the function should return 0. The next bug came from the Excel application. Excel fails to evaluate the boolean expressions in an IF statement that takes the form:

```
=IF(bool, statement)
```

Excel will evaluate that boolean to FALSE on some occasions when TRUE should the value of the boolean expression. This was overcome by simply removing the embedded statement, such that the IF constructs appear as:

```
=IF(bool)

statement_list

=END.IF()
```

The code produced the correct results, but in an unacceptable amount of time on a Macintosh IIci, with a 68030 processor running at 25MHz. Moving the code to the Macintosh Quadra 840AV (68040 at 66MHz) improved the speed but the time to run the code still lagged at 37 minutes for the English network. Looking ahead to several hundred test runs of the program, it was imperative that this process be speeded up. That Excel is an interpreted language meant that even though the code had been extensively restructured, it was clear that it was not possible to reduce the running time by any order of magnitude. It was an essential principle of the project that the spreadsheet

front end and back end be maintained. The internal engine, however, could be implemented any way necessary. Code resources would have offered the most elegant solution, but the unavailability to the Excel 4.0 Software Developers Kit, meant that it was necessary to rewrite the application in C, using the application "Think C".

The final code is approximately 1750 lines and is shown in Appendix B. It maintains as much of the flavor from the Excel code as possible. The English network took approximately 1.5 seconds to run which is nearly 1500 times faster than the code in Excel.

The new application when executed prompts the user to enter an input file and an output file. The input file is opened and read. The NETFLO algorithm runs on this input, then opens the output file and writes to the output. To maintain the integrity of the front and "back-end", the application is written so that it accepts the input in an Excel spreadsheet. The intput file is saved as a text file and can be read by the NETFLO application. Reading the output file is just as simple. When the output file is opened from Excel, it opens into a spreadsheet that has been properly laid out. A great convenience with this method is that it can move to real time mode easier, since the C code has already been written.

4.1.1. Step by Step Code Analysis

A step by step analysis of how the all-artificial start is implemented in NETFLO follows:

1. Retrieve the first node in the demand list. If the demand list is empty then goto step 17.

2. If the unsatisfied demand of the node from the demand list is zero, goto step 12.

3. Find the set of arcs, S, such that they may be set to upper bound. S is such that each arc end at q.

4. Find the set of arcs, T, such that they may become basic. T is such that each arc end at q.

5. If S + T is empty then goto step 8.

6. Compute the minimum of cost (p, q) such that min{cost(p, q) : (p, q) is an element of S + T}. Call the minumum (p, q), m(p, q).

7. If the m(p, q) is an element of S then:
   a. Set the flow of m(p, q) to the upper bound of m(p, q).
   b. Subtract the upper bound of m(p, q) from the unsatisfied demand of q.
   c. The flow from the minimum node p (see above) flow to the artificial node is equal to itself minus the upper bound of m(p, q).
   d. Goto step 1.

   Else:
   a. Set the flow of m(p, q) to the satisfied demand of q.
   b. Set the flow of minimum node p to the artificial node equal to itself minus the unsatisfied demand of q.
   c. Set the unsatisfied demand to q to 0. Remove q from list of nodes with induced demand.
   d. Goto step 1.

8. U is the set of arcs (p, q) that go to q and have a flow from root to p of 0.

9. If |U| = 0 then goto step 12.

10. Compute the minimum of cost (p, q) such that min{cost(p, q) : (p, q) is an element of U}. Call the minumum (p, q), m(p, q).

11. If the upper bound of m(p, q) is less than the unsatisfied demand of q:
   a. Assign the upper bound of m(p, q) to the flow of m(p, q).
   b. Assign the unsatisfied demand of q minus the upper bound of m(p, q) to the unsatisfied demand of p. Place the p of m(p, q) onto the front of the list of nodes with induced demand.
   c. Goto step 1.

   Else:
   a. Set the flow of m(p, q) equal to the unsatisfied demand of q.
   b. Remove q from the list of nodes with induced demand.
   c. Place the p of m(p, q) onto the front of the list of nodes with induced demand.
   d. Set the unsatisfied demand of the above (11c) p to equal the unsatisfied demand of q. Then set the unsatisfied demand of q equal to 0.
   e. Goto step 1.

12. Remove q from the list of nodes with induced demand.

13. Create an artificial arc (a, q).

14. Set the cost and upper bound of (a, q) to infinity.

15. Assign the unsatisfied demand of q to the flow of (a, q).

16. Goto step 1.

17. Set q to 1. (This starts the node counter).

18. If all the node are connected to the tree then END.

19. Assign q to q'.

20. Let V be the set of arcs (p, q) such that p or q is not connected to the tree.

21. If the set V is empty then goto step 25.

22. Compute the minimum of cost (p, q) such that min{cost(p, q) : (p, q) is an element of V}. Call the minumum (p, q), m(p, q).

23. Assign a flow of 0 to the arcs of m(p, q) and then add the arcs of m(p, q) to the tree.

24. Increment the node counter (q). If q is greater than the number of nodes than set the node counter to 1 and goto step 18.

25. Increment the node counter (q) and then check if the node counter is greater than the total number of nodes. If so, then set the node counter to 1.

26. If all the nodes have not been examined goto step 20.

27. Let H be the set of nodes such that the nodes are an element of the network and not connected to the tree.

28. For every node that is in H, p, create an artificial arc from the artificial node to p with the cost of this arc equal to the upper bound which is set to the biggest value possible. The flow in this arc will then be zero.

This is the complete algorithm for the All-Artificial arcs method. By examining this algorithm alongside with the code, the logic of the program is evident. Figure 4.1 shows how the Artificial Arcs provide the final basis tree.

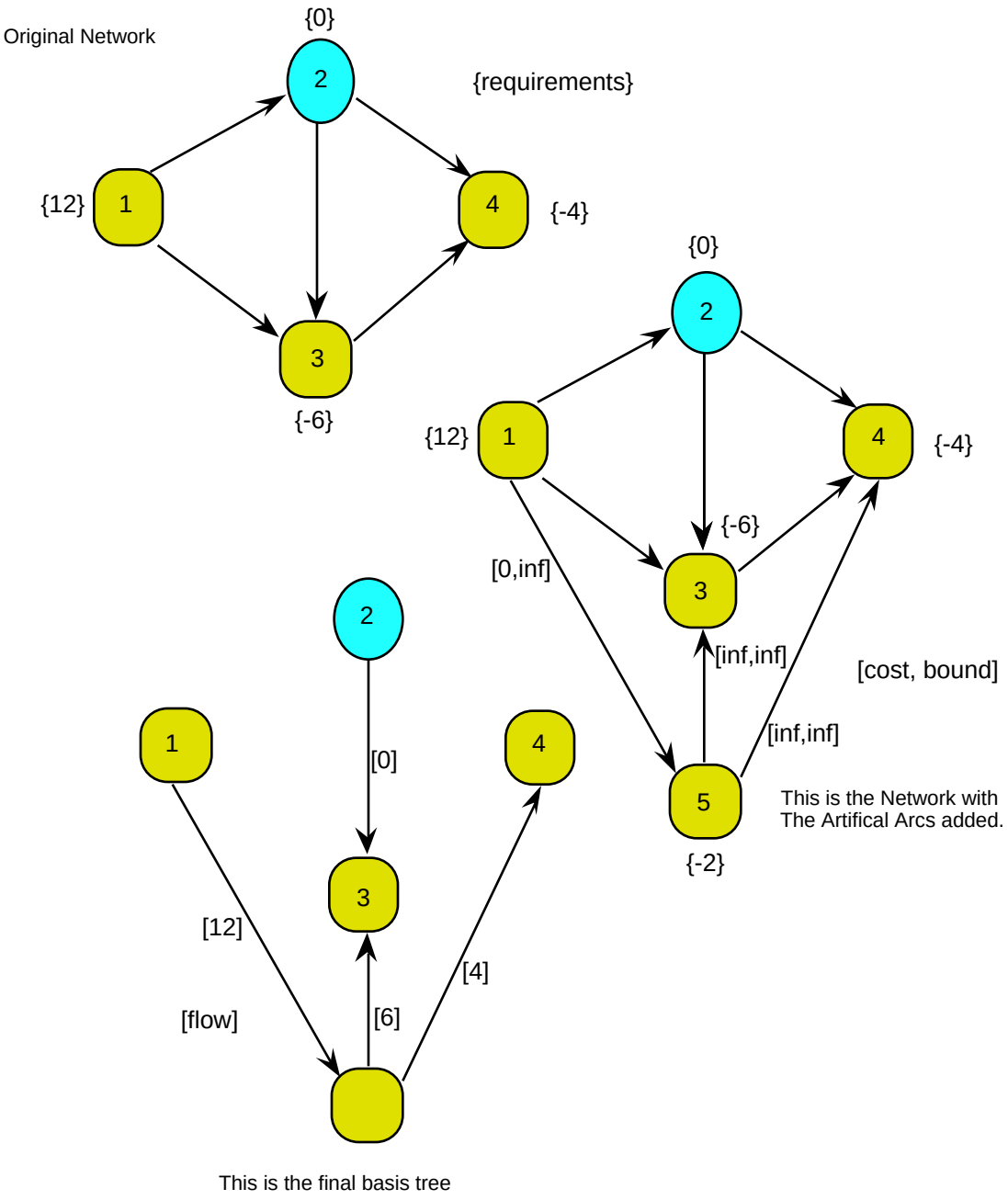


Figure 4.1: Algorithmic Definition

In its current form, a flexibility factor generates a series of upper and lower bound constraints to the detector arcs. Beginning with a restrictive regime, the NETFLO code is repeatedly run until a feasible solution is derived. This iterative procedure whereby the flexibility factor is systematically reduced has been demonstrated to be effective in increasing the accuracy of turning movement flow estimation (see Martin 1992).

The NETFLO algorithm needs "front end" and "back end" interfaces so that the whole of the TMERT model is coded. Hitherto, all input data had to be prepared at a preliminary stage. Spreadsheets have been used to calculate the various parameters needed by the model: network flow imbalances and error arc upper bound constraints. For each run of NETFLO, spreadsheets have been used to compare the turning movement arc output flows with those observed. This is a cumbersome and time consuming process. A "back end" will speed the efficiency of model testing to such an extent that larger, more complex networks will be evaluated. The validation of the model will be automated by coding the comparison of observed and modeled turning flows to provide a performance measure in the form of a correlation coefficient. The code which invokes the NETFLO algorithm, therefore, will be sandwiched between these two automatic procedure components. Large supplies of data, transmitted electronically from the University of Nottingham in England will serve to verify the reliability of the software improvements.

4.2 The "Front - End"

The "front-end" improvements all follow from the ease with which a spreadsheet format can be applied to the input data structures, a substantial advance from the former ASCII type files. Duplication, copying and pre-processing of repeated data sets could now be speedily implemented. Table 4.1 below shows the unimproved data structure for the Six Node Network. It lacks notation. The node-arc incidence matrix is an (8xn) matrix which is not easily comprehended.

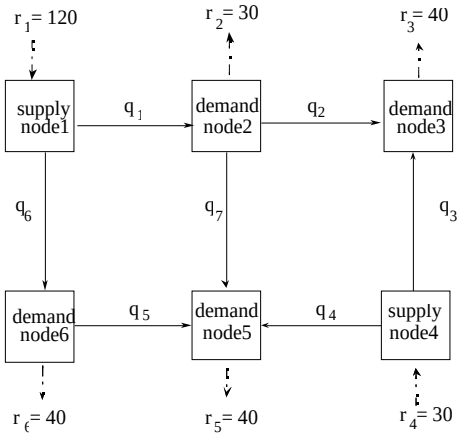


Figure 4.2 The Six Node Network

Table 4.1 Unimproved Data Input Structure for the Six Node Network - ASCII Format

```
7
6
1               120
2               -30
3               -40
4                30
5               -40
6               -40
        0       0
0       1       2       0       3       1
1       1       2       4       60      0
2       2       3       5       15      0
3       4       3       2       35      0
4       4       5       5       25      0
5       6       5       4       30      0
6       1       6       2       60      0
7       2       5       2       15      0
0       0       0       0       0       0
```

Table 4.2 shows how a spreadsheet input data structure has made the input both more easily interpreted and manipulated. Note that the awkward (8xn) node-arc incidence matrix has been replaced by a new “Node Precedence” column which is aligned with the node and requirement columns.

Table 4.2 Improved Data Input Structure for the Six Node Network - Spreadsheet Format

Number of arcs	7				
Number of Nodes	6				
Node	Requirement, r	Node Precedence			
1	120	0			
2	-30	1			
3	-40	2			
4	30	0			
5	-40	3			
6	-40	1			
check sum r =	0				
Terminator Card					
	0	0			
check sum node prec = nr arcs					
Arc	from	to	c	u	v
1	1	2	4	60	0
2	2	3	5	15	0
3	4	3	2	35	0
4	4	5	5	25	0
5	6	5	4	30	0
6	1	6	2	60	0
7	2	5	2	15	0
Terminator Card					
0	0	0	0	0	0

Appendix A contains a comprehensive example of model input and output in spreadsheet format.

4.3 The "Back - End"

Table 4.3 shows the unimproved output format for the Six Node Network. Note that the flows are output in an arbitrary order without arc labels.

Table 4.3 Unimproved Data Output Structure for the Six Node Network - ASCII Format

FROM	TO	COST	FLOW
1	1	0	0
1	2	240	60
2	3	75	15
4	3	50	25
4	4	0	0
4	5	25	5
6	5	80	20
2	5	30	15
1	6	120	60
0	0	0	0
0	0	0	0
0	0	0	0

Table 4.4 shows how the Improved data output structure in spreadsheet form with an Arc Label column to make the output more accessible.

Table 4.4 Improved Data Output Structure for the Six Node Network - Spreadsheet Format

Arc Num	From	To	Weight	Flow
1	1	2	240	60
2	2	3	75	15
3	4	3	50	25
4	4	5	25	5
5	6	5	80	20
7	2	5	30	15
6	1	6	120	60

4.4 Performance Index

The conversion to a spreadsheet structure for both input and output has permitted faster analysis of model performance through easier data input manipulation and faster output validation. Figure 4.3 depicts a graphical

representation of how output analysis has been streamlined through immediate automated spreadsheet analyses.

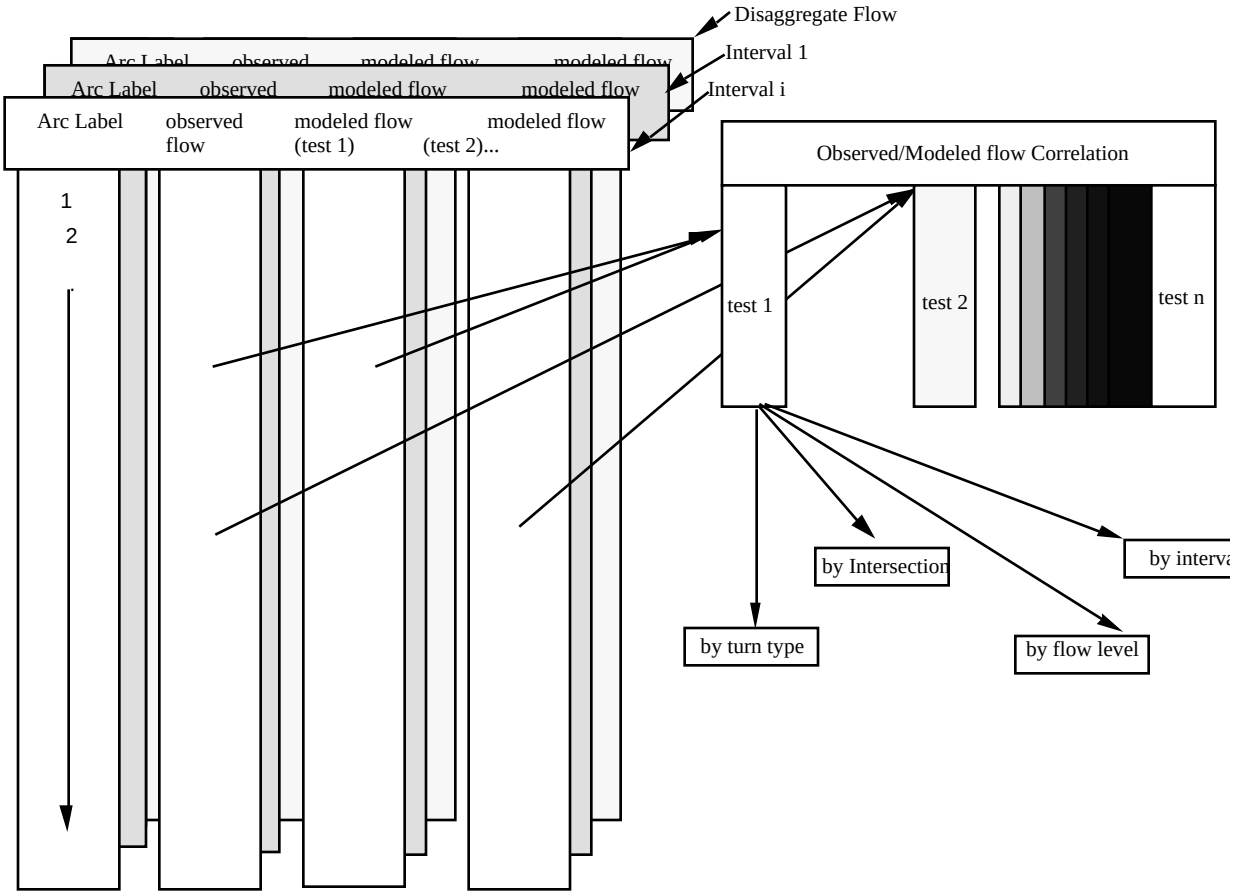


Figure 4.3 Performance Index Method

The output flows that the model provides are compared to the observed flows to determine the correlation of the values through the coefficient of determination (r^2%). Table 4.5 shows the vastly different flows that could be the actual flows of the network which also satisfy the external load constraints. This is why internal link detectors are needed to add another constraint. The number and location of the internal detectors needed to provide acceptable correlation between the modeled and observed flows is suspected to be related to the network size and the number of entrance and exit locations to the network.

4.5 Collection and Validation of English Data

The English Network is a SCOOT sub-area Region "R" of the City of Leicester in the heart of England. It had been selected because it has a spatial geometry which offers alternative routes to traffic both when entering and leaving the City Center. Figure 4.4 shows the Street Layout of the network with six signal controlled junctions.

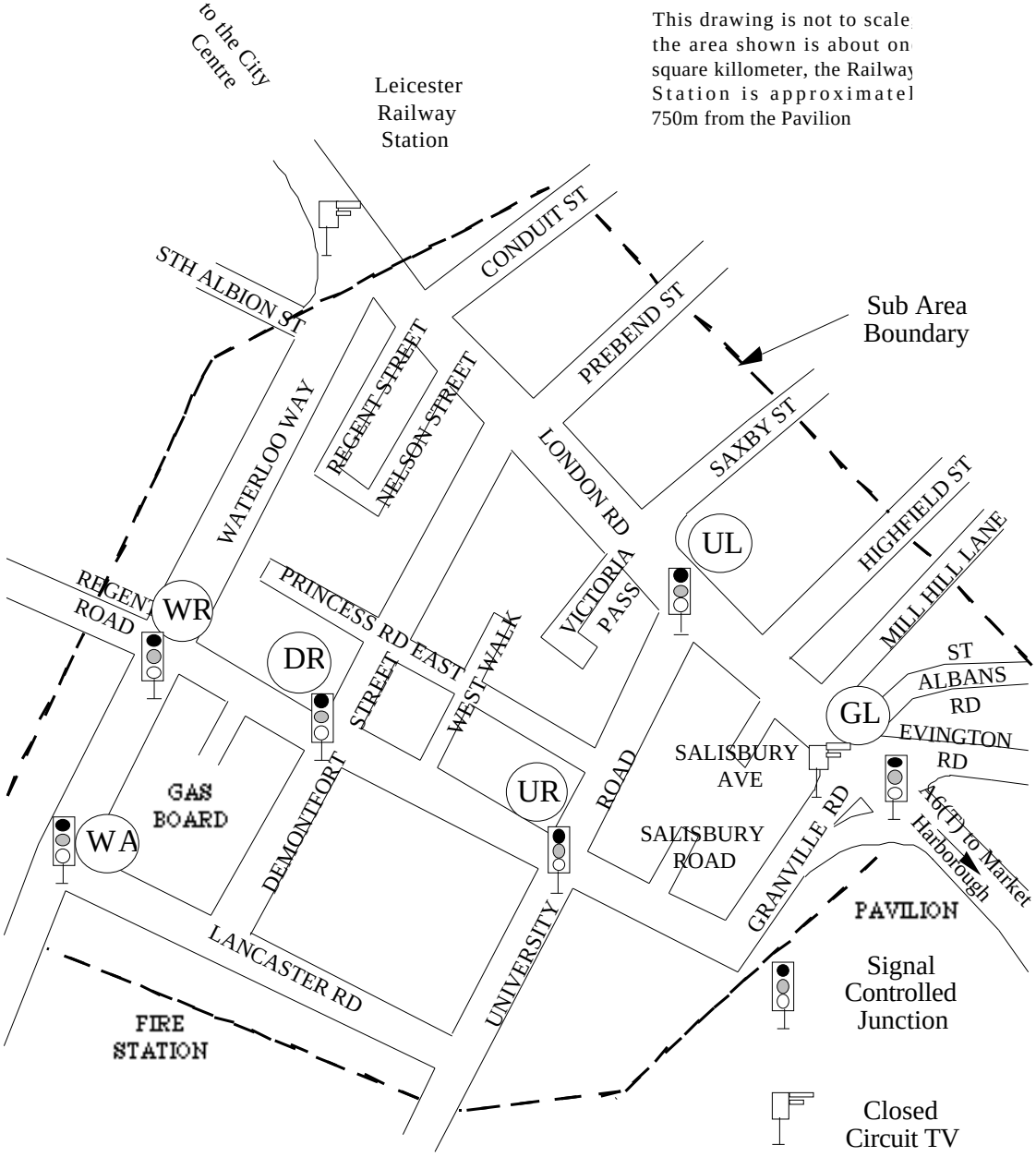


Figure 4.4 Street Layout - SCOOT Region R, Central Leicester

Input files from surveys made in May of 1990 and 1991 in Leicester were collected and tested with the renewed code. Output files from the earlier work were compared to those provided by the renewed code. The two sets of output proved identical.

Section 5 Model Development

5.1. Error Arc Definition

5.1.1. Detector Constraint

Fixing internal flows by constraints on the upper and lower bounds simulates the flow measured by the detector. The known detector flow is forced along the arc.

Let q_d = detected flow
and q_m = predicted flow

then the flows along this arc can be constrained thus:

$$v_d = q_d = u_d$$

where v_d = detector arc lower bound,
and u_d = detector arc upper bound,
so that $q_d = q_m$

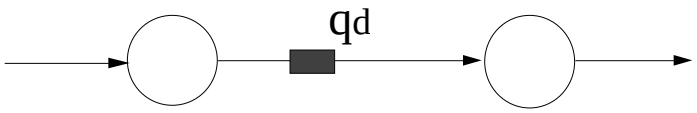


Figure 5.1 Detector Arc

5.1.2. Error Arc Constraints

The weight on each internal arc was set to unity and the minimum flow on each arc was fixed at zero. Each chain of arcs connecting junctions contains a detector arc which effectively controls the upper bounds of each of these internal arcs. NETFLO, however, must have an upper bound specified for each arc. The detector arcs are constrained by fixing upper and lower bounds thus:

$$v_d = q_d = u_d$$

Given the large number of constraints imposed by the more complex geometry of the real network, NETFLO was unable to find a feasible

solution. Bell et al (1987) proposed a method of introducing a flexibility system by introducing two additional "error arcs" (e^+ and e^-) alongside the arc representing the detected flow as shown below.

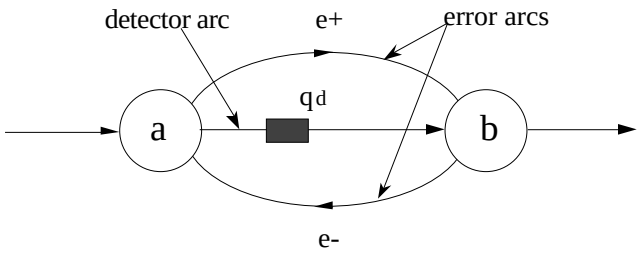


Figure 5.2 Error Arc Configuration

Flexibility is provided by two virtual arcs, or error arcs, which permit additional flows along arc e^+ or a reverse flow along arc e^- so that the detector flow from node a to node b is represented as:

$$q(a \text{ to } b) = [q_d + q(e^+) - q(e^-)]$$

where q_d = known detector flow

and $[q_d + q(e^+) - q(e^-)]$ = modified detector flow

where $q(e^+)$ and $q(e^-)$ are defined as error arc flows. The model is constrained to allocate flows, so that:

$\Sigma(q(e^+) - q(e^-))$ is a minimum, for all arcs in the network.

The error arcs provide flexibility and a means of achieving a feasible solution. The manipulation of the upper and lower bounds and weights of the error arcs provide a control mechanism to ensure a feasible solution. The following sections describe the set of tests which were carried out to assess the influence of error arc upper bound, lower bound and weight. The modified detector flows are compared with the detector flows.

The parameter, ϕ, is defined such that,

$$\phi = \frac{u_e}{q_d}$$

where u_e is the upper bound of each error arc.

5.1.3. ϕ Factor Constraint

The model is constrained tighter by the reduction of the amount of flow allowed to flow along the error arcs. This is accomplished through incremental reduction of the ϕ factor. The upper bounds of the error arc, u_e, for any ϕ value is found by multiplying the ϕ factor value by the detector arc flow.

$$U_e = \phi * q_d$$

The ϕ factor values were calculated using an inverse square relationship.

The ϕ factor value is defined by $\Phi = \frac{1}{\delta^2}$

where δ takes the values tabulated below:

Table 5.1: Incremental ϕ Factor

increment	δ	ϕ
1	1.00	1.000
2	1.15	0.750
3	1.33	0.563
4	1.54	0.422
5	1.78	0.316
6	2.05	0.237
7	2.37	0.178
8	2.74	0.133
9	3.16	0.100
10	3.65	0.075
11	4.21	0.056
12	4.87	0.042
13	5.62	0.032
14	6.49	0.024
15	7.49	0.018
16	8.65	0.013
17	9.99	0.010

Figure 5.3 indicates the path of the incremental feasibility method used to tighten the constraint on the model by lowering the upper bounds limit of the error arcs through a step interval in which the ϕ factor is reduced.

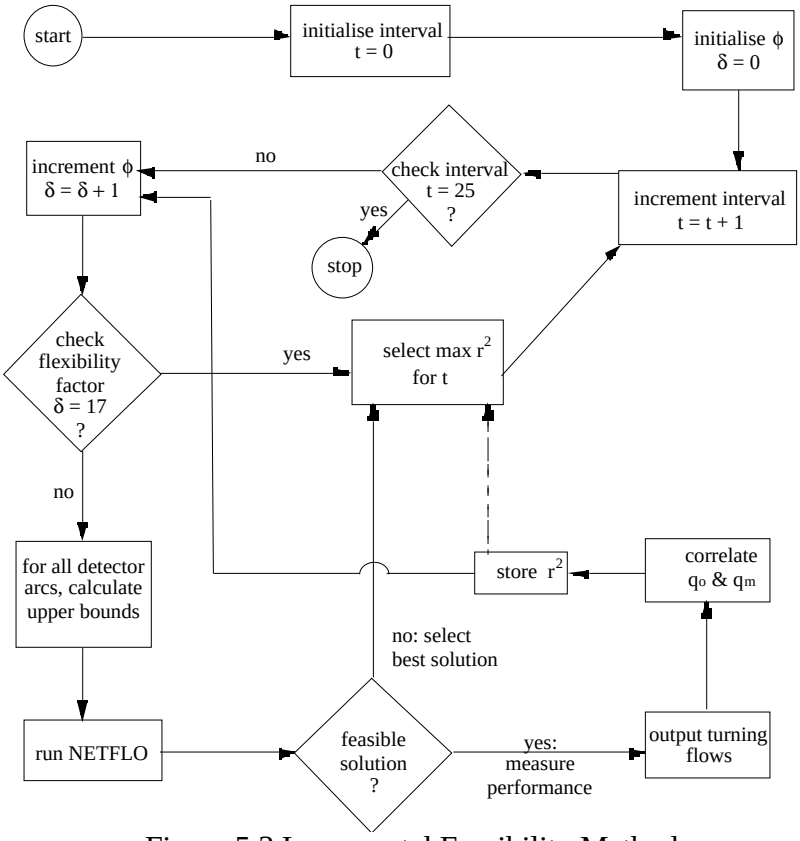


Figure 5.3 Incremental Feasibility Method

5.2. Detector Placement and Number

The City of San Luis Obispo neither has nor needs the high degree of detection associated with an adaptive signal control system. For model development, therefore, on-street flow detection was simulated. Links were identified as detector links on the assumption that the network was a sub-area of a SCOOT controlled network. Intersection turning movement flows were then aggregated to provide simulated detector flows from selected links.

The final detector layout configuration and network used for the remainder of the analysis, is seen in Figure 5.4.

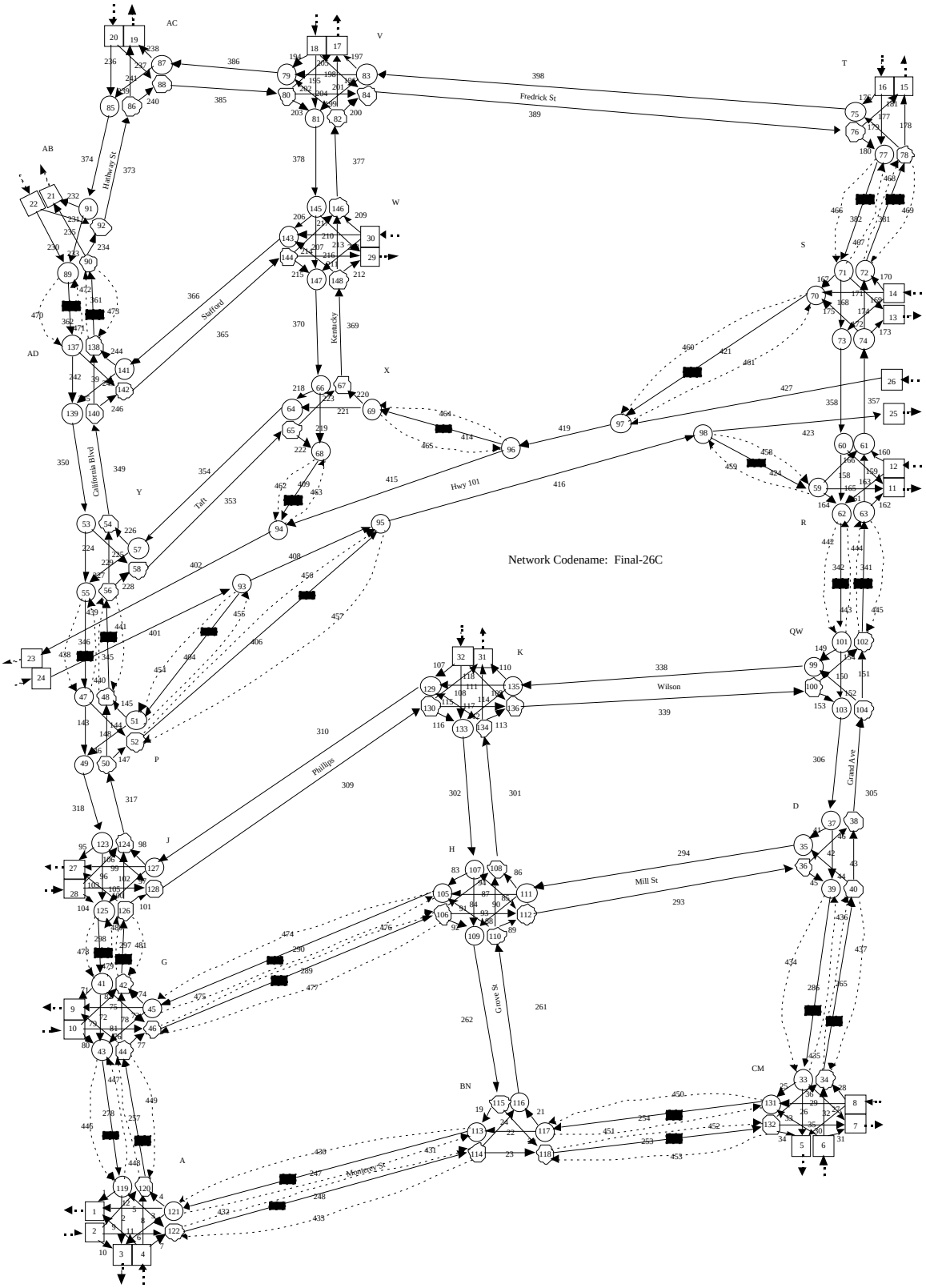


Figure 5.4: Final Detector Layout

Section 6 Model Performance

The SLO network was modeled. Observed turning movements were compared to observed. The Performance Index supplied coefficient of determination (100r2%) for the constraint regime established for the English network (see Martin and Bell 1992a). The flows for the full two hour intervals were aggregated with an error arc upper bound constraint factor ϕ of 0.025. The correlations are summarized below:

Table 6.1: Coefficient of Determination of Observed and Modeled Turning Movement Flows, 2-hr aggregated, ϕ set to 0.025

Turn	English Network	r^2 (%) SLO Network
Overall	92%	90%
Through	94%	89%
Cross-flow (Left US)	83%	66%
With-flow (Right US)	2%	67%

6.1 Varying The Weights Of Arcs

Model development continued with an investigation into the effects of applying different weights to the different arc types. Four different weighting schemes were used in the analysis of TMERT. The more difficult movements were weighted higher than the simpler movements. The order from least weighted to most was determined to be: Link, straight, protected left, right, permitted left, error arcs with left protected and right equivalent for all cases but W3 weighting regime. Table 6.2 shows the weight assigned to each arc type for each of the different trials and the maximum r^2 value obtained for that weight case.

Table 6.2: Arc Weights for each Trial

Arc	**W0**	**W1**	**W2**	**W3**
Link	1	1	1	1
Through	1	2	3	2
Left Protected	1	3	6	3
Right	1	3	6	4
Left Permitted	1	5	9	7
Error	3	15	30	20
2 hr--r^2 %	90.37	96.65	96.31	96.60

The logic applied to the weighting schemes was that driving on a link was the least difficult of all the maneuvers. The next simplistic maneuver was a through movement at an intersection. A protected left and unprotected right were considered to be of the same difficulty except for W3 were the unprotected right was given a 4 weighting while the protected left was given a weight of 3. Left permitted turns were weighted most heavily of all turning movements because of the conflict with both oncoming vehicles and crossing pedestrians. Error arcs were the most weighted arc being approximately 3 times the value of the next highest arc as set forth by Martin and Bell (1992b).

6.2 Coefficient of Determination (r^2)

The Performance Index supplied analyses of the models overall performance, intersection by intersection, and by turning movement. A two hour aggregate test, from 4:30-6:30 PM, was performed on the four different weighting schemes. The incremental increase of the error arc upper bound factor ϕ was applied. This overall performance of the model is shown in Table 6.3.

Table 6.3: Coefficient of Determination of Observed and Modeled Turning Movement Flows, 2-hr aggregated, ϕ increasing for various weight regimes - for all Turning Movements

All Turns ϕ	W0	W1	W2	W3
1.000	87.47%	95.64%	96.31%	96.60%
0.750	89.58%	96.65%	96.31%	96.52%
0.563	87.89%	96.49%	96.31%	96.57%
0.422	88.31%	95.39%	94.90%	95.11%
0.316	87.55%	95.64%	95.33%	95.55%
0.237	85.94%	95.39%	94.90%	95.14%
0.178	88.71%	95.59%	94.90%	95.14%
0.133	90.20%	95.39%	94.90%	95.12%
0.100	89.06%	95.66%	95.14%	95.14%
0.075	90.29%	95.70%	95.15%	95.15%
0.056	90.37%	95.52%	94.96%	95.17%
0.042	89.84%	95.55%	95.18%	95.19%
0.032	88.30%	95.74%	95.35%	95.35%
0.024	Infeas	Infeas	Infeas	Infeas

Infeas = Infeasible

Underlined values are the maximum r^2 obtained associated with ϕ

These results show:

- the higher the weighting regime, the better the correlation between observed and modeled flows
- some form of weighting is better than none
- the incremental increase of ϕ has a marginal impact upon model performance

6.3 Turning Movements

Tables 6.4 to 6.6 represent how the right and left turns were modeled. They all indicate improved model performance with higher weighting regimes.

Table 6.4: Coefficient of Determination of Observed and Modeled Turning Movement Flows, 2-hr aggregated, ϕ increasing for various weight regimes - for Right Turns

Right Turns ϕ	W0	W1	W2	W3
1.000	51.74%	80.37%	79.92%	82.60%
0.750	63.12%	85.69%	79.92%	80.83%
0.563	50.36%	84.01%	79.92%	82.56%
0.422	62.76%	76.98%	71.42%	72.31%
0.316	53.18%	80.37%	76.03%	76.88%
0.237	48.18%	76.98%	71.42%	72.35%
0.178	58.43%	77.73%	71.42%	72.35%
0.133	65.77%	76.98%	71.42%	72.25%
0.100	62.06%	77.96%	72.39%	72.39%
0.075	63.25%	77.97%	72.29%	72.27%
0.056	63.51%	77.08%	71.31%	72.14%
0.042	51.66%	76.14%	72.57%	72.51%
0.032	50.65%	77.89%	74.06%	74.05%
0.024	Infeas	Infeas	Infeas	Infeas

Infeas = Infeasible

Underlined values are the maximum r^2 obtained associated with ϕ

The right turning movements obtained a maximum r^2 term of 65% when no weighting method was applied to the network but an 80% r^2 term was found for all three weighting schemes.

Table 6.5: Coefficient of Determination of Observed and Modeled Turning Movement Flows, 2-hr aggregated, ϕ increasing for various weight regimes - for Left Turns

Left Turns ϕ	r^2(%) W0	W1	W2	W3
1.000	46.49%	30.33%	46.05%	53.41%
0.750	61.15%	52.19%	46.05%	54.95%
0.563	52.70%	44.88%	46.05%	53.19%
0.422	51.28%	30.33%	30.46%	39.46%
0.316	42.74%	30.33%	30.46%	39.46%
0.237	18.89%	30.33%	30.46%	39.46%
0.178	12.13%	39.36%	30.46%	39.46%
0.133	15.38%	30.33%	30.46%	39.46%
0.100	11.72%	39.24%	39.29%	39.29%
0.075	13.38%	39.17%	39.11%	39.12%
0.056	13.44%	30.20%	30.03%	39.01%
0.042	67.06%	36.59%	37.89%	38.02%
0.032	11.68%	38.28%	39.20%	39.19%
0.024	Infeas	Infeas	Infeas	Infeas

Infeas = Infeasible

Underlined values are the maximum r^2 obtained associated with ϕ

The aggregate test on the left turning movements indicates that the models ability to estimate left turns is poor. An average of 50% r^2 was obtained for the weighted schemes while the non weighted scheme produced a 67% r^2.

Table 6.6: Coefficient of Determination of Observed and Modeled Turning Movement Flows, 2-hr aggregated, ϕ increasing for various weight regimes - for Through Movements

Through ϕ	r^2(%) W0	W1	W2	W3
1.000	86.42%	95.76%	96.39%	96.44%
0.750	87.70%	96.33%	96.39%	96.39%
0.563	86.43%	96.43%	96.39%	96.39%
0.422	87.08%	95.63%	95.34%	95.34%
0.316	85.81%	95.76%	95.63%	95.65%
0.237	85.38%	95.63%	95.34%	95.38%
0.178	87.80%	95.64%	95.34%	95.38%
0.133	88.21%	95.63%	95.34%	95.35%
0.100	88.33%	95.72%	95.39%	95.39%
0.075	89.05%	95.79%	95.43%	95.43%
0.056	88.96%	95.84%	95.47%	95.47%
0.042	87.63%	95.83%	95.52%	95.53%
0.032	87.33%	95.97%	95.66%	95.67%
0.024	Infeas	Infeas	Infeas	Infeas

Infeas = Infeasible

Underlined values are the maximum r2 obtained associated with f

The through turning movements are by far the most accurately estimated movement. The non weighted scheme produced results of 89% for the r^2 term and above 96% for each of the weighted schemes. Table 6.7 summarizes the maximum 2 hour aggregated model performance by turning movement.

Table 6.7: Maximum Model Performance by Turn, 2-hr Analysis

Turn	W0	W1	W2	W3
Overall	90.37	96.65	96.31	96.60
Through	89.05	96.43	96.39	96.44
Right	65.77	85.69	79.92	82.60
Left	67.06	52.19	46.05	54.95

The results indicate that a logical weighting method is desirable. The overall inference performs well with a 96% correlation between the observed and modeled flows. The through movements were also highly inferred by the model achieving a 96% correlation. The right turns were fairly accurately estimated with a 85% correlation. The left turns were less accurately estimated with a 52% correlation. The results indicate that the optimum r^2 value does not occur when the model is most constrained (ϕ is the smallest). Figures 6.1 to 6.4 illustrate the correlation relationship for the 2 hour interval of the W3 weighting scheme. Figure 6.1 depicts the models overall correlation of the modeled turning movements and the observed turning movements. A 96% r^2 correlation exists for the overall comparison.

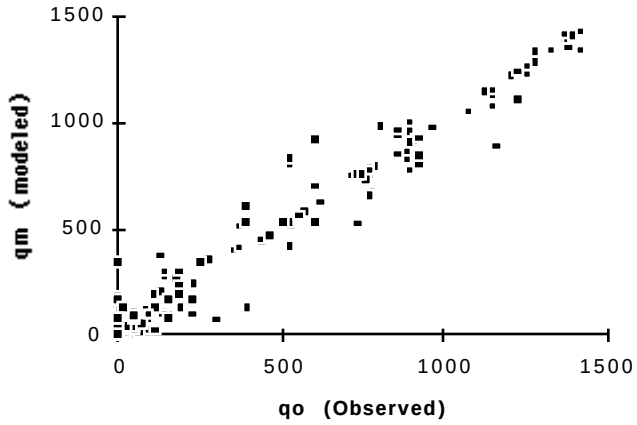


Figure 6.1: Overall Correlation for 2 hr Interval

Figure 6.2 shows the through correlation which acquired a 96% r^2 term.

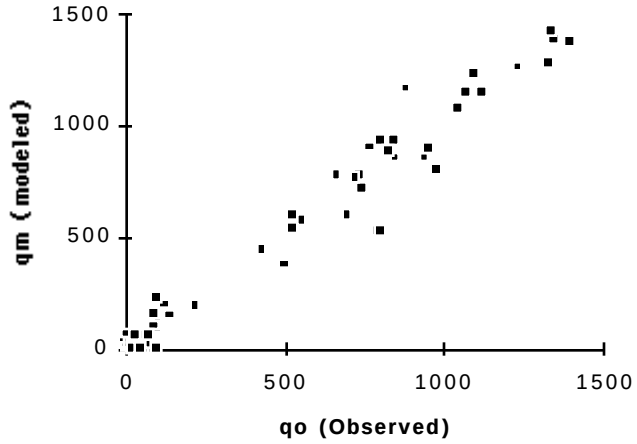


Figure 6.2: Through Movement Correlation for 2 hr Interval

Figure 6.3 shows the right r^2 correlation of 85% is less accurate than TMERT's modeling of the through movements.

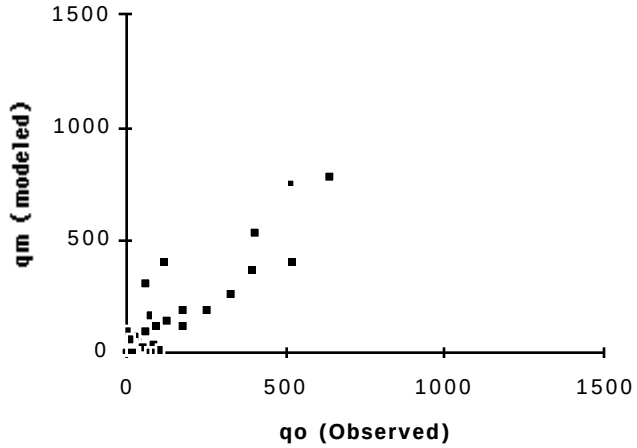


Figure 6.3: Right Movement Correlation for 2 hr Interval

Figure 6.4 shows that the model's ability to estimate the left movements was poor with an r^2 of 52% correlation.

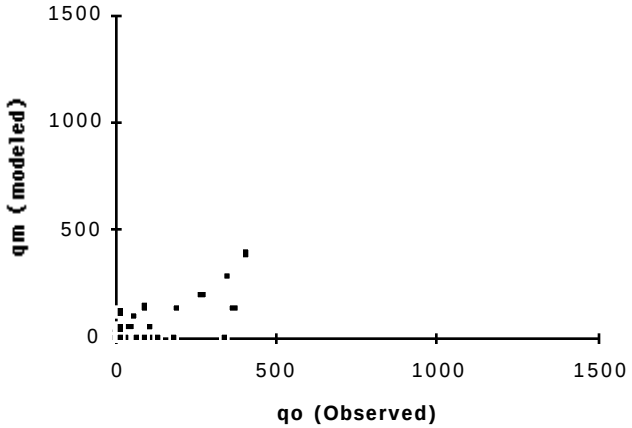


Figure 6.4: Left Movement Correlation for 2 hr Interval

6.4 Dynamic 5-minute Modeling

TMERT's quick execution time is what allows it to be a real time application. Two hours and 25 minutes, Twenty-nine 5 minute intervals, of data were analyzed for the W0 and W3 weighting regimes. The W1 and W2 weighting regimes were evaluated for 1/2 of an hour, (six 5 minute intervals from 4:55 - 5:25), to look at how they compared with W3. The model was again evaluated for its overall correlation and its performance by turning movement.

Table 6.8 depicts the overall 5 minute analysis for the non weighted hierarchy (W0). A maximum difference of 14% is found between the maximum and minimum r^2 terms. Of the 29 intervals, in 7 the maximum r2 correlation occurs at the last ϕ factor interval before unfeasibility occurs. Of the twenty-nine, seven of the time intervals also produce the minimum r^2 term for the first ϕ factor interval. 0 maximum r^2 terms appear in the first ϕ factor interval and 3 r^2 minimums occur last. In only seven of the twenty-nine time intervals is the first r^2 term larger than the last r^2 term.

Table 6.8 Dynamic Modeling Analysis by 5 minute Interval - All Turning Movements, Weighting Regime W0

Int	r2 (%)																	Max	Min
	ϕ= 1.00	ϕ= .750	ϕ= .563	ϕ= .422	ϕ= .316	ϕ= .237	ϕ= .178	ϕ= .133	ϕ= .100	ϕ= .075	ϕ= .056	ϕ= .042	ϕ= .032	ϕ= .024	ϕ= .018	ϕ= .013	ϕ= .010		
4:05	86%	86%	86%	85%	86%	89%	89%	89%	88%	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	89%	85%
4:10	86%	86%	89%	86%	90%	87%	89%	87%	84%	88%	87%	87%	Inf	Inf	Inf	Inf	Inf	90%	84%
4:15	85%	84%	84%	88%	89%	86%	88%	88%	93%	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	93%	84%
4:20	81%	81%	83%	83%	82%	87%	85%	85%	87%	85%	85%	84%	Inf	Inf	Inf	Inf	Inf	87%	81%
4:25	82%	82%	84%	83%	87%	82%	85%	84%	85%	86%	84%	85%	Inf	Inf	Inf	Inf	Inf	87%	82%
4:30	81%	81%	82%	90%	89%	91%	83%	83%	82%	78%	84%	93%	93%	Inf	Inf	Inf	Inf	93%	78%
4:35	88%	90%	86%	92%	94%	92%	92%	87%	87%	91%	91%	Inf	Inf	Inf	Inf	Inf	Inf	94%	86%
4:40	91%	91%	91%	92%	91%	93%	93%	92%	93%	94%	94%	95%	95%	Inf	Inf	Inf	Inf	95%	91%
4:45	90%	90%	87%	86%	87%	88%	90%	92%	91%	91%	90%	91%	91%	Inf	Inf	Inf	Inf	92%	86%
4:50	85%	85%	88%	87%	86%	87%	85%	88%	87%	88%	88%	85%	84%	86%	85%	Inf	Inf	88%	84%
4:55	79%	83%	78%	85%	85%	88%	85%	86%	87%	82%	81%	Inf	Inf	Inf	Inf	Inf	Inf	88%	78%
5:00	87%	88%	87%	92%	88%	88%	89%	89%	89%	88%	88%	86%	Inf	Inf	Inf	Inf	Inf	92%	86%
5:05	87%	85%	88%	90%	89%	89%	90%	88%	90%	92%	Inf	Inf	Inf	Inf	Inf	Inf	Inf	92%	85%
5:10	85%	85%	94%	90%	87%	91%	88%	86%	87%	90%	88%	88%	88%	Inf	Inf	Inf	Inf	94%	85%
5:15	87%	87%	87%	87%	85%	86%	87%	86%	89%	89%	92%	Inf	Inf	Inf	Inf	Inf	Inf	92%	85%
5:20	85%	83%	87%	81%	82%	86%	89%	90%	91%	87%	88%	87%	82%	80%	82%	84%	Inf	91%	80%
5:25	78%	78%	83%	79%	77%	83%	85%	84%	85%	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	85%	77%
5:30	86%	86%	86%	85%	88%	85%	87%	88%	92%	90%	91%	Inf	Inf	Inf	Inf	Inf	Inf	92%	85%
5:35	88%	88%	87%	88%	85%	84%	83%	86%	85%	86%	87%	Inf	Inf	Inf	Inf	Inf	Inf	88%	83%
5:40	84%	88%	91%	90%	84%	89%	88%	86%	92%	91%	Inf	Inf	Inf	Inf	Inf	Inf	Inf	92%	84%
5:45	88%	90%	89%	88%	88%	87%	87%	87%	88%	86%	88%	88%	86%	Inf	Inf	Inf	Inf	90%	86%
5:50	87%	89%	87%	88%	88%	89%	89%	88%	90%	90%	90%	91%	91%	91%	Inf	Inf	Inf	91%	87%
5:55	90%	89%	89%	91%	93%	93%	90%	90%	90%	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	93%	89%
6:00	90%	90%	89%	91%	92%	87%	87%	88%	88%	87%	Inf	Inf	Inf	Inf	Inf	Inf	Inf	92%	87%
6:05	89%	89%	85%	86%	90%	89%	90%	88%	88%	90%	90%	90%	Inf	Inf	Inf	Inf	Inf	90%	85%
6:10	83%	83%	84%	84%	84%	85%	84%	84%	86%	84%	83%	83%	83%	83%	Inf	Inf	Inf	86%	83%
6:15	88%	87%	89%	88%	88%	88%	92%	88%	90%	90%	Inf	Inf	Inf	Inf	Inf	Inf	Inf	92%	87%
6:25	84%	86%	87%	90%	87%	89%	85%	88%	85%	88%	87%	89%	89%	91%	Inf	Inf	Inf	91%	84%
6:30	88%	88%	86%	90%	83%	88%	85%	88%	82%	84%	83%	Inf	Inf	Inf	Inf	Inf	Inf	90%	82%
4:30 6:30	87%	90%	88%	88%	88%	86%	89%	90%	89%	90%	90%	90%	88%	Inf	Inf	Inf	Inf	90%	86%
																	mean	91%	84%

Table 6.9 shows the mean maxima and mean minima for the 5 minute intervals of each of the 4 weighting schemes, 3 weighted and 1 non weighted. The overall and each individual turning movement, Left, Through, and Right, are calculated. The means for the W0 and W3 weighting regimes are presented for the entire 29 intervals analyzed. The means for the W1 and W2 weighting regimes are obtained from the six 5 minute intervals investigated from the intervals modeled between 4:55-5:25.

Table 6.9: Average Max and Min r^2 Value for the Movements of the 5 minute Intervals

Movement	Weighting	r^2(%) Max	r^2(%) Min
Overall	W0	90.65%	84.14%
	W1	94.69%	92.26%
	W2	94.42%	92.46%
	W3	94.45%	92.58%
Left	W0	52.00%	21.72%
	W1	44.88%	24.08%
	W2	45.25%	27.61%
	W3	48.92%	32.82%
Rights	W0	67.68%	41.43%
	W1	77.62%	62.25%
	W2	71.40%	60.22%
	W3	72.53%	61.02%
Through	W0	89.97%	82.85%
	W1	93.60%	91.67%
	W2	93.55%	92.02%
	W3	93.87%	92.24%

The dynamic modeling indicates that TMERT has the ability to infer turning movement flows in real time. Every 5 minute time interval has a maximum r^2 value of above 90%. The largest change between maximum and minimum r^2 was less than 5 % indicating that changing the ϕ factor value has little effect on the model. The 5 minute detailed analyses of the individual turning movements: Through, Right, and Left, for each of the W0, W1, W2 and W3 weighting regimes can be found in Appendices E through G.

Section 7 Discussion and Conclusion

In Section 7 the model performance is critically examined. The principal findings of the research are summarized. The report closes with suggestions for future work.

7.1. Model Formulation

As with all models which are based on a linear program, the real question is one of problem formulation, rather than the mathematical solution. The TMERT model has the NETFLO algorithm, which is a network form of a minimum cost flow problem, at its mathematical core because it is computationally very quick. The assumptions and approximations which clothe the NETFLO code formulate the problem to constitute the TMERT model. The formulation encompasses a network notation, detector flow simulation, flow modeling of internal sources and sinks and a means of constraining flow throughout. The method of constraining the network, called the constraint regime, is at the heart of the TMERT model.

When the constraint regime is too severe, no feasible solution can be found. When the constraint regime is lax, feasibility is readily accomplished, but a lax constraint regime estimates flows which do not correlate well with observed flows. So the challenge of the research presented here has been to constrain the model as tightly as possible while retaining a feasible solution. This had to be done in such a way that all the information available was used in a correct and sensible way so that a particular feasible solution was derived.

It is important that this constraint mechanism conforms to some logical structure which will enable the model to be both repeatable and transferable. The model should perform consistently on different data sets for it to be repeatable. For a model to be transferable, it should perform consistently for a different network. Repeatability has been demonstrated. The TMERT model been shown to perform consistently on data from English surveys in 1990 and 1991, for both aggregated one hour flows, and pseudo-dynamically using five minute flows for both years. It has now been shown

to be transferable, through its ability to estimate turning movement flows on a completely different US network surveyed in 1994.

7.2. Model Limitations

There are several limitations associated with the assumption that a linear relationship prevails between traffic flows on arcs which enter and leave nodes. The formulation of the model assumes that arcs accept and discharge flows with infinite elasticity. There is no recognition of the time taken for an arc to discharge a queue. So the nodal continuity principle can be accepted provided the modeling interval is long enough to permit flows to be absorbed and discharged before the onset of the next modeling interval. The longest arc, therefore will govern the minimum interval. Furthermore, as the modeling interval is reduced, it approaches the journey time across the network. In the limit, TMERT could be trying to model a picture of traffic activity which is changing more rapidly than the small time frame permits. At this point, the modeling intervals can no longer be regarded as independent populations. Intuitively, an interval shorter than say 30 seconds, would not be sensible. While the model can fairly be described as a real-time model, therefore, a stepped or interval approach must be preserved with flows estimated pseudo-dynamically. It could not perform continuously.

The TMERT model was fed simulated detector flows every five minutes. The turning movement capacities are derived from signal timings which are calculated from fixed parameters such as saturation flows and lost times, and variable parameters such as cycle time and green time. A five minute modeling interval will inevitably give on-line turning movement capacities which include truncated cycles. This means that a truncated cycle time is likely to contain a truncated green time and a missing portion of lost time from either the start or end of one stage. The turning movement capacities, therefore, suffer cyclic irregularities which depend upon the temporal relationship between the modeling interval and the cycle time encompassed. This problem could be obviated by synchronizing the modeling interval with the cycle time or multiples of the cycle time. For an adaptive signal control system such as SCOOT, this would mean that the model would have to follow changing cycle times and, like SCOOT, operate for groups of signals

with common cycle times. The TMERT model could be set up to estimate turning movement flows in a cellular structure relating to SCOOT sub-areas, but this would detract from its strategic application.

The external flows entering and leaving the network, modeled as Cordon Node flows, were supplied by survey data. A practical demonstration of the TMERT model in real time, would need detectors associated with each cordon node, including side roads. However, the TMERT model could be applied to an extended network with a cordon which circumscribed a whole city, crossing arterial roads with detectors, whereby lower status roads could be avoided.

The flows which are generated and attracted internally, are mostly due to small local parking facilities. Their magnitude relates to parking capacity. A large capacity office car park, for example, would generate high flows, while a small group of on-street parking spaces with short stay parking restrictions would earn low flows. If the large car park were to fill up during the morning peak and empty during the evening peak, with no activity during the day, while the on-street parking spaces were busy all day with short stay visitors, then unrealistic flows would be estimated. So, further developments of the model would implement some form of energy index as a measure of the liveliness of a parking facility to simulate source and sink flows more reliably.

The flows and constraints within the SLO network are subject to inconsistencies because of averaging data over modeling intervals. Internally generated flows are approximations, while detected flows have to co-exist with turning movement capacities which are derived from variable signal settings. The error arcs associated with each detector seek to accommodate these inconsistencies enabling feasible solutions to be found. In its working form, TMERT would be installed so that its error arcs would also have to absorb the effect of noisy calibrated detector flows. While the model presented has been shown to cope with the inconsistencies turning movement capacities, the simulation of detector flows means that it has yet to be burdened with real detector irregularities. Furthermore, in its current state of development, there is no means of either identifying or quantifying the degree of inconsistency generated by each of the foregoing disruptive elements. The

optimum error arc upper bound factor, however, provides an indication of the degree of adjustment which has been necessary to provide a particular feasible solution.

7.3. Applications of the TMERT Model

As technology advances with more sophisticated communication methods, data monitoring in co-ordinated signal control is increasing in volume and accuracy. The manipulation of information through relational data bases is developing in its range and subtlety so that processed data is readily available in real time. The research presented here provides a mechanism which enables these developing technologies to be exploited.

The TMERT model could not only take information from a database such as flow and capacity, but could continually feed a database with turning movement flow estimates. This long term collection of link flows, supplemented by estimated turning flows, could provide flow patterns which could be continuously compared with on-line flows and estimates. This could lead to the identification of the onset of congestion. With knowledge of link and turning movement capacities supplied through flow data and signal settings, the turning movement estimates could enable alternative routes to be identified. At this point, traffic could be controlled by route guidance through variable message signs, in-car guidance, radio broadcasts or even mandatory signal control.

The TMERT model could be used as a modeling component of an on-line traffic control device. As the model uses flows, capacities and estimates of turning movement flows, it could then enable the identification of the spare capacity in a network which would prove valuable for re-routing vehicles. There could be a quick source of advice on optimum routes for diversions following an incident. These would be derived immediately before the incident. Since the estimates would be no older than the modeling interval of five minutes or perhaps a multiple of cycle time, then a traffic control center could relay alternative routes to drivers, in real time. In the early implementation of a prototype TMERT model, this information could be advisory, to drivers of emergency services and later to all drivers.

Alternatively, as an off-line tool, the knowledge base could be used to model both recurrent congestion and to develop a set of remedial fixed-time signal plans for sudden incident response which could be set up with TRANSYT using link flows from SCOOT and turning proportions supplied by the TMERT model.

7.4. The Future of Area Traffic Signal Control

Traffic control has developed from the simple expedient of keeping drivers to one side of the road, through isolated signal control to dynamically responsive systems which respond to the demand for travel made by drivers. Perhaps Area Traffic Signal Control is an inappropriate term because the widely implemented dynamically responsive UTC Systems such as SCOOT and SCATS monitor link flows and respond, rather than control. While drivers may be delayed by red signal aspects to optimize network performance, their choice of route remains sovereign. In contrast to this type of control philosophy, commercial air traffic, for example, is quite different. Airline pilots have no route choice, except in emergencies, and are controlled from the ground by a series of Air Traffic Control Centers. Responsibility for the height, speed and bearing of the aircraft is passed from one air traffic control center to the next.

The research detailed in this report shows that turning movement flows may be estimated in real time without the need to identify individual vehicles. This could become an essential component of a new generation of traffic control system. This contributes vital information to significantly the enhance traffic monitoring, which in time could lead to better control, and perhaps provides the foundation for a fourth generation of traffic control systems. These advanced systems would need a complete description of the signals and other traffic control devices over an entire city network and surrounding area and a comprehensive set of criteria and rules defining traffic constraints and policy. Prior to congestion, dynamically responsive systems would provide control, while beyond congestion, TMERT could be implemented to control through route guidance and restraint, through ramp and meter implementation, with traffic restrained from using critical links and junctions.

However, the enthusiasm for improving the capacity of urban traffic networks through optimizing time and space, with ever more sophisticated flow optimization techniques must be tempered with caution. By maximizing the volume of traffic in an urban network, there is a danger that incidents could lead to catastrophic congestion within what can be described as a "highly strung" network. The congestion following an incident in a maximized network could lock up an entire traffic system for hours giving rise to the "super-jam".

7.5 Conclusions

The model has shown its ability to apply the NETFLO algorithm to minimize a weighted objective function to balance nodal continuity throughout a network and accurately estimate turning movements. TMERT has also shown it repeatability on a second network producing correlation coefficients of determination (r^2) of above 90%. This correlation varies between 90 and 96% depending on the weighting hierarchy used in describing the network. The analysis showed that all 3 of the different weighting systems produced maximum correlation of 96% which was approximately 6% better than the no weighting hierarchy with a correlation of 90.4%. This indicates that some type of weighting system should be applied to the network to differentiate the difficulties of the various maneuvers.

Martin (1992) had obtained an overall correlation of 92% on a 1 hour aggregate test of flow data. This study provided correlations of up to 96.65% between observed and modeled turning movement flows depending on the weighting hierarchy. The current analysis also provided higher results for the pseudo-dynamic modeling of 5 minute data. The model further showed its performance and flexibility by providing these results without the need for internal source or sink constraint loads as further data input.

This research has demonstrated how a turning movement traffic survey has supplied enough information to test the TMERT model on a US network. The principal findings of the research are summarized as follows:

1. The model proposed by Martin (1993) has been applied to a US traffic network using actual traffic data, to demonstrate that turning

flows can be estimated reliably, in real time from flows simulated to be detected on links.

2. The model can be constrained by a combination of arc upper bounds, arc lower bounds, arc weights and detector flows in a systematic way whereby each constraint relates to measurable system parameters.

3. The reliability of turning flow estimation improves with the severity of constraint.

4. Flow inconsistencies associated with noisy traffic data, can be overcome by using virtual "error arcs" which can be controlled through a network wide constraint parameter, ϕ, the error arc upper bound constraint factor. However, the lowest ϕ factor value, where the model is most constrained, does not always provide the best correlation of observed and modeled flows.

5. The model can predict flows over five minute intervals thereby simulating a pseudo-dynamic environment.

4. TMERT has now been shown to work on two networks: a US and UK and thus has demonstrated that it is a repeatable model.

5. TMERT performs as well on a US network as it did on a UK network.

6. The TMERT model can now accept and supply its Input and Output in an easily manipulated spreadsheet format. Future modeling will be quicker. The model now has "front end" and "back end" interfaces so that the whole of the TMERT model is coded.

    6.1 The "front-end" pre-processing calculates the various parameters needed by the model: error arc upper bound constraints from the assigned constraint factor and detector flows, and turning movement arc capacities from signal data.

    6.2 The "back end" speeds the efficiency of model testing so that larger, more complex networks can now be evaluated. The process is now automated by coding the comparison of

observed and modeled turning flows to provide a performance measure in the form of a correlation coefficient.

7. The code which invokes the NETFLO algorithm, which is sandwiched between the two automatic procedure components has been re-written in a structured form in C.

8. When turning movement arcs were weighted, the observed to modeled turning movement flows were highly correlated (96% r^2). Correlation deteriorated when weights were removed (90% r^2). Therefore, turning movement weighting, by type of turn is desirable.

9. In both the UK and the US network, approximately 15% of the internal links were detected. This suggests that a minimum number of detectors is necessary for reliable model performance.

7.6 Suggestions For Future Work

For future modeling, a network should be configured before data collection to stream line data management.

TMERT's effectiveness should be examined on a large, heavily congested network with actual detector flows. A large city with more signal controlled intersections and flows operating closer to capacity, with an adaptive control system would provide a full scale demonstration of the model's flow estimating ability.

A link analysis would determine TMERT's ability to estimate the internal link flows which has practical value. In this way, the model could be configured to supply real-time link flows from a large set of detector flows in place of a small number of defective detectors.

Validation into TMERT's ability to work on several different networks is needed to ensure that the model is capable of working on any detector layout as the model will be used in conjunction with existing detectors of a network.

The relationship between the number and location of detectors and their effect on TMERT's ability to estimate turning movements should be explored.

A series of traffic incidents could be simulated so that spare capacity routes could be identified and the effect of diverting flows along spare capacity routes could be compared to the performance of a remedial fixed-time signal plan.

TMERT has shown its potential as an on-line traffic monitoring tool that can be used for congestion or incident alleviation. The future direction of the model is clear, an analysis of the model's performance involving a large urban network with congestion flow levels at or near capacity is needed.

References

Bell Margaret C., Bretherton R.D. (1986) "Ageing of fixed-time traffic signal plans"; paper presented to the IEE international conference on Road Traffic Control, 15 - 18 April 1986.

Bell Margaret C., Gault H.E. (1982) "An empirical study of plan change algorithms for area traffic control systems"; Paper presented at the I.E.E. International conference on road traffic signalling, London 30 March - 1 April.

Bell Margaret C., Irvine M., Geary G (1987) "The use of automatic control algorithms to define urban traffic routes", Paper presented to the 19th annual UTSG Annual Conference, University of Sheffield, January 1987, (unpublished).

Bell Margaret C., Kerridge J. (1992) "COMIS - a real time transportation management system"; Paper presented to the 24th UTSG conference at The University of Newcastle -Upon-Tyne, January.1992, unpublished.

Bell Margaret C., Martin P.T. (1990) "The Use of Traffic Detector Data for Traffic Control Strategies"; The Institution of Electrical Engineers, May 1990.

Bell M.G.H. (1983) "The estimation of O-D flows and their confidence intervals from measurements of link volumes: a computer program"; Traffic Engineering and Control, April 1983.

Bell M.G.H., Inaudi D., Lange J., Maher M. (1991) "Techniques for the dynamic estimation of O-D Matrices in traffic networks"; Proceedings of the DRIVE Conference: Advanced Telematics in Road Transport, 4-6 February 1991, Brussels, pp 1040.

Cascetta E., Nguyen S. (1988) "A unified framework for estimating or updating origin/destination matrices from traffic counts"; Transportation Research, 22B(6), pp 437-455.

Cremer M., Keller H. (1981) "Dynamic identification of flows from traffic counts at complex intersections"; Proceedings of the eighth international symposium on transportation and traffic theory, pp 121 - 142, Toronto, Canada, June 24 - 26 1981.

Cremer M., Keller H. (1987) "A new class of dynamic methods for the identification of origin-destination flows". Transportation. Res.-B, Vol 21B, No 2, pp. 117-132.

Danzig G.B., Orden A., Wolfe P. (1951) "The Generalised Simplex Method for Minimizing a Linear Form under Linear Inequality Restraints"; Pacific Journal of Mathematics, 5, 2, 183-195.

Davies P., Salter D.R., Bettison M. (1982) "Loop sensors for vehicle classification"; Traffic Engineer and Control, February, pp55 - 59.

Echenique M.L., Williams J. (1982) "O-D matrix production from cordon survey data"; Traffic Engineering and Control, Dec. 82, 584-589.

Euler G. (1988) "Issues in real time control of traffic"; Workshop Report. Management and Control of Urban Traffic Systems. US Engineering Foundation Press, New York, pp 53-63.

Fisk C.S. (1988) "On combining maximum entropy trip matrix estimation with user optimal assignment"; Transportation Research, 22B(1), pp 69-79.

Fisk C.S., Boyce D.E. (1983) "A note on trip matrix estimation from link traffic count data"; Transportation Research, 17B(3), pp 245-250.

Gartner N.H. (1982) "Development and testing of a demand responsive strategy for traffic signal control"; Proc. 1982 Americal Control Conf. pp 578-83.

Gartner N.H. (1983) "OPAC: A demand-responsive strategy for traffic signal control"; Trans. Res. Rec. 906, 75-81.

Glover F., Hultz J., Klingman D.(1977) "Improved Computer-Based Planning Techniques"; Research Report CCS 283, Center for Cybernetic Studies, The University of Texas, Austin, Texas.

Glover F., Karney D., Klingman D., Napier A. (1974) "A Computational Study on Start Procedures, Basis Change Criteria, and Solution Algorithms for Transportational Problems"; Management Science, 20, 5, 793-813

Hall M.D., Van Vliet D., Willumsen L.G. (1980) "SATURN - a simulation-assignment model for the evaluation of traffic management schemes"; Traffic Engineering and Control, April 1980.

Hauer E., Shin B-Y. T.(1981) "Origin-destination matrices from traffic counts: application and validation on simple systems"; Traffic Engineering and Control, Vol 22, No 3, pp 118-121.

Henry J.J., Farges J.L.(1989);"PRODYN"; Proc., 6th IFAC-IFIP-FORS Symp. Transportation. Pergamon, Oxford, pp.505-7.

Highway Capacity Manual (1985), Special Report 209, Second Edition, Revised, Transportation Research Board, Washington, DC., 1992.

Highway Capacity Software (1992) Rd. 1.30, Distributed by McTrans Center for Federal Highway Authority.

Holroyd J., Robertson D.I. (1973) "Strategies for area traffic control systems: present and future"; TRRL Report LR 569, Crowthorne.

Humphrey T.L., Wong, P.J. (1976) "Improved control logic for use with computer-controlled traffic"; NCHRP Report 3-18 (1)/1 by Stanford Research Institute, March 1976.

Hunt P.B., Robertson D.I., Bretherton R.D., Winton R.I. (1981) "SCOOT - A traffic responsive method of coordinating signals"; TRRL Report LR1014, Crowthorne.

Irving J.M., Oakley, C.F., Ramsey J.B.H. (1986) "The updating of an O-D matrix: a maximum likelihood approach"; Traffic Engineering and Control, 27(9), pp442-446, September 1986.

Jarrett D.F., Wright, C.C. (1990) "Estimating Origin-Destination traffic flows from the random variability in automatic detector counts: a preliminary analysis"; PTRC Summer Annual Meeting, September 1990.

Jeffreys M., Norman M. (1977) "On finding realistic turning flows at road junctions"; Traffic Engineering and Control, 18(1), January 1977, 19-21, 25 (an addendum to this article was published in the April 1977 issue p. 207).

Johnson E. L. (1966) "Networks and Basic Solutions"; Operational Research, 14, 619-623.

Kennington,J.L., Helgason R.V. (1980) "Algorithms for network programming"; A Wiley Inter-Science Publication, John Wiley and Sons.

Koshi M. (1988) "An overview of area control systems and motor vehicle navigation/ route guidance developments in Japan"; Setember 1988.

Logie M., Hynd A. (1989) "Trip matrix estimation from varied data sources"; Proceedings of Seminar C, PTRC 17th Summer Meeting, University of Sussex, September 1989.

Luk J.Y.K., Sims A.G., Lowrie P.R. (1982) "SCATS - application and field comparison with a TRANSYT optimized fixed time system"; Paper presented to the IEE international conference on Road Traffic Signalling, 3 March - 1 April 1982.

Maher M.J. (1983) "Inferences on trip matrices from observations on link volumes: a Bayesian statistical approach"; Transportation Research B, Vol 17B, No 6, pp 435-447.

Maher M.J. (1987) "Bias in the estimating of O-D flows from link counts"; Traffic Engineering and Control, 28(12), pp 624-627.

Martin P.T. and Bell Margaret C. (1991) "Turning Movements from Detector Flows Using State Estimation Theory: Some Early Findings" , Universities Transport Study Group (UTSG) Annual Conference, University of Nottingham, January 1991.

Martin P.T. and Bell Margaret C.(1992a) "Linear Programming and Flow Detection", Universities Transport Study Group Annual Conference, Universities of Newcastle, January 1992.

Martin P.T. and Bell Margaret C.(1992b) "Network Programming to Derive Turning Movements from Link Flows", Transportation Research Board Record 1365, 147-154, 1992

Martin P.T.(1992) "Urban Traffic Movement Prediction From Automatic Flow Detectio," PhD thesis, University of Nottingham, March 1992.

Martin P.T.(1993) "Can Operational Research Tackle Traffic Congestion?", Operational Research Insight, volume 6, issue 1, pp 4-9, January 1993.

McDonald M., Hounsell,N.B., Sittampolam, N., McLeod F.N. (1987) "Traffic Incidents and route Guidance in a SCOOT Network"; Final Report to the SERC, Contract No. GR/D/36824.

Mountain L.J. (1983) "The accuracy of turning flows estimates at intersections"; Presented at the 15th UTSG Conference 1983.

Murchland J. (1977) "The multi-proportional problem"; Research Note JDM 263, Transport Studies Group, University College, London.

Nguyen S. (1977) "Estimating an O-D matrix from network data, a network equilibrium approach. Publication no. 60"; Center de recherche sur les transports, Universite of Montreal.

Oh J. (1989) "Estimation of trip matrices in networks with equilibrium flows"; Proceedings of Seminar C, PTRC 17th Summer Meeting, University of Sussex, September 1989.

Ploss G., Keller,H. (1986) "Dynamic Estimation of Origin and Destination Flows from Traffic Counts in Networks". Transportation System Analysis and Policy Studies. Proc. of the Int. Conference on Transportation System Studies (P S Satsangi, A L Agarwal Ed), Tata McGraw-Hill, Delhi, pp. 211-221.

Rach L. (1976) "The development and evaluation of Metropolitan Toronto's read time program for computerised traffic control devices"; Paper to the 3rd IFAC International Symposium on Control in Transportation Systems, Columbus, Ohio. August 1976.

Robertson D.I. (1969) "TRANSYT: A Traffic Network Study Tool"; Road Research Laboratory Report LR 253, Crowthorne, Berkshire.

Robertson D.I. (1984) "Estimating origin-destination flows by simulating trip choice"; Traffic Engineering and Control, July 1984.

Robillard P. (1975) "Estimating the origin-destination matrix from observed link volumes", Transportation Research 9, pp 123-128.

Selby D.L., Powell R.J. (1985) "SCOOT in Southampton", PTRC, Procedings of the Annual Summer Meeting, pp 97-109.

Sheffi Y., Daganzo C. (1979) "Hypernetworks and supply / demand equilibrium obtained with disaggregate demand models"; Transport Research Record 673, Transport Research Board, Washington DC, 1979, 113-120.

Smith M.J. (1979) "Traffic control and route choice: a simple example"; Transportation Research, 13B, pp289-294.

Spiess H. (1987) "A maximum likelihood model for estimating origin-destination matrices", Transportation Research -B Vol 21B, No 5, pp 395-412.

Srinivasan V., Thompson G.L. (1973) "Benefit-Cost of Coding Techniques for the Primal Transportational Algorithm" Journal for the Association for Computing Machinery, 20, 194-213.

Stark D.C. (1989) "Estimating trip matrices from traffic counts", Proceedings of Seminar C, PTRC 17th Summer Meeting, University of Sussex, September 1989.

Timms P. (1990) "The estimation of origin-destination matrices using link counts and "old" matrix information"; Universities Transport Study Group Annual Conference, Hatfield Polytechnic, January 1990 (unpublished).

Van Vliet D., Dow P.D.C. (1979) "Capacity-restrained road assignment"; Traffic Engineering and Control, June, 296 - 305.

Van Zuylen H.J. (1979) "The estimation of turning flows on a junction"; Traffic Engineering and Control, Vol 20, pp 539-541.

Van Zuylen H.J. (1981) "Some improvements on the estimation of an origin-destination matrix from traffic counts"; Proceedings of the 8th International Symposium on Transportation and Traffic Theory, Toronto University, June 1981.

Van Zuylen H.J., Willumsen L.G. (1980) "The most likely trip matrix estimated from traffic counts"; Transport Research B, Vol 14B, pp 281-293.

Williams H.C.W.L. (1976) "On the formation of travel demand models and economic evaluation measures of user benefit."; SRC Transport Study Working Paper 80, Institute of Transport Studies, University of Leeds

Willumsen L.G. (1981) "Simplified transport models based on traffic counts"; Transportation Research, 10, 257-278

Willumsen L.G. (1982) "Estimation of trip matrices from volume counts validation of a model under congested conditions"; PTRC 10th Summer Annual Meeting, 12-15 July 1982, Transportation Analysis and Models Seminar.

Wilson A.G. (1970) "Entropy in urban and regional modeling"; Pion, London 1970.

Wright C.C. (1974) "A second method of estimating traffic speeds from flows observed at the ends of a road link"; Traffic Engineering and Control, 15, 9, 432-434.

Contact Author for Appendix information

Appendix

Appendix A Input and Output Files for overall 2-hr Modeling

Appendix B NETFLO Code

Appendix B: NETFLO Code

NetTest.c

```
/*      TMERT
        rewritten by Kang Su Gatlin
        5/24/94
        THINK C 6.0

        This program utilizes the algorithm outlined by JL Kennington and
RV Helgason
        from their text "Algorithms for Network".  This network uses an All-
Artificial
        Start heuristic.  (See corresponding TMERT paper for more in-depth
explanation).

        It uses an input that is readable and writable from MS-Excel in text
format.

        for Dr. Peter Martin
*/

#include <stdio.h>
#include <math.h>
#include "NetTest.h"

#define MAX_ARCS 500
#define MAX_NODES       150
#define FALSE       0
#define TRUE1
#define BIG    9000         /* Simply just a big number that can be
doubled and still be an int */
#define SLACK 1024
#define ARTIF 2048
#define DUMMY 4096
```

```
#define XCESS 8182


/* This huge amount of global variables has been used for simplicity. This is
a port
        from Excel where all variables are global and the function calls
change a huge number
        of values.  There wasn't enough time to make logical structures.
*/
FILE *fp;
int fnode[MAX_ARCS+1], tnode[MAX_ARCS+1], carc[MAX_ARCS+1],
uarc[MAX_ARCS+1],flow[MAX_NODES+1];
int lwarc[MAX_ARCS+1], named[MAX_ARCS+1], narc, mnode, i, j, l,
cat[MAX_NODES+1], lnodp1, mnodp2;
int lnode,  down[MAX_NODES+1], next[MAX_NODES+1],
level[MAX_NODES+1], arcid[MAX_NODES+1];
int     dual[MAX_NODES+1], larc, marc, from[MAX_ARCS+1],
cost[MAX_ARCS+1], capac[MAX_ARCS+1];
int     flor[MAX_ARCS+1], name[MAX_ARCS+1], to, price0, too, cst,
mnodp1, k, mtree, mslk;
int     try, price, newarc, newpr, newfrm, newto, thd, dw[3], ch[3], dwn,
chg, theta;
int net, i200, nxt, nstop, mreg, msorc, iarc, m, n, ii, jj, kk, inod;
int     jtheta, ktheta, poss, jposs, frm, lvj, fm, lst, dwe, flw, aid, q1, q2, dir,
ref;
int counti, countj, iter, larcp1, ldiff, countk, countl, itheta,
copy_from[MAX_ARCS+1];
int lstar, lsave, ideg, kdeg, iwarn, counter, j80, i70, k70,
copy_to[MAX_ARCS+1];
long kost, kost0;
char skipalot, loop, input_name[20], output_name[20];
char infeas, optim, stuffthenbig, bigloop, skipit, loopagain, alldone, skipit2;

void main()
{
        /*      Gets the names of the input */

                printf("NETFLO\n");
                printf("Enter the name of the file with the input data\n");
                scanf("%s", input_name);
                printf("Enter the name of the desired output file.  If it exists it
will be overwritten.\n");
                scanf("%s", output_name);
                fp = fopen(input_name, "r");

        /*      Gets the number of arcs and nodes. */

                fscanf(fp, "%d%d", &narc, &mnode);
                for(;;)
```

```
		{
			fscanf(fp, "%d ", &i);
			if(i == 0)
				break;
			fscanf(fp, "%d", &j);
			flow[i] = j;
			fscanf(fp, "%d", &cat[i]);
		}


		i = 0;
		for(i = 1; ;i++)
		{
			fscanf(fp, "%d%d%d%d%d%d", &named[i],
&fnode[i], &tnode[i], &carc[i], &uarc[i], &lwarc[i]);
			copy_to[i] = tnode[i];
			copy_from[i] = fnode[i];
			if(fnode[i] == 0)
				break;
		}
		netf();
}


/*	This is the main network flow algorithm.  This is essentially the main
part of the
code.  */
void netf()
{
/*	Initialize all the node sets */
	iwarn = 0;
	lnode = MAX_NODES;
	larcp1 = MAX_ARCS;
	larc = larcp1 - 1;
	lnodp1 = lnode + 1;
	i = mnode;
	nstop = 3;
	if(i <= 0)
		end_code();
	mnodp1 = mnode + 1;
	mnodp2 = mnode + 2;
	nstop = 4;
	if(mnodp1 > lnode)
		end_code();
	for(counter = 1; counter <= mnodp1; counter++)
	{
		down[counter] = 0;
		next[counter] = 0;
		level[counter] = 0;
```

```
        arcid[counter] = 0;
}

/* Here we initialize the artificial arcs */
initialize_artf();

 /* This part right here is the basic feasibility test */
 nstop = 8;
 if(net < 0)
        end_code();
 nstop = 10;
 i = 1;
 j = 0;
 for(j80 = 1; j80 <= mnode; j80++)
 {
        i = -i;
        k70 = max(1, cat[j80]);
        if(j+k70 > larc)
                end_code();
        cat[j80] = isign(j + 1, i);
        for(i70 = 1; i70 <= k70; i70++)
        {
                j++;
                from[j] = isign(j80, i);
                cost[j] = 0;
                capac[j] = -BIG;
                flor[j] = 0;
                name[j] = DUMMY;
        }
}
marc = j +1;
if(marc > larc)
        end_code();
from[marc] = isign(mnodp1, -i);
kost0 = 0;
iarc = 0;

/* We now move the arcs to prepare for the artificial arcs */
move_arc();


eliminate_arcs();

/* Add the artificial arcs */
add_excess_arc();

/*      link the arcs to the tree */
set_link();
```

```
/* Expand the tree if necessary */
set_expansion();
clear_stuff();
while(TRUE)
{
        while(TRUE)
        {
                while(TRUE)
                {
                        skipalot = FALSE;

                        /* Do the incremental Flow change */
                        flow_change();

                        if(skipalot)
                                break;

                        /* Check the ratios of the new set of flows.
Check if the new flows
                                are optimal (check the objective
function  */
                        ratio_test();
                        /* Update the flow */
                        update_flow();
                        if(jtheta != 0)
                                break;
                        capac[newarc] = -capac[newarc];
                }
                if(skipalot)
                        break;
                /* Set the new flow if it seems satisfactory */
                setnflow();

                /*      Change the tree based on star structure */
                dostar();

                /* Get the next node ready */
                next[k] = nxt;
                next[thd] = next[q2];
                next[q2] = q1;
                down[q1] = q2;
        }
        skipalot = FALSE;
        infeas = FALSE;
        /* Compute the final kost of the network */
        kost = kost0;
        for(counti = 1; counti <= mnode; counti++)
        {
                i = abs(arcid[counti]);
```

```
                        if(flow[counti] != 0 && cost[i] == BIG)
                                infeas = TRUE;
                        kost = kost + cost[i] *flow[counti];
                }
                for(counti = 1; counti <= mslk; counti++)
                {
                        if(capac[counti] >= 0)
                                continue;
                        countj = -capac[counti];
                        kost = kost + cost[counti]*countj;
                }

                /*      Check to make sure the network isn't funny! */
                if(infeas)
                {
                        if(optim)
                                printf("\nUh oh there is an error here.
Infeasible but yet optimal!\n");
                        nstop = 20;
                }
                if(!optim)
                {
                        iter--;
                        continue;
                }
                if(infeas)
                {
                        printf("\nThe Network is infeasible, and not
optimal.\n");
                        end_code();
                }

                /* Sends the output data the final arrays */
                flows2cells();
                /* End of the code and the final output */
                end_code();
        }
}

/* Name: flows2cells
        This function computes the final flows and costs of the arcs.  It
moves the data
        to new arrays to do this.  Essentially simple computational.
*/
void flows2cells()
{
        for(counti = 1; counti <= mnode; counti++)
        {
                countj = abs(arcid[counti]);
```

```
                capac[countj] = -flow[counti];
        }
        to = 1;
        try = 1;
        frm = from[try];
        while(TRUE)
        {
                lst = isign(lnodp1, frm);
                while(TRUE)
                {
                        if(0 > -capac[try])
                                flw = 0;
                        else
                                flw = -capac[try];
                        flw += flor[try];
                        fm = abs(frm);
                        cst = flw * cost[try];
                        fnode[try] = fm;
                        tnode[try] = to;
                        carc[try] = cst;
                        uarc[try] = flw;
                        try++;
                        frm = from[try];
                        if((frm ^ lst) > 0)
                                continue;
                        break;
                }
                to++;
                if(to != mnodp1)
                        continue;
                break;
        }
}

/* Name: end_code
        This code is at the end of the program.  It prints out the output data to
an output
        file and ends the program.
*/
void end_code()
{
        FILE *wfp;
        int arc_num[MAX_ARCS+1];
        char dupl_arc[MAX_ARCS+1];
        int temp;

        if(infeas)
                printf("\nThe Network is infeasible\n");
        else
```

```
                printf("\nThe Network is feasible\n");

        if(optim)
                printf("\nThe Network is optimal.\n");
        else
                printf("\nThe Network is NOT optimal.\n");

        for(temp = 0; temp <= MAX_ARCS+1; temp++)
                dupl_arc[temp] = 0;

        fclose(fp);
        wfp = fopen(output_name, "w");
        fprintf(wfp,"Cost = %ld\n", kost);
        fprintf(wfp, "Arc Num\tFrom\tTo\tCost\tFlow\n");
        for(n = 1; n <= marc; n++)
        {
                if(fnode[n] == tnode[n])
                        continue;
                arc_num[n] = find_arc(fnode[n], tnode[n]);
                temp = arc_num[n];
                dupl_arc[temp] = dupl_arc[temp] + 1;
                if(dupl_arc[temp] > 1)
                        arc_num[n] = find_arc_nextn(fnode[n], tnode[n],
dupl_arc[temp]);
        }

        sort_arcs(arc_num);

        for(n = 1; n <= marc; n++)
        {
                if(fnode[n] == tnode[n])
                        continue;
                fprintf(wfp,"%d\t%d\t%d\t%d\t%d\n", arc_num[n], fnode[n],
tnode[n], carc[n], uarc[n]);
        }
        printf("\nALL DONE!\n");
        fclose(wfp);
        exit(0);
}

/* Name: find_arc_nextn
        This function will find the nth arc in the list that has a fnode equal to
from and
        a tnode equal to 'to'.
*/
int find_arc_nextn(int from, int to, int arcs)
{
        int counter;
        for(counter = 1; counter <= marc; counter++)
```

```
	{
		if(from == copy_from[counter] && to == copy_to[counter])
		{
			arcs--;
			if(arcs == 0)
				return(named[counter]);
		}
	}
}

/* Name: sort_arcs
	This sorts the list of arcs and the corresponding data.
	Currently this uses the slow bubblesort.
*/
void sort_arcs(int arc_name[])
{
	int a, b;
	int temp1, temp2, temp3, temp4, temp5;
	for(a = 1; a <= marc; a++)
		for(b = 1; b <= marc; b++)
		{
			if(arc_name[a] < arc_name[b])
			{
				temp1 = arc_name[b];
				temp2 = fnode[b];
				temp3 = tnode[b];
				temp4 = carc[b];
				temp5 = uarc[b];
				arc_name[b] = arc_name[a];
				fnode[b] = fnode[a];
				tnode[b] = tnode[a];
				carc[b] = carc[a];
				uarc[b] = uarc[a];
				arc_name[a] = temp1;
				fnode[a] = temp2;
				tnode[a] = temp3;
				carc[a] = temp4;
				uarc[a] = temp5;
			}
		}
}

/* Name: find_arc
	This will find the first arc in the list that corresponds to from, to.
*/
int find_arc(int from, int to)
{
	int counter;
	for(counter = 1; counter <= marc; counter++)
```

```
                if(from == copy_from[counter] && to == copy_to[counter])
                        return(named[counter]);
}

/* Name: ratio_test
        This ratio_test function will check the ratio's of the newly computed
arcs to
        see if the new flows have helped improve the network
*/
void ratio_test()
{
        newfrm = abs(from[newarc]);
        theta = abs(capac[newarc]);
        jtheta = 0;
        ch[2] = isign(larcp1, capac[newarc]);
        ch[1] = -ch[2];
        dw[1] = newfrm;
        dw[2] = newto;
        ldiff = level[newfrm] - level[newto];
        ktheta = 1;
        if(ldiff == 0)
                goto fourfifty;
        if(ldiff > 0)
                goto threeninety;
        ktheta = 2;
threeninety:
        dwn = dw[ktheta];
        chg = ch[ktheta];
        countk = abs(ldiff);
        for(counti = 1; counti <= countk; counti++)
        {
                if((chg ^ arcid[dwn]) > 0)
                        goto fourten;
                i = abs(arcid[dwn]);
                poss = capac[i] - flow[dwn];
                jposs = -dwn;
                goto fourtwenty;
        fourten:
                poss = flow[dwn];
                jposs = dwn;
        fourtwenty:
                if(theta <= poss)
                        goto fourthirty;
                theta = poss;
                jtheta = jposs;
                if(theta == 0)
                        return;
        fourthirty:
                dwn = down[dwn];
```

```
        }
        dw[ktheta] = dwn;
fourfifty:
foursixty:
        if(dw[1] == dw[2])
                goto fivetwenty;
        for(countl = 1; countl <= 2; countl++)
        {
                dwn = dw[countl];
                if((ch[countl] ^ arcid[dwn]) > 0)
                        goto foureighty;
                i = abs(arcid[dwn]);
                poss = capac[i] - flow[dwn];
                jposs = -dwn;
                goto fourninety;
        foureighty:
                poss = flow[dwn];
                jposs = dwn;
        fourninety:
                if(theta <= poss)
                        goto fivezero;
                theta = poss;
                jtheta = jposs;
                ktheta = countl;
                if(theta == 0)
                        return;
        fivezero:
                dw[countl] = down[dwn];
        }
        goto foursixty;
fivetwenty:
        dwe = dw[1];
}

/* Name: dostar
        This function continuously updates the duals and creates a star like
structure
        (where all the nodes eventually have arcs to the root)
*/
void dostar()
{
        if(theta != 0)
        {
                chg = isign(theta, ch[ktheta]);
                while(TRUE)
                {
                        updatedual();
                        if(i == itheta)
                                return;
```

```
                        flw = n - chg *dir;
                        aid = -isign(ref, dir);
                        next[k] = nxt;
                        next[thd] = j;
                        k = i;
                        i = j;
                        j = down[j];
                        down[i] = k;
                        lstar++;
                }
        }
        while(TRUE)
        {
                updatedual();
                if(i == itheta)
                        return;
                flw = n;
                aid = -isign(ref, dir);
                next[k] = nxt;
                next[thd] = j;
                k = i;
                i = j;
                j = down[j];
                down[i] = k;
                lstar++;
        }
        return;
}

/* Name: updatedual
        Simply updates the duals as we tighten the tree into the star.
*/
void updatedual()
{
        dual[i] = dual[i] + newpr;
        n = flow[i];
        flow[i] = flw;
        dir = isign(1, arcid[i]);
        ref = abs(arcid[i]);
        arcid[i] = aid;
        lsave = level[i];
        ldiff = lstar - lsave;
        level[i] = lstar;
        thd = i;
        while(TRUE)
        {
                nxt = next[thd];
                if(level[nxt] <= lsave)
                        break;
```

```
                level[nxt] = level[nxt] + ldiff;
                dual[nxt] = dual[nxt] + newpr;
                thd = nxt;
        }
        k = j;
        while(TRUE)
        {
                l = next[k];
                if(l == i)
                        return;
                k = l;
        }
}

/* Name: setnflow
        Set the negative flows so that we can't send flows through an arc
more than
        can be contained in the arc.
*/
void setnflow()
{
        itheta = abs(jtheta);
        if(jtheta <= 0)
        {
                j = abs(arcid[itheta]);
                capac[j] = -capac[j];
        }
        flw = theta;
        if(capac[newarc] <= 0)
        {
                capac[newarc] = -capac[newarc];
                flw = capac[newarc] - flw;
                newpr = -newpr;
        }
        if(ktheta == 2)
        {
                q1 = newto;
                q2 = newfrm;
                aid = newarc;
        }
        else
        {
                q1 = newfrm;
                q2 = newto;
                aid = -newarc;
                newpr = -newpr;
        }
        i = q1;
        j = down[i];
```

```
        lstar = level[q2] + 1;
}


/* Name: update_flow
        This function updates the flows in an incremental manner.  This is
defined
        by the chg variable.
*/
void update_flow()
{
        ideg++;
        if(theta != 0)
        {
                ideg--;
                if(iter < kdeg)
                        kdeg = iter;
                dw[1] = newfrm;
                dw[2] = newto;
                if(jtheta != 0)
                        dw[ktheta] = abs(jtheta);
                for(counti = 1; counti <= 2; counti++)
                {
                        dwn = dw[counti];
                        chg = isign(theta, ch[counti]);
                        while(TRUE)
                        {
                                if(dwn == dwe)
                                        break;
                                flow[dwn] = flow[dwn] - chg * isign(1,
arcid[dwn]);
                                dwn = down[dwn];
                        }
                }
        }
}

/* Name: flow_change
        This function changes the flows depending on the weights.  It does
this by changing
        the newxxx variables and then later applying them to the arrays.
*/
void flow_change()
{
        iter++;
        too = to;
        newpr = 0;
        while(TRUE)
        {
```

```
                price0 = -dual[to];
                lst = isign(lnodp1, frm);
                while(TRUE)
                {
                        fm = abs(frm);
                        price = dual[fm] + price0 - cost[try];
                        if(capac[try] < 0)
                                price = -price;
                        if(capac[try] != 0)
                        {
                                if(price > newpr)
                                {
                                        newarc = try;
                                        newpr = price;
                                        newto = to;
                                }
                        }
                        try++;
                        frm = from[try];
                        if((frm ^ lst) > 0)
                                continue;
                        break;
                }
                to++;
                if(to == mnodp2)
                {
                        to = 1;
                        try = 1;
                        frm = from[try];
                }
                if(newpr != 0)
                        return;
                if(to != too)
                        continue;
                optim = TRUE;
                skipalot = TRUE;
                return;
                break;
        }
}

/* Name: clear_stuff
        Just initializes alot of the arrays.  Either sets them to zero or
someother well
        defined value (like the BIG value or its negation)
*/
void clear_stuff()
{
        for(counti = 1; counti <= mnode; counti++)
```

```
        {
                countj = abs(arcid[counti]);
                capac[countj] = -capac[countj];
        }
        for(counti = 1; counti <= marc; counti++)
        {
                if(capac[counti] + BIG == 0)
                        capac[counti] = 0;
        }
        capac[0] = BIG;
        capac[marc] = BIG;
        to = 1;
        try = 1;
        frm = from[try];
        iter = 0;
        optim = FALSE;
        ideg = 0;
        kdeg = BIG;
}

/* Name: set_expansion
        Here is where we expand the tree to add all of the nodes that may not
have got
        connected to the artificial arc directly.
*/
void set_expansion()
{
        loop = FALSE;
        to = 1;
        try = 1;
        frm = from[try];
        while(TRUE)
        {
                if(mtree == mnode)
                        return;
                too = to;
                newpr = BIG;
                while(TRUE)
                {
                        lvj = level[to];
                        lst = isign(lnodp1, frm);
                        while(TRUE)
                        {
                                while(TRUE)
                                {
                                        loop = FALSE;
                                        if(capac[try] <= 0)
                                                break;
                                        m = cost[try];
```

```
                        if(newpr < m)
                                break;
                        fm = abs(frm);
                        if(level[fm] == 0)
                        {
                                if(lvj == 0)
                                        break;
                                i = to;
                                j = fm;
                                k = -m;
                                l = -try;
                        }
                        else
                        {
                                if(lvj != 0)
                                        break;
                                i = fm;
                                j = to;
                                k = m;
                                l = try;
                        }
                        newpr = m;
                        break;
                }
                try++;
                frm = from[try];
                if((frm ^ lst) > 0)
                        continue;
                break;
        }
        to++;
        if(!(to != mnodp1))
        {
                to = 1;
                try = 1;
                frm = from[try];
        }
        if(newpr != BIG)
        {
                arcid[j] = l;
                down[j] = i;
                next[j] = next[i];
                next[i] = j;
                level[j] = level[i] + 1;
                dual[j] = dual[i] - k;
                newarc = abs(l);
                capac[newarc] = -capac[newarc];
                mtree++;
                loop = TRUE;
```

```
                                        break;
                                }
                                if(to != too)
                                        continue;
                                break;
                        }
                        if(loop)
                        {
                                loop = FALSE;
                                continue;
                        }
                        for(counti = 1; counti <= mnode; counti++)
                        {
                                if(level[counti] != 0)
                                        continue;
                                if(!(arcid[counti] == -1))
                                {
                                        countj = cat[counti];
                                        if(!(abs(from[countj]) != counti))
                                        {
                                                printf("\nWe have an isolated NODE
in the Network\n");
                                                iwarn = 1;
                                        }
                                }
                                arcid[counti] = 0;
                                flow[counti] = 0;
                                next[counti] = next[mnodp1];
                                next[mnodp1] = counti;
                                down[counti] = mnodp1;
                                level[counti] = 1;
                                dual[counti] = -BIG;
                                loop = FALSE;
                        }
                        break;
                }
}


/* Name: unavailable
        If the flow in a particular arc is negative, then it is unavailable then
        we must find alternate routes for the flow.  This function works on
the
        alternative routes.
*/
void unavailable()
{
        while(TRUE)
        {
```

```
            skipit2 = FALSE;
            if(capac[try] > 0)
            {
                    fm = abs(frm);
                    if(level[fm] == 0)
                    {
                            price = cost[try];
                            if(price < newpr)
                            {
                                    newarc = try;
                                    newpr = price;
                                    if(newpr == 0)
                                            skipit2 = TRUE;
                            }
                    }
            }
            if(!skipit2)
            {
                    try++;
                    frm = from[try];
                    if((frm ^ lst) > 0)
                            continue;
                    if(newarc == 0)
                    {
                            k = 0;
                            addchain();
                            bigloop = TRUE;
                            return;
                    }
            }
            break;
    }
    fm = abs(from[newarc]);
    if(capac[newarc] <= flow[to])
    {
            flw = capac[newarc];
            capac[newarc] = -flw;
            flow[fm] = flw;
            flow[to] -= flw;
            down[fm] = to;
            down[mnodp1] = fm;
            level[fm] = -1;
            bigloop = TRUE;
            return;
    }
    capac[newarc] = -capac[newarc];
    flow[fm] = flow[to];
    down[fm] = down[to];
    down[to] = fm;
```

```
        down[mnodp1] = fm;
        next[fm] = to;
        arcid[to] = newarc;
        level[fm] = level[to] - 1;
        dual[to] = newpr;
        mtree++;
        bigloop = TRUE;
}


/* Name: setfrom
        This function just checks the last node and the from node aren't on
        the same side.
*/
void setfrom()
{
        try++;
        frm = from[try];
        if((frm ^ lst) > 0)
                loopagain = TRUE;
        else
        {
                skipit = FALSE;
                loopagain = FALSE;
        }
}

/* Name: enoughsupply
        enough supply set will set a newpr and arc based upon where more
flow is going
        , to the 'to' node or the 'from' node.
*/
void enoughsupply()
{
        if(flow[fm] < flow[to])
        {
                setfrom();
                return;
        }
        newarc = try;
        newpr = price;
        if(newpr == 0)
        {
                skipit = TRUE;
                alldone = TRUE;
                return;
        }
        setfrom();
}
```

```
/* Name: nomarc
        This applies negative flows to an arc that is filled to capacity.
*/
void nomarc()
{
        capac[newarc] = -capac[newarc];
        fm = abs(from[newarc]);
        flow[fm] -= flow[to];
        k = BIG;
}

/* Name: addchain
        addchain will add the new arcs to the tree.  This is relatively
straightforward.
*/
void addchain()
{
        bigloop = FALSE;
        down[mnodp1] = down[to];
        fm = abs(from[newarc]);
        arcid[to] = newarc;
        dual[to] = newpr;
        down[to] = fm;
        i = next[fm];
        next[fm] = to;
        j = level[fm] - level[to] + 1;
        thd = fm;
        while(TRUE)
        {
                thd = next[thd];
                l = level[thd];
                level[thd] = l + j;
                k -= dual[thd];
                dual[thd] = k;
                if(l != -1)
                        continue;
                break;
        }
        next[thd] = i;
        mtree++;
        bigloop = TRUE;
}

/* Name: set_link
        This will set up the links to the newly created artificial nodes.
        See the algorithm as outlines in the corresponding paper by Martin
and Gatlin.
```

```
        This part is very complex and can't be given justice in the code.
*/
void set_link()
{
        while(TRUE)
        {
                stuffthenbig = FALSE;
                to = down[mnodp1];
                if(to == mnodp1)
                        break;
                while(TRUE)
                {
                        skipit = FALSE;
                        stuffthenbig = FALSE;
                        bigloop = FALSE;
                        newarc = 0;
                        newpr = BIG;
                        if(flow[to] == 0)
                        {
                                k = 0;
                                addchain();
                                bigloop = TRUE;
                                break;
                        }
                        try = cat[to];
                        frm = from[try];
                        lst = isign(lnodp1, frm);
                        check_all();
                        if(!skipit)
                        {
                                if(newarc == 0)
                                {
                                        stuffthenbig = TRUE;
                                        bigloop = FALSE;
                                        break;
                                }
                        }
                        if(newarc <= 0)
                        {
                                newarc = -newarc;
                                fm = abs(from[newarc]);
                                flw = capac[newarc];
                                capac[newarc] = -flw;
                                flow[fm] -= flw;
                                flow[to] -= flw;
                                continue;
                        }
                        nomarc();
                        addchain();
```

```
                        bigloop = TRUE;
                        break;
                }
                if(bigloop)
                        continue;
                if(!stuffthenbig)
                        break;
                stuffthenbig = FALSE;
                try = cat[to];
                frm = from[try];
                unavailable();
        }
}

/* Name: check_all
        This function will check the negatives in the matrix and the objective
        function to see if we have improved.
*/
void check_all()
{
        while(TRUE)
        {
                loopagain = FALSE;
                alldone = FALSE;
                skipit = TRUE;
                if(capac[try] <= 0)
                {
                        setfrom();
                        if(loopagain)
                                continue;
                        return;

                }
                fm = abs(frm);
                if(level[fm] != 1 || arcid[fm] == 0)
                {
                        setfrom();
                        if(loopagain)
                                continue;
                        return;
                }
                price = cost[try];
                if(price >= newpr)
                {
                        setfrom();
                        if(loopagain)
                                continue;
                        return;
                }
```

```
                if(capac[try] > flow[to])
                {
                        enoughsupply();
                        if(loopagain)
                                continue;
                        return;
                }
                if(flow[fm] < capac[try])
                {
                        setfrom();
                        if(loopagain)
                                continue;
                        return;
                }
                newarc = -try;
                newpr = price;
                if(newpr == 0)
                        return;
                setfrom();
                if(loopagain)
                        continue;
                return;
        }
}


/* Name: add_excess_arc
        Add the new arcs to the new excess node.  This becomes the
        basis of the artificial arcs heuristic.
*/
void add_excess_arc()
{
        mslk = marc;
        marc++;
        from[marc] = isign(mnodp2, -i);
        cost[marc] = BIG;
        capac[marc] = 0;
        flor[marc] = 0;
        name[marc] = XCESS;
        net = 0;
        mtree = 0;
        thd = mnodp1;
        for(i200 = 1; i200 <= mnode; i200++)
        {
                j = flow[i200];
                net += j;
                if(j < 0)
                {
                        flow[i200] = -j;
```

```
                        dwn = mnodp1;
                        while(TRUE)
                        {
                                nxt = down[dwn];
                                if(flow[nxt] + j <= 0)
                                        break;
                                dwn = nxt;
                        }
                        down[dwn] = i200;
                        down[i200] = nxt;
                        level[i200] = -1;
                }
                if(j > 0)
                {
                        mtree++;
                        arcid[i200] = -marc;
                        flow[i200] = j;
                        next[thd] = i200;
                        down[i200] = mnodp1;
                        next[i200] = mnodp1;
                        level[i200] = 1;
                        dual[i200] = BIG;
                        thd = i200;
                }
        }
        nstop = 16;
        if(net < 0)
                end_code();
        nstop = 0;
}

/* Name: eliminates_arcs
        Despite the name, what we are actually doing is changing a portion of
the
        arcs to SLACK or DUMMY arcs.  This in effect eliminates arcs
from the actual
        network.
*/
void eliminate_arcs()
{
        int j190;

        i = lnodp1;
        k = 0;
        l = 0;
        marc--;
        for(j190 = 1; j190 <= marc; j190++)
        {
                j = from[j190];
```

```
        if(!(((i ^ j) > 0) && (abs(j) == l)))
        {
                if(!((i ^ j) > 0))
                {
                        i = -i;
                        l++;
                        cat[l] = k + 1;
                }
                k++;
                if(k != j190)
                {
                        from[k] = from[j190];
                        cost[k] = cost[j190];
                        capac[k] = capac[j190];
                        flor[k] = flor[j190];
                        name[k] = name[j190];
                }
        }
}
marc = k;
mreg = k;
nstop = 15;
if(marc + max(1, msorc) + 1 > larc)
        end_code();
i = -from[marc];
thd = next[mnodp1];
next[mnodp1] = mnodp1;
if(!(thd != mnodp1))
{
        marc++;
        from[marc] = isign(mnodp1, i);
        cost[marc] = 0;
        capac[marc] = -BIG;
        flor[marc] = 0;
        name[marc] = DUMMY;
}
else
{
        while(TRUE)
        {
                marc++;
                name[marc] = SLACK;
                from[marc] = isign(thd, i);
                cost[marc] = 0;
                capac[marc] = level[thd];
                level[thd] = 0;
                flor[marc] = 0;
                nxt = next[thd];
                next[thd] = 0;
```

```
                        thd = nxt;
                        if(!(thd != mnodp1))
                                break;
                }
        }
}

/* Name: move_arc
        The arcs in the original network may have to be moved to allow for
        proper formation of the artificial arcs.  This is done here.  The setting
        of n to 'a' in the code is actually setting n to the designation 'arc'.
*/
void move_arc()
{
int j120, m120, l120, j130, k120;

        while(TRUE)
        {
                iarc++;
                if(iarc > narc)
                        break;
                n = 'a';
                i = fnode[iarc];
                j = tnode[iarc];
                k = carc[iarc];
                l = uarc[iarc];
                m = lwarc[iarc];
                nstop = 12;
                if((i <= 0) || (i > mnode) || (j > mnode) || (j <= 0))
                        end_code();
                nstop = 13;
                if(l >= BIG)
                        end_code();
                if(l < 0)
                        l = 0;
                if((m >= BIG) || (m < 0) || (m > l))
                        end_code();
                ii = cat[j];
                jj = abs(ii);
                kk = isign(lnodp1, ii);
                if(!((kk ^ from[jj]) > 0))
                {
                        nstop = 14;
                        if(marc == larc)
                                end_code();
                        marc++;
                        k120 = marc - jj;
                        m120 = marc;
                        for(j120 = 1; j120 <= k120; j120++)
```

```
                        {
                                l120 = m120 - 1;
                                from[m120] = from[l120];
                                cost[m120] = cost[l120];
                                capac[m120] = capac[l120];
                                flor[m120] = flor[l120];
                                name[m120] = name[l120];
                                m120 = l120;
                        }
                        for(j130 = j; j130 <= mnode; j130++)
                                cat[j130] += isign(1, cat[j130]);
                }
                from[jj] = isign(i, ii);
                cost[jj] = k;
                kost0 = kost0 + k * m;
                capac[jj] = l - m;
                flor[jj] = m;
                flow[i] -= m;
                flow[j] += m;
                name[jj] = n;
                cat[j] = isign(jj+1, ii);
                arcid[i] = -1;
        }
}

/* Name: isign
        Computes and returns the sign of 'b' times the absolute value of 'a'.
*/
int isign(int a, int b)
{
        if(b == 0)
                return(abs(a));
        if(b > 0)
                return(abs(a));
        return(-abs(a));
}

int max(int a, int b)
{
        if(a > b)
                return(a);
        return(b);
}

/* Name: initialize_artf
        The function will create the artificial arc at position 0 in the arrays.
        This artificial arc becomes the starting point of the heuristic.
*/
void initialize_artf()
```

```
{
        from[0] = mnodp1;
        cost[0] = BIG;
        capac[0] = 0;
        flor[0] = 0;
        name[0] = ARTIF;
        net = 0;
        msorc = 0;
        marc = 0;
        next[mnodp1] = mnodp1;
        down[mnodp1] = mnodp1;
        inod = 0;
        while(TRUE)
        {
                inod++;
                if(inod > mnode)
                        break;
                i = inod;
                j = flow[i];
                net += j;
                if(j <= 0)
                        continue;
                msorc++;
                level[i] = j;
                next[i] = next[mnodp1];
                next[mnodp1] = i;
        }
}
```