Southwest Region University Transportation Center

Development and Evaluation of Light Rail Transit Signal Priority Strategies

SWUTC/98/472840-00072-1

Center for Transportation Research
University of Texas at Austin
3208 Red River, Suite 200
Austin, Texas 78705-2650

Technical Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
SWUTC/98/472840-00072-1		

4. Title and Subtitle	5. Report Date
Development and Evaluation of Light Rail Transit Signal Priority Strategies	August 1998
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Beth Taylor and Randy B. Machemehl	

9. Performing Organization Name and Address	10. Work Unit No. (TRAIS)
Center for Transportation Research The University of Texas at Austin 3208 Red River, Suite 200 Austin, Texas 78705-2650	11. Contract or Grant No. DTOS88-G-0006

12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered
Southwest Region University Transportation Center Texas Transportation Institute The Texas A&M University System College Station, Texas 77843-3135	14. Sponsoring Agency Code

15. Supplementary Notes

Supported by a grant from the U.S. Department of Transportation, University Transportation Centers Program

16. Abstract

Light rail transit (LRT) signal priority strategies are often considered LRT vehicle delay reduction tools. The purpose of this research is to develop and evaluate LRT signal priority strategies to determine which strategies are most likely to produce significant delay reductions. A simulated median running LRT route in Austin, Texas is used for this case study.

Both active and passive priority strategies are simulated using CORSIM with a Run-Time Extension (RTE). The RTE is used to provide the active priority signal timing changes associated with a green signal extension for the LRT phase. The results indicate that passive priority strategies are more effective in reducing delay. Among the passive methods, prohibition of left turns across the LRT tracks and one-way progression segmented to account for stops at LRT stations yield the best results.

17. Key Words	18. Distribution Statement
Light Rail Transit (LRT), Signal Priority, Active and Passive Priority Strategies, Analysis of Variance, Delay Reduction	No Restrictions. This document is available to the public through NTIS: National Technical Information Service 5285 Port Royal Road Springfield, Virginia 22161

19. Security Classif.(of this report)	20. Security Classif.(of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	161	

DEVELOPMENT AND EVALUATION OF LIGHT RAIL TRANSIT SIGNAL PRIORITY STRATEGIES

by

Beth Taylor
Randy Machemehl

Research Report SWUTC/98/472840-00072-1

Southwest Region University Transportation Center
Center for Transportation Research
The University of Texas at Austin
Austin, Texas 78712

August 1998

Disclaimer

The contents of this report reflect the views of the authors, who are responsible for the facts and the accuracy of the information presented herein. This document is disseminated under the sponsorship of the Department of Transportation, University Transportation Centers Program in the interest of information exchange. The U.S. Government assumes no liability for the contents or use thereof.

EXECUTIVE SUMMARY

Several U.S. cities are either planning or operating light rail transit (LRT) systems. In many of these cases, LRT signal priority is considered to reduce LRT delay. Reported results regarding the effectiveness of LRT signal priority have been mixed. The objectives of this research are as follows: 1) Develop a method of analysis for LRT signal priority, 2) Develop both passive and active priority strategies for a given case study network in Austin, Texas, and 3) Evaluate the effectiveness of these strategies.

The CORSIM simulation package is used to analyze several LRT signal priority strategies for a hypothetical median-running LRT route in Austin, Texas. CORSIM does not explicitly model LRT, so the bus operation features are used with some modification to more accurately model LRT.

A previous study on bus signal priority for the same case study network and a review of the relevant literature led to the development of the specific LRT signal priority strategies for this study. Both active and passive priority strategies are analyzed. The active strategies are simulated through the use of a CORSIM Run-Time Extension (RTE) and consist of unconditional and conditional green signal extension for the LRT phase. The passive methods are prohibition of left turns across the tracks, one-way progression, and one-way progression segmented to account for stops at LRT stations.

The simulation runs constitute repeated measures experiments using person delay as the dependent variable. Analysis of variance (ANOVA) is used to analyze the data. The results indicate that passive priority strategies are more effective in reducing LRT delay. Among the passive methods, prohibition of left turns across the LRT tracks and one-way progression segmented to account for stops at LRT stations yield the best results.

ACKNOWLEDGMENTS

The authors recognize that support was provided by a grant from the U.S. Department of Transportation, University Transportation Centers Program to the Southwest Region University Transportation Center.

ABSTRACT

Light rail transit (LRT) signal priority strategies are often considered LRT vehicle delay reduction tools. The purpose of this research is to develop and evaluate LRT signal priority strategies to determine which strategies are most likely to produce significant delay reductions. A simulated median running LRT route in Austin, Texas is used for this case study.

Both active and passive priority strategies are simulated using CORSIM with a Run-Time Extension (RTE). The RTE is used to provide the active priority signal timing changes associated with a green signal extension for the LRT phase. The results indicate that passive priority strategies are more effective in reducing delay. Among the passive methods, prohibition of left turns across the LRT tracks and one-way progression segmented to account for stops at LRT stations yield the best results.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1. INTRODUCTION

Light Rail Transit (LRT) is currently operating in several U.S. cities and is being planned or implemented in others. As traffic congestion grows, cities are turning to transit and more specifically to rail transit as a means to provide enhanced mobility. To make rail transit use attractive, signal priority for rail is often considered to reduce transit vehicle delay. The purpose of this research is to develop priority strategies for LRT based on previous research and to evaluate their effectiveness in reducing LRT delay and their impacts on other traffic.

An earlier study by Garrow (1997) serves as the predecessor to this research. Garrow evaluated signal priority for buses along the same arterial that is used in this work. Since LRT and buses have different operating characteristics that could affect signal priority strategy selection, a follow-up study specifically for LRT was undertaken. The findings of this LRT-based signal priority research are reported in this report.

The Guadalupe-North Lamar arterial in Austin, Texas is the case study LRT route in this research. Capital Metro, the transit authority in Austin, does not currently operate any LRT service; however, for several years light rail has been in the discussion and planning stages. The Guadalupe-North Lamar arterial is a segment of the most heavily used bus route currently operated by Capital Metro and thus was the route segment chosen in the previous bus study by Garrow (1997). In addition, preliminary LRT planning for the Austin area has identified a route along this corridor since it serves the University of Texas campus.

Evaluating the effectiveness of signal priority strategies can be considered as a multiple criteria decision problem. The most common criterion used in the past for evaluating signal priority impacts has been either vehicle delay/travel time or person delay/travel time. The person measure takes into consideration the number of occupants in each vehicle. When the transit mode share is sufficiently high, increased delay to automobiles when transit receives signal priority could result in an overall reduction in person delay/travel time even though vehicle delay/travel time increased. Delay and/or travel time are not the only criteria that could be considered when making a decision regarding signal priority. Other factors that might be considered include qualitative issues such as the concerns of residents and businesses along an arterial when strategies such as turning movement restrictions are proposed. Capital and maintenance costs associated with various strategies could be considered. In addition, impacts on environmental quality and on other users such as

pedestrians and bicyclists might be included as part of the evaluation process. Finally, community objectives in relation to transportation and mobility may cause the numbers such as person delay to be evaluated in a different light. For example, delay to automobiles within a particular range may be deemed acceptable when there is a policy to encourage a mode shift to transit by providing signal timing that is favorable to transit. Although all of these factors are not investigated or measured in this research, it is important to note that any recommendations made based on specific measures of effectiveness are only components of the decision to implement or not implement a particular strategy.

The specific objectives of this study are as follows: 1) Develop a method of analysis for LRT signal priority, 2) Develop both passive and active priority strategies for a given case study network in Austin, Texas, and 3) Evaluate the effectiveness of these strategies.

Chapter Two presents a review of the relevant literature. The research methodology is described in Chapter Three. A discussion of the signal priority strategies is the subject of Chapter Four. The evaluation of these strategies is presented in Chapter Five. Note that the three study objectives are fulfilled by the work described in Chapters Three, Four, and Five, respectively. Finally, Chapter Six presents a summary of the findings and recommendations for future research.

CHAPTER 2. LITERATURE REVIEW

INTRODUCTION

The purpose of this literature review is to present past findings relevant to LRT signal priority. Both techniques used for analysis and results from previous studies and implementations are discussed.

TYPES OF PRIORITY

To clarify the upcoming discussion of signal priority for light rail, some definition of terms is needed. The following definitions are from Korve (1978).

> The terms priority and preemption both refer to preferential treatment given to LRT at traffic signals to minimize delays to LRT caused by the traffic signals or by other vehicles in the traffic stream. Preemption is intended to imply as immediate a response as is consistent with safety, whereas priority is intended to imply that, in addition to safety considerations, the needs of other movements, primarily vehicular, will be evaluated before deciding whether to grant preference to LRT.

Within the realm of priority, a differentiation can be made between passive versus active priority strategies. Passive priority involves presetting the signal timing to favor transit vehicles, whereas active priority involves making a signal timing adjustment that is favorable to transit upon detection of a transit vehicle. The key distinction here is that passive priority is in operation even when a transit vehicle is not present at the intersection (Tyer et al., 1995).

Among the strategies that fall into the category of passive priority are progression favoring the transit vehicle, reduced cycle lengths, and split phasing. Split phasing divides the transit phase into multiple phases and repeats these smaller phases within a cycle (Sunkari et al., 1995).

For active priority, a further distinction can be made between partial and full priority strategies. Partial priority involves increasing the green time for transit. Full priority adds to this the ability to skip certain non-transit phases, so it is generally more disruptive to other intersection users. Preemption, which was described earlier, can be thought of as a full priority strategy that is subject only to safety considerations in determining which phases to skip (Noyce, 1996).

Some examples of partial priority strategies include green extension, which involves extending the transit green phase by a specified amount, and red truncation which terminates the transit red phase early. Additionally, window stretching is a term used to describe the use of green extension and/or red truncation in a coordinated system where the added transit green phase time is deducted from non-transit phases so that coordination is maintained. Red interruption increases the green time for transit, but the green time is not contiguous with the normal green phase. A separate green phase is inserted into the normal red phase for transit (Noyce, 1996).

Full priority methods include phase suppression, lift strategy, and HOV weighted OPAC. Phase suppression skips specified non-transit phases to return to the transit phase as quickly as possible. For systems utilizing actuated signal control, lift strategy ignores non-transit detection calls, so that transit phase can be recalled sooner. High Occupancy Vehicle (HOV) Weighted Optimized Policy for Adaptive Control (OPAC) is a form of demand responsive signal control that utilizes persons rather than vehicles in its calculation of demand. An optimization algorithm is used to determine the signal timing. Continual monitoring of the location and need for priority of the transit vehicle through an AVI system is a component of this strategy (Noyce, 1996).

The final distinction between different types of signal priority strategies is that of unconditional versus conditional priority. Unconditional priority is issued each time a transit vehicle is detected, and can be very disruptive to opposing traffic. Conditional priority is issued when a transit vehicle is detected only if certain conditions are met. Examples of the types of conditions that might be used include schedule adherence, transit vehicle occupancy, queue lengths for opposing movements, current traffic conditions, time since the priority was last issued, effect on coordination, and point in the cycle at which the transit vehicle is detected (Sunkari et al., 1995).

ANALYSIS TECHNIQUES

Development of a method of analysis for LRT signal priority strategies and evaluation of the effectiveness of these strategies are two of the objectives of this research. This section of the literature review is used as an aid in meeting those objectives. Several techniques have been used for analyzing and evaluating the impacts of signal priority for transit. Following is a description of the use of some of these techniques.

Mathematical Models

Stone and Wild (1982) compared criteria for determining if LRT should be given intersection priority. The criteria compared were level of service versus total person delay. Level of service was calculated based on techniques from the Highway Capacity Manual, 1965. For a single intersection, three different intersection configurations, including both near-side and far-side transit stop locations were included. LRT headways of two, six, ten, and twenty minutes were evaluated. Calculations were made with and without LRT preemption, with the exception of near-side configurations where preemption calculations were not made. Regression analysis on average individual vehicle delay was used in determining delay to automobile users if unconditional light rail preemption was used. Delay to light rail passengers if preemption was not granted was estimated based on several assumptions, then the automobile passenger delay was subtracted from this value to determine the savings in person delay due to preemption.

The results indicated that the level of service calculations produced very few situations where preemption would be acceptable. The total person delay results were positive for many more situations. In general, increases in line volume lead to more justification for LRT preemption until the headways reach about six minutes. At this point, the delay to automobiles may be unacceptable. The conclusion reached was that warrants for LRT signal preemption cannot be developed until more research is undertaken.

Radwan et al. (1985) developed control warrants for signalized intersections operating preemption strategies for light rail transit. Mathematical modeling using a modified version of Webster's delay model was the method of analysis chosen. Several assumptions were made to develop the analytical models, and three different signal-timing strategies were evaluated. These strategies included a two-phase plan, a three-phase plan with an exclusive LRT phase, and a three-phase plan with an exclusive left-turn phase for automobile traffic. LRT arrivals were modeled by probability expressions. A computer program coded with the probability expressions and delay equations was used to calculate the delay to both automobiles and LRT under both preemption and non-preemption strategies, and to calculate the total delay due to preemption.

Variations in automobile traffic volumes for the main arterial were used in running the model for all three signal-timing strategies. For the first strategy, variations were also made in cross-street automobile traffic networks, LRT volumes, and advance detection duration. Findings indicated that the second strategy provided no intersection gain for any volume levels. Regression analysis was used for the first and third strategies to develop equations

for establishing signal preemption warrants. For the first option the finding was that the overall intersection gain due to signal preemption is linearly proportional to the LRT volume and that the advance detection duration had no impact on intersection gain. For the third option with a given constant left-turn volume, an optimum main-arterial volume exists at which the overall intersection gain is maximized.

Simulation Models

The use of simulation models in analyzing transit signal priority strategies is described below. This presentation is organized by the particular simulation package or combination of packages used.

Yagar and Heydecker (1988) utilized the TRANSYT model, which was used to model fixed-time traffic signal networks, to study peak hour delays to both streetcars and private vehicles in the Queen Street corridor in Toronto. Modeling of this corridor included the addition of dummy nodes and links to account for delay imposed by streetcars on other road traffic due to streetcar loading at signalized intersections. Fixed stop times were assumed so that an upper bound on time saved due to setting signal timing to accommodate streetcars could be estimated. Weights assigned to streetcars were equivalent to five private vehicles in one scenario and one hundred private vehicles in another. The results indicated that there was a potential reduction in delay to streetcars of up to twenty-five percent with no additional delay to private vehicles. Since fixed stop times were determined to be impractical, the authors suggest further modeling efforts to estimate delays when stop times are random.

According to Bodell and Huddart (1987), for the Hong Kong LRT system, a passive priority system of fixed-time signal progression was rejected due to its inflexibility and inability to deal effectively with variable dwell times and driver speeds. The system chosen was an active priority system based on vehicle-actuated control through the use of a series of request and cancel detector loops. Limits were planned on the amount of priority given in some situations, and fixed-time signals were planned for certain critical intersections. The FLEXSYT simulation tool was used to evaluate the effects on both light rail vehicles and road traffic. The goal was to give priority to LRT while not unduly penalizing road traffic.

Also using FLEXSYT were Chin and Mundy (1992). Results for a lightly loaded intersection indicated that even with high priority for LRT the effect on other traffic was relatively small. For a more complex intersection, although the survey was not of sufficient duration to be statistically significant, the indication was that the simulation results were similar to the field data.

Analysis of LRT projects is complicated by many factors including the sporadic and random nature of arrivals and the interdependence of traffic signal coordination and LRT arrivals, according to Fehon et al. (1988). As a result, for LRT analysis, they recommend the use of a microscopic network simulator, ROADTEST, which they developed. ROADTEST simulates various modes including rail, automobiles, buses, pedestrians, and other modes if needed. Also included is the ability to model traffic and train control signals.

Wu and McDonald (1996) stated that "TRGMSM is an object-oriented microscopic simulation model, which has been specifically developed to study the interactions between at-grade LRT and normal road traffic, and has been calibrated against U.K. data." Some features of the simulation model include vehicle characteristics and driver behavior, geometric aspects, signals, and LRT priority measures. There are seven types of priority that can be simulated: 1) Fixed-time signal control without LRT priority (FN), 2) Fixed-time signal control with partial LRT priority (FP), 3) Vehicle-actuated signal control without LRT priority (VN), 4) Vehicle-actuated signal control with full LRT priority (VF), 5) Vehicle-actuated signal control with LRT rail-gate-close priority (VRC), 6) Fixed-time signal with LRT driver-triggered priority (FDTP), and 7) Vehicle-actuated signal with LRT triggered priority (VDTP).

A simulation of the introduction of a new at-grade crossing LRT into an existing signalized intersection was conducted using a mean LRT headway of five minutes and three levels of traffic demand which were 1112, 2779, and 4445 vehicles/hour. A mid-block station location was assumed and the VF type of priority was used. The introduction of LRT resulted in increases in vehicle delay by 9.9, 24.25, and 74.48 percent respectively for each of the traffic demand levels. However, the number of passengers transported per hour increased by 115, 45, and 25 percent respectively.

Several conclusions were reached as result of this analysis. The first of these is that significant vehicle and person delay does not necessarily result from increasing LRT frequency. Another conclusion was that the VF type of control resulted in the least amount of person delay without causing significant extra vehicle delay. Thirdly, the location of the LRT station has an effect on delay, especially person delay. One such situation is the use of mid-block stations that can result in significant reductions in person delay at junctions. Finally, the restriction or reduction of right-turn traffic results in decreases in both vehicle and person delay at junctions. (Note: this research was done in Great Britain, so this usage of right-turn corresponds to left-turn in the U.S.)

Bauer et al. (1995) conducted an analysis of three signal control strategies for the proposed Central Area Circulator light rail project in Chicago. This light rail line will have

dedicated travel lanes. The signal priority strategy chosen should provide LRT priority along with preservation of a safe pedestrian environment and reasonable automobile traffic performance. Two LRT routes, seven LRT intersections, and one LRT junction were included in the study area.

Four alternatives were considered as the methodology for this analysis. The first alternative, intersection capacity analyses using the Highway Capacity Manual (HCM) could not consider several of the needed analysis factors, such as the effects of other intersections, queuing, and variable signal phases due to LRT-actuated calls.

Time-space diagrams that were developed to show network-wide effects were used in combination with the HCM results for the second alternative. For this method, the variable nature of the signal timings was still a problem.

The use of a microscopic simulation program, TRAF-NETSIM, was able to handle some of the variability introduced by the LRT-actuated signal calls but could not handle some of the unique signal capabilities of this project and the constant communication between the light rail vehicle and the controllers.

The final alternative, which was the one chosen, utilized a transit simulation tool, TransSim II™ for simulation of transit operations and signal controllers in conjunction with TRAF-NETSIM for simulation of traffic operations. The models were joined by inputting the signal timing data produced by TransSim II™ into TRAF-NETSIM.

The three signal control strategies evaluated via simulation were 1) fixed-time signalization, 2) addition of "early green" and "green extension" capabilities to Strategy 1, and 3) two-way communication between LRT and signal controllers. Calibration of the models with field data was conducted before testing the three strategies. Several measures of effectiveness (MOEs) are output from both simulation tools. The results indicated significant improvement in the average LRT operating speed for Strategy 3 when compared to Strategies 1 and 2. Automobile performance was best with Strategy 1, while Strategies 2 and 3 showed identical results for automobile performance.

Venglar et al. (1995) described the calibration and validation of simulation models constructed using the TRAF-NETSIM and TransSim II™ simulators. Since simulation is often used in planning future LRT systems or operational changes to existing systems, knowledge of the accuracy and reliability of such models is important.

Field data from five existing networks were used to test the models of these systems. Two of these networks were in Los Angeles, California; one was in Long Beach, California; and two were in Portland, Oregon. Calibration was required primarily for queue discharge

and platoon dispersion in NETSIM. For TransSim II™, the primary calibration needed was the location of the vehicle detector. Results of the validation for both NETSIM and TransSim II™ indicated that the model outputs were more representative for system-wide travel times than for individual intersection MOEs.

Koch et al. (1995) outlined a network strategy for real-time optimization of traffic signal timings where LRT occupies a reserved right-of-way except at intersections. The goal is to minimize person delay in an intermodal system. This approach is based on a neural network utilizing weights (parameters) estimated by the method of simultaneous perturbation stochastic approximation (SPSA). An adaptive process updates these weights on a daily basis. Real-time traffic data and transit vehicle location are utilized to determine the weights and the optimal signal timing strategy for a given MOE. In this case, the MOE chosen was person delay. The person delay MOE was modified to also include transit delay from its target schedule.

The Baltimore CBD containing seventeen signalized intersections, four of which include LRT, was used as the study area for a simulation of the strategy described above. The period simulated was the evening peak period from 4:00 to 6:00 p.m when the LRT headways were 15 minutes. The simulation determined the timing for all seventeen intersections. This strategy was found to be more effective than both fixed signal timing (current system) and LRT preemption in reducing person delay. LRT preemption was much more successful than the integrated strategy described above in reducing LRT person delay, but also resulted in more automobile passenger delay with the result that the overall decrease in person delay was two percent. The corresponding figure for the integrated strategy was eight percent.

Tyer et al. (1995) compared ten traffic simulation models based on their ability to model transit and HOV (high-occupancy vehicle) facilities. Supply features that should be able to be modeled included buses, carpools, bus/HOV lanes, bus preemption, station characteristics (layout, spacing, and capacity). Demand modeling requirements were mode choice, mode shift, route choice, destination choice, and temporal diversion. Several MOEs, both system and bus-specific were required.

The ten models evaluated were CONTRAM, CORFLO, INTEGRATION, JAM, LATM, MICRO-ASSIGNMENT, NETSIM, SATURN, TRAFFICQ, and TRANSYT. This list was narrowed down to five models that most closely satisfied the requirements. These models were then evaluated further with respect to the specific requirements. These models were:

CONTRAM, CORFLO, NETSIM, SATURN, and TRAFFICQ. In terms of overall score, SATURN was ranked highest followed by CORFLO, NETSIM, TRAFFICQ, and CONTRAM.

Though SATURN was ranked higher, it was developed for use in England, so it would have to be modified for use in the United States; therefore, CORFLO was selected for a case study since this was the best model developed in the United States. The report concluded with recommendations for transit-related enhancements to CORFLO.

Noyce (1996) gave a brief description of four simulation software packages that could be used for the evaluation of the effects of signal priority for transit. These were TRGMSM, PREEMPT, PTV Vision, and TransSim II™.

Rymer et al. (1988) evaluated the delays imposed upon cross street automobile traffic due to at-grade LRT at an isolated intersection. The simulation was conducted using NETSIM with the light rail vehicles simulated as buses with crossing clearance times modified to account for the light rail vehicle size.

Garrow (1997) chose simulation for evaluation of signal priority for buses since it provides for more efficient data collection than field testing and also allows sensitivity analyses of key parameters. Although simulation cannot replicate all conditions in the real traffic environment, it was the best option for this case study. TRAF-NETSIM was the simulation package chosen since its features allow for the modeling of various signal priorities. The Guadalupe-N. Lamar arterial was chosen as the network to be simulated. Three models were developed for 1) peak period local bus, 2) off-peak period local bus, and 3) off-peak period express bus. Models were calibrated against existing traffic data collected in a University of Texas class project.

During off-peak times, both local and express transit services were evaluated. Local transit service was evaluated for two passive priority techniques (reduced signal cycle lengths and split phasing), and express transit service was evaluated for an active priority technique (unconditional priority). The passive techniques are less expensive to implement and have less potential for negative impacts on cross street traffic.

The results of the simulation for reduced cycle lengths indicated increased levels of service for both transit vehicles and private automobiles on the arterial and cross streets. Split phasing results were not as conclusive as the results for reduced cycle lengths, as there was improvement for the northbound bus but not for the southbound bus. Any implementation of split phasing would need to consider any progression timing that was already in place on either the arterial or cross streets since split phasing could interfere with progression.

Simulation results of unconditional priority given to the express bus service indicated that bus travel times were reduced. The effects on the cross streets were related to the degree of saturation. Greater delays were incurred for cross streets with greater degrees of saturation. The conclusion reached was that unconditional priority should only be used at minor cross streets; however, it is at the major cross streets where transit vehicles are more likely to need priority treatment.

For the investigation of transit signal priority impacts during peak times, the scenarios considered were impacts upon cross streets, effects of various levels of saturation of the arterial, and effects of various bus stop locations. Finally, the overall impacts on both an individual intersection and the arterial network were evaluated.

The analysis of impacts on cross streets indicated that as the saturation level varied from 0.8 to 1.0 the negative impacts increased from minimal to significant for green extensions of ten seconds duration. When the duration increased to twenty seconds, the impacts were moderate at the 0.8 level of saturation and significant for the other levels.

Near-side and far-side bus stop locations were evaluated similarly (saturation levels, green extension durations) as stated above. Overall, the success rate of being able to utilize the green extension decreased with increasing arterial saturation, with twenty second extensions faring better than ten second extensions. The most dramatic result was the differentiation in success rates between near-side and far-side bus stop locations, with far-side locations being much more successful.

Performance at an individual intersection was evaluated based on the travel time per person. The results indicated that signal priority did not have a significant impact on travel time per person for all persons at the intersection. Since this intersection is largely populated by automobile traffic, the bus passengers do not contribute greatly to the total number of persons at the intersection. Similar results were found for the total travel time on the arterial network. If similar studies were conducted in an environment where transit had a larger mode share, the results would probably be more positive for signal prioritization.

Combination of Several Techniques

According to Kuah and Allen (1992), for the Baltimore LRT line running along Howard Street in the central business district (CBD), an earlier study had indicated that providing signal progression between Camden Street and Preston/Dolphin Street could result in travel time savings of three to five minutes. The study described below was needed to assess the effects of the suggested signal progression on other traffic in the downtown grid.

A manual method utilizing time-space diagrams was used to determine the LRT operating profile which provided progression on Howard Street for LRT while keeping disruption to cross street timing to a minimum. Average headways of fifteen minutes (7.5 minutes in each direction) and constant station dwell times of thirty seconds were assumed. To select an appropriate green bandwidth, LRT clearance times and green intervals were determined for best and worst cases. Thirty seconds was the bandwidth chosen.

The effects on cross-street progression indicated a few cases where moderate to large changes would occur. In most of these cases, the effects could be compensated by changing upstream and/or downstream signal offsets. One situation resulted in a study to change the alignment at that intersection from center to east-side. Another situation resulted in a division of the existing cross-street progression into two pieces.

Capacity analysis indicated a major reduction in level of service (LOS) at one intersection, so fifteen seconds of additional green time was added for that cross street. A few seconds of green time were also added to another cross street to alleviate some LOS degradation.

Evaluation of the street network as a whole was conducted using a TRANSYT-7F simulation. Forty-seven intersections, fifteen of which were utilized by LRT, were included in the simulation. The base-year model was calibrated by comparing maximum calculated queue lengths with actual queue lengths obtained in the field. Several MOEs including delay and travel time indicated some degradation in performance, however, the conclusion was that the traffic effects from LRT as illustrated by the system-wide MOEs were only moderate.

Provision of preferential treatment for the Hudson-Bergen Light Rail Transit System (HBLRTS) in downtown Jersey City, New Jersey was designed with the aid of the TRANSYT-7F model and Time-Space Diagram-Windows (TSD-WIN) (Abebe et al., 1996). A first cut at a timing plan was developed in TRANSYT-7F using the bus components of the model after adjusting these to emulate LRT operations. At most intersections, the additional clearance times needed for LRT due to its deceleration rate and train length resulted in LRT yellow and red lengths of four and six seconds, respectively. The LRT phase was usually able to operate concurrently with the non-conflicting automobile phase. The type of signal control utilized was semi/fully actuated. After obtaining the initial timing plan from TRANSYT-7F, TSD-WIN was used to develop timing patterns for two-way coordinated green bands between LRT stations. The final plan from TSD-WIN was then imported back into TRANSYT-7F and the model was rerun to achieve optimal vehicular network performance. During this process,

the LRT progression band developed in TSD-WIN was locked in so that preferential treatment was retained.

PLANNED IMPLEMENTATIONS

Following is a brief description of several implementations of transit signal priority which were planned at the time of publication of the referenced source. This information is helpful in the development of specific LRT signal priority strategies, which is one of the objectives of this research.

The Downtown-Holladay Street section of the Banfield Light Rail line in Portland, Oregon is in an area of small city blocks and one-way streets (Fox, 1985). Traffic signal progression is used in this section and LRT trains will utilize this progression. At stops, the trains will wait until the next green to resume progression. Phase extension is used only at minor streets when the train is detected by traffic loops for LRT directions that run against the signal progression.

In the five-mile Burnside section, which has median-running LRT, there are seventeen intersections and eight stations all with far-side platforms. Signal progression actuated by loop detectors causes the traffic controller to go to the clearance phase for conflicting movements. The preempt phase is then entered while the train is at least stopping distance plus two seconds away from the intersection.

Tighe and Patterson (1985) reported on the planned implementations of LRT priority at signalized intersections for median-running LRT in Detroit, Michigan and Santa Clara County, California. The first segment described, the Woodward Corridor in Detroit, is four miles long with three stations. Headways are projected to be as short as four minutes. Traffic signal coordination currently exists along Woodward Avenue. The large number of median openings was consolidated to a few openings for U-turns and some left turns. Left turns from side streets were not permitted. Signals along Woodward Avenue will have only two phases and partial priority for LRT will be used. Signal preemption will only be allowed at one of the two signals at U-turn slot pairs and will not be allowed in consecutive cycles. Three different signal timing plans are used at different times of the day. Total delay to "typical" trains should be approximately twelve percent of the segment run time.

The Guadalupe Corridor in Santa Clara County, an eight-mile segment which will contain thirteen stations with LRT headways as short as six minutes, differs from the Woodward Corridor described above in a few important ways. First, due to several factors, it is not possible to introduce any significant changes to left-turn opportunities. Also, many

intersections are operating at capacity. Finally, irregular intersection spacing, lack of good two-way progression, and multiple phase signal timings complicate the situation. The plan selected involves multi-phase vehicle-actuated traffic signals which will be operating in a partial priority mode when signals are coordinated, and in a full priority mode during uncoordinated operation. Any one of three algorithms for recovery of normal signalization after a full priority call will be available. The conclusion reached was that the use of traffic signal controllers for providing LRT priority while minimizing peak-hour automobile traffic delay is flexible and relatively inexpensive.

Design of signal coordination for the Long Beach to Los Angeles LRT line (LB/LA) is based on experience with other LRT systems, namely, Sacramento and San Jose (Fehon et al., 1990). The LB/LA system is more complex due to the number of jurisdictions involved and the number of intersections and gates. The design objective is to minimize LRT travel time while not having a significant negative impact on the level of service for other road traffic.

Due to the amount of existing congestion and the number of intersections involved, an active priority system is needed. The headways range from eleven to twenty minutes, and there is some spare capacity at most intersections. The resulting system design involves LRT detectors, travel time predictors, signal and sign controllers, central master computers and the SCADA center. Upon notification of an approaching train, the system first attempts partial priority by lengthening compatible phases or shortening incompatible phases so that the compatible phase will be active upon the arrival of the LRT at the intersection.

In cases where full priority is enabled, partial priority is attempted first. If this will not achieve the desired objective, then full priority, which involves omitting one or more phases, is invoked. Recovery of the normal signal timing can be recovered by a variety of procedures. For both partial and full priority, the system operator has control of various parameters to specify limits on the actions taken. Finally, the flexibility of this system allows for future fine tuning and policy changes.

FIELD RESULTS

Field results provide another source of information for development of LRT signal priority strategies. Several implementations of LRT signal priority, both in the U.S. and abroad are described below.

Transit signal priority was implemented for Phase 1 of the Sheffield Supertram System using a transponder-detector loop system (Saffer and Wright, 1994). The design

goal for the Supertram system was to maximize the priority capability at all signals, including those intersections where this capability may not be used initially, but would be available if desired in the future. Another goal was to minimize delays to other traffic as a result of the use of priority for the Supertram.

Urban Traffic Control (UTC) has the ability to disable LRT priority but still allow LRT phases to be served normally. The UTC strategy might be used for closely spaced complex intersections or at peak times. Facilities are provided to quickly identify failures in the LRT detector system so that delays to other traffic can be minimized.

The signal control strategy for a 1.6 mile section of the Light Rail Rapid Transit (LRRT) system in Buffalo, New York was described by Kessman and Cooper (1986). Most of the LRRT system is located in an underground subway, but this section runs through a transit mall in downtown, has eight signalized and coordinated intersections, and can have headways as small as two minutes. A standard Type 170 microcomputer traffic controller is used at each of the intersections. Minor wiring changes were made to the controllers and software was added to perform the additional functions needed for the LRRT interactions.

Loop detectors sense the activation of a Ready to Depart switch by the train operator. The controller receives the preempt request and notifies the train operator of when to proceed by illuminating a signal lamp, the Station Clearance Indicator (SCI). In determining which action to take the controller follows a priority sequence that is described next. The first choice would be the option that results in no delay to the train or the cross-street traffic. The next choice would be no delay to the train. Third in priority order would be an option which resulted in minimum delay to the train. Finally, if none of the above were possible, the option selected would result in minimum delays to cross-street traffic.

The constraints which must be adhered to during the controller's selection process are 1) minimum phase lengths, 2) maximum phase lengths, and 3) maximum time for the preempt. The results reported are that the system efficiently resolves conflicts between the LRRT and road traffic.

When the San Diego trolley began operation in 1981, traffic signal preemption was used at the intersections in downtown (Celniker and Terry, 1992). As the system has expanded over the years, the maximum number of trains at downtown intersections increased from eight per hour to twenty-seven per hour in 1992. Under these conditions, the preemption system was no longer adequate, so a fixed-time signal progression plan favorable to the trolley was implemented. Under this plan, trolley operators wait at the station until a fresh green is issued. They then have five seconds within which to depart the station

so as to receive signal progression to the next station. If unable to depart within the five second interval, the trolley should remain at the station until the next green signal. This easily implemented system has been successful, as indicated by the typical savings of two to three minutes per trip through downtown.

Hood et al. (1995) discussed the design and field testing of several traffic control strategies used for light rail at signalized intersections in Maryland. Prior to designing the strategies for light rail, Maryland had successfully implemented bus priority at signalized intersections using three techniques, 1) queue jumping, 2) extending the green time, and 3) phase reservicing. Queue jumping gives the bus an exclusive phase at near-side stops. Green extension is used at far-side stops. Phase reservicing, previously referred to as split phasing, is used at one location and involves adding a lagging left turn phase if a bus is present (a leading left turn phase was already in place). Results indicated a fourteen to eighteen percent reduction in bus travel time without any loss of arterial signal coordination.

For the light rail study of a future line along MD 170 at BWI airport, three options were being considered for a specific intersection where preemption is expected to occur about every fifteen minutes. The first option is preemption issued on demand by the transit vehicle. Minor movements would be served while the main street traffic is stopped. The second option limits preemption to a specific "window" within the traffic signal cycle. The third option is similar to the queue jump described above in that the controller notifies the train, which is waiting in a holding location, when to proceed through the intersection.

The second option was the one chosen since it provided a better balance between transit and non-transit needs than the other two options. Preemption techniques are already in use at an existing segment of the light rail line that runs parallel to an arterial, MD 648, with signal progression. Preemption occurs about once every seven minutes along this section of closely spaced traffic signals. Fine-tuning of the signal operations are continuing. The goal of these strategies is to provide a balance between light rail and automobile traffic which gives priority to light rail without resulting in automobile traffic gridlock.

Colquhoun et al. (1995) indicated that the signal progression along 7th avenue in downtown Calgary has been designed to minimize delays to LRT. During the A.M. peak period and during off-peak daytime hours, the cycle length is 70 seconds and the signal green split for 7th avenue is 24 - 46 seconds. For the P.M. peak period, the corresponding figures are 80 seconds and 24 –56 seconds, and the night figures are 60 seconds and 24 – 36 seconds.

Median operation occurs along 36th Street N.E., which is a four-lane arterial roadway with average daily traffic flow of approximately 35,000 to 40,000 vehicles and a high percentage of left turn and cross-street traffic on several sections. LRT is given preemption at traffic signals along this street. Stop-line loop detectors are used on all approaches and advance detectors are being considered to achieve more efficient signal operations. The LRT operation is efficient, but there have been significant delays to other traffic due to several factors of which preemption is only one.

RECOMMENDATIONS

Several authors have made recommendations regarding the use of signal priority for transit. These recommendations are considered in the selection of specific signal priority strategies for this research and are described below.

Fox (1988) suggested that if near-side stop locations are used, then either preemption should not be used or should be conditional on cross street traffic queue length. This could be done by a detector or by operators issuing the preempt call. In the latter case, operators should be trained to issue the calls only after an assessment of the traffic conditions.

Regarding median operation of LRT, Korve (1978) stated that progression speed favoring LRT generally means lowering the progression speed to include average dwell time at passenger stops. Although this means increased delay to motor vehicles, this disadvantage might be balanced by the diversion of motor vehicles to alternate routes as a result of the increased delay.

Korve also indicated that unconditional preemption should only be used in cases where the intersection is not very busy and where far-side transit stops are used. Frequent unconditional preemptions used at a busy intersection will result in unacceptable negative impacts on other traffic. Far-side locations are preferred due to the better prediction of the time of the train arrival at the intersection, thereby resulting in more preemption phase efficient timing. Conditional preemption or priority techniques consider the needs of other intersection traffic movements. This usually involves detectors for both the LRT and conflicting approaches. The controller would then assign the green time to the movement that is predicted to arrive first, or in the case of simultaneous arrivals, the controller could assign the green to the movement containing the greatest number of people. There is a great deal of communication required in this technique between the controller and the detectors.

Noyce (1996) provides guidelines for the implementation of a signal priority system. These are grouped into the following categories: 1) Transit Organization, 2) Transit Type, 3) Transit Network Characteristics, 4) Level and Type of Transit Service, 5) Stop/Station Characteristics, 6) Traffic Characteristics, 7) Signal Control System, and 8) Transit Identification System. For purposes of this work, the categories most applicable are 3, 4, 5, and 6.

For category 3, Transit Network Characteristics, the guidelines are to use transit priority when transit operates in an exclusive lane and to use signal progression favoring transit in congested areas such as the CBD. Other guidelines in this category deal with characteristics of the transit vehicles on the cross street during the peak hour. These include the use of priority when the cross-street bus volume is less than five vehicles/200 passengers and the LRT volume is less than two trains/600 passengers.

For category 4, Level and Type of Transit Service, the guidelines indicate that only fixed route service should be considered and that during the peak hour, the LRT volume should exceed two trains/600 passengers and should not exceed ten trains. If the LRT headway is less than ten minutes the delay to cross streets may be at an unacceptable level. Conditional service should be used whenever possible.

Guidelines for category 5, Stop/Station Characteristics, include the consideration of intersection spacings in relation to potential queues, LRT vehicle length, and stop locations. There have been successful implementations of transit signal priority when the cross-street intersection spacings were as small as 250 feet.

Finally, for category 6, Traffic Characteristics, the guidelines for cross streets are that the V/C ratio should be less than or equal to 0.85 and the ADT should be 25,000 or less with further reductions if a transit route is present on the cross-street. For the intersection as a whole, the V/C ratio should be less than or equal to 0.7 with 0.5 being the desirable limit. Additional considerations include the storage needed for computed queue lengths, the total person delay, maintenance of the cycle length, and the reductions in acceptable V/C ratio and cross-street ADTs due to decreasing transit headways.

For local bus service during off-peak periods, reducing signal cycle lengths and split phasing are recommended by Garrow (1997) as strategies which may be useful assuming that appropriate attention is paid to high volume intersections and progression along the bus approach and cross streets. For express bus service during off-peak periods, unconditional priority may be useful with regulations on the length of green extension and red truncation lengths, particularly at high volume cross streets.

For the peak period, the implementation of active signal priority should be used with caution due to the lack of excess capacity available especially at high volume cross streets. In general, far-side bus stops should be used, and the success of signal priority when measured in terms of overall person delay improves as the mode share for transit increases (Garrow, 1997).

SUMMARY

The analysis and implementation of various forms of LRT signal priority to date indicate mixed results regarding the effectiveness of these strategies. Results are more likely to be successful when consideration is given to the characteristics of the particular route and intersection before deciding on a priority strategy. Simulation seems to be the most flexible analysis tool for evaluation of LRT signal priority strategies.

CHAPTER 3. RESEARCH METHODOLOGY

INTRODUCTION

For this research, the method chosen to analyze transit signal priority effects was simulation. The particular simulation package chosen was CORSIM. Following is a brief description of CORSIM and the features it possesses which make it useful for simulating LRT signal priority.

CORSIM

CORSIM is a link-node based micro-simulation model that is the successor to NETSIM and is one of the tools provided as part of the Traffic Software Integrated System (TSIS) developed by ITT Systems & Sciences Corporation (ITT) for the Federal Highway Administration (FHWA). Micro-simulation models are useful when analyzing detailed strategies, since they simulate the movements of individual vehicles. Links are generally used to represent roadways and CORSIM does not explicitly model LRT guideways, so links are used in this research for both the roadways and the LRT guideway. Since links are unidirectional, a pair of links is needed to simulate two-way travel. Nodes represent intersections and have some form of traffic control device associated with them, with the exception of entry/exit nodes. A dummy node should be placed between entry/exit nodes and nodes representing actual intersections so that statistics can be collected on the traffic approaching and exiting the intersection. The traffic control associated with these dummy nodes should be a perpetual green indication (FHWA, 1998a).

CORSIM employs the concepts of time periods and time intervals. Certain aspects of a model that vary over time can be simulated through the use of different time periods. The time periods are further subdivided into time intervals that indicate the points during the simulation when intermediate MOEs are output. These MOEs include statistics such as average vehicle speed, vehicle stops, delays, travel time, distance traveled, and number of trips.

A random number generator is used to randomly assign, from user specified distributions, vehicle routing and other driver/vehicle characteristics. One random number string is used for all time-dependent stochastic decision-making processes, such as gap acceptance and duration of lane blockages. Using different random number seeds, a user can obtain replicate runs of a particular scenario.

A graphical animation tool, TRAFVU™, is also included as part of TSIS. This tool is very useful in debugging and analysis of simulations. Some items that are displayed include traffic networks, traffic, signals, MOEs, bus stations, and bus routes.

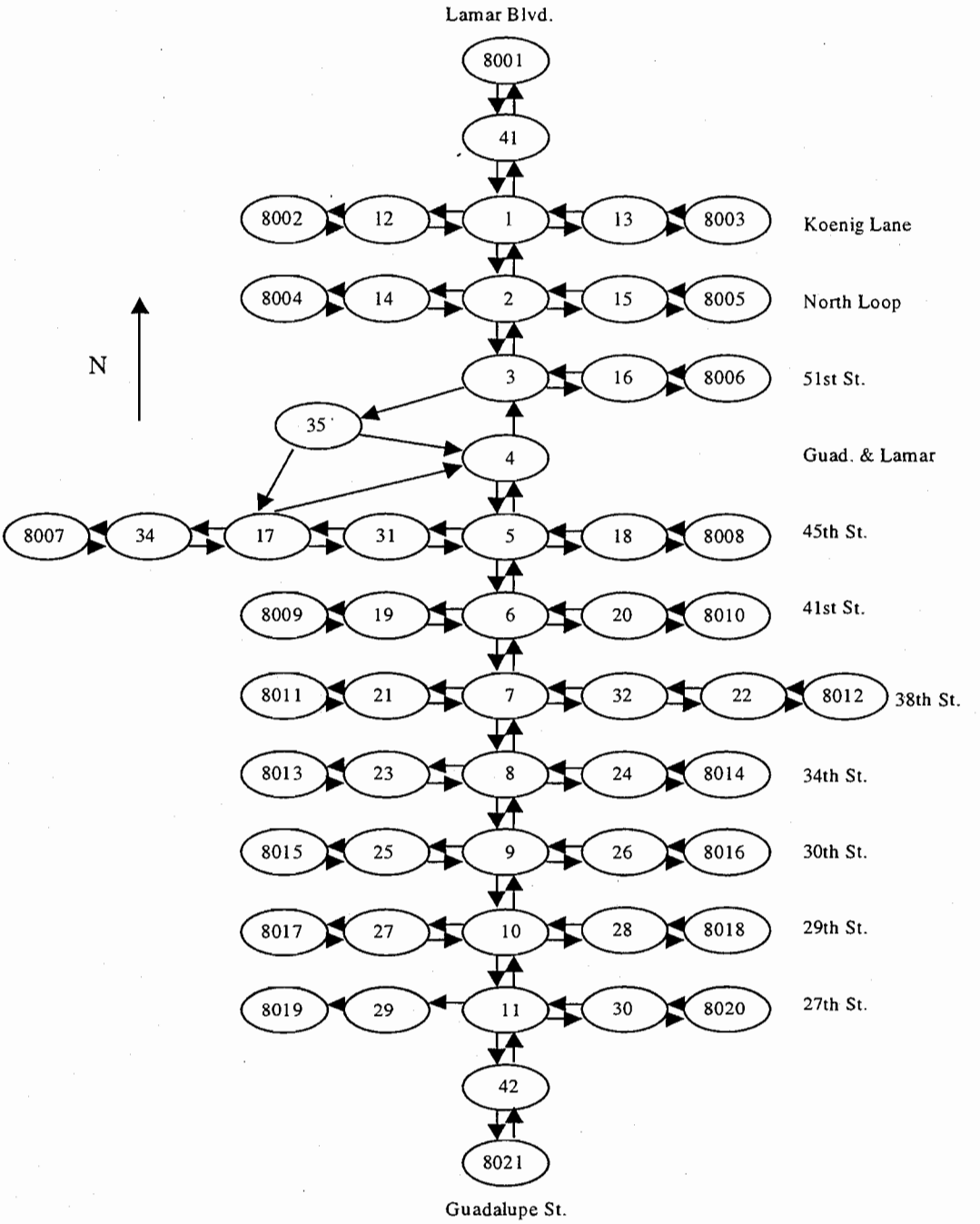
To run a simulation using CORSIM, the traffic environment must be specified by the user. For purposes of this research, the components required consist of the following (FHWA 1998a):

1) A link-node representation of the network topology
2) Roadway geometrics, such as number of lanes and turn pockets
3) Lane channelizations, such as left-turn only and bus only lanes
4) Traffic control devices, such as signs, signals, and detectors
5) Traffic volumes at network entry links
6) Turn movement percentages for each approach (link) to each node
7) Bus system specifications, such as routes, stations, headways, and dwell time distribution (which are used as a substitute for LRT)

SIMULATING LIGHT RAIL IN CORSIM

As mentioned earlier, CORSIM does not explicitly model LRT. Instead, the features of CORSIM that are used to model bus system operations can be used for modeling LRT operations with some modification to account for the operating conditions required by LRT.

Since LRT requires its own signal phase due to deceleration rates and clearance times that differ from those of automobiles, a separate set of links and nodes is used to model the LRT guideway. This separate set of nodes is required since CORSIM only allows separate signal phases for up to five approaches to an intersection. Four approaches are already in use by the automobile traffic along the simulated corridor. Since the simulated LRT travels in both the north and south directions, a total of six approaches are needed. The LRT alignment chosen for this research is in the median of the case study arterial. This implies that LRT operates in the leftmost arterial "lane". So, even if the total number of approaches needed is satisfied, there would still be a problem placing LRT in the leftmost lane since CORSIM does not allow through lanes left of left turn lanes. In addition, all cross streets are included in the LRT system even though they are not used by LRT, since the graphical animation will not work correctly without these streets present. See Figures 3.1 and 3.2 for depictions of the automobile and LRT link-node diagrams that represent the case study arterial for this research.



3.1 Automobile link-node representation of Guadalupe-N. Lamar arterial

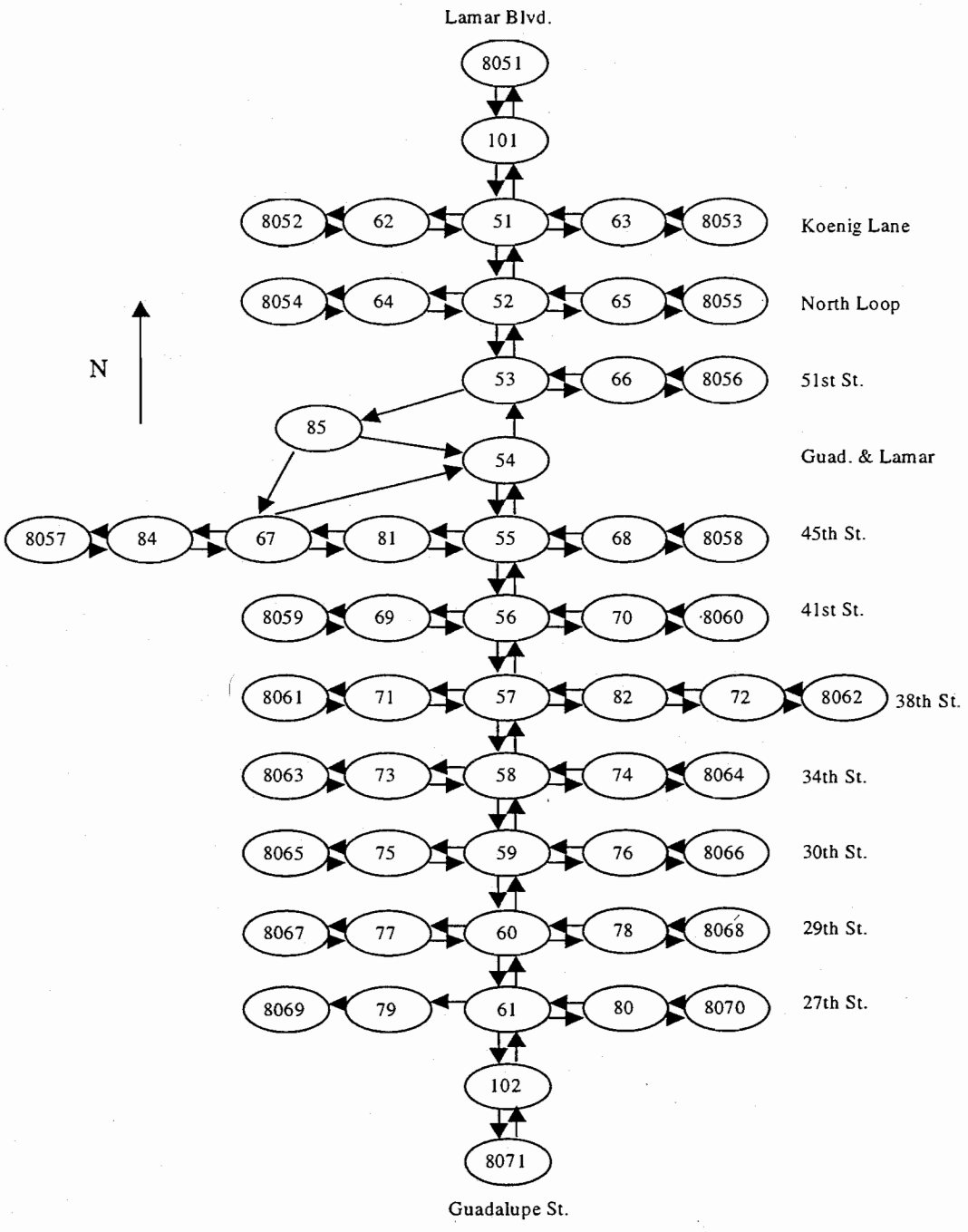


Figure 3.2 LRT link-node representation of Guadalupe-N. Lamar arterial

The only traffic that is present in the LRT link-node system is the LRT. The signal timing is a modified version of the timing in place at the corresponding nodes in the automobile system. The modifications allow for the longer clearance needed by the LRT. As mentioned above, LRT has different deceleration rates and clearance intervals from those of automobiles. The equation used to calculate the LRT clearance interval was the following (Homburger et al., 1996):

$y = t + v/2a + ((w + l)/v)$

where:
- y = nondilemma clearance interval (seconds)
- t = perception-reaction time (seconds)
- v = approach speed (feet per second squared)
- a = deceleration rate (feet per second squared)
- w = width of intersection (feet)
- l = length of vehicle (feet)

Using a perception-reaction time of 1.5 seconds, a speed of 58.66667 feet per second (i.e., 40 mph), a deceleration rate of 5 feet per second squared, an intersection width of 60 feet, and an LRT length of 200 feet, the resulting calculation is as follows:

$y = 1.5 + 58.66667/(2 * 5) + ((60 + 200) / 58.66667)$

y = approximately 12 seconds

So, a clearance time of twelve seconds is needed to allow the train to completely clear the intersection. This is divided into five seconds of yellow and seven seconds of red. These twelve seconds overlap the last twelve seconds of the associated auto green and yellow phases. The logic that CORSIM uses when determining if a vehicle will proceed through the intersection or stop upon receiving a yellow signal indication is based on an acceptable deceleration rate, which is a driver characteristic. CORSIM calculates the deceleration rate required for the vehicle to stop based on the current position and vehicle speed. Then, the driver characteristic code is used to extract the acceptable deceleration rate for the driver from a decile statistical distribution. If the acceptable deceleration extracted from this distribution exceeds the value required to stop, then the vehicle will stop; otherwise, it will continue through the intersection. To more accurately model the yellow indication reactions that would be taken by a train, the LRT clearance time calculated above should be modified.

This is needed to account for the fact that the train is simulated by a bus that uses the default distribution of deceleration rates provided in CORSIM. Since the distribution of deceleration rates applies to all vehicles in CORSIM, it is not possible to modify the distribution to account for the lower deceleration rates of a train; therefore, the clearance time calculated above for LRT must be modified for use in CORSIM.

If the LRT clearance interval is not modified from the calculation shown above, the train would stop at intersections more often than it should. If the train receives a yellow indication while outside the safe stopping distance (SSD) for automobiles but inside the SSD for trains, CORSIM would stop the vehicle at the intersection. In reality a train would not be able to stop at this position due to its lower deceleration rate and would proceed through the intersection. Since the train travels at the same speed as the parallel automobile traffic and the signal indications for the train overlap those of the parallel through auto traffic, the starting point for the modified clearance interval to use in CORSIM LRT simulation is the automobile clearance interval. A red interval is still needed, but should be shorter than the seven seconds calculated above. The red interval is only needed to account for the time that would be needed for a train to clear the intersection rather than this clearance time plus the additional time needed due to a lower deceleration rate. The time needed for the train to fully clear the intersection from the equation above is as follows:

$x = (w + l) / v$

where: x = time to traverse the intersection (seconds)
v = approach speed (feet per second squared)
w = width of intersection (feet)
l = length of vehicle (feet)

Using a speed of 58.66667 feet per second (i.e., 40 mph), an intersection width of 60 feet, and an LRT length of 200 feet, the resulting calculation is as follows:

$x = (60 + 200) / 58.66667$
x = approximately 5 seconds

So, a yellow interval of five seconds and a red interval of five seconds should result in more realistic behavior in those situations when a yellow indication begins when the vehicle a train SSD from the intersection.

To summarize, there is a two-second LRT clearance time difference between that used in CORSIM versus that obtained using LRT deceleration rates. This accounts for the time that the simulated LRT is beyond the LRT SSD when the yellow signal indication appears but before the point where CORSIM would allow the vehicle to proceed through the intersection. In these cases, the vehicle should proceed through the intersection rather than stop.

RUN-TIME EXTENSIONS IN CORSIM

With TSIS Version 4.3 alpha, users can extend certain CORSIM capabilities by designing and implementing a Run-Time Extension (RTE) (FHWA, 1998b). FHWA provided an example RTE developed by ITT that replicates the signal timing defined by the user via CORSIM input records. This example can be used as a starting point for constructing an RTE to execute various signal priority strategies. Column 77 of CORSIM input record type 36 or 43 (depending on if the node is using pre-timed or actuated control) is used to indicate those nodes whose signal timing is controlled by an RTE. The programming language used to code the RTE is usually FORTRAN or C/C++. After the RTE has been compiled and linked as a Dynamic Link Library (DLL), the user adds a new CORSIM configuration using the TSIS Traffic Tool Configuration option. When this CORSIM configuration is subsequently used to execute a simulation, the RTE extension will be called by CORSIM once per second to update the signal states. More details on the process of creating and configuring an RTE can be found in documentation provided with the RTE example code (FHWA, 1998b).

The example RTE includes one FORTRAN file, several C++ files, associated header files for the C++ files (i.e., files with a .h extension), and a make file used for compiling and linking the above mentioned files to form a DLL. The basic function of the code provided in these files is to read the CORSIM input file, store the data defining the network including links, nodes, signal timings, and detectors into object-oriented data structures, and update the signal states once per second when called by CORSIM.

GREEN EXTENSION SIGNAL PRIORITY USING A CORSIM RUN-TIME EXTENSION

As described above, the starting point for coding the signal priority RTE was the example code provided by FHWA. For the purposes of this research, the signal priority method to be tested is green extension; therefore, the example code was modified to implement a strategy providing green extensions under specific conditions. The details of the conditions and the reasons they were selected will be discussed in Chapter 4, but a general

algorithm description and steps taken to implement this algorithm using a CORSIM RTE seems appropriate at this point. The basic logic involved in the implemented green extension algorithm is to extend the LRT green phase and associated auto phase when a train is detected and the LRT signal indication is green. Since the intersections in the case study arterial are all operating under fixed-time control, this translates to switching to an alternate timing plan for both the LRT node and the associated auto node for one signal cycle. The alternate timing plan is defined using a new input file type developed for use by this RTE. The new input file has the same filename as the associated CORSIM input file, but has an extension of '.alt' rather than '.trf'. The contents of this file include record types 35 and 36 used to specify fixed-time signal control for the nodes that are to utilize green extensions. Any record number larger than 36 can be used to terminate the input file. As stated above, there may be additional conditions to be met before a switch to the alternate timing plan can be made, but those will be discussed later.

The RTE example code was modified to read the contents of the '.alt' file and store the alternate timing plans in addition to the regular timing plans. To know when to check if a green extension should be issued, surveillance detectors (using CORSIM record type 42) are specified for the LRT links immediately upstream of the nodes for which green extensions could be issued. For a green extension to occur, an alternate timing plan which provides the green extension by removing that same amount of time from the cross street through approaches must have been specified in the '.alt' file. Otherwise, an error will not occur, but the regular timing plan will be used instead. Each time a train is detected, the current time in the simulation, cumulative count of vehicles passing the detector, current signal state, and green or red time remaining are written to an new output file created by this RTE. This output file has the same filename as the associated CORSIM input file, but has an extension of '.det' rather than '.trf'. If the conditions for switching to the alternate signal timing plan are met, a log entry is made in the '.det' file to indicate that alternate signal timing is in effect. Included in this entry are the current time in the simulation and the nodes affected. At the end of one signal cycle, when the alternate timing ends, another log entry is made similar to the aforementioned one. This entry indicates that the alternate signal timing has ended. The complete source code listing for this RTE is included in Appendix A.

SUMMARY

CORSIM bus operation features modified to more accurately model LRT are used in this research to simulate LRT operation. In addition, a custom RTE is used to provide signal priority for LRT. Specific priority scenarios simulated are described in Chapter 4.

CHAPTER 4. SIMULATION SCENARIOS AND RESULTS

INTRODUCTION

Descriptions of several simulation scenarios used for evaluation of LRT signal priority strategies are presented in this chapter. Also included are person delay measures for each of the scenarios.

BASE CASE

Before developing and testing signal priority strategies, a base case for comparison purposes needs to be established. Normally, the existing conditions could serve as the base case, but there is currently no LRT operating in Austin. Therefore, some assumptions had to be made to provide an appropriate starting point for testing of LRT signal priority strategies. As stated previously, since this research follows that of Garrow (1997), the same arterial is used. One assumption is that an LRT route would replace the bus route. Since the bus route operates at ten-minute headways during the peak period, the simulated LRT will also operate at this frequency.

The particular locations for LRT stations were chosen based on an assumption of a station at the University of Texas Campus. Stations are approximately three-quarters to one mile apart at mid-block locations where the intersection spacing is large enough to accommodate a station and where the surrounding area is appropriate for an LRT station. See Figure 4.1 for a depiction of the LRT station locations, signal cycle lengths, and free flow speeds. Signal cycle lengths and free flow speeds for automobiles are identical to those for LRT.

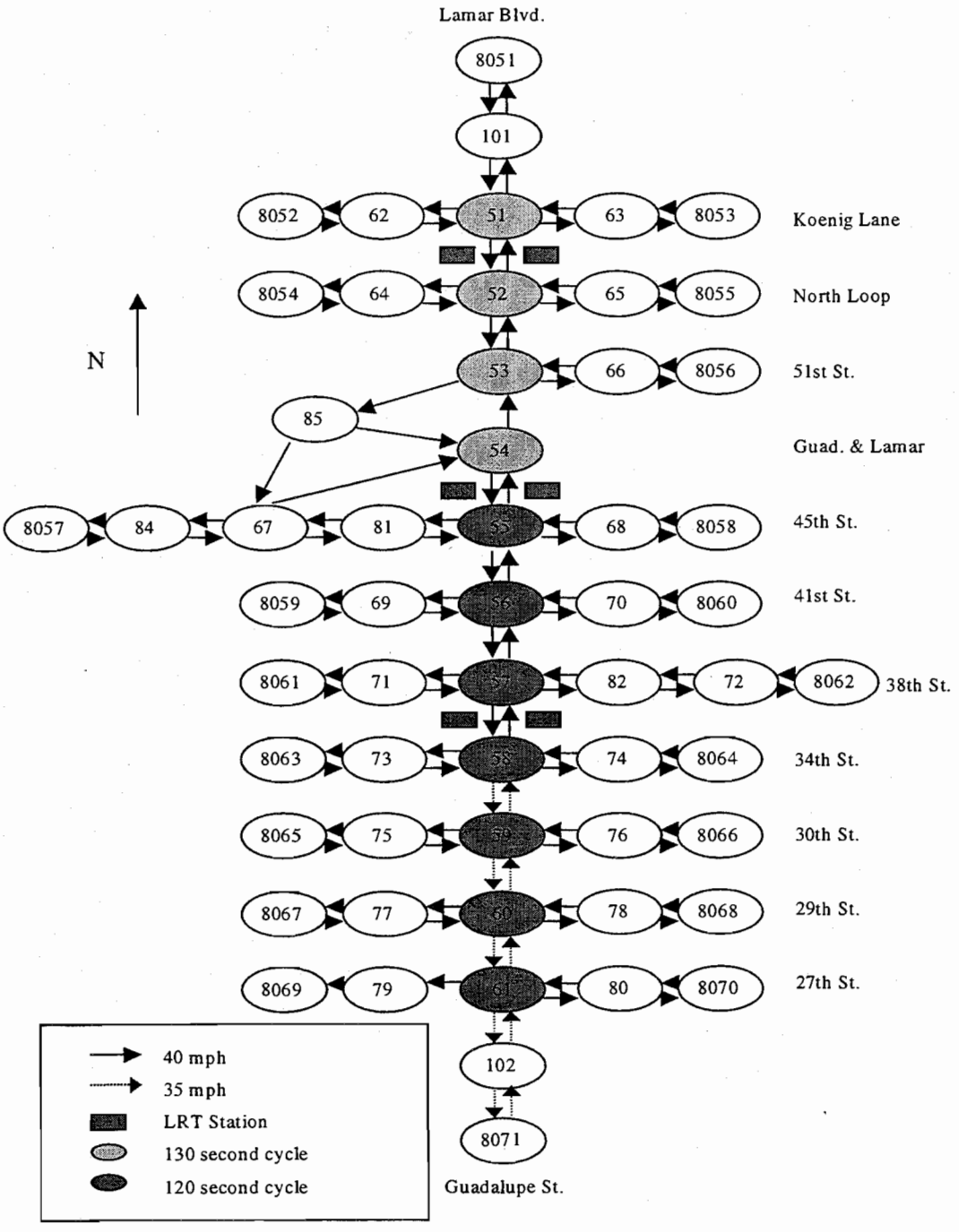


Figure 4.1 LRT link-node representation of Guadalupe-N. Lamar arterial

Regarding LRT occupancy, an initial value of 500 passengers per train in the peak direction is used. The occupancy for the non-peak direction is computed using the same proportion of non-peak to peak as that for automobiles. This proportion is 0.79 yielding a non-peak LRT occupancy of 395 passengers per train. This is intended to represent a future peak period situation when LRT along this route has a significant mode share. The assumed automobile occupancy is 1.2 persons per vehicle. With the automobile and LRT volumes used in these simulations, this translates to a mode split of 43 percent for automobiles versus 57 percent for LRT.

The automobile volumes and signal control gathered from the City of Austin during Garrow's work serve as the starting point for these components of this research. These data items are modified as discussed below due to the specific needs of LRT operating in the median of an arterial, but both the fixed-time signal operation and the existing cycle lengths are retained. Another assumption made is that no automobile travel lanes will be lost due to the LRT addition.

The choice of the peak time period for analysis is based on the findings from Garrow (1997). Some form of priority for transit during off-peak times was generally not difficult to justify due to the excess capacity available at that time. Implementation of signal priority during the peak period was found to be much more problematic due to the needs of cross street traffic at certain critical intersections. In addition, the need for transit priority was often more urgent during the peak period. One of the major reasons for this research is that the differences in the operating characteristics of LRT versus bus service could have an impact on the success of various signal priority strategies. Given the findings stated above and the differences between the two transit modes, the peak time period seemed more likely to yield results illuminating the differences between signal priority for LRT versus buses. The signal timing data mentioned above is for the afternoon peak period when northbound is the peak travel direction.

A median alignment for LRT is assumed since this is the general alignment recommendation for two-way streets (Korve et al., 1996). Median operation minimizes impacts on driveways and provides a pedestrian refuge. The choice of a median operation implies that safety measures are needed for traffic turning left across the tracks. Permissive left turns across the tracks should not be allowed when a train is approaching. With actuated control, it is possible to prohibit the permissive left turn phase only when a train is approaching; however, since this study involves fixed-time control, all permissive lefts across the tracks are prohibited. To accomplish this, some assumptions are made for those

intersections that formerly had only a permissive left phase, and now have no left turns allowed.

First, an assumption is made that twenty-five percent of this traffic chooses another route that is not part of the study network. The remaining seventy-five percent goes to the next downstream intersection and makes a loop movement consisting of two right turns followed by a left onto the cross street that was to be the receiving link for the prohibited left turn. In some instances, due to geometry or heavy volume present on the next downstream cross street, the prohibited left turn is replaced by a right turn prior to the intersection, followed by two left turns. See Figures 4.2 and 4.3 diagrams of these situations.

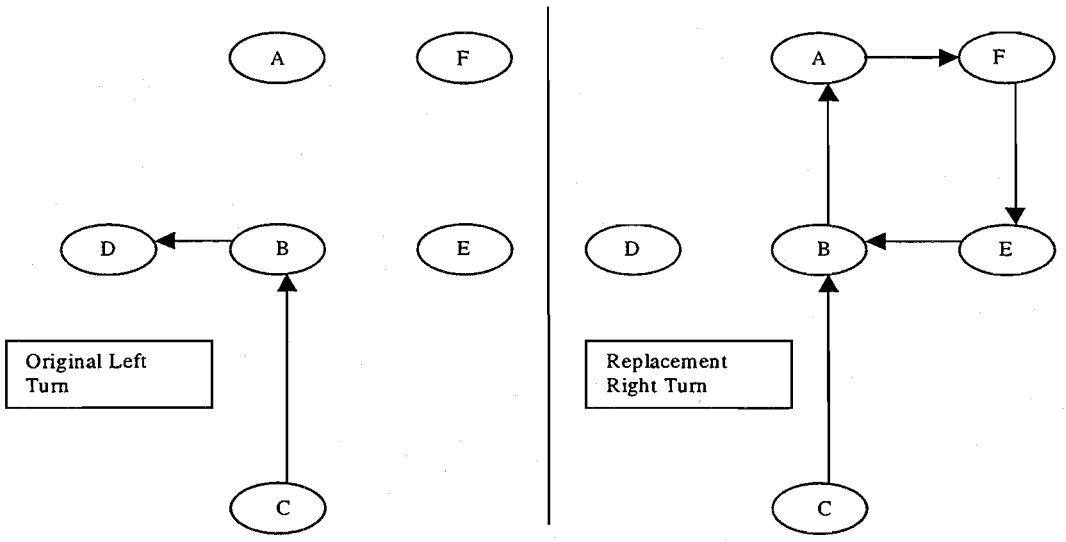


Figure 4.2 Left turn replaced by downstream right turn loop

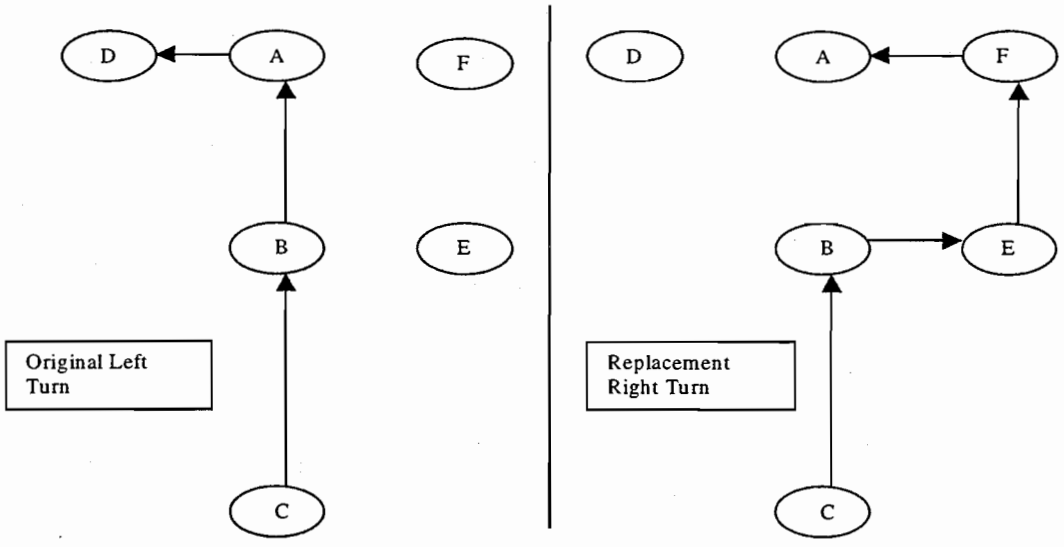


Figure 4.3 Left turn replaced by upstream right turn loop

Since the simulated network contains only signalized intersections, minor streets that would likely receive some of this diverted traffic are not included. Streets parallel to the study arterial are also not included in the model. The underestimation of travel time and delay due to travel on the parallel street during the diversion is compensated by overestimation of travel along the arterial before beginning the detour. This overestimation occurs when the vehicle travels to the next downstream intersection that could be several blocks away. For those instances where the vehicles were detoured prior to the intersection, some underestimation is still present; however, the choice of twenty-five percent leaving the network is intended to be a conservative estimate that could offset this difference.

Although it is possible that some of the permissive left turning traffic might have chosen to turn left at another intersection having a protected phase, this method of diversion is not used in the simulation. Since the protected phases are generally located at intersections with very busy cross streets with no spare green time available to add to the protected left phase, excessive left turn queues would result. In this environment, it seems likely that drivers would choose to make the detour maneuvers described above or a percentage larger than the assumed twenty-five would choose another route entirely.

The signal priority strategies chosen for simulation and evaluation include both active and passive methods. Each of these will be discussed in the following sections.

ACTIVE PRIORITY

The particular method of active priority simulated is green extension, but it is not used at specific intersections (i.e., 38th Street and 45th Street) where the cross street traffic demand is greater than the arterial. In an earlier study of this arterial, the use of active priority during the peak period at intersections with busy cross streets was found to be very disruptive to automobile traffic (Garrow, 1997). In addition, priority is only used in the peak travel direction.

Unconditional Priority

Green extension signal priority is simulated in CORSIM through use of the RTE extension mentioned in Chapter 3. The green time added for the LRT approach is taken in equal amounts from the green time for the eastbound and westbound through approaches. The signal timing modifications are made to both the automobile and LRT timing, so when a train is detected both the automobile and LRT signals switch to the alternate timing plan for one signal cycle.

The green extension duration is estimated as the amount of time needed for the train to travel at cruise speed from just beyond the upstream intersection to the SSD for the downstream intersection. At some intersections the cross street through green is not long enough to provide this amount of extension. In those cases, the green extension is the length of the cross street through green minus five seconds, so that all cross street green lengths are at least five seconds. The distance traveled during the green extension at cruise speed is used for locating the surveillance detector. The location of the SSD point for the downstream intersection minus the green extension distance provides the point for locating the surveillance detector.

For links less than six hundred feet in length, priority is not used at the downstream intersection since the SSD for LRT is approximately 330 – 440 feet depending on the cruise speed. The intervening distance leaves only a few seconds before the train would reach the SSD where the need for priority ends because from that point, the clearance interval time provides safe passage through the intersection. There is only one intersection like this along the case study arterial, and that intersection has a large green split for LRT. In each of the simulation runs, the train did not stop at this intersection.

Conditional Priority

For this research, the conditional priority strategy extends the green when a train is detected only when the green time remaining is less than the green extension time. The green extension is still the fixed amount of time as described above. This strategy is less disruptive to cross street traffic and provides the same level of priority for LRT as unconditional priority. Actual implementation of this strategy would require a controller and signal priority equipment with the capability to issue priority based on the point in the cycle when the train is detected.

The CORSIM RTE is coded to provide this type of conditional priority through the use of two columns, 74 – 75, on record type 36 in the '.alt' file. These columns are currently not used by CORSIM, so they are used by the RTE to specify the green extension length in seconds. The RTE checks these columns before switching to the alternate timing plan. If the columns are blank unconditional priority is issued; otherwise, priority is issued only if the LRT green time remaining is less than that specified by these columns.

PASSIVE PRIORITY

The base case described above provides some degree of progression which is neither one-way nor pure two-way progression. The accommodation to some extent of the off-peak direction and possibly the need to accommodate progression on particular cross streets could account for the signal timing plan obtained from the City of Austin and used in the base case.

One-Way Progression

The passive priority method chosen for evaluation in this research is progression. One-way progression is used since the simulation represents peak period travel.

Progression Segmented for LRT

To provide a progression scenario most favorable to peak direction LRT travel, the signal offsets determined above for one-way progression are modified to account for time lost by LRT due to stops at stations. The time lost includes the mean dwell time, time decelerating to a stop at the station, and time accelerating when leaving the station. The values are summed and added to the one-way offset to restart the progression at intersections immediately downstream of an LRT station.

LEFT TURN PROHIBITION

Signal timing that provides protected left turn phases for the arterial automobile traffic reduces the amount of green time for LRT by the amount of these protected phases. Protected lefts for the arterial in combination with large cross street green splits result in a small LRT green window at certain intersections. For example, the length of the green phase for northbound through automobile traffic at 38th street is 46 seconds. The corresponding value for LRT is 19 seconds. This small green window makes it very unlikely that a train would be able to maintain progression after stopping at a station. Due to demand considerations, reducing the cross street green time is not feasible. Increasing the cycle lengths at problem intersections would disrupt progression. Longer cycle lengths tend to increase delay, so increased cycle lengths along the length of the arterial are not used, particularly since the cycle lengths are already at 120 or 130 seconds. Prohibiting the remaining left turns (i.e., the protected phases) and redistributing the traffic as described above is therefore the option chosen for providing a wider LRT green band. Any excess time due to a longer protected phase for one direction (i.e. either northbound or southbound) is given to the cross street through phase since most left turn arterial traffic is diverted to cross street through approaches.

RESULTS

Person delay is the measure of effectiveness used to compare the effectiveness of the various signal priority strategies. This is computed by multiplying the number of vehicle trips for each link by the vehicle occupancy.

The person delay measures for each link are summed to form directional totals. The sum of the north and south directional totals yields the arterial total. Similarly, the cross street total is the sum of the east and west directional totals. Each of these totals is computed for LRT, automobile, and all links. Finally, the network total is the sum of all directional totals.

Protected Lefts

Two active priority simulations are used for situations where the protected left turn phases along the arterial are retained. The results from those simulations are presented below.

Base Case. The first case simulated is the base case using the existing signal timing data modified to remove permitted left turns along the arterial. The results in terms of person delay in minutes from that simulation are presented below in Table 4.1.

TABLE 4.1 PERSON DELAY (MINUTES) FOR BASE CASE

LRT

Run Number	*North*	*South*	*Arterial Total*
1	130,515	185,826	316,341
2	138,325	196,757	335,082
3	139,480	185,826	325,306
Average	136,107	189,470	325,577

Auto

Run Number	*North*	*South*	*Arterial Total*	*East*	*West*	*Cross Street Total*	*Network Total*
1	90,668	62,946	153,614	34,462	43,936	78,398	232,012
2	89,127	62,655	151,783	32,901	44,789	77,690	229,473
3	87,929	63,504	151,433	32,666	44,751	77,418	228,850
Average	89,241	63,035	152,276	33,343	44,492	77,835	230,112

Overall

Run Number	*North*	*South*	*Arterial Total*	*East*	*West*	*Cross Street Total*	*Network Total*
1	221,183	248,772	469,955	34,462	43,936	78,398	548,353
2	227,452	259,413	486,865	32,901	44,789	77,690	564,555
3	227,409	249,330	476,739	32,666	44,751	77,418	554,157
Average	225,348	252,505	477,853	33,343	44,492	77,835	555,688

Priority. Following are the results for two active priority strategies simulated, unconditional and conditional green extensions.

1. Unconditional

The results for unconditional priority in Table 4.2 show a small decrease in LRT delay and a small increase for automobile delay. The overall delay decreases slightly with this strategy.

TABLE 4.2 PERSON DELAY (MINUTES) FOR UNCONDITIONAL PRIORITY

LRT

Run Number	*North*	*South*	*Arterial Total*
1	128,480	185,826	314,306
2	120,945	196,757	317,702
3	121,935	185,826	307,761
Average	123,787	189,470	313,257

Auto

Run Number	*North*	*South*	*Arterial Total*	*East*	*West*	*Cross Street Total*	*Network Total*
1	94,592	61,794	156,386	38,259	54,001	92,259	248,646
2	90,207	62,749	152,956	38,396	50,269	88,664	241,620
3	83,239	61,463	144,701	37,952	53,210	91,161	235,863
Average	89,346	62,002	151,348	38,202	52,493	90,695	242,043

Overall

Run Number	*North*	*South*	*Arterial Total*	*East*	*West*	*Cross Street Total*	*Network Total*
1	223,072	247,620	470,693	38,259	54,001	92,259	562,952
2	211,152	259,506	470,658	38,396	50,269	88,664	559,322
3	205,174	247,289	452,463	37,952	53,210	91,161	543,624
Average	213,133	251,472	464,605	38,202	52,493	90,695	555,299

2. Conditional

For conditional priority, Table 4.3 indicates the same change in LRT delay as unconditional priority, but there is now a small decrease in automobile delay. The overall delay decreases slightly more with this strategy than with unconditional priority.

TABLE 4.3 PERSON DELAY (MINUTES) FOR CONDITIONAL PRIORITY

LRT

Run Number	*North*	*South*	*Arterial Total*
1	128,480	185,826	314,306
2	120,945	196,757	317,702
3	121,935	185,826	307,761
Average	123,787	189,470	313,257

Auto

Run Number	*North*	*South*	*Arterial Total*	*East*	*West*	*Cross Street Total*	*Network Total*
1	93,324	65,373	158,696	35,224	44,201	79,424	238,121
2	87,703	62,845	150,548	35,670	45,137	80,806	231,354
3	90,616	63,937	154,553	35,246	46,462	81,708	236,261
Average	90,548	64,051	154,599	35,380	45,266	80,646	235,245

Overall

Run Number	*North*	*South*	*Arterial Total*	*East*	*West*	*Cross Street Total*	*Network Total*
1	221,804	251,199	473,003	35,224	44,201	79,424	552,427
2	208,648	259,602	468,250	35,670	45,137	80,806	549,056
3	212,551	249,763	462,314	35,246	46,462	81,708	544,023
Average	214,334	253,521	467,856	35,380	45,266	80,646	548,502

Progression. Two types of progression are simulated. The results for each of these strategies are described below.

1. One-Way Progression

For one-way progression, LRT delay increases while automobile delay decreases by a small amount with the result that overall delay increases slightly. The person delay measures for this strategy are presented in Table 4.4.

TABLE 4.4 PERSON DELAY (MINUTES) FOR ONE-WAY PROGRESSION

LRT

Run Number	*North*	*South*	*Arterial Total*
1	159,445	183,102	342,547
2	152,625	194,060	346,685
3	145,805	194,252	340,057
Average	152,625	190,471	343,096

Auto

Run Number	*North*	*South*	*Arterial Total*	*East*	*West*	*Cross Street Total*	*Network Total*
1	75,051	67,868	142,919	35,897	45,189	81,086	224,005
2	71,378	66,962	138,340	35,143	47,514	82,657	220,997
3	69,283	68,120	137,403	34,584	45,203	79,787	217,189
Average	71,904	67,650	139,554	35,208	45,969	81,176	220,730

Overall

Run Number	*North*	*South*	*Arterial Total*	*East*	*West*	*Cross Street Total*	*Network Total*
1	234,496	250,970	485,466	35,897	45,189	81,086	566,552
2	224,003	261,021	485,024	35,143	47,514	82,657	567,681
3	215,088	262,371	477,459	34,584	45,203	79,787	557,246
Average	224,529	258,121	482,650	35,208	45,969	81,176	563,827

2. One-Way Progression Segmented for LRT

When one-way progression is segmented to account for stops at LRT stations, LRT delay shows a much larger decrease than in any of the previous strategies. Automobile delay decreases by a small amount, and the overall delay shows a sizeable decrease. More details can be found in Table 4.5.

TABLE 4.5 PERSON DELAY (MINUTES) FOR ONE-WAY LRT PROGRESSION

LRT

Run Number	*North*	*South*	*Arterial Total*
1	118,250	147,841	266,091
2	121,220	151,947	273,167
3	110,550	151,899	262,449
Average	116,673	150,562	267,235

Auto

Run Number	*North*	*South*	*Arterial Total*	*East*	*West*	*Cross Street Total*	*Network Total*
1	75,004	65,960	140,964	35,231	43,459	78,690	219,655
2	77,619	66,398	144,017	34,716	45,288	80,004	224,020
3	75,613	64,463	140,076	33,122	44,050	77,172	217,248
Average	76,079	65,607	141,686	34,357	44,265	78,622	220,308

Overall

Run Number	*North*	*South*	*Arterial Total*	*East*	*West*	*Cross Street Total*	*Network Total*
1	193,254	213,801	407,055	35,231	43,459	78,690	485,745
2	198,839	218,345	417,184	34,716	45,288	80,004	497,188
3	186,163	216,361	402,525	33,122	44,050	77,172	479,696
Average	192,752	216,169	408,921	34,357	44,265	78,622	487,543

No Lefts

Each of the strategies above is also simulated with all left turns prohibited along the arterial. The results of those simulations are described below.

Base Case. The base case with no left turns allowed produced a large decrease in LRT delay and a moderate decrease in automobile delay. A large decrease in delay is thus reflected in the overall measure. The specific amounts can be seen in Table 4.6.

TABLE 4.6 PERSON DELAY (MINUTES) FOR NO LEFT TURNS

LRT

Run Number	*North*	*South*	*Arterial Total*
1	128,480	121,107	249,587
2	133,595	110,679	244,274
3	135,190	121,391	256,581
Average	132,422	117,726	250,147

Auto

Run Number	*North*	*South*	*Arterial Total*	*East*	*West*	*Cross Street Total*	*Network Total*
1	82,131	48,572	130,703	34,948	48,981	83,929	214,632
2	80,772	50,725	131,497	34,506	46,222	80,727	212,224
3	79,021	50,050	129,071	34,100	47,262	81,362	210,434
Average	80,641	49,782	130,424	34,518	47,488	82,006	212,430

Overall

Run Number	*North*	*South*	*Arterial Total*	*East*	*West*	*Cross Street Total*	*Network Total*
1	210,611	169,679	380,290	34,948	48,981	83,929	464,219
2	214,367	161,404	375,771	34,506	46,222	80,727	456,498
3	214,211	171,442	385,653	34,100	47,262	81,362	467,015
Average	213,063	167,508	380,571	34,518	47,488	82,006	462,577

Priority. Results for the green extension strategies in conjunction with the left turn prohibition are presented below.

1. Unconditional

Results for unconditional priority indicate a slight improvement for LRT and a similar degradation for automobiles from the base case with no left turns allowed. Overall delay improves by a very small amount as can be seen in Table 4.7.

TABLE 4.7 PERSON DELAY (MINUTES) FOR NO LEFT TURNS & UNCONDITIONAL PRIORITY

LRT

Run Number	*North*	*South*	*Arterial Total*
1	126,720	121,107	247,827
2	119,240	110,679	229,919
3	120,230	121,391	241,621
Average	122,063	117,726	239,789

Auto

Run Number	*North*	*South*	*Arterial Total*	*East*	*West*	*Cross Street Total*	*Network Total*
1	80,328	47,334	127,662	40,483	55,178	95,661	223,323
2	75,090	47,997	123,087	39,371	53,199	92,570	215,657
3	75,257	47,389	122,646	38,202	55,046	93,248	215,894
Average	76,891	47,574	124,465	39,352	54,475	93,827	218,292

Overall

Run Number	*North*	*South*	*Arterial Total*	*East*	*West*	*Cross Street Total*	*Network Total*
1	207,048	168,441	375,489	40,483	55,178	95,661	471,150
2	194,330	158,676	353,006	39,371	53,199	92,570	445,576
3	195,487	168,780	364,268	38,202	55,046	93,248	457,516
Average	198,955	165,299	364,254	39,352	54,475	93,827	458,081

2. Conditional

LRT delay for the conditional and unconditional priority strategies are the same; however, the automobile delay decreases by a small amount with a similar decrease in overall delay. Specific delay measures are shown in Table 4.8.

TABLE 4.8 PERSON DELAY (MINUTES) FOR NO LEFT TURNS & CONDITIONAL PRIORITY

LRT

Run Number	*North*	*South*	*Arterial Total*
1	126,720	121,107	247,827
2	119,240	110,679	229,919
3	120,230	121,391	241,621
Average	122,063	117,726	239,789

Auto

Run Number	*North*	*South*	*Arterial Total*	*East*	*West*	*Cross Street Total*	*Network Total*
1	84,170	49,062	133,232	34,551	47,092	81,643	214,875
2	79,066	51,441	130,508	35,340	46,643	81,983	212,491
3	78,967	49,180	128,148	34,534	47,564	82,098	210,246
Average	80,735	49,894	130,629	34,808	47,100	81,908	212,537

Overall

Run Number	*North*	*South*	*Arterial Total*	*East*	*West*	*Cross Street Total*	*Network Total*
1	210,890	170,169	381,059	34,551	47,092	81,643	462,702
2	198,306	162,120	360,427	35,340	46,643	81,983	442,410
3	199,197	170,572	369,769	34,534	47,564	82,098	451,867
Average	202,798	167,620	370,418	34,808	47,100	81,908	452,326

Progression. The results for two progression strategies with left turns prohibited along the arterial are described below.

1. One-Way Progression

LRT delay for the one-way progression strategy shows a small decrease when measured against the delay for the base case when left turns are allowed, but when compared to the prohibited left turns base case, LRT delay actually increases by a sizeable amount. Automobile delay shows an improvement from the base case for no left turns allowed, but the overall delay is larger due to the degradation in LRT performance. See Table 4.9 below for specifics.

TABLE 4.9 PERSON DELAY (MINUTES) FOR NO LEFT TURNS & ONE-WAY PROGRESSION

LRT

Run Number	*North*	*South*	*Arterial Total*
1	155,210	164,913	320,123
2	144,760	166,757	311,517
3	144,540	167,829	312,369
Average	148,170	166,500	314,670

Auto

Run Number	*North*	*South*	*Arterial Total*	*East*	*West*	*Cross Street Total*	*Network Total*
1	57,337	54,937	112,274	36,732	46,807	83,539	195,813
2	58,003	56,951	114,955	36,107	47,984	84,091	199,046
3	55,124	56,408	111,533	35,129	45,869	80,997	192,530
Average	56,822	56,099	112,920	35,989	46,887	82,876	195,796

Overall

Run Number	*North*	*South*	*Arterial Total*	*East*	*West*	*Cross Street Total*	*Network Total*
1	212,547	219,849	432,397	36,732	46,807	83,539	515,936
2	202,763	223,709	426,472	36,107	47,984	84,091	510,563
3	199,664	224,237	423,901	35,129	45,869	80,997	504,899
Average	204,992	222,598	427,590	35,989	46,887	82,876	510,466

2. One-Way Progression Segmented for LRT

Table 4.10 indicates that the final strategy evaluated shows the best results for overall delay. LRT delay is approximately the same as that for the active priority strategies with left turns prohibited. Automobile delay increases from the one-way progression case but is still decreased from the no left turns base case.

TABLE 4.10 PERSON DELAY (MINUTES) FOR NO LEFT TURNS & ONE-WAY LRT PROGRESSION

LRT

Run Number	*North*	*South*	*Arterial Total*
1	97,020	143,006	240,026
2	92,400	147,651	240,051
3	99,880	146,929	246,809
Average	96,433	145,862	242,295

Auto

Run Number	*North*	*South*	*Arterial Total*	*East*	*West*	*Cross Street Total*	*Network Total*
1	66,665	60,185	126,850	36,458	45,803	82,261	209,111
2	69,284	59,936	129,220	34,837	47,979	82,816	212,036
3	65,643	57,897	123,541	34,825	45,484	80,308	203,849
Average	67,197	59,340	126,537	35,373	46,422	81,795	208,332

Overall

Run Number	*North*	*South*	*Arterial Total*	*East*	*West*	*Cross Street Total*	*Network Total*
1	163,685	203,191	366,876	36,458	45,803	82,261	449,137
2	161,684	207,587	369,271	34,837	47,979	82,816	452,087
3	165,523	204,827	370,350	34,825	45,484	80,308	450,658
Average	163,630	205,202	368,832	35,373	46,422	81,795	450,628

SUMMARY

The person delay measures for the strategies described in this chapter exhibit some degree of variation. Table 4.11 below summarizes the network delay for each of the simulation scenarios. Chapter 5 presents a statistical evaluation of these person delay measures to elucidate meaningful differences.

TABLE 4.11 NETWORK PERSON DELAY (MINUTES) FOR ALL PRIORITY STRATEGIES

Strategy	*LRT Network Total*	*Auto Network Total*	*Overall Network Total*
Base Case	325,577	230,112	555,688
Unconditional Priority	313,257	242,043	555,299
Conditional Priority	313,257	235,245	548,502
One-Way Progression	343,096	220,730	563,827
One-Way LRT Progression	267,235	220,308	487,543
No Lefts Base Case	250,147	212,430	462,577
No Lefts Unconditional Priority	239,789	218,292	458,081
No Lefts Conditional Priority	239,789	212,537	452,326
No Lefts One-Way Progression	314,670	195,796	510,466
No Lefts One-Way LRT Progression	242,295	208,332	450,628

CHAPTER 5. EVALUATION

INTRODUCTION

The experiments conducted in this research are repeated measures designs using person delay as the dependent variable. The same three random number seeds (i.e., the subjects) are used to test each experimental condition which consist of chosen levels of three factors. The left turn factor has two levels, protected left turns allowed and no left turns allowed. Three signal priority levels are used including none, unconditional, and conditional. The final factor is progression with three levels including existing, one-way, and one-way segmented for LRT. Since priority and progression are not used together, there are actually two separate experiments. The first has factors of left turns and priority, and the second has factors of left turns and progression.

Analysis of variance (ANOVA) was used to analyze the data. Certain assumptions should be met before using ANOVA for a particular data set. Independence of observations within the groups was met by using the random number generator in CORSIM to produce the "subjects". Since all the factors in these experiments are within subjects factors, this is the only independence of observations assumption to be met. The normality assumption was tested using the using the One-Sample Kolmogorov-Smirnov test, and Mauchly's test was used for the sphericity assumption. Neither of these assumptions was violated at an alpha level of .05 in any of the experiments so proceeding with the ANOVA was appropriate.

An experiment-wide alpha level of .05 was selected as the decision rule for null hypotheses rejection. When specific contrasts were conducted, the Bonferroni step-down adjustment was used to maintain a .05 alpha level. This method was chosen to provide more power since the pure Bonferroni adjustment is extremely conservative.

LEFT TURNS AND PRIORITY

For the experiment with factors for left turns and priority, the test of the main effect for priority indicates that priority does not produce a significant effect; however left turns does produce a significant effect. There is no significant interaction effect. The specifics including the F-statistic, significance level, and effect size (Eta Squared) are shown in Table 5.1.

TABLE 5.1 ANOVA RESULTS FOR FACTORS OF LEFT TURNS AND PRIORITY

Tests of Main and Interaction Effects	*Df*	*F*	*Sig.*	*Eta Squared*
Priority	2, 4	1.656	0.299	0.453
Lefts	1, 2	186.633	0.005	0.989
Priority * Lefts	2, 4	1.027	0.437	0.339

Figure 5.1 displays overall person delay for each of the simulation runs in the experiment involving left turns and priority.

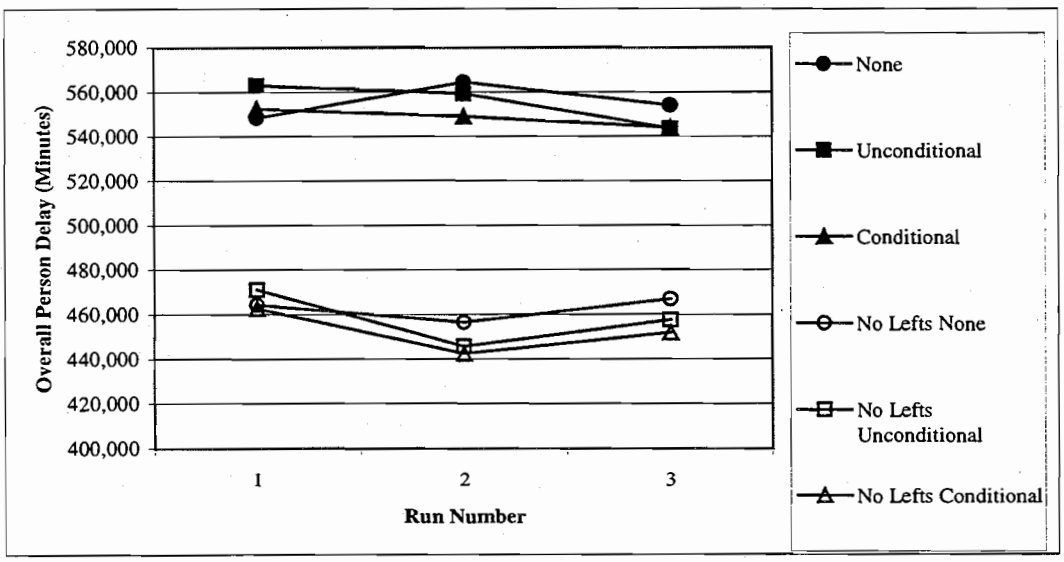


Figure 5.1 Overall person delay for left turns and priority

LEFT TURNS AND PROGRESSION

For the experiment examining left turn and progression effects, the main effects (both left turns and progression) and the interaction effect are significant. Due to the finding of a significant interaction effect, simple effects analyses were conducted. For left turns at each progression level, all the tests indicate a significant effect using a reduced alpha level. An alpha level of .05/3 = .0167 was used for the test with the smallest significance level. Using the step-down adjustment, the alpha level for the test with the next smallest significance level

was .05/2 = .025, and .05 was the alpha level used for the final test. Alpha levels of .008, .010, .0125, .0167, .025, and .05 were used for the pairwise comparisons among progression levels at each left turn level. The results for protected left turns allowed indicate a significant difference between existing progression and one-way progression segmented for LRT. Also significant is the difference between one-way progression and one-way progression segmented for LRT. There is no significant difference between existing progression and one-way progression. With no left turns allowed, significant differences are found between existing versus one-way progression and one-way versus one-way LRT progression. See Table 5.2 for details on the results of each of these tests.

TABLE 5.2 ANOVA RESULTS FOR FACTORS OF LEFT TURNS AND PROGRESSION

Tests of Main and Interaction Effects	*Df*	*F*	*Sig.*	*Eta Squared*
Progression	2, 4	207.353	0.000	0.990
Lefts	1, 2	185.373	0.005	0.989
Progression * Lefts	2, 4	77.902	0.001	0.975
Tests of Simple Effects				
Left Turn Factor comparisons at each level of Progression Factor				
Protected vs. No Lefts at Existing Progression	1, 2	153.175	0.006	0.987
Protected vs. No Lefts at One-Way Progression	1, 2	753.318	0.001	0.997
Protected vs. No Lefts at One-Way LRT Progression	1, 2	63.31	0.015	0.969
Progression Factor comparisons at each level of Left Turn Factor (Protected Lefts)				
Existing vs. Auto Progression at Protected Lefts	1, 2	2.617	0.247	0.567
Existing vs. One-Way LRT Progression at Protected Lefts	1, 2	391.575	0.003	0.995
One-Way vs. One-Way LRT Progression at Protected Lefts	1, 2	628.011	0.002	0.997
(No Lefts)				
Existing vs. Auto Progression at No Lefts	1, 2	89.995	0.011	0.978
Existing vs. One-Way LRT Progression at No Lefts	1, 2	9.955	0.087	0.833
One-Way vs. One-Way LRT Progression at No Lefts	1, 2	263.165	0.004	0.992

Table 5.3 below summarizes test results for the experiment involving left turns and progression. The values displayed are mean person delays for the condition represented by the row and column headings. The display shows whether a significant effect represents an increase or a decrease in person delay. Subsets with no significance difference have a common superscript. For example, when no left turns are allowed, there is no significant difference between existing and one-way LRT progression, so these entries have a common superscript (i.e., c). The significant difference between one-way and one-way LRT progression is shown by the lack of a common superscript. The superscripts should be interpreted in terms of row and column factors only, no diagonal comparisons, since all possible pairwise combinations of the two factors were not tested for significant differences.

TABLE 5.3 HOMOGENEOUS SUBSETS FOR LEFT TURNS AND PROGRESSION (OVERALL PERSON DELAY MINUTES)

	Existing Progression	**One-Way Progression**	**One-Way LRT Progression**
Protected Lefts	555,688[a]	563,826[a]	487,543[b]
No Lefts	462,577[c]	510,466[d]	450,627[c]

Figure 5.2 displays overall person delay for each of the simulation runs in the left turns and progression experiment.

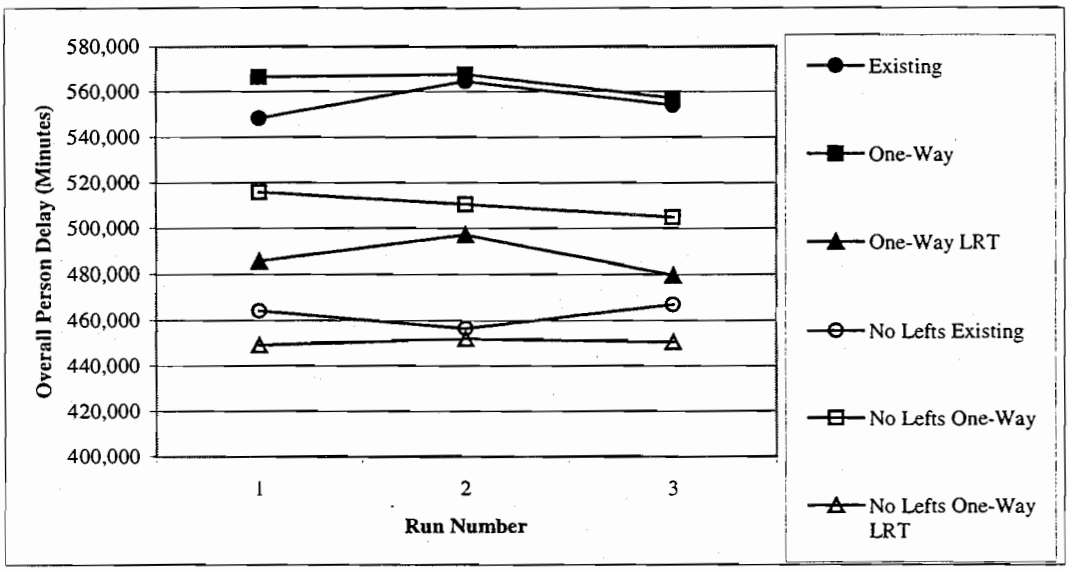


Figure 5.2 Overall person delay for left turns and progression

LRT PERSON DELAY

Although overall person delay provides a global measure of effectiveness by which to evaluate signal timing changes, when the reason for considering a change to signal timing is to provide priority for LRT, the LRT person delay measure should be used in conjunction with overall delay in evaluating various strategies. It is possible that the results of a particular strategy could show significant improvement in overall delay without providing any significant improvement for LRT delay. In that case, the original objective of providing priority for LRT has not been met. In addition, if a policy of encouraging a mode shift to transit through signal timing is pursued, criteria for selection of a particular strategy might be based on a significant improvement in LRT delay with some restriction on the amount of automobile delay degradation.

When the experiments described above are evaluated using LRT person delay as the dependent variable, none of the findings of significant differences change. The detailed results of these tests can be found in Appendix B.

LRT OCCUPANCY

One of the recommendations for further research from Garrow (1997) was evaluation of transit signal priority in a network where transit has a substantial mode share. The LRT occupancy level assumed in computing the person delay measures above was 500 per

northbound train and 395 per southbound train. This level of LRT occupancy represents a large transit mode share. Occupancy for the southbound train was computed using the same proportion of southbound to northbound travel represented by the automobile traffic. It seems likely, however, that in the case of LRT, a larger percentage of the trips are for commuting purposes. In that case, the southbound occupancy is more on the order of 50 percent of the northbound occupancy, and the mode split is 49 percent automobiles and 51 percent LRT.

When the data is analyzed assuming a southbound LRT occupancy of 250 per train, one test changes from not significant to significant. This change is for the comparison of existing versus one-way LRT progression when no left turns are allowed. The finding is significant when either overall person delay or LRT person delay is used as the dependent variable. Appendix B contains the ANOVA results and Table 5.4 below shows the means and homogeneous subsets when the southbound LRT occupancy is 250 per train.

TABLE 5.4 HOMOGENEOUS SUBSETS WITH SB LRT OCCUPANCY = 250
(OVERALL PERSON DELAY MINUTES)

	Existing Progression	**One-Way Progression**	**One-Way LRT Progression**
Protected Lefts	486,136[a]	493,907[a]	432,274[b]
No Lefts	419,362 [c]	449,346[d]	397,083[e]

If a more conservative estimate of LRT occupancy is used, such as 150 persons per train in the northbound direction, some changes in significance occur when overall person delay is used as the dependent variable. LRT person delay findings are affected by changes in the ratio of southbound to northbound occupancy as described above, but not by the occupancy level.

With 150 and 119 persons per train in the northbound and southbound directions respectively, the mode split is 72 percent for automobiles and 28 percent for LRT. The results for protected lefts are not affected by the change in occupancy. When no left turns are allowed, the findings change to indicate that the only significant difference in progression strategies is between one-way and one-way LRT progression. See Table 5.5 below for the means and homogeneous subsets in this situation.

TABLE 5.5 HOMOGENEOUS SUBSETS WITH SB LRT OCCUPANCY = 119
(OVERALL PERSON DELAY MINUTES)

	Existing Progression	One-Way Progression	One-Way LRT Progression
Protected Lefts	327,785 [a]	323,659[a]	300,478[b]
No Lefts	287,474[cd]	290,197[c]	281,021[d]

Finally, when the northbound occupancy is 150 persons per train and the southbound occupancy is 75 persons per train, the mode split is 76 percent for automobiles and 24 percent for LRT. There are no longer any significant differences detected between progression strategies when left turns are prohibited. The only significant difference with protected left turns is between existing and one-way LRT progression. Table 5.6 shows the means and homogeneous subsets.

TABLE 5.6 HOMOGENEOUS SUBSETS WITH SB LRT OCCUPANCY = 75
(OVERALL PERSON DELAY MINUTES)

	Existing Progression	One-Way Progression	One-Way LRT Progression
Protected Lefts	306,919[a]	302,683[ab]	283,897[b]
No Lefts	274,510[b]	271,861[b]	264,957[b]

SUMMARY

The use of LRT delay as the dependent variable does not pose any conflicts with overall delay. In situations where findings of significance differ based on the dependent variable, a particular factor provides a significant LRT delay improvement but shows no significant difference for overall delay.

Since prohibiting left turns is significant at each level of progression and was the only significant effect in the experiment involving the factors of left turns and priority, this seems like a promising approach for favorable LRT signal timing. This strategy may be problematic when evaluated in a multiple criteria decision framework due to adjacent land users concerns.

The success of the various progression scenarios when measured in terms of LRT person delay indicates that in all cases one-way progression segmented for LRT provides a significant improvement over one-way progression. One-way progression is probably not very effective due to the large proportion of automobile traffic in the non-peak direction. Comparison to existing progression depends on the prohibition of left turns and the occupancy ratio of the southbound LRT to the northbound LRT. With protected left turns, one-way LRT progression significantly reduces delay. When left turns are prohibited, if the southbound occupancy is 79 percent of the northbound occupancy, there is no significant difference between these progression strategies. There is a significant improvement with one-way LRT progression when the southbound occupancy is 50 percent of the northbound occupancy.

The use of green extension signal priority does not seem to be a very useful strategy for reducing LRT person delay since there was no finding of significant differences in delay due to priority.

CHAPTER 6. CONCLUSIONS

RESEARCH FINDINGS

The findings of this research indicate that passive priority strategies are more effective than active priority strategies in reducing LRT person delay. In most of the passive priority scenarios overall person delay is also significantly reduced. The particular technique that seems most effective is prohibition of left turns. This finding seems reasonable since it increases the size of the green window for LRT, thereby making it more likely that the train will be able to maintain progression. It is recognized, however, that implementing this strategy may not be feasible due to other considerations.

Signal progression segmented to account for stops at LRT stations also shows favorable results. The use of this method with or without left turn prohibitions shows significant reductions in LRT person delay subject to some considerations of non-peak LRT occupancy.

An active priority technique, green extension signal priority, shows no significant reduction in delay. There are other forms of active priority, particularly full priority, which may show better results in other situations. In this research, certain intersections are excluded from the use of any form of active priority due to the findings of earlier research regarding the amount of disruption to cross street automobile traffic (Garrow, 1997). Since these are the very intersections where priority is usually needed, the limitation here is not with the type of active priority tested but with the lack of intersections which are good candidates for the use of these techniques.

INDICATIONS FOR FURTHER RESEARCH

The finding of the effectiveness of left turn prohibition in reducing delay in this research was based on several assumptions as to the dispersal of the traffic that formerly made left turns. Since this traffic was assumed to utilize some parallel streets in making right turn loops to get to the desired cross street, an expansion of the study network to include minor (unsignalized) and parallel streets (particularly arterials) would provide a better picture of impacts of this strategy. In particular, the reduction in delay in the smaller network represented by this study may not exist in the larger network. The offloading of left turn traffic to other streets in this larger network could result in increases to overall delay depending on the capacity available to handle this extra traffic.

Another assumption made regarding the left turning traffic was that twenty-five percent of this traffic left the network. This percentage could be varied to see how the resulting delay measures are affected. This variation could be done within the current study network or in the larger network described above.

The success of one-way progression segmented for LRT suggests further research using other types of segmented progression strategies during the peak period. These strategies would directly consider the southbound traffic when designing progression.

In a network with characteristics more favorable to active priority strategies, an evaluation of other types of active priority could be useful. In addition, results for active priority may differ if priority is available in both directions of LRT travel.

APPENDIX A. SOURCE CODE FOR CORSIM RTE

```
// Copyright 1997 <Kaman Sciences Corporation>.  All Rights Reserved
// CBinarySequence

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <afxtempl.h>       // MFC templates

class CInteger;

class CBinarySequence:public CObject
{
public:  //implementation
            CBinarySequence();
            ~CBinarySequence();

public:  //Overrides

public:  //data
            CTypedPtrList<CPtrList, CInteger*> m_sequence;

};
```

```
// Copyright 1997 <Kaman Sciences Corporation>.  All Rights Reserved
//CDetector
#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <afxtempl.h>

class CLane;
class CLink;

class CDetector:public CObject
{

public:  //implementation
            CDetector();
            ~CDetector();

public:  //data
            int m_id;
            int m_length;
            CString m_type;  //presence or pulse
            CLink* m_Link;
            int m_distanceFromDownstreamNode;
            CLane* m_Lane;
            int m_CorsimId;
            int m_count;
            BOOL m_state;  //activated=TRUE or not=FALSE
            float m_activationTime;
            float m_deactivationTime;

};
```

```
// Created by Beth Taylor in May-June 1998
//initvars.h
extern int endOfInit;
extern int prevInit;
extern int prevTime;
extern FILE* fptr;
extern BOOL altSignal;
extern CNetwork* pNetwork;
```

```
// Copyright 1997 <Kaman Sciences Corporation>.  All Rights Reserved
//CInteger

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <afxtempl.h>

class CInteger:public CObject
{

public:  //implementation
            CInteger();
            ~CInteger();

public:  //Overrides

public:  //data
            int data;
};
```

```
//CInterface
#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <afxtempl.h>

class CNetwork;

class CInterface:public CObject
{

public:  //implementation
            CInterface();
            ~CInterface();

public:  //data
            CString m_trfFileName;
            CString m_Simulator;
            CString m_ProtoName;
            CNetwork* m_Network;

};
```

```
// Copyright 1997 <Kaman Sciences Corporation>.  All Rights Reserved
// CLane
#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <afxtempl.h>

class CLink;
class CDetector;

class CLane:public CObject
{

public:  //implementation
            CLane();
            ~CLane();

public:  //data
            int m_id;       //lane id 1,2,..,7 see card type 11
            CString m_type;  //bay or full
            CLink* m_Link;
            int m_length;
            int m_leftMovementPercent;
            int m_thruMovementPercent;
            int m_rightMovementPercent;
            CTypedPtrList<CPtrList,CDetector*> m_DetectorList;
            CTypedPtrList<CPtrList,CDetector*> m_StopBarDetectorList;

};
```

```
// Copyright 1997 <Kaman Sciences Corporation>.  All Rights Reserved
// Modified by Beth Taylor in May-June 1998
//          - Added pSignalState parameter to ProcessDetectors declaration
//          - Added m_altcode[12]
//CLink class
#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <afxtempl.h>

class CNode;
class CLane;
class CDetector;
class CBinarySequence;
class CSignalState;

class CLink:public CObject
{

public:  //implementation
            float ComputeTravelTime();
            CLane* FindLane(int corsimId);
            void ProcessDetectors(CSignalState* pSignalState);
            CBinarySequence* ConvertToBinary(int j);
            CLink();
            ~CLink();

public:  //data
            void SetSDCCode();
            void CreateSignalStates();
            int   m_id;
            int   m_CorsimId;
            CNode* m_upnode;
            CNode* m_dnnode;
            CNode* m_thrunode;
            CNode* m_leftnode;
            CNode* m_rightnode;
            CTypedPtrList<CPtrList,CLane*> m_listOfLanes;
            CTypedPtrList<CPtrList,CDetector*> m_listOfDetectors;
            CTypedPtrList<CPtrList,CSignalState*> m_signalStates;
            int m_length;
            int m_numOfFullLanes;
            int m_numOfLeftTurnBays;
            int m_numOfRightTurnBays;
            int m_lengthOfLeftBay;
            int m_lengthOfRightBay;
            int m_freeFlowSpeed;
            float m_travelTime;
            int m_leftMovementPercent;
            int m_thruMovementPercent;
            int m_rightMovementPercent;
            CString m_channelCode[7];
            int m_NumOfDetectors;
            int m_code[12];
            int m_altcode[12];
            int m_offset;
            POSITION m_pos;
            CLink* m_opposingLink;
            int m_opposingID;

};
```

```
// Copyright 1997 <Kaman Sciences Corporation>.  All Rights Reserved
// Modified by Beth Taylor in May-June 1998
//          - Moved some variables to upcntrl.h
#ifndef _NETSIM_H_
#define _NETSIM_H_
#define DLL_IMPORT extern __declspec( dllimport )


#define IMXLNK 500
#define IMXNOD 250
#define IMXACT 100
#define IMXDET 700
#define IMXVEH 10000

//Moved to upcntrl.h
//int endOfInit;
//int prevInit;
//int prevTime;

extern "C"
{

DLL_IMPORT struct {int CNTENT[IMXLNK];}      SIN026;
#define cntent                  SIN026.CNTENT

DLL_IMPORT struct {int LANEF[IMXLNK * 7];}   SIN038;
#define lanef                   SIN038.LANEF

DLL_IMPORT struct {int FOLOWR[IMXVEH];}      SIN824;
#define folowr                  SIN824.FOLOWR

DLL_IMPORT struct {int TCODE[IMXVEH];}       SIN003;
#define tcode                   SIN003.TCODE

DLL_IMPORT struct {int VSTATE[IMXVEH];}      SIN007;
#define vstate                  SIN007.VSTATE

DLL_IMPORT struct {int LKSTOP[IMXVEH];}      SIN009;
#define lkstop                  SIN009.LKSTOP

DLL_IMPORT struct {int DISTUP[IMXVEH];}      SIN004;
#define distup                  SIN004.DISTUP

DLL_IMPORT struct {int SPDLN[IMXVEH];}       SIN008;
#define spdln                   SIN008.SPDLN

DLL_IMPORT struct {int VTYPLE[IMXVEH];}      SIN135;
#define vtyple                  SIN135.VTYPLE

DLL_IMPORT struct {int NVHTYP[IMXVEH];}      SIN133;
#define nvhtyp                  SIN133.NVHTYP

DLL_IMPORT struct {int YINIT;}               GLR091;
#define yinit                   GLR091.YINIT

DLL_IMPORT struct {int DWNOD [IMXLNK];}      SIN036;
#define dwnod                   SIN036.DWNOD

DLL_IMPORT struct {int LANEGD[IMXLNK];}      SIN039;
#define lanegd                  SIN039.LANEGD

DLL_IMPORT struct {int SDCODE[IMXLNK];}      SIN050;
#define sdcode                  SIN050.SDCODE

DLL_IMPORT struct {int UPNOD [IMXLNK];}      SIN062;
```

```
#define upnod                    SIN062.UPNOD

DLL_IMPORT struct {int LINK [IMXLNK];}      SIN171;
#define link                     SIN171.LINK

DLL_IMPORT struct {int NACT  [IMXNOD];}     SIN073;
#define nact                     SIN073.NACT

DLL_IMPORT struct {int SIGT  [IMXNOD];}     SIN074;
#define sigt                     SIN074.SIGT

DLL_IMPORT struct {int NMAP  [IMXNOD];}     SIN075;
#define nmap                     SIN075.NMAP

DLL_IMPORT struct {int SIGI [IMXNOD *5];}   SIN076;
#define sigi                                                SIN076.SIGI

DLL_IMPORT struct {int CLOCK;}              SIN104;
#define sclock                   SIN104.CLOCK

DLL_IMPORT struct {int TTLILK;}             SIN113;
#define ttlilk                   SIN113.TTLILK

DLL_IMPORT struct {int TTLND;}              SIN115;
#define ttlnd                    SIN115.TTLND

DLL_IMPORT struct {int TTLNK;}                                  SIN116;
#define ttlnk                                               SIN116.TTLNK

DLL_IMPORT struct {int XSIGT [IMXACT];}     SIN390;
#define xsigt                    SIN390.XSIGT

DLL_IMPORT struct {int AMOVSP[IMXACT * 8 * 5];} SIN305;
#define amovsp                   SIN305.AMOVSP

DLL_IMPORT struct {int DTLANE[IMXDET * 2];}  SIN311;
#define dtlane                   SIN311.DTLANE

DLL_IMPORT struct {int DTLNK[IMXDET];}      SIN700;
#define dtlnk                    SIN700.DTLNK

DLL_IMPORT struct {int DTLEN [IMXDET];}     SIN313;
#define dtlen                    SIN313.DTLEN

DLL_IMPORT struct {int DTMOD [IMXDET];}     SIN314;
#define dtmod                    SIN314.DTMOD

DLL_IMPORT struct {int DETID [IMXDET];}     SIN719;
#define sdetid                   SIN719.DETID

DLL_IMPORT struct {int DTNLNK[IMXDET];}     SIN308;
#define dtnlnk                   SIN308.DTNLNK

DLL_IMPORT struct {int DTPOS [IMXDET];}     SIN312;
#define dtpos                    SIN312.DTPOS

DLL_IMPORT struct {int WDTYPE[IMXDET];}     SIN368;
#define wdtype                   SIN368.WDTYPE

DLL_IMPORT struct {int WDET  [IMXDET];}     SIN340;
#define wdet                     SIN340.WDET

DLL_IMPORT struct {int DPPOUT[48];}         SIN345;
#define dppout                   SIN345.DPPOUT
```

```
DLL_IMPORT struct {int LOWRAM[751];}          SIN355;
#define lowram                    SIN355.LOWRAM

DLL_IMPORT struct {int DTFLNK[IMXLNK];}       SIN307;
#define dtflnk                    SIN307.DTFLNK

DLL_IMPORT struct {int PSDCOD[IMXLNK];}       SIN365;
#define psdcod                    SIN365.PSDCOD

DLL_IMPORT struct {long int AMBSPC[IMXLNK];}  SIN366;
#define ambspc                    SIN366.AMBSPC

DLL_IMPORT struct {int TOTDET;}               SIN109;
#define totdet                    SIN109.TOTDET

DLL_IMPORT struct {int DETON [IMXDET];}       SIN070;
#define deton                     SIN070.DETON

}

#endif
```

```
// Copyright 1997 <Kaman Sciences Corporation>.  All Rights Reserved
// Modified by Beth Taylor in May-June 1998
//          - Added altfile parameter to ReadCard35 and ReadCard36
//          - Added new variables for signal priority
// CNetwork
#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <afxtempl.h>       // MFC templates

class CDetector;
class CLink;
class CNode;
class CSignal;
class CBinarySequence;

class CNetwork:public CObject
{

public:  //implementation
          void CreateSignalStates();
          void ReadCard36(FILE* fpin, BOOL altFile);
          void ReadCard35(FILE* fpin, BOOL altFile);
          void CreateLanes(FILE* fpin);
          void GetDetectorCorsimId(CDetector* pDetector);
          void GetDetectors(FILE* fpin);
          CLink* FindLink(int up, int dn);
          void UpdateNodeSignalStates();
          int GetLinkCorsimId(int upnode, int dnnode);
          CNode* FindNode(int id);
          void GetNodes(FILE* fpin);
          void GetLinks(FILE* fpin);
          void ReadTrafFile();
          CNetwork();
          ~CNetwork();

public:  //data
          CString m_NetworkName;
          CString m_TrafInputFile;
          CString m_altSignalFile;
          CString m_detOutputFile;
          CTypedPtrList<CPtrList,CLink*> m_LinkList;
          CTypedPtrList<CPtrList,CNode*> m_NodeList;

};
```

```
// Copyright 1997 <Kaman Sciences Corporation>.  All Rights Reserved
// Modified by Beth Taylor in May-June 1998
//          - Added new variables for signal priority
// CNode
#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <afxtempl.h>

class CLink;
class CMovement;

class CNode:public CObject
{

public:  //implementation
          void SetSDCCode();
          CNode();
          ~CNode();
          void SetSignalState();

public:  //data
          int    m_id;
          int    m_xPos;
          int    m_yPos;
          CLink* m_Link1;
          CLink* m_Link2;
          CLink* m_Link3;
          CLink* m_Link4;
          CLink* m_Link5;
          CString m_typeOfControl;
          int m_duration[12];
          int m_altduration[12];
          CSignalState* m_altSignalBegin;
          int m_green_remaining;
};
```

```
// Modified by Beth Taylor in May-June 1998
//          - Added alt parameter to SetSDCODE declaration
//          - Added alt parameter to SetAMBSPC declaration
//          - Added new variables for signal priority
//CSignalState
#include <afxwin.h>       // MFC core and standard components
#include <afxext.h>       // MFC extensions
#include <afxtempl.h>

class CLink;

class CSignalState:public CObject
{

public: //implementation
          CSignalState();
          ~CSignalState();
          void SetSDCODE(BOOL alt);
          void SetAMBSPC(BOOL alt);
          int GetAMBSPC1(int nextActualCode);
          int GetAMBSPC3(int nextActualCode);
          int GetAMBSPC4(int nextActualCode);
          int GetAMBSPC5(int nextActualCode);
          int GetAMBSPC6(int nextActualCode);
          int GetAMBSPC7(int nextActualCode);
          int GetAMBSPC8(int nextActualCode);
          int GetAMBSPC9(int nextActualCode);

public: //data
          int signalCode;   //code from the TRAF file card 36
          int m_actualCode; //code used to actually set the SDCODE and
                    //AMBSPC
          int SDCODE;
          int AMBSPC;
          CLink* m_Link;

          int altsignalCode;   //code from the alt file card 36
          int m_altactualCode; //code used to actually set the SDCODE and
                    //AMBSPC
          int altSDCODE;
          int altAMBSPC;
};
```

```
// Copyright 1997 <Kaman Sciences Corporation>.  All Rights Reserved
// Modified by Beth Taylor in May-June 1998
//          - Added #include <stdio.h>
//          - Moved variables from netsim.h
#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <stdio.h>

class CNetwork;

CNetwork* pNetwork;

//Moved from netsim.h
int endOfInit;
int prevInit;
int prevTime;
FILE* fptr;
BOOL altSignal;
```

```
// Copyright 1997 <Kaman Sciences Corporation>.  All Rights Reserved
// CBinarySequence Implementation
//
#include "binarySequence.h"
#include "integer.h"

CBinarySequence::CBinarySequence()
{

}

CBinarySequence::~CBinarySequence()
{
        POSITION pos;
        CInteger* pI;

        pos=m_sequence.GetHeadPosition();
        while (pos!=NULL)
        {
                pI=m_sequence.GetNext(pos);
                delete pI;
        }
        m_sequence.RemoveAll();

}
```

```
// Copyright 1997 <Kaman Sciences Corporation>.  All Rights Reserved
//CDetector Implementation
//

#include "link.h"
#include "detector.h"
#include "lane.h"

CDetector::CDetector()
{
        m_activationTime=(float)0.0;
        m_deactivationTime=(float)0.0;
        m_state=FALSE;

}

CDetector::~CDetector()
{

}
```

```
// Copyright 1997 <Kaman Sciences Corporation>.  All Rights Reserved
//CInteger Implementation
//An integer class separate from the regular integers
//is necessary list provided by MFC are to be used.
#include "integer.h"

CInteger::CInteger()
{

}

CInteger::~CInteger()
{

}
```

```
//CInterface Implementation
//
//#include "netsim.h"
#include "interface.h"
#include "network.h"

CInterface::CInterface()
{
        m_Network=new CNetwork();
}

CInterface::~CInterface()
{
        if (m_Network!=NULL)
        {
                delete m_Network;
                m_Network=NULL;
        }
}
```

```
// Copyright 1997 <Kaman Sciences Corporation>.  All Rights Reserved
//CLane Implementation

#include "link.h"
#include "lane.h"
#include "node.h"
#include "netsim.h"

CLane::CLane()
{

}

CLane::~CLane()
{
        m_DetectorList.RemoveAll();
}
```

```
// Copyright 1997 <Kaman Sciences Corporation>.  All Rights Reserved
//CLink Implementation
//
// Modified by Beth Taylor in May-June 1998
//          - Added #include "initvars.h" and #include "network.h"
//          - Reordered some of the #include statements
//          - Added pSignalState parameter to ProcessDetectors
//          - Added code to ProcessDetectors, CreateSignalStates,
//            and SetSDCCode to do signal priority
#include "detector.h"
#include "link.h"
#include "lane.h"
#include "integer.h"
#include "binarySequence.h"
#include "signalState.h"
#include "node.h"
#include "netsim.h"
#include "network.h"
#include "initvars.h"
#include <math.h>

CLink::CLink()
{

        //default constructor
        m_id=0;
        m_NumOfDetectors=0;
        m_pos=NULL;
        m_opposingLink=NULL;

}

CLink::~CLink()
{
        //default destructor
        POSITION pos;
        CLane* pLane;
        CDetector* pDetector;
        CSignalState* pSignalState;

        //delete the lanes on this link
        pos=m_listOfLanes.GetHeadPosition();
        while (pos!=NULL)
        {
                pLane=m_listOfLanes.GetNext(pos);
                delete pLane;
        }
        m_listOfLanes.RemoveAll();

        //delete the detectors on this link
        pos=m_listOfDetectors.GetHeadPosition();
        while (pos!=NULL)
        {
                pDetector=m_listOfDetectors.GetNext(pos);
                delete pDetector;
        }
        m_listOfDetectors.RemoveAll();

        //delete the signal states for this link
        pos=m_signalStates.GetHeadPosition();
        while (pos!=NULL)
        {
                pSignalState=m_signalStates.GetNext(pos);
                delete pSignalState;
        }
```

```
        m_signalStates.RemoveAll();

}

void CLink::ProcessDetectors(CSignalState* pSignalState)
{
        //routine processes the information from the detectors
        //on this link
        POSITION pos;
        CDetector* pDetector;
        int  i;
        int  det;
        int  detinfo;
        int  type;
        int  num;

        POSITION posI;
        CInteger* pI;
        int newstate;
        int oldstate;
        CBinarySequence* sequence;

        int time;
        POSITION posS;
        CSignalState* nextSignalState;
        BOOL done;
        int greenTime;
        int redTime;
        CNode* autoNode;
        BOOL in;

        //Flag to indicate if initialization is over
        in=yinit;

        //compute the current time
        time=sclock+endOfInit;

        //loop through the detectors
        pos=m_listOfDetectors.GetHeadPosition();
        while (pos!=NULL)
        {
                pDetector=m_listOfDetectors.GetNext(pos);
                detinfo=dtmod[pDetector->m_CorsimId];
                //detinfo is bit packed
                //the first 3 bits contain the type of detector
                type=detinfo&7;
                //bits 11 - 23 contain the vehicle count since
                //start of simulation
                num=detinfo&8387584;
                num=num>>10;

                det=deton[pDetector->m_CorsimId];
                sequence=ConvertToBinary(det);
                //loop through the binary sequence starting at the end
                //because when the sequence was created it is in reverse
                //order
                posI=sequence->m_sequence.GetTailPosition();
                oldstate=(int)pDetector->m_state;
                if (num>pDetector->m_count)
                {
                        fprintf(fptr,"Detector %d count %d at time %4d, signalcode %d",
                                pDetector->m_id,num,time,pSignalState->signalCode);
                        //if current signal code is green for through traffic then find out
                        //how much green time remains
                        if ((pSignalState->signalCode==1) || (pSignalState->signalCode==7) ||
```

```
        (pSignalState->signalCode==9))
{
        //find the next signal state with a zero signal code
        //a zero signal code means amber
        posS=m_pos;
        done=FALSE;
        greenTime=0;
        while (!done)
        {
                if (posS==NULL)
                {
                        posS=m_signalStates.GetHeadPosition();
                        nextSignalState=m_signalStates.GetNext(posS);
                }
                else
                {
                        nextSignalState=m_signalStates.GetNext(posS);
                }
                greenTime++;
                done=nextSignalState->signalCode==0;
        }

        fprintf(fptr,", Green Time remaining = %3d seconds\n",greenTime);
        if (((m_dnnode->m_green_remaining==0) ||
          (greenTime<m_dnnode->m_green_remaining))
          && (m_dnnode->m_altSignalBegin==NULL) && altSignal && !in)
        {
                m_dnnode->m_altSignalBegin=pSignalState;
                fprintf(fptr,"AltTiming for node %3d began at time %4d\n",
                        m_dnnode->m_id, time);
                //Get auto node and set altSignalBegin for that one also
                autoNode=pNetwork->FindNode(m_dnnode->m_id-50);
                autoNode->m_altSignalBegin=pSignalState;
                fprintf(fptr,"AltTiming for node %3d began at time %4d\n",
                        autoNode->m_id, time);
        }
}
else
{
        if (pSignalState->signalCode!=0)
        {
                //find the next signal state with a green thru signal code
                posS=m_pos;
                done=FALSE;
                redTime=0;
                while (!done)
                {
                        if (posS==NULL)
                        {
                            posS=m_signalStates.GetHeadPosition();
                            nextSignalState=m_signalStates.GetNext(posS);
                        }
                        else
                        {
                            nextSignalState=m_signalStates.GetNext(posS);
                        }
                        redTime++;
                        done=((nextSignalState->signalCode==1) ||
                                (nextSignalState->signalCode==7) ||
                                (nextSignalState->signalCode==9));
                }
                fprintf(fptr,", Red Time remaining = %3d seconds",redTime);
        }
        fprintf(fptr,"\n");
}
```

```
                }
                for (i=0;i<10;i++)
                {
                        pI=sequence->m_sequence.GetPrev(posI);
                        newstate=pI->data;
                        if ((oldstate==0)&&(newstate==1))
                        {
                                //activation
                                pDetector->m_activationTime=(float)0.1;
                                pDetector->m_deactivationTime=(float)0.0;
                                pDetector->m_state=TRUE;
                        }
                        if ((oldstate==1)&&(newstate==1))
                        {
                                pDetector->m_activationTime=
                                        pDetector->m_activationTime+(float)0.1;
                                pDetector->m_deactivationTime=(float)0.0;
                                pDetector->m_state=TRUE;
                        }
                        if ((oldstate==1)&&(newstate==0))
                        {
                                //deactivation
                                pDetector->m_activationTime=(float)0.0;
                                pDetector->m_deactivationTime=(float)0.1;
                                pDetector->m_state=FALSE;
                        }
                        if ((oldstate==0)&&(newstate==0))
                        {
                                pDetector->m_activationTime=(float)0.0;
                                pDetector->m_deactivationTime=
                                        pDetector->m_deactivationTime+(float)0.1;
                                pDetector->m_state=FALSE;
                        }
                        oldstate=newstate;
                }

                //clean up
                delete sequence;
                pDetector->m_count=num;
        }

}

CBinarySequence* CLink::ConvertToBinary(int j)
{
        //function converts an integer into a binary sequence
        CBinarySequence* pSequence;
        CInteger* pI;
        int remainder;
        int i;
        int k;
        int l;

        pSequence=new CBinarySequence();
        //assume the binary number is less than 2^10
        k=9;
        remainder=j;
        for (i=k;i>-1;i--)
        {
                if (remainder>=(int)pow(2,i))
                {
                        pI=new CInteger;
                        pI->data=1;
                        pSequence->m_sequence.AddTail(pI);
                        l=1;
```

```
                }
                else
                {
                        pI=new CInteger;
                        pI->data=0;
                        pSequence->m_sequence.AddTail(pI);
                        I=0;
                }
                remainder=remainder-I*(int)pow(2,i);
        }

        //the sequence will be reversed
        return pSequence;
}

CLane* CLink::FindLane(int corsimId)
{
        //finds the lane on the link based on the lane's
        //ID from the TRAF file.
        POSITION pos;
        BOOL found;
        CLane* pLane;

        pLane=NULL;
        found=FALSE;
        pos=m_listOfLanes.GetHeadPosition();
        while ((pos!=NULL)&&(!found))
        {
                pLane=m_listOfLanes.GetNext(pos);
                found=pLane->m_id==corsimId;
        }

        return pLane;

}

float CLink::ComputeTravelTime()
{
        //computes the travel time to traverse a link
        float travelTime;

        travelTime=(float)m_length/(float)m_freeFlowSpeed;

        return travelTime;
}


void CLink::CreateSignalStates()
{
        //creates the list of signal states for
        //each of the links not under CORSIM control
        int cycleLength;
        int offset;
        int duration;
        int alt;
        int maxalt;
        int i;
        int j;
        int code;
        int prevcode;
        POSITION pos;
        CSignalState* pSignalState;

        //sum the durations to get the cycle length
```

```
//for the fixed time control
cycleLength=0;
for (i=0;i<12;i++)
{
        cycleLength=cycleLength+m_dnnode->m_duration[i];
}

//create the signal states for every second
for (i=0;i<cycleLength;i++)
{
        pSignalState=new CSignalState();
        pSignalState->m_Link=this;
        m_signalStates.AddTail(pSignalState);
}
if (altSignal)
        maxalt=2;
else
        maxalt=1;
//do this twice - once for regular signal timing and once
//for alternate signal timing
for (alt=0;alt<maxalt;alt++)
{
        offset=m_offset;
        //use the offset to determine where to begin creating the
        //the signal states
        //move to that position in the signal state list
        pos=m_signalStates.GetHeadPosition();
        for (i=0;i<offset;i++)
        {
                pSignalState=m_signalStates.GetNext(pos);
        }
        //for each duration set signal code
        //and actual code
        //if the signal code is 0 for amber then
        //the actual code will be the previous
        //signal code that was not amber
        for (i=0;i<12;i++)
        {
                if (alt==0)
                {
                        duration=m_dnnode->m_duration[i];
                        code=m_code[i];
                }
                else
                {
                        duration=m_dnnode->m_altduration[i];
                        code=m_altcode[i];
                }
                for (j=0;j<duration;j++)
                {
                        if (alt==0)
                                pSignalState->signalCode=code;
                        else
                                pSignalState->altsignalCode=code;

                        if (code==0)
                        {
                                if (alt==0)
                                        pSignalState->m_actualCode=prevcode;
                                else
                                        pSignalState->m_altactualCode=prevcode;
                        }
                        else
                        {
                                if (alt==0)
```

```
                                        pSignalState->m_actualCode=code;
                                else
                                        pSignalState->m_altactualCode=code;
                                prevcode=code;
                        }
                        //advance to the next signal state in the list
                        if (pos==NULL)
                                pos=m_signalStates.GetHeadPosition();
                        pSignalState=m_signalStates.GetNext(pos);
                }
        }
}

//for each signal state get the appropriate values
//for the SDCODE and AMBSPC
pos=m_signalStates.GetHeadPosition();
while (pos!=NULL)
{
        m_pos=pos;
        pSignalState=m_signalStates.GetNext(pos);
        pSignalState->SetSDCODE(FALSE);
        pSignalState->SetAMBSPC(FALSE);
        if (altSignal)
        {
                pSignalState->SetSDCODE(TRUE);
                pSignalState->SetAMBSPC(TRUE);
        }
}

}

void CLink::SetSDCCode()
{
        //set the SDCODE and AMBSPC for this link
        CSignalState* pSignalState;
        int time;
        CNode* autoNode;

        time=sclock+endOfInit;

        if ((time==0)||(m_pos==NULL))
        {
                //at the end of the signal state list
                //so go back to the beginning of the list
                m_pos=m_signalStates.GetHeadPosition();
        }

        pSignalState=m_signalStates.GetNext(m_pos);

        if (m_dnnode->m_altSignalBegin==pSignalState)
        {
                m_dnnode->m_altSignalBegin=NULL;
                fprintf(fptr,"AltTiming for node %3d ended at time %4d\n", m_dnnode->m_id, time);
                //Get auto node and set altSignalBegin for that one also
                autoNode=pNetwork->FindNode(m_dnnode->m_id-50);
                autoNode->m_altSignalBegin=NULL;
                fprintf(fptr,"AltTiming for node %3d ended at time %4d\n", autoNode->m_id, time);
        }

        ProcessDetectors(pSignalState);

        if (m_dnnode->m_altSignalBegin==NULL)
        {
                sdcode[m_CorsimId]=pSignalState->SDCODE;
                ambspc[m_CorsimId]=pSignalState->AMBSPC;
```

```
        }
        else
        {
                sdcode[m_CorsimId]=pSignalState->altSDCODE;
                ambspc[m_CorsimId]=pSignalState->altAMBSPC;
        }

}
```

```
// Copyright 1997 <Kaman Sciences Corporation>.  All Rights Reserved
// Modified by Beth Taylor in May-June 1998
//          - Added #include "initvars.h"
//          - Modified code to work for input files without Card Type 43
//          - Added altfile parameter to ReadCard35 and ReadCard36
//          - Added code for signal priority
// CNetwork Implementation

#include "netsim.h"
#include "network.h"
#include "link.h"
#include "node.h"
#include "detector.h"
#include "lane.h"
#include "integer.h"
#include "binarySequence.h"
#include "initvars.h"
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

CNetwork::CNetwork()
{
        //default constructor
}

CNetwork::~CNetwork()
{
        POSITION pos;
        CNode* pNode;
        CLink* pLink;

        //delete the link list
        pos=m_LinkList.GetHeadPosition();
        while (pos!=NULL)
        {
                pLink=m_LinkList.GetNext(pos);
                delete pLink;
        }
        m_LinkList.RemoveAll();

        //delete the node list
        pos=m_NodeList.GetHeadPosition();
        while (pos!=NULL)
        {
                pNode=m_NodeList.GetNext(pos);
                delete pNode;
        }
        m_NodeList.RemoveAll();

}


void CNetwork::ReadTrafFile()
{

        FILE* fpin;
        POSITION pos;
        CLink* pLink;
        int up;
        int down;

        //create the node list
        fpin=fopen(m_TrafInputFile,"r");
        GetNodes(fpin);
```

```
        fclose(fpin);

        //create the link list
        fpin=fopen(m_TrafInputFile,"r");
        GetLinks(fpin);
        fclose(fpin);

        //find the opposing link for each link
        pos=m_LinkList.GetHeadPosition();
        while (pos!=NULL)
        {
                pLink=m_LinkList.GetNext(pos);
                up=pLink->m_opposingID;
                down=pLink->m_dnnode->m_id;
                pLink->m_opposingLink=FindLink(up,down);
        }

        //create the lanes on each link
        fpin=fopen(m_TrafInputFile,"r");
        CreateLanes(fpin);
        fclose(fpin);

        //create the signal timings for the nodes to
        //be controlled by the algorithm
        fpin=fopen(m_TrafInputFile,"r");
        ReadCard35(fpin,FALSE);
        fclose(fpin);

        fpin=fopen(m_TrafInputFile,"r");
        ReadCard36(fpin,FALSE);
        fclose(fpin);

        altSignal=FALSE;
        fpin=fopen(m_altSignalFile,"r");

        if (fpin!=NULL)
        {
                fprintf(fptr,"*** Alternate Signal Timing File Input ***\n");
                altSignal=TRUE;

                ReadCard35(fpin,TRUE);
                fclose(fpin);

                fpin=fopen(m_altSignalFile,"r");
                ReadCard36(fpin,TRUE);
                fclose(fpin);
                fprintf(fptr,"*** End of Alternate Signal Timing File Input ***\n");
        }

        CreateSignalStates();

        //create the detectors that exist in the
        //TRAF file
        fpin=fopen(m_TrafInputFile,"r");
        GetDetectors(fpin);
        fclose(fpin);

}

void CNetwork::GetLinks(FILE* fpin)
{
        //get the information about the links in the
        //TRAF file and creates the link list
        int   endnum;
        char* end;
```

```
int   i;
char  line[82];
char  temp1[4];
BOOL  found;
BOOL  done;
CLink* pLink;

end=fgets(line,sizeof(line),fpin);
found=FALSE;

//find the card type 11
while((end!=NULL)&&(!found))
{
        endnum=80;
        found=(line[endnum-2]=='1')&&(line[endnum-1]=='1');
        if (!found)
                end=fgets(line,sizeof(line),fpin);
}

//create the links
done=FALSE;
while ((end!=NULL)&&(!done))
{
        char up[5];
        char dn[5];
        char th[5];
        char le[5];
        char ri[5];
        char op[5];
        char length[5];
        char speed[5];
        char lengthOfLeftBay[5];
        char lengthOfRightBay[5];
        char numOfFullLanes[5];
        char numOfLeftBays[5];
        char numOfRightBays[5];
        char ch[7][5];

        //initialize all the character strings
        for (i=0;i<5;i++)
        {
                up[i]='\0';
                dn[i]='\0';
                th[i]='\0';
                le[i]='\0';
                ri[i]='\0';
                op[i]='\0';
                length[i]='\0';
          speed[i]='\0';
          lengthOfLeftBay[i]='\0';
          lengthOfRightBay[i]='\0';
          numOfFullLanes[i]='\0';
          numOfLeftBays[i]='\0';
          numOfRightBays[i]='\0';
          ch[0][i]='\0';
          ch[1][i]='\0';
          ch[2][i]='\0';
          ch[3][i]='\0';
          ch[4][i]='\0';
          ch[5][i]='\0';
          ch[6][i]='\0';
        }

        //parse the line
        //the line contains the upstream node number
```

```
//              the downstream node number
//              the link length
//              turn bay length
// etc.. see documentation describing CARD 11
for (i=0;i<4;i++)
{
        up[i]=line[i];
        dn[i]=line[i+4];
        length[i]=line[i+8];
        lengthOfLeftBay[i]=line[i+12];
        lengthOfRightBay[i]=line[i+16];
        th[i]=line[i+40];
        le[i]=line[i+36];
        ri[i]=line[i+44];
        op[i]=line[i+52];
        speed[i]=line[i+64];
}
//read the channelization codes for each lane
for (i=0;i<7;i++)
{
        ch[i][1]=line[i+29];
}

numOfFullLanes[0]=line[21];
numOfLeftBays[0]=line[23];
numOfRightBays[0]=line[25];

if ((atoi(dn)<8000)&&(atoi(up)<8000))
{
        //link is not a source link so create it
        pLink=new CLink();
        pLink->m_numOfFullLanes=atoi(numOfFullLanes);
        pLink->m_numOfLeftTurnBays=atoi(numOfLeftBays);
        pLink->m_lengthOfLeftBay=atoi(lengthOfLeftBay);
        pLink->m_numOfRightTurnBays=atoi(numOfRightBays);
        pLink->m_lengthOfRightBay=atoi(lengthOfRightBay);
        pLink->m_length=atoi(length);
        pLink->m_freeFlowSpeed=atoi(speed);
        if (pLink->m_freeFlowSpeed == 0)
                pLink->m_freeFlowSpeed = 44;
        pLink->m_upnode=FindNode(atoi(up));
        pLink->m_dnnode=FindNode(atoi(dn));
        pLink->m_thrunode=FindNode(atoi(th));
        pLink->m_leftnode=FindNode(atoi(le));
        pLink->m_rightnode=FindNode(atoi(ri));
        pLink->m_CorsimId=GetLinkCorsimId(pLink->m_upnode->m_id,
                                          pLink->m_dnnode->m_id);
        pLink->m_opposingID=atoi(op);
        if ((atoi(up)<8000)&&(atoi(dn)<8000))
                pLink->m_travelTime=pLink->ComputeTravelTime();
        for (i=0;i<7;i++)
        {
                pLink->m_channelCode[i]=CString(ch[i][1]);
        }
        m_LinkList.AddTail(pLink);
}
//get the next line
end=fgets(line,sizeof(line),fpin);

//check for end of card type 11
endnum=80;
temp1[0]=line[endnum-3];
temp1[1]=line[endnum-2];
temp1[2]=line[endnum-1];
temp1[3]='\0';
```

```
                done=atoi(temp1)>11;
        }
        return;

}

void CNetwork::GetNodes(FILE* fpin)
{
        //get the node information from the TRAF file
        //and create the node list
        int  endnum;
        char*  end;
        int  i;
        char line[82];
        char temp1[4];
        BOOL found;
        BOOL done;
        CNode* pNode;

        //search for card type 36
        end=fgets(line,sizeof(line),fpin);
        found=FALSE;
        done=FALSE;
        while((end!=NULL)&&(!found)&&(!done))
        {
                endnum=80;
                found=(line[endnum-2]=='3')&&(line[endnum-1]=='6');
                if (!found)
                {
                        temp1[0]=line[endnum-2];
                        temp1[1]=line[endnum-1];
                        temp1[2]='\0';
                        temp1[3]='\0';
                        int temp2=atoi(temp1);
                        done=temp2>36;
                        if (!done)
                                end=fgets(line,sizeof(line),fpin);
                }
        }

        //read in all of the 36 cards
        //done=FALSE;
        while((end!=NULL)&&(!done))
        {
                char id[5];
                char type[2];

                for (i=0;i<5;i++)
                {
                        id[i]='\0';
                }

                //parse the 36 card
                type[0]='\0';
                type[1]='\0';
                id[0]=line[0];
                id[1]=line[1];
                id[2]=line[2];
                id[3]=line[3];
                type[0]=line[endnum-4];

                if (atoi(id)<9000)
                {
                        pNode=new CNode();
                        pNode->m_id=atoi(id);
```

```
                    if (pNode->m_id>8000)
                    {
                            //source node
                            pNode->m_typeOfControl=CString("source");
                    }
                    else
                    {
                            if ((type[0]=='2')||(type[0]=='3'))
                            {
                                    //node is controlled by algorithm
                                    //external to CORSIM
                                    pNode->m_typeOfControl=CString("external");
                            }
                            else
                            {
                                    //node is controlled by CORSIM
                                    pNode->m_typeOfControl=CString("corsim");
                            }
                    }
                    m_NodeList.AddTail(pNode);
            }

            //read next line
            //if card type > 36 then all the 36 cards have been read
            end=fgets(line,sizeof(line),fpin);
            if (end!=NULL)
            {
                    endnum=80;
                    temp1[0]=line[endnum-2];
                    temp1[1]=line[endnum-1];
                    temp1[2]='\0';
                    temp1[3]='\0';
                    int temp2=atoi(temp1);
                    done=temp2>36;
            }

    }

    //search for card type 43
    found=FALSE;
    done=FALSE;
    while((end!=NULL)&&(!found)&&(!done))
    {
            endnum=80;
            found=(line[endnum-2]=='4')&&(line[endnum-1]=='3');
            if (!found)
            {
                    temp1[0]=line[endnum-2];
                    temp1[1]=line[endnum-1];
                    temp1[2]='\0';
                    temp1[3]='\0';
                    int temp2=atoi(temp1);
                    done=temp2>43;
                    if (!done)
                            end=fgets(line,sizeof(line),fpin);
            }
    }

    //read in the card type 43's
    //done=FALSE;
    while((end!=NULL)&&(!done))
    {
            char id[5];
            char type[2];
            for (i=0;i<5;i++)
```

```
{
        id[i]='\0';
}

//parse the 43 card
type[0]='\0';
type[1]='\0';
id[0]=line[0];
id[1]=line[1];
id[2]=line[2];
id[3]=line[3];
type[0]=line[endnum-4];

if (atoi(id)<9000)
{
        pNode=new CNode();
        pNode->m_id=atoi(id);
        if (pNode->m_id>8000)
        {
                //source node
                pNode->m_typeOfControl=CString("source");
        }
        else
        {
                if ((type[0]=='2')||(type[0]=='3'))
                {
                        //node control is external to CORSIM
                        pNode->m_typeOfControl=CString("external");
                }
                else
                {
                        //node is controlled by CORSIM
                        pNode->m_typeOfControl=CString("corsim");
                }
        }
        m_NodeList.AddTail(pNode);
}

end=fgets(line,sizeof(line),fpin);
if (end!=NULL)
{
        endnum=80;
        temp1[0]=line[endnum-2];
        temp1[1]=line[endnum-1];
        temp1[2]='\0';
        temp1[3]='\0';
        int temp2=atoi(temp1);
        done=temp2>43;
}

}

//search for card type 195
found=FALSE;
done=FALSE;
while((end!=NULL)&&(!found)&&(!done))
{
        endnum=80;
        found=(line[endnum-3]=='1')&&(line[endnum-2]=='9')&&
                (line[endnum-1]=='5');
        if (!found)
        {
                temp1[0]=line[endnum-3];
                temp1[1]=line[endnum-2];
                temp1[2]=line[endnum-1];
```

```
                temp1[3]='\0';
                int temp2=atoi(temp1);
                done=temp2>195;
                if (!done)
                        end=fgets(line,sizeof(line),fpin);
        }
}

//get the x,y coordinates of the nodes
//done=FALSE;
while((end!=NULL)&&(!done))
{
        char xpos[6];
        char ypos[7];
        char nodeid[7];

        //initialize the character strings
        for (i=0;i<6;i++)
        {
                nodeid[i]='\0';
        }

        for (i=0;i<7;i++)
        {
                xpos[i]='\0';
                ypos[i]='\0';
        }

        //parse line
        nodeid[0]=line[0];
        nodeid[1]=line[1];
        nodeid[2]=line[2];
        nodeid[3]=line[3];

        xpos[0]=line[6];
        xpos[1]=line[7];
        xpos[2]=line[8];
        xpos[3]=line[9];
        xpos[4]=line[10];
        xpos[5]=line[11];
        ypos[0]=line[14];
        ypos[1]=line[14];
        ypos[2]=line[14];
        ypos[3]=line[14];
        ypos[4]=line[14];
        ypos[5]=line[19];

        //search the node list for the corresponding node
        POSITION pos;
        pos=m_NodeList.GetHeadPosition();
        while (pos!=NULL)
        {
                pNode=m_NodeList.GetNext(pos);
                if (pNode->m_id==atoi(nodeid))
                {
                        pNode->m_xPos=atoi(xpos);
                        pNode->m_yPos=atoi(ypos);
                }
        }

        //read the next line and see if
        //all the 195 cards have been read
        end=fgets(line,sizeof(line),fpin);
        if (end!=NULL)
        {
```

```
                    endnum=80;

                    temp1[0]=line[endnum-3];
                    temp1[1]=line[endnum-2];
                    temp1[2]=line[endnum-1];
                    temp1[3]='\0';
                    int temp2=atoi(temp1);
                    done=temp2>195;
                }
            }
            return;
}

CNode* CNetwork::FindNode(int id)
{
            //find the node (id) in the node list
            POSITION pos;
            CNode* pNode;
            BOOL found;

            pNode=NULL;
            found=FALSE;
            pos=m_NodeList.GetHeadPosition();
            while ((pos!=NULL)&&(!found))
            {
                        pNode=m_NodeList.GetNext(pos);
                        found=pNode->m_id==id;
            }

            if (found)
            {
                        //node found
                        return pNode;
            }
            else
            {
                        //node not found
                        return NULL;
            }
}

int CNetwork::GetLinkCorsimId(int upnode, int dnnode)
{
            //find the CORSIM link ID for the link (upnode,dnnode)
            int id;
            int i;
            int dnode;
            int unode;

            id=0;

            //search through all the links in CORSIM
            for (i=0;i<ttlnk;i++)
            {
                        //for the ith link get the downstream node and
                        //upstream node as represented in CORSIM
                        dnode=dwnod[i];
                        unode=upnod[i];
                        //for nonsource nodes (<7000), use the nmap array to map
                        //the node number in CORSIM back to the user defined
                        //node number in the TRAF file
                        //use an offset of -1 (i.e. dnode-1), because C arrays
                        //start at 0 FORTRAN arrays start at 1
                        if (dnode<7000) dnode=nrnap[dnode-1];
                        if (unode<7000) unode=nmap[unode-1];
```

```
                if ((dnode==dnnode)&&(unode==upnode))
                {
                        id=i;
                }
        }

        return id;
}

void CNetwork::UpdateNodeSignalStates()
{
        //update the signal states for all the
        //nodes not controlled by CORSIM
        POSITION pos;
        CNode* pNode;

        int time;
        time=sclock+endOfInit;

        pos=m_NodeList.GetHeadPosition();
        while (pos!=NULL)
        {
                pNode=m_NodeList.GetNext(pos);
                if (pNode->m_typeOfControl==CString("external"))
                {
                        //node is not controlled by CORSIM
                        pNode->SetSignalState();
                }
        }

        return;
}

CLink* CNetwork::FindLink(int up, int dn)
{
        //finds the link between nodes (up,dn)
        POSITION posLink;
        CLink* pLink;
        BOOL found;

        posLink=m_LinkList.GetHeadPosition();
        found=FALSE;
        while ((posLink!=NULL)&&(!found))
        {
                pLink=m_LinkList.GetNext(posLink);
                found=(pLink->m_upnode->m_id==up)&&
                        (pLink->m_dnnode->m_id==dn);
        }
        if (!found)
                pLink=NULL;

        return pLink;

}

void CNetwork::GetDetectors(FILE* fpin)
{
        //read the detectors information from the
        //TRAF input file and create the detector
        //list
        int  endnum;
        char*  end;
        int  i;
        char line[82];
        char temp1[4];
```

```
char up[5];
char dn[5];
char lane[2];
char distance[6];
char type[2];
char id[5];
char length[5];
BOOL found;
BOOL done;

CLink* pLink;
CLane* pLane;
POSITION posLink;
POSITION posLane;
CDetector* pDetector;

//search for card type 42
found=FALSE;
end=fgets(line,sizeof(line),fpin);
while((end!=NULL)&&(!found))
{
        endnum=80;
        found=(line[endnum-2]=='4')&&(line[endnum-1]=='2');
        if (!found)
                end=fgets(line,sizeof(line),fpin);
}

done=FALSE;
while((end!=NULL)&&(!done))
{
        //initialize the character strings
        for (i=0;i<5;i++)
        {
                up[i]='\0';
                dn[i]='\0';
                id[i]='\0';
                length[i]='\0';
        }
        for (i=0;i<6;i++)
        {
                distance[i]='\0';
        }
        for (i=0;i<2;i++)
        {
                lane[i]='\0';
                type[i]='\0';
        }

        //parse the line
        for (i=0;i<4;i++)
        {
                up[i]=line[i];
                dn[i]=line[i+4];
                id[i]=line[i+22];
                length[i]=line[i+28];
        }
        lane[0]=line[11];
        type[0]=line[34];
        for (i=0;i<5;i++)
        {
                distance[i]=line[i+15];
        }

        //create the detector
        pDetector=new CDetector();
```

```
                pDetector->m_count=0;
                pDetector->m_id=atoi(id);
                pDetector->m_length=atoi(length);
                pDetector->m_distanceFromDownstreamNode=
                        (int)((float)atoi(distance)/(float)10.0);

                //search for the link the detector is on
                posLink=m_LinkList.GetHeadPosition();
                found=FALSE;
                while ((posLink!=NULL)&&(!found))
                {
                        pLink=m_LinkList.GetNext(posLink);
                        found=(pLink->m_upnode->m_id==atoi(up))&&
                                (pLink->m_dnnode->m_id==atoi(dn));
                }
                pDetector->m_Link=pLink;

                //search for lane the lane the detector is on
                posLane=pLink->m_listOfLanes.GetHeadPosition();
                found=FALSE;
                while ((posLane!=NULL)&&(!found))
                {
                        pLane=pLink->m_listOfLanes.GetNext(posLane);
                        found=pLane->m_id==atoi(lane);
                }

                pDetector->m_Lane=pLane;
                GetDetectorCorsimId(pDetector);
                pLink->m_listOfDetectors.AddTail(pDetector);
                pLane->m_DetectorList.AddTail(pDetector);
                pLink->m_NumOfDetectors++;

                //read the next line and see if the we are
                //at the end of the 42 cards
                end=fgets(line,sizeof(line),fpin);
                if (end!=NULL)
                {
                        endnum=80;
                        temp1[0]=line[endnum-2];
                        temp1[1]=line[endnum-1];
                        temp1[2]='\0';
                        temp1[3]='\0';
                        int temp2=atoi(temp1);
                        done=temp2>42;
                }

        }

}

void CNetwork::GetDetectorCorsimId(CDetector* pDetector)
{
        //finds the CORSIM detector ID for pDetector

        int distance;
        CLink* pLink;
        CLane* pLane;
        BOOL found;
        int i;
        int id;

        distance=pDetector->m_distanceFromDownstreamNode;
        pLink=pDetector->m_Link;
        pLane=pDetector->m_Lane;
```

```
		//get the id of the first detector on the same
		//link as pDetector
		id=dtflnk[pLink->m_CorsimId];

		//compare the position and lane for the first detector
		//with that of pDetector
		found=(pLane->m_id==dtlane[2*id-2])&&
			(distance==dtpos[id-1]/10);
		i=0;
		while ((!found)&&(i<pLink->m_NumOfDetectors))
		{
			//not found so get the next detector on this link
			//and compare again
			id=dtnlnk[id-1];
			found=(pLane->m_id==dtlane[2*id-2])&&
			    (distance==dtpos[id-1]/10);
			i++;
		}

		if (found)
		{
			//detector found
			pDetector->m_CorsimId=id-1;
		}

		return;
}

void CNetwork::CreateLanes(FILE* fpin)
{
		//create the lanes
		POSITION pos;
		CLink* pLink;
		CLane* pLane;
		int i;
		int lastid;
		CString channelCode;

		int  endnum;
		char*  end;
		char line[82];
		char temp1[4];
		char up[5];
		char dn[5];
		char left[5];
		char thru[5];
		char right[5];
		BOOL found;
		BOOL done;

		//search for card type 21
		found=FALSE;
		end=fgets(line,sizeof(line),fpin);
		while((end!=NULL)&&(!found))
		{
			endnum=80;
			found=(line[endnum-2]=='2')&&(line[endnum-1]=='1');
			if (!found)
				end=fgets(line,sizeof(line),fpin);
		}

		done=FALSE;
		while((end!=NULL)&&(!done))
		{
```

```
		for (i=0;i<5;i++)
		{
			up[i]='\0';
			dn[i]='\0';
			left[i]='\0';
			thru[i]='\0';
			right[i]='\0';
		}
		for (i=0;i<4;i++)
		{
			up[i]=line[i];
			dn[i]=line[i+4];
			left[i]=line[i+8];
			thru[i]=line[i+12];
			right[i]=line[i+16];
		}

		//search for the link between nodes up and down
		found=FALSE;
		pos=m_LinkList.GetHeadPosition();
		while ((pos!=NULL)&&(!found))
		{
			pLink=m_LinkList.GetNext(pos);
			found=(pLink->m_upnode->m_id==atoi(up))&&
				(pLink->m_dnnode->m_id==atoi(dn));
		}

		if (found)
		{
			//assign the turning percentages for the link
			pLink->m_leftMovementPercent=atoi(left);
			pLink->m_thruMovementPercent=atoi(thru);
			pLink->m_rightMovementPercent=atoi(right);
		}

		//check to see if this is the end of the 21 cards
		end=fgets(line,sizeof(line),fpin);
		if (end!=NULL)
		{
			endnum=80;
			temp1[0]=line[endnum-2];
			temp1[1]=line[endnum-1];
			temp1[2]='\0';
			temp1[3]='\0';
			int temp2=atoi(temp1);
			done=temp2>21;
		}
}

//for each link create the lanes for the link
pos=m_LinkList.GetHeadPosition();
while (pos!=NULL)
{
		pLink=m_LinkList.GetNext(pos);

		for (i=0;i<pLink->m_numOfFullLanes;i++)
		{
			pLane=new CLane();
			pLane->m_id=i+1;
			channelCode=pLink->m_channelCode[i];
			if (channelCode==CString("T"))
			{
				//lane is thru only
				pLane->m_leftMovementPercent=0;
				pLane->m_rightMovementPercent=0;
```

```
                pLane->m_thruMovementPercent=100;
        }
        if (channelCode==CString("1"))
        {
                //lane is left turn only
                pLane->m_leftMovementPercent=100;
                pLane->m_rightMovementPercent=0;
                pLane->m_thruMovementPercent=0;
        }
        if (channelCode==CString("4"))
        {
                //lane is right turn only
                pLane->m_leftMovementPercent=0;
                pLane->m_rightMovementPercent=100;
                pLane->m_thruMovementPercent=0;
        }
        if (channelCode==CString("7"))
        {
                //lane is right and thru
                float right;
                float thru;
                right=(float)pLink->m_rightMovementPercent;
                thru=(float)pLink->m_thruMovementPercent;
                pLane->m_leftMovementPercent=0;
                pLane->m_rightMovementPercent=(int)(100*(right/(right+thru)));
                pLane->m_thruMovementPercent=(int)(100*(thru/(thru+right)));
        }
        if (channelCode==CString("8"))
        {
                //lane is left and thru
                float thru;
                float left;
                thru=(float)pLink->m_thruMovementPercent;
                left=(float)pLink->m_leftMovementPercent;
                pLane->m_leftMovementPercent=(int)(100*(left/(left+thru)));
                pLane->m_thruMovementPercent=(int)(100*(thru/(left+thru)));
                pLane->m_rightMovementPercent=0;
        }
        if (channelCode==CString("9"))
        {
                //lane is left, thru, and right
                float left;
                float thru;
                float right;
                left=(float)pLink->m_leftMovementPercent;
                thru=(float)pLink->m_thruMovementPercent;
                right=(float)pLink->m_rightMovementPercent;
                pLane->m_leftMovementPercent=(int)(100*(left/(left+right+thru)));
                pLane->m_rightMovementPercent=(int)(100*(right/(left+right+thru)));
                pLane->m_thruMovementPercent=(int)(100*(thru/(left+right+thru)));
        }
        pLane->m_Link=pLink;
        pLane->m_type=CString("Full");
        pLink->m_listOfLanes.AddTail(pLane);
}

//compute lane number for the first right turn bay
//see documentation for CARD type 11
lastid=7-pLink->m_numOfRightTurnBays-pLink->m_numOfLeftTurnBays+1;
for (i=0;i<pLink->m_numOfRightTurnBays;i++)
{
        pLane=new CLane();
        pLane->m_id=lastid+i;
        pLane->m_leftMovementPercent=0;
        pLane->m_Link=pLink;
```

```
                        pLane->m_rightMovementPercent=100;
                        pLane->m_thruMovementPercent=0;
                        pLane->m_type=CString("Bay");
                        pLink->m_listOfLanes.AddTail(pLane);
                }

                //compute the lane number for the first left turn bay
                //see documentation for CARD type 11
                lastid=7-pLink->m_numOfLeftTurnBays+1;
                for (i=0;i<pLink->m_numOfLeftTurnBays;i++)
                {
                        pLane=new CLane();
                        pLane->m_id=lastid+i;
                        pLane->m_leftMovementPercent=100;
                        pLane->m_Link=pLink;
                        pLane->m_rightMovementPercent=0;
                        pLane->m_thruMovementPercent=0;
                        pLane->m_type=CString("Bay");
                        pLink->m_listOfLanes.AddTail(pLane);
                }
        }
}

void CNetwork::ReadCard35(FILE * fpin, BOOL altFile)
{
        //read card type 35 to get the duration for each signal interval
        //and the approach links for the nodes
        int   endnum;
        char* end;
        int   i;
        char  line[82];
        char  temp1[4];
        BOOL  found;
        BOOL  done;
        CNode* pNode;
        int up;
        int down;

        end=fgets(line,sizeof(line),fpin);
        found=FALSE;

        //find the card type 35
        while((end!=NULL)&&(!found))
        {
                endnum=80;
                found=(line[endnum-2]=='3')&&(line[endnum-1]=='5');
                if (!found)
                        end=fgets(line,sizeof(line),fpin);
        }

        //process all the 35 cards
        done=FALSE;
        while ((end>0)&&(!done))
        {
                char node[5];
                char offset[5];
                char upnode1[5];
                char upnode2[5];
                char upnode3[5];
                char upnode4[5];
                char upnode5[5];
                char dur1[4];
                char dur2[4];
                char dur3[4];
```

```
char dur4[4];
char dur5[4];
char dur6[4];
char dur7[4];
char dur8[4];
char dur9[4];
char dur10[4];
char dur11[4];
char dur12[4];

//inialize the character strings
for (i=0;i<5;i++)
{
        node[i]='\0';
        offset[i]='\0';
        upnode1[i]='\0';
        upnode2[i]='\0';
        upnode3[i]='\0';
        upnode4[i]='\0';
        upnode5[i]='\0';
}
for (i=0;i<4;i++)
{
        dur1[i]='\0';
        dur2[i]='\0';
        dur3[i]='\0';
        dur4[i]='\0';
        dur5[i]='\0';
        dur6[i]='\0';
        dur7[i]='\0';
        dur8[i]='\0';
        dur9[i]='\0';
        dur10[i]='\0';
        dur11[i]='\0';
        dur12[i]='\0';
}

//parse the line
for (i=0;i<4;i++)
{
        //get the node number, offset, and
        //upstream nodes for the 5 approaches
        node[i]=line[i];
        offset[i]=line[i+4];
        upnode1[i]=line[i+8];
        upnode2[i]=line[i+12];
        upnode3[i]=line[i+16];
        upnode4[i]=line[i+20];
        upnode5[i]=line[i+24];
}
for (i=0;i<3;i++)
{
        //get the durations the 12 signal intervals
        dur1[i]=line[i+29];
        dur2[i]=line[i+33];
        dur3[i]=line[i+37];
        dur4[i]=line[i+41];
        dur5[i]=line[i+46];
        dur6[i]=line[i+49];
        dur7[i]=line[i+53];
        dur8[i]=line[i+57];
        dur9[i]=line[i+61];
        dur10[i]=line[i+65];
        dur11[i]=line[i+69];
        dur12[i]=line[i+73];
```

```
}

up=atoi(upnode1);
down=atoi(node);

pNode=FindNode(down);

// Do not do this part for alternate timing file
if (!altFile)
{
        //assign the 5 approach links for this node
        pNode->m_Link1=FindLink(up,down);
        if (pNode->m_Link1)
                pNode->m_Link1->m_offset=atoi(offset);

        up=atoi(upnode2);
        pNode->m_Link2=FindLink(up,down);

        if (pNode->m_Link2)
                pNode->m_Link2->m_offset=atoi(offset);

        up=atoi(upnode3);
        pNode->m_Link3=FindLink(up,down);
        if (pNode->m_Link3)
                pNode->m_Link3->m_offset=atoi(offset);

        up=atoi(upnode4);
        pNode->m_Link4=FindLink(up,down);
        if (pNode->m_Link4)
                pNode->m_Link4->m_offset=atoi(offset);

        up=atoi(upnode5);
        pNode->m_Link5=FindLink(up,down);
        if (pNode->m_Link5)
                pNode->m_Link5->m_offset=atoi(offset);
}

//store the durations for the 12 alternate signal states
pNode->m_altduration[0]=atoi(dur1);
pNode->m_altduration[1]=atoi(dur2);
pNode->m_altduration[2]=atoi(dur3);
pNode->m_altduration[3]=atoi(dur4);
pNode->m_altduration[4]=atoi(dur5);
pNode->m_altduration[5]=atoi(dur6);
pNode->m_altduration[6]=atoi(dur7);
pNode->m_altduration[7]=atoi(dur8);
pNode->m_altduration[8]=atoi(dur9);
pNode->m_altduration[9]=atoi(dur10);
pNode->m_altduration[10]=atoi(dur11);
pNode->m_altduration[11]=atoi(dur12);
if (!altFile)
{
        //store the durations for the 12 signal states
        pNode->m_duration[0]=atoi(dur1);
        pNode->m_duration[1]=atoi(dur2);
        pNode->m_duration[2]=atoi(dur3);
        pNode->m_duration[3]=atoi(dur4);
        pNode->m_duration[4]=atoi(dur5);
        pNode->m_duration[5]=atoi(dur6);
        pNode->m_duration[6]=atoi(dur7);
        pNode->m_duration[7]=atoi(dur8);
        pNode->m_duration[8]=atoi(dur9);
        pNode->m_duration[9]=atoi(dur10);
        pNode->m_duration[10]=atoi(dur11);
        pNode->m_duration[11]=atoi(dur12);
```

```
                }

                if (altFile)
                        fprintf(fptr,"%s\n",end);
                //get the next line
                end=fgets(line,sizeof(line),fpin);

                //check for end of card type 35
                endnum=80;
                temp1[0]=line[endnum-3];
                temp1[1]=line[endnum-2];
                temp1[2]=line[endnum-1];
                temp1[3]='\0';
                done=atoi(temp1)>35;
        }

        return;

}

void CNetwork::ReadCard36(FILE * fpin, BOOL altFile)
{
        //read the 36 cards in the input TRAF file
        int   endnum;
        char* end;
        int   i;
        int   j;
        int   l;
        int   m;
        char  line[82];
        char  temp1[4];
        BOOL  found;
        BOOL  done;
        CNode* pNode;
        int down;

        //find the card type 36
        end=fgets(line,sizeof(line),fpin);
        found=FALSE;

        while((end!=NULL)&&(!found))
        {
                endnum=80;
                found=(line[endnum-2]=='3')&&(line[endnum-1]=='6');
                if (!found)
                        end=fgets(line,sizeof(line),fpin);
        }

        //store the control codes for the 5 approach links
        //for the 12 signal intervals
        done=FALSE;
        while ((end>0)&&(!done))
        {
                //parse line
                char node[5];
                char code[2];
                int controlCode[12][5];   //intervals,links

                //initialize the character strings
                for (i=0;i<5;i++)
                {
                        node[i]='\0';
                }
                for (i=0;i<2;i++)
                {
```

```
		code[i]='\0';
	}
	for (i=0;i<12;i++)
	{
		for (j=0;j<5;j++)
		{
			controlCode[i][j]=0;
		}
	}

	//parse the line
	for (i=0;i<4;i++)
	{
		node[i]=line[i];
	}

	down=atoi(node);

	pNode=FindNode(down);

	if (pNode!=NULL)
	{
		for (i=5;i<65;i++)
		{
			//l=interval
			//m=approach link
			code[0]=line[i];
			code[1]='\0';
			l=(int)((i-5)/5);
			m=(i-5)-5*l;
			controlCode[l][m]=atoi(code);
		}

		//store the control codes for each of the approach links
		//for each of the 12 alternate signal intervals
		for (l=0;l<12;l++)
		{
			if (pNode->m_Link1!=NULL)
				pNode->m_Link1->m_altcode[l]=controlCode[l][0];
			if (pNode->m_Link2!=NULL)
				pNode->m_Link2->m_altcode[l]=controlCode[l][1];
			if (pNode->m_Link3!=NULL)
				pNode->m_Link3->m_altcode[l]=controlCode[l][2];
			if (pNode->m_Link4!=NULL)
				pNode->m_Link4->m_altcode[l]=controlCode[l][3];
			if (pNode->m_Link5!=NULL)
				pNode->m_Link5->m_altcode[l]=controlCode[l][4];
		}
		if (!altFile)
		{
			//store the control codes for each of the approach links
			//for each of the 12 signal intervals
			for (l=0;l<12;l++)
			{
				if (pNode->m_Link1!=NULL)
					pNode->m_Link1->m_code[l]=controlCode[l][0];
				if (pNode->m_Link2!=NULL)
					pNode->m_Link2->m_code[l]=controlCode[l][1];
				if (pNode->m_Link3!=NULL)
					pNode->m_Link3->m_code[l]=controlCode[l][2];
				if (pNode->m_Link4!=NULL)
					pNode->m_Link4->m_code[l]=controlCode[l][3];
				if (pNode->m_Link5!=NULL)
					pNode->m_Link5->m_code[l]=controlCode[l][4];
			}
```

```
                        }
                        else
                        {
                                endnum=75;
                                temp1[0]=line[endnum-2];
                                temp1[1]=line[endnum-1];
                                temp1[2]='\0';
                                pNode->m_green_remaining=atoi(temp1);
                        }
                }

                if (altFile)
                        fprintf(fptr,"%s\n",end);
                //get the next line
                end=fgets(line,sizeof(line),fpin);

                //check for end of card type 36
                endnum=80;
                temp1[0]=line[endnum-3];
                temp1[1]=line[endnum-2];
                temp1[2]=line[endnum-1];
                temp1[3]='\0';
                done=atoi(temp1)>36;

        }
        return;
}

void CNetwork::CreateSignalStates()
{
        //create the list of signal states
        //for each link
        CLink* pLink;
        POSITION pos;

        pos=m_LinkList.GetHeadPosition();
        while (pos!=NULL)
        {
                pLink=m_LinkList.GetNext(pos);
                pLink->CreateSignalStates();
        }

}
```

```
// Copyright 1997 <Kaman Sciences Corporation>.  All Rights Reserved
// Modified by Beth Taylor in May-June 1998
//          - Added #include "initvars.h" and #include "network.h"
//          - Reordered some of the #include statements
//          - Added code to initialize new variables
//CNode Implementation

#include "link.h"
#include "node.h"
#include "lane.h"
#include "netsim.h"
#include "detector.h"
#include "network.h"
#include "initvars.h"
#include <stdlib.h>
#include <stdio.h>

CNode::CNode()
{

        //default constructor

        //initialize the approach links
        m_Link1=NULL;
        m_Link2=NULL;
        m_Link3=NULL;
        m_Link4=NULL;
        m_Link5=NULL;

        m_altSignalBegin=NULL;
        m_green_remaining=0;

}

CNode::~CNode()
{

}

void CNode::SetSignalState()
{
        //set the signal state for each node
        int time;
        BOOL in;

        time=sclock+endOfInit;
        in=yinit;

        SetSDCCode();

        return;

}

void CNode::SetSDCCode()
{
        //set the SDCODE and AMBSPC for each of the
        //5 approach links
        if (m_Link1!=NULL)
                m_Link1->SetSDCCode();

        if (m_Link2!=NULL)
                m_Link2->SetSDCCode();

        if (m_Link3!=NULL)
```

```
        m_Link3->SetSDCCode();

    if (m_Link4!=NULL)
        m_Link4->SetSDCCode();

    if (m_Link5!=NULL)
        m_Link5->SetSDCCode();

    return;
}
```

```
// Copyright 1997 <Kaman Sciences Corporation>.  All Rights Reserved
// Modified by Beth Taylor in May-June 1998
//          - Added alt parameter to SetSDCODE
//          - Added alt parameter to SetAMPSPC
//          - Added code to do signal priority
//          - Added code to handle some other values for nextActualCode
// CSignalState Implementation

#include "link.h"
#include "signalState.h"

CSignalState::CSignalState()
{
        //default constructor
        //set member variables to default values
        signalCode=-10;
        SDCODE=0;
        AMBSPC=0;
        m_Link=NULL;
}

CSignalState::~CSignalState()
{

}

void CSignalState::SetSDCODE(BOOL alt)
{

        int actualCode;
        int tempSDCODE;

        //if alt is TRUE then we are setting the alternate
        //signal timing codes, else the regular signal timing
        //codes
        if (alt)
                actualCode=m_altactualCode;
        else
                actualCode=m_actualCode;
        //the actualCode is the code read from the 36 cards
        //in the input TRAF file.
        //it ranges from 1 - 9
        //see the documentation for Card type 36 in the
        //TSIS users guide
        if (actualCode==1)
        {
                //green ball
                if (m_Link->m_opposingLink==NULL)
                {
                        //without permitted left
                        tempSDCODE=0;
                }
                else
                {
                        //with permitted left
                        tempSDCODE=16;
                }
        }

        if (actualCode==2)
        {
                //red ball
                tempSDCODE=15;
        }
```

```
		if (actualCode==3)
		{
			//green right turn only
			tempSDCODE=14;
		}

		if (actualCode==4)
		{
			//green left turn only
			tempSDCODE=7;
		}

		if (actualCode==5)
		{
			//stop sign
			tempSDCODE=15;
		}

		if (actualCode==6)
		{
			//green diagonal only
			tempSDCODE=11;
		}

		if (actualCode==7)
		{
			//green through only
			tempSDCODE=4;
		}

		if (actualCode==8)
		{
			//green right and left turn only
			tempSDCODE=2;

		}

		if (actualCode==9)
		{
			//green thru and right only
			tempSDCODE=8;
		}

		if (alt)
			altSDCODE=tempSDCODE;
		else
			SDCODE=tempSDCODE;

		return;
}

void CSignalState::SetAMBSPC(BOOL alt)
{
		//set the AMBSPC for the signal state
		//the AMBSPC for the current signal state
		//will depend on the next non-yellow or non-red
		//signal state
		CSignalState* nextSignalState;
		POSITION pos;
		int nextActualCode;
		BOOL done;
		int actualCode;
		int tempAMBSPC;
		int sigCode;
```

```
//find the next signal state with a nonzero signal code
//a zero signal code means amber
pos=m_Link->m_pos;
done=FALSE;
if (pos==NULL)
{
        pos=m_Link->m_signalStates.GetHeadPosition();
        nextSignalState=m_Link->m_signalStates.GetNext(pos);
}
else
{
        nextSignalState=m_Link->m_signalStates.GetNext(pos);
}
if (alt)
{
        done=nextSignalState->altsignalCode!=0;
        actualCode=m_altactualCode;
        sigCode=altsignalCode;
}
else
{
        done=nextSignalState->signalCode!=0;
        actualCode=m_actualCode;
        sigCode=signalCode;
}

while (!done)
{
        //signal code is 0 for yellow so get the next
        //signal state
        if (pos==NULL)
        {
                pos=m_Link->m_signalStates.GetHeadPosition();
                nextSignalState=m_Link->m_signalStates.GetNext(pos);
        }
        else
        {
                nextSignalState=m_Link->m_signalStates.GetNext(pos);
        }
        if (alt)
                done=nextSignalState->altsignalCode!=0;
        else
                done=nextSignalState->signalCode!=0;
}

if (alt)
        nextActualCode=nextSignalState->m_altactualCode;
else
        nextActualCode=nextSignalState->m_actualCode;

//based on the current signal state and the
//next signal state set the AMBSPC
if (sigCode==0)
{
        if (actualCode==1)
        {
                //green ball
                tempAMBSPC=GetAMBSPC1(nextActualCode);
        }
        if (actualCode==3)
        {
                //green right turn only
                tempAMBSPC=GetAMBSPC3(nextActualCode);
        }
        if (actualCode==4)
```

```
		{
			//green left turn only
			tempAMBSPC=GetAMBSPC4(nextActualCode);
		}
		if (actualCode==5)
		{
			//stop sign
			tempAMBSPC=GetAMBSPC5(nextActualCode);
		}
		if (actualCode==6)
		{
			//green diagonal only
			tempAMBSPC=GetAMBSPC6(nextActualCode);
		}
		if (actualCode==7)
		{
			//green thru only
			tempAMBSPC=GetAMBSPC7(nextActualCode);
		}
		if (actualCode==8)
		{
			//green rght and left turn only
			tempAMBSPC=GetAMBSPC8(nextActualCode);
		}
		if (actualCode==9)
		{
			//greem thru and right turn only
			tempAMBSPC=GetAMBSPC9(nextActualCode);
		}
	}
	else
	{
		//red ball
		tempAMBSPC=15;
	}

	if (alt)
		altAMBSPC=tempAMBSPC;
	else
		AMBSPC=tempAMBSPC;

	return;
}

int CSignalState::GetAMBSPC1(int nextActualCode)
{
	//this routine is limited to use with the INTCH.TRF file
	//provided in this example
	//if other files are to be used this routine may
	//need to be expanded to handle other values for the
	//input argument nextActualCode

	//m_actualCode==1
	//green ball
	int ambspc;

	if (nextActualCode==2)
	{
		//green ball to all red
		ambspc=0;
	}
	if (nextActualCode==3)
	{
		//green ball to green right turn only
		ambspc=1;
```

```
	}
	if (nextActualCode==9)
	{
		//green ball to green right & thru only
		ambspc=3;
	}

	return ambspc;
}

int CSignalState::GetAMBSPC3(int nextActualCode)
{
	//this routine is limited to use with the INTCH.TRF file
	//provided in this example
	//if other files are to be used this routine may
	//need to be expanded to handle other values for the
	//input argument nextActualCode

	//m_actualCode==3
	//green right turn only
	int ambspc;

	if (nextActualCode==2)
	{
		//green right turn only to all red
		ambspc=14;
	}
	if (nextActualCode==4)
	{
		//green right turn only to green left turn only
		ambspc=14;
	}

	return ambspc;
}

int CSignalState::GetAMBSPC4(int nextActualCode)
{
	//this routine is limited to use with the INTCH.TRF file
	//provided in this example
	//if other files are to be used this routine may
	//need to be expanded to handle other values for the
	//input argument nextActualCode

	//m_actualCode==4
	//green left turn only
	int ambspc;

	if (nextActualCode==1)
	{
		//green left turn only
		//to green ball
		ambspc=7;
	}
	if (nextActualCode==2)
	{
		//green left turn only
		//to red ball
		ambspc=7;
	}
	if (nextActualCode==9)
	{
		//green left turn only
		//to green right & thru only
		ambspc=7;
```

```
	}

	return ambspc;
}

int CSignalState::GetAMBSPC5(int nextActualCode)
{
	//this routine is limited to use with the INTCH.TRF file
	//provided in this example
	//if other files are to be used this routine may
	//need to be expanded to handle other values for the
	//input argument nextActualCode

	//m_actualCode==5
	//stop sign
	int ambspc;

	if (nextActualCode==2)
	{
		//stop sign to all red
		ambspc=15;
	}

	return ambspc;
}

int CSignalState::GetAMBSPC6(int nextActualCode)
{
	//this routine is limited to use with the INTCH.TRF file
	//provided in this example
	//if other files are to be used this routine may
	//need to be expanded to handle other values for the
	//input argument nextActualCode

	//m_actualCode==6
	//green diagonal only
	int ambspc;

	if (nextActualCode==2)
	{
		//green diagonal only
		//to all red
		ambspc=11;
	}

	return ambspc;
}

int CSignalState::GetAMBSPC7(int nextActualCode)
{
	//this routine is limited to use with the INTCH.TRF file
	//provided in this example
	//if other files are to be used this routine may
	//need to be expanded to handle other values for the
	//input argument nextActualCode

	//m_actualCode==7
	//green thru only
	int ambspc;

	if (nextActualCode==2)
	{
		//green thru only
		//to all red
		ambspc=13;
```

```
        }

        return ambspc;
}

int CSignalState::GetAMBSPC8(int nextActualCode)
{
        //this routine is limited to use with the INTCH.TRF file
        //provided in this example
        //if other files are to be used this routine may
        //need to be expanded to handle other values for the
        //input argument nextActualCode

        //m_actualCode==8
        //green right and left turn only
        int ambspc;

        if (nextActualCode==2)
        {
                //green right and left turn only
                //to all red
                ambspc=6;
        }

        return ambspc;
}

int CSignalState::GetAMBSPC9(int nextActualCode)
{
        //this routine is limited to use with the INTCH.TRF file
        //provided in this example
        //if other files are to be used this routine may
        //need to be expanded to handle other values for the
        //input argument nextActualCode

        //m_actualCode==9
        //green right and thru only
        int ambspc;

        if (nextActualCode==2)
        {
                //green right and thru only
                //to all red
                ambspc=12;
        }

        return ambspc;
}
```

```
// Copyright 1997 <Kaman Sciences Corporation>.  All Rights Reserved
// Modified by Beth Taylor in May-June 1998
//          - Fixed code to allow blanks in input filename
//          - Added code for new file types (.alt and .det) used in signal priority


#include "netsim.h"
#include "network.h"
#include "upcntrl.h"
#include "stdlib.h"
#include "stdio.h"

extern "C" { void UPINIT(char *fname); }
extern "C" { void UPCNTRL(void); }
extern "C" { void UPEXIT(void); }

void UPINIT( char *fname )
{
        //initialization routine
        //called once at the beginning of simulation
        int i;
        BOOL done;

        endOfInit=0;
        prevInit=0;

        pNetwork=new CNetwork();

        //fname must be null terminated
        done=FALSE;
        i=511;
        while ((!done)&&(i>=0))
        {
                if (fname[i]==' ')
                        i--;
                else
                        done=TRUE;
        }
        fname[i+1]='\0';

        pNetwork->m_TrafInputFile=CString(fname);
        //This assumes that the TrafInputFile has a .trf extension
        fname[i-2]='\0';
        pNetwork->m_altSignalFile=CString(strcat(fname,"alt"));
        fname[i-2]='\0';
        pNetwork->m_detOutputFile=CString(strcat(fname,"det"));

        //open the detector output file
        fptr=fopen(pNetwork->m_detOutputFile,"w");

        //read the traf file
        pNetwork->ReadTrafFile();

        return;
}

void UPCNTRL()
{
        int time;
        BOOL init;

        init=yinit;

        //the algorithm that controlls the signal states at the
        //intersections assumes time is always increasing, but
```

```
        //the CORSIM clock starts over after initialization
        //so the time at which initialization is over must
        //be recorded
        if ((!init)&&(prevInit))
        {
                //end of initialization
                endOfInit=prevTime+1;
        }

        //adjust the time by adding the end of initialization
        time=sclock+endOfInit;

        //get signal state for the node under corsim control
        pNetwork->UpdateNodeSignalStates();

        //record whether the simulation has reached equilibrium
        //or not, so the time at which initialization can be
        //recorded
        prevInit=init;
        prevTime=time;
}

void UPEXIT()
{
        //clean up
        //delete all objects that were created
        delete pNetwork;
        fclose(fptr);
}
```

```
c KSC.for
c
c This file contains the interfaces for TSIS
c

c Define the interface for the C function UPINIT
c It accepts the TRAF input filename as an argument
      INTERFACE TO SUBROUTINE UPINIT(FILENAME)
            !MS$ATTRIBUTES C, ALIAS:'_UPINIT' :: UPINIT
            CHARACTER*(512) FILENAME
            !MS$ATTRIBUTES REFERENCE :: FILENAME
            END SUBROUTINE UPINIT
          END

c Define the interface for the C function UPCNTRL
      INTERFACE TO SUBROUTINE UPCNTRL()
            !MS$ATTRIBUTES C, ALIAS:'_UPCNTRL' :: UPCNTRL
          END

c Define the interface for the C function UPEXIT
          INTERFACE TO SUBROUTINE UPEXIT()
            !MS$ATTRIBUTES C, ALIAS:'_UPEXIT' :: UPEXIT
      END

          SUBROUTINE MSGBOX(IMESSAGE)
      CHARACTER*132 IMESSAGE
      CALL MYMSGBOX(IMESSAGE,"interfac.dll")
      RETURN
      END

      SUBROUTINE SETIOFILES
      INCLUDE 'KSC.FD'
      common /Liofiles/ linfname,loutfname,linflen,loutflen
      character*512 linfname,loutfname
      common /screen/ iscreen
      LINFNAME = INFNAME
      LOUTFNAME = OUTFNAME
      LINFLEN = INFLEN
      LOUTFLEN = OUTFLEN
      ISCREEN = GHWND
      RETURN
      END


      SUBROUTINE CWRITE(CWRTBUF)
      COMMON /SCREEN/ ISCREEN
      CHARACTER*132 CWRTBUF
      CALL SCREENIO(ISCREEN,CWRTBUF,'R')
      RETURN
      END

      SUBROUTINE SENDERRORMSG(CWRTBUF)
      COMMON /SCREEN/ ISCREEN
      CHARACTER*132 CWRTBUF
      CALL SCREENIO(ISCREEN,CWRTBUF,'E')
      RETURN
      END

c The JMAIN subroutine is called once every second by TSIS
      INTEGER*2 FUNCTION JMAIN[DLLEXPORT, STDCALL]()
            common /Liofiles/ linfname,loutfname,linflen,loutflen
      character*512 linfname,loutfname
            CALL UPCNTRL
            JMAIN = 0
      RETURN
```

```
      END

c The INIT function is called once at the beginning of simulation
      INTEGER*2 FUNCTION INIT[DLLEXPORT, STDCALL]()
            common /Liofiles/ linfname,loutfname,linflen,loutflen
            character*512 linfname,loutfname
            CALL SETIOFILES
            CALL UPINIT(linfname)
      INIT = 0
      RETURN
      END

c The JEXIT function is called at the end of simulation
            INTEGER*2 FUNCTION JEXIT[DLLEXPORT, STDCALL]()
            CALL UPEXIT
            JEXIT = 0
      RETURN
      END
```

APPENDIX B. ANOVA RESULTS

ANOVA RESULTS FOR LEFT TURNS AND PRIORITY USING *LRT PERSON DELAY*

Tests of Main and Interaction Effects	*Df*	*F*	*Sig.*	*Eta Squared*
Priority	2, 4	5.766	0.066	0.742
Lefts	1, 2	102.083	0.010	0.981
Priority * Lefts	2, 4	5.291	0.075	0.726

ANOVA RESULTS FOR FACTORS OF LEFT TURNS AND PRIORITY
(SB LRT OCCUPANCY = 250)

Tests of Main and Interaction Effects	*Df*	*F*	*Sig.*	*Eta Squared*
Priority	2, 4	1.656	0.299	0.453
Lefts	1, 2	251.666	0.005	0.992
Priority * Lefts	2, 4	1.027	0.437	0.339

ANOVA RESULTS FOR LEFT TURNS AND PRIORITY USING *LRT PERSON DELAY*
(SB LRT OCCUPANCY = 250)

Tests of Main and Interaction Effects	*Df*	*F*	*Sig.*	*Eta Squared*
Priority	2, 4	5.766	0.066	0.742
Lefts	1, 2	103.080	0.010	0.981
Priority * Lefts	2, 4	5.291	0.075	0.726

ANOVA RESULTS FOR FACTORS OF LEFT TURNS AND PRIORITY
(SB LRT OCCUPANCY = 119)

Tests of Main and Interaction Effects	*Df*	*F*	*Sig.*	*Eta Squared*
Priority	2, 4	3.804	0.119	0.655
Lefts	1, 2	550.246	0.002	0.996
Priority * Lefts	2, 4	2.138	0.234	0.517

ANOVA RESULTS FOR FACTORS OF LEFT TURNS AND PRIORITY USING *LRT PERSON DELAY* (SB LRT OCCUPANCY = 119)

Tests of Main and Interaction Effects	*Df*	*F*	*Sig.*	*Eta Squared*
Priority	2, 4	5.765	0.066	0.742
Lefts	1, 2	102.081	0.010	0.991
Priority * Lefts	2, 4	5.280	0.075	0.725

ANOVA RESULTS FOR FACTORS OF LEFT TURNS AND PRIORITY
(SB LRT OCCUPANCY = 75)

Tests of Main and Interaction Effects	*Df*	*F*	*Sig.*	*Eta Squared*
Priority	2, 4	3.804	0.119	0.655
Lefts	1, 2	1110.868	0.001	0.998
Priority * Lefts	2, 4	2.138	0.234	0.517

ANOVA RESULTS FOR FACTORS OF LEFT TURNS AND PRIORITY USING *LRT PERSON DELAY* (SB LRT OCCUPANCY = 75)

Tests of Main and Interaction Effects	*Df*	*F*	*Sig.*	*Eta Squared*
Priority	2, 4	5.765	0.066	0.742
Lefts	1, 2	103.074	0.010	0.981
Priority * Lefts	2, 4	5.283	0.075	0.725

ANOVA RESULTS FOR LEFT TURNS AND PROGRESSION USING *LRT PERSON DELAY*

Tests of Main and Interaction Effects	*Df*	*F*	*Sig.*	*Eta Squared*
Progression	2, 4	363.442	0.000	0.995
Lefts	1, 2	72.026	0.014	0.973
Progression * Lefts	2, 4	71.888	0.001	0.973
Tests of Simple Effects				
Left Turn Factor comparisons at each level of Progression Factor				
Protected vs. No Lefts at Existing Progression	1, 2	95.699	0.010	0.980
Protected vs. No Lefts at One-Way Progression	1, 2	59.111	0.016	0.967
Protected vs. No Lefts at One-Way LRT Progression	1, 2	24.140	0.039	0.923
Progression Factor comparisons at each level of Left Turn Factor (Protected Lefts)				
Existing vs. Auto Progression at Protected Lefts	1, 2	15.591	0.059	0.886
Existing vs. One-Way LRT Progression at Protected Lefts	1, 2	207.050	0.005	0.990
One-Way vs. One-Way LRT Progression at Protected Lefts	1, 2	3881.551	0.000	0.999
(No Lefts)				
Existing vs. Auto Progression at No Lefts	1, 2	208.409	0.005	0.990
Existing vs. One-Way LRT Progression at No Lefts	1, 2	18.705	0.050	0.903
One-Way vs. One-Way LRT Progression at No Lefts	1, 2	293.997	0.003	0.993

ANOVA RESULTS FOR FACTORS OF LEFT TURNS AND PROGRESSION
(SB LRT OCCUPANCY = 250)

Tests of Main and Interaction Effects	*Df*	*F*	*Sig.*	*Eta Squared*
Progression	2, 4	117.750	0.000	0.983
Lefts	1, 2	198.042	0.005	0.990
Progression * Lefts	2, 4	43.622	0.002	0.956
Tests of Simple Effects				
Left Turn Factor comparisons at each level of Progression Factor				
Protected vs. No Lefts at Existing Progression	1, 2	183.829	0.005	0.989
Protected vs. No Lefts at One-Way Progression	1, 2	1137.355	0.001	0.998
Protected vs. No Lefts at One-Way LRT Progression	1, 2	55.783	0.017	0.965
Progression Factor comparisons at each level of Left Turn Factor (Protected Lefts)				
Existing vs. Auto Progression at Protected Lefts	1, 2	1.772	0.315	0.470
Existing vs. One-Way LRT Progression at Protected Lefts	1, 2	170.620	0.006	0.998
One-Way vs. One-Way LRT Progression at Protected Lefts	1, 2	276.278	0.004	0.993
(No Lefts)				
Existing vs. Auto Progression at No Lefts	1, 2	42.199	0.023	0.955
Existing vs. One-Way LRT Progression at No Lefts	1, 2	95.794	0.010	0.980
One-Way vs. One-Way LRT Progression at No Lefts	1, 2	217.828	0.005	0.991

ANOVA RESULTS FOR LEFT TURNS AND PROGRESSION USING *LRT PERSON DELAY*
(SB LRT OCCUPANCY = 250)

Tests of Main and Interaction Effects	*Df*	*F*	*Sig.*	*Eta Squared*
Progression	2, 4	182.385	0.000	0.989
Lefts	1, 2	57.624	0.017	0.966
Progression * Lefts	2, 4	44.743	0.002	0.957
Tests of Simple Effects				
Left Turn Factor comparisons at each level of Progression Factor				
Protected vs. No Lefts at Existing Progression	1, 2	92.886	0.011	0.979
Protected vs. No Lefts at One-Way Progression	1, 2	47.971	0.020	0.960
Protected vs. No Lefts at One-Way LRT Progression	1, 2	20.362	0.046	0.911
Progression Factor comparisons at each level of Left Turn Factor (Protected Lefts)				
Existing vs. Auto Progression at Protected Lefts	1, 2	11.608	0.076	0.853
Existing vs. One-Way LRT Progression at Protected Lefts	1, 2	113.817	0.009	0.983
One-Way vs. One-Way LRT Progression at Protected Lefts	1, 2	1408.848	0.001	0.999
(No Lefts)				
Existing vs. Auto Progression at No Lefts	1, 2	105.610	0.009	0.981
Existing vs. One-Way LRT Progression at No Lefts	1, 2	1396.916	0.001	0.999
One-Way vs. One-Way LRT Progression at No Lefts	1, 2	250.601	0.004	0.992

ANOVA RESULTS FOR FACTORS OF LEFT TURNS AND PROGRESSION
(SB LRT OCCUPANCY = 119)

Tests of Main and Interaction Effects	*Df*	*F*	*Sig.*	*Eta Squared*
Progression	2, 4	54.405	0.001	0.965
Lefts	1, 2	1072.923	0.001	0.998
Progression * Lefts	2, 4	57.048	0.001	0.966
Tests of Simple Effects				
Left Turn Factor comparisons at each level of Progression Factor				
Protected vs. No Lefts at Existing Progression	1, 2	353.986	0.003	0.994
Protected vs. No Lefts at One-Way Progression	1, 2	2040.536	0.000	0.999
Protected vs. No Lefts at One-Way LRT Progression	1, 2	248.755	0.004	0.992
Progression Factor comparisons at each level of Left Turn Factor (Protected Lefts)				
Existing vs. Auto Progression at Protected Lefts	1, 2	3.886	0.187	0.660
Existing vs. One-Way LRT Progression at Protected Lefts	1, 2	215.965	0.005	0.991
One-Way vs. One-Way LRT Progression at Protected Lefts	1, 2	94.593	0.010	0.979
(No Lefts)				
Existing vs. Auto Progression at No Lefts	1, 2	1.327	0.368	0.399
Existing vs. One-Way LRT Progression at No Lefts	1, 2	6.559	0.125	0.766
One-Way vs. One-Way LRT Progression at No Lefts	1, 2	139.156	0.007	0.986

ANOVA RESULTS FOR LEFT TURNS AND PROGRESSION USING *LRT PERSON DELAY*
(SB LRT OCCUPANCY = 119)

Tests of Main and Interaction Effects	*Df*	*F*	*Sig.*	*Eta Squared*
Progression	2, 4	363.414	0.000	0.995
Lefts	1, 2	72.018	0.014	0.973
Progression * Lefts	2, 4	71.894	0.001	0.973
Tests of Simple Effects				
Left Turn Factor comparisons at each level of Progression Factor				
Protected vs. No Lefts at Existing Progression	1, 2	95.688	0.010	0.980
Protected vs. No Lefts at One-Way Progression	1, 2	59.112	0.016	0.967
Protected vs. No Lefts at One-Way LRT Progression	1, 2	24.138	0.039	0.923
Progression Factor comparisons at each level of Left Turn Factor (Protected Lefts)				
Existing vs. Auto Progression at Protected Lefts	1, 2	15.582	0.059	0.886
Existing vs. One-Way LRT Progression at Protected Lefts	1, 2	207.035	0.005	0.990
One-Way vs. One-Way LRT Progression at Protected Lefts	1, 2	3880.443	0.000	0.999
(No Lefts)				
Existing vs. Auto Progression at No Lefts	1, 2	208.459	0.005	0.990
Existing vs. One-Way LRT Progression at No Lefts	1, 2	18.714	0.050	0.903
One-Way vs. One-Way LRT Progression at No Lefts	1, 2	294.016	0.003	0.993

ANOVA RESULTS FOR FACTORS OF LEFT TURNS AND PROGRESSION
(SB LRT OCCUPANCY = 75)

Tests of Main and Interaction Effects	*Df*	*F*	*Sig.*	*Eta Squared*
Progression	2, 4	41.368	0.002	0.954
Lefts	1, 2	1797.581	0.001	0.999
Progression * Lefts	2, 4	32.169	0.003	0.941
Tests of Simple Effects				
Left Turn Factor comparisons at each level of Progression Factor				
Protected vs. No Lefts at Existing Progression	1, 2	546.631	0.002	0.996
Protected vs. No Lefts at One-Way Progression	1, 2	846.299	0.001	0.998
Protected vs. No Lefts at One-Way LRT Progression	1, 2	227.484	0.004	0.991
Progression Factor comparisons at each level of Left Turn Factor (Protected Lefts)				
Existing vs. Auto Progression at Protected Lefts	1, 2	3.082	0.221	0.606
Existing vs. One-Way LRT Progression at Protected Lefts	1, 2	108.899	0.009	0.982
One-Way vs. One-Way LRT Progression at Protected Lefts	1, 2	52.069	0.019	0.963
(No Lefts)				
Existing vs. Auto Progression at No Lefts	1, 2	1.666	0.326	0.455
Existing vs. One-Way LRT Progression at No Lefts	1, 2	21.493	0.044	0.915
One-Way vs. One-Way LRT Progression at No Lefts	1, 2	93.828	0.010	0.979

ANOVA RESULTS FOR LEFT TURNS AND PROGRESSION USING *LRT PERSON DELAY*
(SB LRT OCCUPANCY = 75)

Tests of Main and Interaction Effects	*Df*	*F*	*Sig.*	*Eta Squared*
Progression	2, 4	182.328	0.000	0.989
Lefts	1, 2	57.635	0.017	0.966
Progression * Lefts	2, 4	44.734	0.002	0.957
Tests of Simple Effects				
Left Turn Factor comparisons at each level of Progression Factor				
Protected vs. No Lefts at Existing Progression	1, 2	92.891	0.011	0.979
Protected vs. No Lefts at One-Way Progression	1, 2	47.984	0.020	0.960
Protected vs. No Lefts at One-Way LRT Progression	1, 2	20.365	0.046	0.911
Progression Factor comparisons at each level of Left Turn Factor (Protected Lefts)				
Existing vs. Auto Progression at Protected Lefts	1, 2	11.603	0.076	0.853
Existing vs. One-Way LRT Progression at Protected Lefts	1, 2	113.818	0.009	0.983
One-Way vs. One-Way LRT Progression at Protected Lefts	1, 2	1407.519	0.001	0.999
(No Lefts)				
Existing vs. Auto Progression at No Lefts	1, 2	105.574	0.009	0.981
Existing vs. One-Way LRT Progression at No Lefts	1, 2	1398.884	0.001	0.999
One-Way vs. One-Way LRT Progression at No Lefts	1, 2	250.497	0.004	0.992

BIBLIOGRAPHY

Abebe, Yohannes, Stewart Gordon, Mustafa Kamal, and Robert Conte (1996). "New Jersey: Coordinating LRV Movements Under Traffic Signal Control for Downtown Area", *PB Network*, Summer, Issue No. 35, pp.18-19, 59.

Bauer, Thomas, Mark P. Medema, and Subbarao V. Jayanthi (1995). "Testing of Light Rail Signal Control Strategies by Combining Transit and Traffic Simulation Models", *Transportation Research Record*, Number 1494, pp. 155-160.

Bodell, Graham and Ken Huddart (1987). "Tram Priority in Hong Kong's First Light Rail Transit System", *Traffic Engineering and Control*, Volume 28, Number 9, pp. 446-451, 470.

Celniker, Stephen and E. Wayne Terry (1992). "Trolley Priority on Signalized Arterials in Downtown San Diego", *Transportation Research Record*, Number 1361, pp. 184-187.

Chin, Kim and John Mundy (1992). "Control of the Light Rail Transit/traffic Conflict: An Update from Hong Kong, and Simulation Using the FLEXSYT Program", *Traffic Engineering and Control*, Volume 33, Number 2, pp. 65-71.

Colquhoun, Dave, John Morrall, and John Hubbell (1995). "Calgary Light Rail Transit Surface Operations and Grade-Level Crossings", *Transportation Research Record*, Number 1503, pp. 127-136.

Fehon, Kevin J., Warren A. Tighe, and Peter L. Coffey (1988). "Operational Analysis of At-Grade Light Rail Transit", *Transportation Research Board Special Report 221; Light Rail Transit: New System Successes at Affordable Prices*, pp. 593-605.

Fehon, K. J., W. A. Tighe, and A. O. Albers (1990). "Advanced Integration of LRT and Traffic Signals", *Third International Conference on Road Traffic Control*, May, pp. 209-214.

Federal Highway Administration (FHWA) (1998a). TSIS 4.3 alpha, Washington, D.C.

Federal Highway Administration (FHWA) (1998b). *TSIS 4.2 Run-Time CORSIM Run-Time Extension User's Guide*, Washington, D.C.

Fox, Gerald (1985). "Traffic Interface on the Banfield Light Rail Project", *Transportation Research Board State-of-the-Art Report 2, Light Rail Transit: System Design for Cost-Effectiveness*, pp. 203-212.

Fox, Gerald D. (1988). "Designing At-Grade Light Rail Transit", *Transportation Research Board Special Report 221; Light Rail Transit: New System Successes at Affordable Prices*, pp. 606-620.

Garrow, Michael Charles (1997). *Development and Evaluation of Transit Signal Priority Strategies*, Master's thesis, The University of Texas at Austin, Austin, Texas.

Homburger, Wolfgang S., Jerome W. Hall, Roy C. Loutzenheiser, and William R. Reilly (1996). *Fundamentals of Traffic Engineering,* Fourteenth Edition, Institute of Transportation Studies, University of California, Berkley.

Hood III, Woodrow, Thomas Hicks, and Linda I. Singer (1995). "Light Rail Preemption of Traffic Signals: A Question of Balance", *Seventh National Conference on Light Rail Transit*, Volume 1, pp. 285-293.

Kessmann, Roy W. and D. L. Cooper (1986). "Resolution of Automobile and Light Rail Rapid Transit Vehicle Conflicts: An American Approach", *Second International Conference on Road Traffic Control*, pp. 162-165.

Koch, Matthew I., Daniel C. Chin, and Richard H. Smith (1995). "Network Approach to Optimal Signal Timing for Integrated Transit Vehicle and Traffic Operations", *Seventh National Conference on Light Rail Transit*, Volume 2, pp. 126-131.

Korve, Hans W. (1978). "Traffic Engineering for Light-Rail Transit", *Transportation Research Board Special Report 182; Light Rail Transit: Planning and Technology*, pp. 107-115.

Korve, Hans W., Jose I. Farran, Douglas M. Mansel, Herbert S. Levinson, Ted Chira-Chavala, and David R. Ragland (1996*). TCRP Report 17: Integration of Light Rail Transit into City Streets*, National Academy Press, Washington, D.C.

Kuah, Geok K. and Jeffrey B. Allen (1992). "Designing At-Grade LRT Progression: Proposed Baltimore Central Light Rail", *Transportation Research Record*, Number 1361, pp. 207-216.

Noyce, David A. (1996). "Barriers to Implementation of Signal Priority Systems for Transit", *Compendium: Graduate Student Papers on Advanced Surface Transportation Systems*, Research Report SWUTC/96/72840/00003-1, Southwest Region University Transportation Center, Texas Transportation Institute, The Texas A&M University System, College Station, TX.

Radwan, A. Essam, and Kuo-Ping Hwang (1985). "Preferential Control Warrants of Light Rail Transit Movements", *Transportation Research Board State-of-the-Art Report 2, Light Rail Transit: System Design for Cost-Effectiveness*, pp. 234-240.

Rymer, Bruce, Thomas Urbanik II, and James C. Cline, Jr. (1988). "Delay at Light Rail Transit Grade Crossings", *Transportation Research Board Special Report 221; Light Rail Transit: New System Successes at Affordable Prices*, pp. 621-634.

Saffer, H. G. and T. Wright (1994). "The Development of LRT and Signal Control Techniques for the Sheffield Supertram System", *Seventh International Conference on Road Traffic Monitoring and Control*, April, pp. 89-92.

Stone, Thomas J. and William A. Wild (1982). "Design Consideration for LRT in Existing Medians: Developing Warrants for Priority Treatments", *Transportation Research Board Special Report 195; Light Rail Transit: Planning, Design, and Implementation*, pp. 170-175.

Sunkari, Srinivasa R., Phillip S. Beasley, Thomas Urbanik II, and Daniel B. Fambro (1995). "Model to Evaluate the Impacts of Bus Priority on Signalized Intersections", *Transportation Research Record*, Number 1494, pp. 117-123.

Tighe, Warren A., and Larry Patterson (1985). "Integrating LRT into Flexible Traffic Control Systems", *Transportation Research Board State-of-the-Art Report 2, Light Rail Transit: System Design for Cost-Effectiveness*, pp. 213-220.

Tyer, Kevin D., Kethireddiapli S. Rao, Amadou Oumarou, and Raymond A. Krammes (1995). *Urban Public Transit Systems Modeling Capabilities*, Research Report SWUTC/95/60040-1, Southwest Region University Transportation Center, Texas Transportation Institute, The Texas A&M University System, College Station, TX.

Venglar, Steven, Daniel B. Fambro, and Thomas Bauer (1995). "Validation of Simulation Software for Modeling Light Rail Transit", *Transportation Research Record*, Number 1494, pp. 161-166.

Wu, Jianping and Mike McDonald (1996). "TRGMSM: A Simulation Model for Light Rail Transit (LRT) At-Grade Crossing Design", *Traffic Engineering and Control*, Volume 37, Number 3, pp. 173-177.

Yagar, Sam and Ben Heydecker (1988). "Potential Benefits to Transit in Setting Traffic Signals", *Transportation Research Board Special Report 221; Light Rail Transit: New System Successes at Affordable Prices*, pp. 657-667.