A MODULAR IMPLEMENTATION OF THE SAVaNT ANTICIPATORY ROUTE GUIDANCE ALGORITHM*

Daniel J. Reaume
Department of Industrial & Operations Engineering
The University of Michigan
Ann Arbor, Michigan 48109-2117

Technical Report 95-27

November 1995

*This work was supported in part by the Intelligent Transportation Systems Research Center of Excellence at the University of Michigan.

A Modular Implementation of the SAVaNT Anticipatory Route Guidance Algorithm

Daniel J. Reaume

Department of Industrial and Operations Engineering
University of Michigan, Ann Arbor MI

Abstract

The SAVaNT anticipatory route guidance algorithm determines user-optimal routings for vehicles in a traffic network by iteratively executing a route optimization program and a traffic simulation. In this paper, we describe an implementation of the SAVaNT algorithm using a modular system architecture. We then examine various implementations of SAVaNT employing rolling horizon procedures to reduce computational effort.

Introduction

The SAVaNT (Simulation of Anticipatory Vehicle Network Traffic) system, developed at the University of Michigan , optimizes of vehicle routings in a traffic network. [1] This system consists of a routing program and a simulation program that are called iteratively as depicted in *Figure 1*.

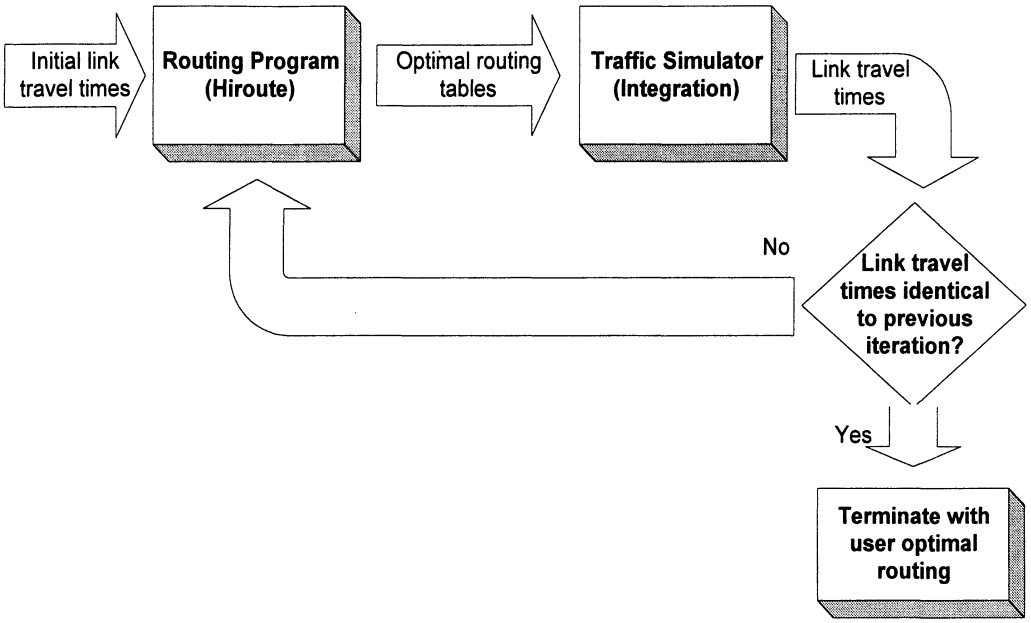


Figure 1

Given an initial set of time dependent link travel times, the routing program computes a set of routings. These routings are optimal in the sense that if the provided link travel times are the actual travel times experienced by vehicles, then the vehicles will experience the shortest possible trip times. Unfortunately, this generally is not the case since a new routing of vehicles changes congestion patterns, thus modifying the link travel times from those provided to the routing program.

Applying the routings provided by the routing program, the traffic simulator simulates the passage of vehicles through the traffic network and outputs the time dependent link travel times experienced by these vehicles. In [1], it was proven that if these travel times are identical to those used by the routing program to produce the routings used by the traffic simulation in this iteration, then these routings are user optimal. User optimality implies that no vehicle may shorten its travel time by unilaterally deviating from the provided routings. In the event that tables of time dependent link travel times differ in consecutive SAVaNT iterations, the travel times most recently generated by the simulation are input to the routing program to begin another iteration of the optimization scheme.

As originally implemented, on a Macintosh Computer, the SAVaNT system consisted of the Hiroute vehicle routing program, written in C, and the Integration traffic simulator, written in FORTRAN. A top-level Hypercard script controlled the iterative execution of these two modules. For two reasons, this architecture proved to be less than optimal. First, and most importantly, Hypercard is a Macintosh specific application, hence its use prohibited the execution of SAVaNT on speedier workstations to analyze the behavior of very large traffic

networks. Secondly, many research problems in traffic optimization may be examined by modifying the SAVaNT package appropriately, hence it would be desirable for SAVaNT to be coded in a modular manner to facilitate such modifications.

To remedy these limitations, we have developed a new modular implementation of SAVaNT around a set of high level, platform independent C subroutines. These routines also simplify the replacement of SAVaNT components, such as the traffic simulator, with a minimum of effort. Using these subroutines, we have extended SAVaNT to allow it to optimize vehicle routings using rolling horizon procedures.

In the first section of this paper, we describe the details of the subroutines with which the new SAVaNT optimizer was constructed. The second section of this paper is devoted to a discussion of the issues to be addressed in the construction of a rolling horizon implementation of SAVaNT. Finally, the third section of this paper describes the actual implementation of a SAVaNT in a particular rolling horizon framework.

Subroutines for Traffic Simulation

The principal task of the SAVaNT optimization program is to iteratively execute various modules, and to allow communication between these modules. This task may be subdivided into three major classes of functions performing the following duties:

- Execute a module and wait for that module to terminate.
- Move a data file from one location to another.
- Truncate or modify a data file.

Executing a Module

To execute a module, we have developed the C subroutine ***launchprogram*** with the following functionality:

Calling convention:
int launchprogram(const char* filename)

Return values:
0 if program executes and terminates successfully
1 otherwise

Moving a Data File

Although certain implementations of the C language contain library functions to perform this task, many do not. Hence, to preserve code portability, we have developed the subroutine ***copyfile*** to make an identical copy of a file having filename ***oldfilename*** in a file having filename ***newfilename***. This subroutine does not modify the original file in any way.

If a file is to be moved rather than copied, the ANSI-C library function ***remove()*** may be employed to delete the original file after a copy operation takes place.

Calling convention:
int copyfile(char* oldfilename, char* newfilename)

Return values:
0 if file is copied successfully
1 otherwise

Modifying a Data File

This class of functions encompasses a wide variety of subroutines, often specific to the particular modules used in an optimization system. We present those used in one rolling horizon implementation of SAVaNT.

Set the simulation length to equal *n* time periods: This subroutine modifies the input files to the traffic simulator to indicate that ***n*** time periods must be simulated. In the case of Integration, this task reduces to modifying a parameter in the file **master.fil**, hence the name of the subroutine.

Calling convention:

int modifymast(int n)

Return values:

0 if successful

1 otherwise

Set the routing horizon to equal *n* time periods: This subroutine modifies the input files to the traffic routing module to indicate that link travel times are only provided for the first ***n*** time periods, and that routing information should be produced for ***n*** time periods. The routing program uses its own conventions to account for vehicles left on the network at the end of these **n** time periods. In the case of Hiroute, this task reduces to modifying a parameter in the file **siminfo.dat**, hence the name of the subroutine.

Calling convention:

int modifydat(int n)

Return values:

0 if successful

1 otherwise

Set the accuracy with which the simulation will report experienced link travel times: To ensure the convergence of SAVaNT, a parameter called ***delta*** must sometimes be modified to modify the link travel times reported by the traffic simulator. In the case of Integration, a parameter must be changed in the file **master.fil**. We will discuss this issue in the third section of this paper.

Calling convention:

int changedelta(float delta)

Return values:

0 if successful

1 otherwise

Merge routing tables: In many instances, it is useful to fix the first part of a routing table, and merge the end of a newly generated routing table onto this fixed portion in order to cover the entire simulation horizon. The following subroutine concatenates the time periods ***len-rhorizon*** to ***len*** of a routing table, ***genroute*** to the first ***len-rhorizon-1*** time periods of fixed routing table, **fixedroute**, to generate the new routing table in file ***newroute.*** We must provide the subroutine with ***highnode,*** the number of origin/destination zones in the traffic network, since the number of such nodes determines the size of vehicle routing tables.

Calling convention:

int mergefiles(int len, int rhorizon, int highnode, char *fixedroute, char *genroute, char *newroute)

Return values:

0 if successful

1 otherwise

Issues in Constructing a Rolling Horizon Implementation of SAVaNT

Rolling Horizon Algorithms

In a rolling horizon algorithm, difficult problems are broken into a sequence of smaller problems which may be more efficiently solved in the manner depicted in *Figure 2*.

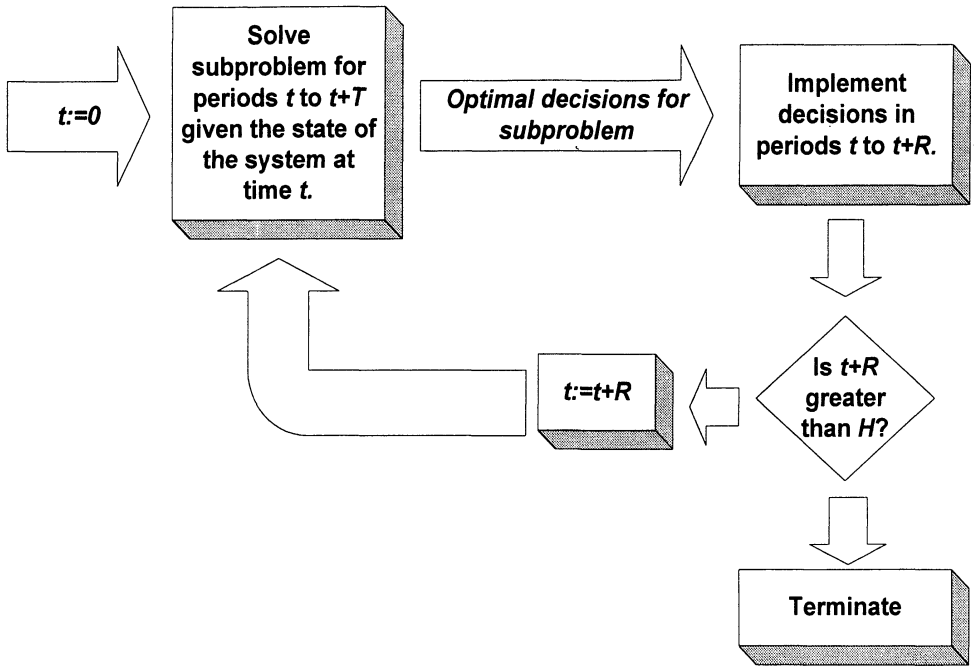


Figure 2

Consider the optimization of a system over time periods 0 to H, a potentially difficult problem. Instead of solving this single large problem, let us begin by solving the subproblem corresponding to time periods 0 to T and implement the decisions thus computed until some time R, where R may not be greater than T. Given the current state of the system at time R, we then solve the subproblem corresponding to time periods R through $R+T$, and implement the decisions thus computed from time periods R through $R+R$. Proceeding in this manner, we will eventually construct a policy providing decisions for time periods 0 through H, the timespan of the original problem. We call H, the length of the original optimization problem, the study horizon. T, the extent to which we look forward in each subproblem, we call the rolling horizon. R, the length of time during which the decisions computed in each subproblem are implemented, we call the horizon roll. The term rolling horizon arises since at every R time periods we "roll" forward our subproblem horizon of length T by a distance R.

For a more practical example, consider the difficult problem of optimally routing vehicles in a traffic network from 7:00am to 7:00pm. Suppose we instead first solve the problem of optimally routing vehicles from 7:00am to 8:00am, a much smaller problem, and implement these routing decisions from 7am to 7:30am. Then, at 7:30am, suppose we determine optimal routing decisions for the next 60 minutes, given the current state of the traffic network, and implement those decisions for the next 30 minutes. Proceeding in this manner, we may reduce the optimization of 12 hours of traffic into 24 much smaller problems of optimizing only a single hour of traffic at a time. In this case, the study horizon in 720 minutes, the rolling horizon is 60 minutes, and the horizon roll is 30 minutes.

Note that a rolling horizon solution is generally not an optimal solution to a problem. In the vehicle routing example, for instance, myopically looking ahead only one hour at 7am

might cause one to overlook a congestion causing event occurring at 8:15, such as a shift change at a factory, thus leading to the computation suboptimal routings. The extent of this suboptimality for various choices of rolling horizons and horizon rolls is an open research question.

Starting a Simulation with a Loaded Network

In a rolling horizon algorithm, each subproblem begins with the system to be optimized in a state that depends on previously implemented decisions. Hence, to solve a routing subproblem using the SAVaNT algorithm, it is necessary to be able to apply SAVaNT to a loaded network.

Due to the principal of optimality of dynamic programming, the route previously taken by a vehicle will not influence its optimal route from its current position to its destination. No changes are therefore required to the routing program if SAVaNT is applied to a previously loaded network.

The traffic simulator currently employed by SAVaNT, Integration, unfortunately only allows simulations to begin with an empty traffic network. To simulate a loaded traffic network from time *T1* to time *T2* using Integration we actually simulate from time 0 to *T2* using

- a table of traffic flows corresponding to the interval from 0 to *T2*
- a routing table consisting of the routings which produced the loaded network at *T1*, to which is concatenated the routing table for the interval from *T1* to *T2*

The routing table used in the above step may be generated using the subroutine ***mergefiles()*** discussed in the first section of this paper. Note that in practice, implementing a rolling horizon version of SAVaNT using the Integration traffic simulator is extremely time consuming, since rather than simulating a sequence of short time intervals, one must simulate a sequence of ever longer time intervals. In theory, however, one could easily substitute a traffic simulator allowing non-empty initial networks in place of Integration to overcome this limitation.

Integration Technicalities

In the course of developing subroutines for interfacing with Integration, several previously undocumented limitations of the simulator were discovered. First, if we only change the simulation length parameter in the input file **master.fil** to truncate an Integration simulation set up to run for n time periods to run for only $m<n$ time periods, Integration may crash or produce incorrect results. This occurs because travel demands may be scheduled for time periods between m and n. Though Integration recognizes such a condition as an error and notes it in the error file **runerr.out**, it does not discard the invalid vehicle arrivals as one would expect, but rather tries to schedule them, with disastrous consequences.

To accommodate this shortcoming of Integration, the input file listing the rates of travel demand must be pre-processed to eliminate demands occurring after the end of the simulation. We have programmed the application **odgen** to accomplish this task, reading in the desired simulation length from a text file called **simlen**.

The second undocumented limitation of Integration is that each link in the traffic network is assigned 1000 spots in "driveways". If a link is filled, due to congestion, array space is allocated so that up to 1000 vehicles scheduled to begin their trip on the link may wait in a buffer at any one time. Unfortunately, if a 1001st vehicle is scheduled to enter the link, but is

blocked, the error condition is not detected. Rather a 1001st entry will be added to the buffer, leading to memory corruption that may completely crash the system or produce incorrect results. We have implemented error checking code in Integration to eliminate this condition.

Cycling in SAVaNT

The SAVaNT optimization procedure terminates when two consecutive iterations yield identical time dependent link travel times. Consider the case that occurs when no set of link travel times repeats in consecutive iterations of SAVaNT, but such a set of link travel times repeats in two *non-consecutive* iterations. Since the routing program uses these travel times to compute routes that are then employed by the traffic simulator, such an occurrence will lead to infinite cycling.

In [2], Wunderlich demonstrates that sufficient perturbations of link travel times will always eliminate cycling in SAVaNT, at a cost of accuracy. Since inaccurate link travel times are used, the "optimal" solution reported by SAVaNT will not truly be optimal for the traffic network being optimized, but rather for a network which would yield these travel times.

To perturb link travel times, we employ a parameter called *delta*. Suppose a vehicle *V* enters link *l* at time *T1* and reaches the end of the link at time *T2*. We therefore have that the travel time on link *l* for vehicles entering the link at time *T1* is *(T2-T1)*. Instead of recording this data, we record that vehicles entering link *l* at time *T1+delta(T2-T1)* experience travel time *(T2-T1)*. We therefore record experienced link travel times to have occurred at later points in time than when they actually did occur. Accurately reported link travel times correspond to a *delta* value of 0.0. A *delta* value of 1.0 corresponds to probe accounting, whereby *V's* experienced travel time on *l* is recorded to be the travel time to be experienced by a vehicle entering *l* at *T2*, which is in fact *V's* departure time from *l*. By making *delta* large enough, no experienced travel times will be recorded before the study horizon, and free flow link travel times will always be reported to the routing program, thus causing SAVaNT to converge.

Because of the many subproblems to solve in a rolling horizon implementation of SAVaNT and the fact that cycling in any single subproblem will cause the entire optimization procedure to fail, it is highly desirable to allow *delta* to be dynamically incremented to deal with such cycling. In the case of Integration, the subroutine ***changedelta()*** modifies delta to an appropriate value in the simulation control file, **master.fil**, thus allowing such control over *delta* to be easily implemented.

What is a SAVaNT subproblem?

To implement a rolling horizon solution to a problem, one must first be able to decompose the problem into a sequence of subproblems. The definition of such a subproblem in the context of the SAVaNT optimizer is an open research question, with several possible alternatives.

Consider a routing subproblem corresponding to optimizing traffic from time *A* through *B*. What is the appropriate Integration simulation to employ in this case? One alternative is to terminate the simulation at time *B*, and then apply the routing program to optimize the routing tables for the interval from *A* to *B*. A difficulty with this case is that for short roll horizons, this interval will be short, and hence since few vehicles could complete their trips in this interval, end-of-study affects will dominate the computations performed by the routing program.

An alternative to this situation is to allow the simulation to run to some time *H'* after *B*, but to allow no new trips to begin after *B*, and then to apply the routing program to optimize

routes over the interval from *A* to *H'*. For large enough *H'*, the traffic network will always empty, thus eliminating the end-of-study effects discussed in connection with the previous alternative, but introducing new end-of-study effects instead.

Further complications are introduced by the use of *delta*. Recall that a vehicle traversing a link during the interval from *T1* to *T2* will have its travel time on the link recorded as the link travel time experienced by a vehicle entering the link at time *T1+delta(T2-T1)*. In some cases, for positive *delta* values, a link travel time experienced *before* time *A* could be recorded as occurring *after* time *A*. Thus the subproblem of optimizing traffic over the interval from *A* through *B* would not only be influenced by the state of the network at time *A*, but by the routes used by vehicles *before* time *A*. Whether such influence is beneficial or detrimental to solution quality is currently unknown. This issue is particularly significant in the case of Integration, where such situations may occur due to the scheme used to begin a simulation with a loaded traffic network.

Parameters Influencing SAVaNT Performance

Several parameters of a traffic optimization problem may affect the performance of SAVaNT on the problem. These parameters include the time between routing table updates, the time between updates of link travel times, exponential smoothing parameters, the initial state of the traffic network, network loading, and the method of applying and modifying *delta*.

Let us first consider the effect of the length of time between routing table updates. In Integration, routing tables are updated every *tree* seconds. A larger value for *tree* ensures that fewer routing decisions need be made, thus speeding convergence of Integration, at the cost of decreasing the accuracy of the simulation. In the case of Integration, this is doubly true since for each link, only a single experienced travel time is reported for each interval of *tree* seconds. This travel time is currently obtained by exponentially smoothing the sequence of travel times experienced in the interval over the link according to a smoothing parameter. Alternatively one could also use the mean of the link travel times experienced in the interval or even the most recent link travel time experienced. The final effect of *tree* is that when implementing a rolling horizon version of SAVaNT using Integration, both rolling horizon and horizon roll must be in multiples of *tree* seconds to because of the way Integration reports travel times and deals with routing tables.

Rather than computing a new link travel time each time a vehicle enters a link, Integration employs an exponential smoothing procedure to update link travel times every *sample* seconds. The flow of vehicles into the link in the past *sample* seconds is exponentially smoothed with the current estimate of flow on the link to compute a new vehicle flow on the link. This flow is then used to compute a link travel time. Finally, this time is exponentially smoothed with the current link travel time to determine a travel time for the link over the next *sample* seconds. While a smaller value for *sample* improves the accuracy of the simulation, it also entails considerable computational burden.

Currently, the SAVaNT optimization algorithm has only been applied to traffic networks that are initially empty because of the difficulties of simulating previously loaded networks using Integration. This introduces a start-of-study effect which may affect algorithm performance. Using the routing-table merging technique to apply SAVaNT to a previously loaded traffic network is an ongoing research problem.

In addition to its initial loading, the loading of a traffic network throughout the optimization window may significantly affect the performance of SAVaNT. Conditions of light loading cause dynamic routing to yield few benefits over free-flow routing. Another important factor is the level of penetration of intelligent vehicles in the traveling population.

With many vehicles all seeking optimal routings, the gains available to the individual drivers are not as great as if few vehicles were given such routing information.

A final parameter influencing the performance of SAVaNT is the *delta* value used to end cycling conditions. Since an increase in *delta* leads to a more inaccurate simulation, it is desirable to increase *delta* as slowly as possible. Conversely, if *delta* is not increased sufficiently quickly, several increases to *delta* might be required to escape a cycle, with the intervening SAVaNT iterations being wasted. Once a cycle has been escaped and SAVaNT converges, allowing the algorithm to roll forward to the next subproblem, the issue arises as to what value *delta* should be reset. To obtain the most accurate simulations possible, *delta* could be reset to 0, or it could be kept at its current value in anticipation of cycling conditions. Once convergence occurs for a particular value of *delta*, it is also possible to attempt to reduce *delta* as far as possible before cycling occurs in order to improve solution accuracy.

Currently, the use of *delta* causes an experienced link travel time of t time units to be recorded *(delta)(t)* time units after the time the corresponding vehicle entered the link. This scheme allows probe accounting to be easily implemented by using a *delta* value of 1.0, but has the possibly undesirable effect of reporting far more inaccurate travel times for long, congested links than for short, quick links, since reporting delay is proportional to link travel time. Another possible scheme that could be investigated is to report all experienced link travel times as occurring at *delta* time units after the corresponding vehicle enters a link.

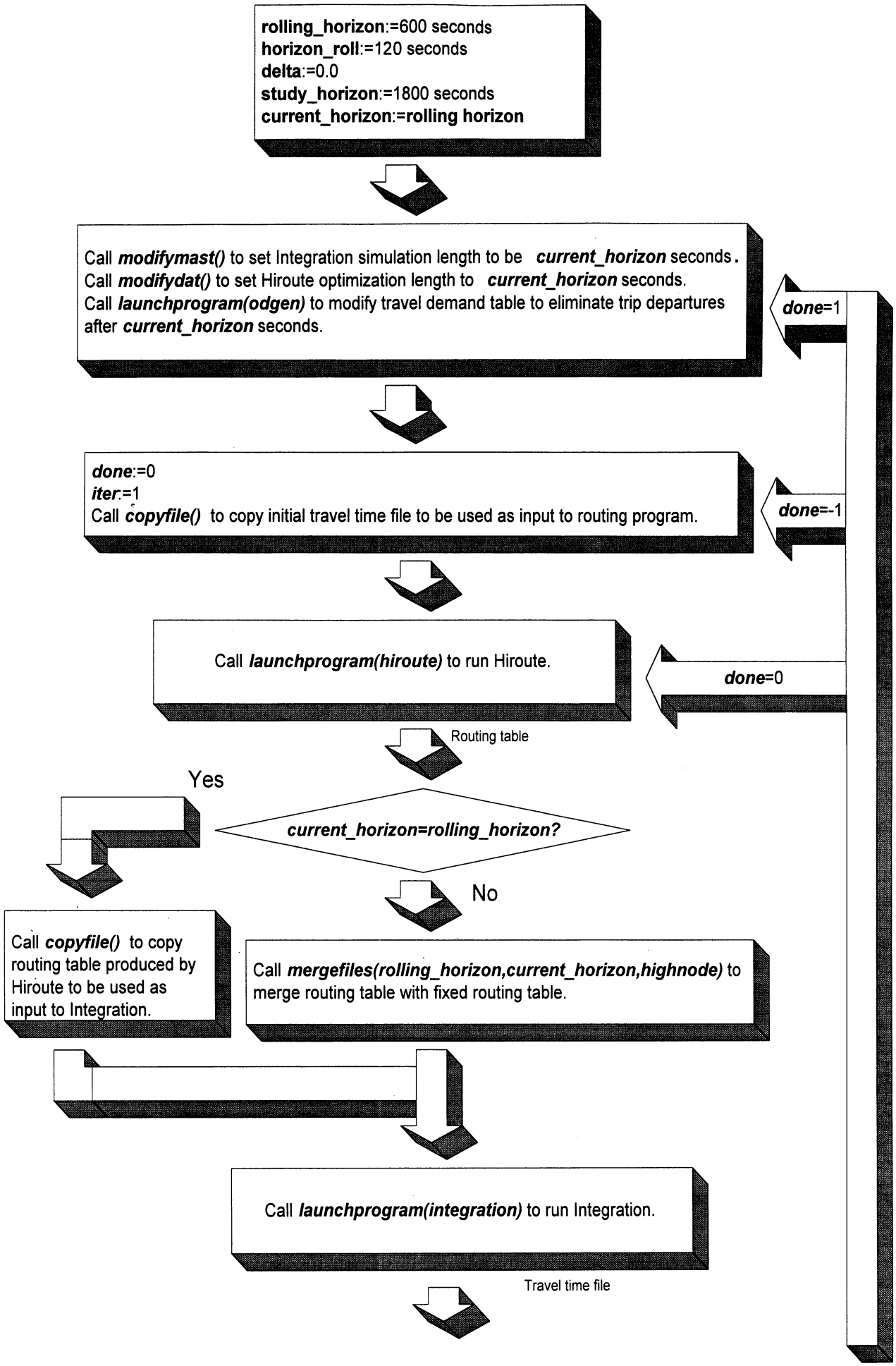
A Rolling Horizon Implementation of SAVaNT

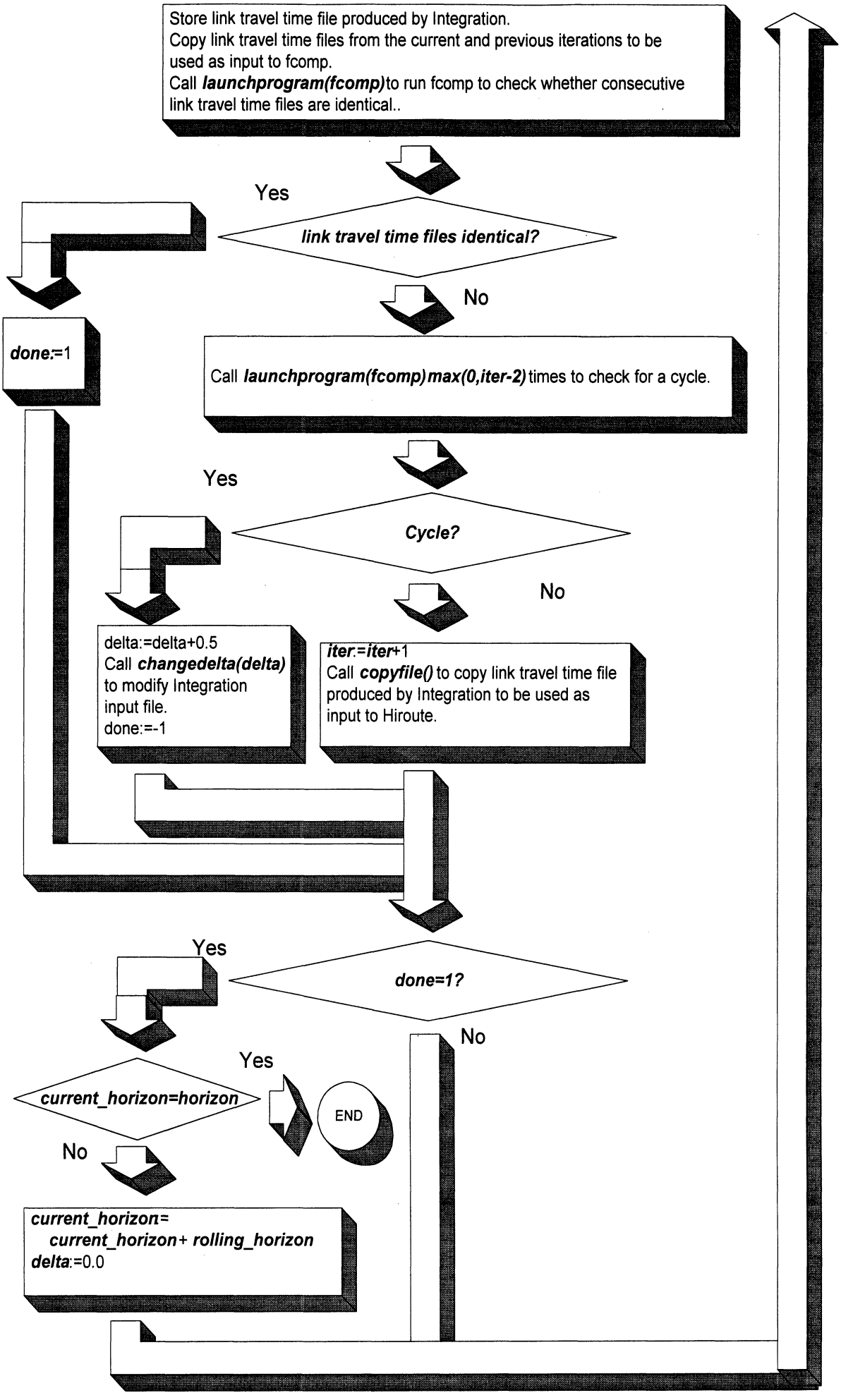
In this section, we present a particular rolling horizon implementation of SAVaNT. This implementation has been applied to the optimization of a model of the Troy Michigan traffic network, whose construction is detailed in [3], over a 30 minute period. We will use the Integration and Hiroute modules this implementation, though other comparable modules could be easily substituted.

Initially, we will assume that the traffic network is empty. We set the parameters of Integration such that routing tables and reported link travel times are updated every 120 seconds, and that link travel times are updated internally every 30 seconds. The default exponential smoothing method is used by Integration to compute link travel times. We will use a rolling horizon of 600 seconds and a horizon roll of 600 seconds.

The value of *delta* is initially set to 0.0, yielding the most accurate link travel times, and will be incremented by 0.5 every time cycling occurs. Once convergence occurs and a subproblem is solved, *delta* will be reduced to back to 0.0. We will allow vehicles traveling in the "fixed" portion of simulations to have their travel times recorded in intervals corresponding to the "variable" portion of the simulation if perturbations by *delta* lead to such circumstances. We define a routing subproblem over the interval from *T1* through *T2* to be such that at time *T2*, Integration will terminate, and only link travel times from time 0 through *T2* will be reported to the routing program.

Using the above problem specification, we obtain the rolling horizon implementation of SAVaNT expressed by the following flowchart:


rolling_horizon:=600 seconds
horizon_roll:=120 seconds
delta:=0.0
study_horizon:=1800 seconds
current_horizon:=rolling horizon
Call modifymast() to set Integration simulation length to be current_horizon seconds.
Call modifydat() to set Hiroute optimization length to current_horizon seconds.
Call launchprogram(odgen) to modify travel demand table to eliminate trip departures after current_horizon seconds.
done=1
done:=0
iter:=1
Call copyfile() to copy initial travel time file to be used as input to routing program.
done=-1
Call launchprogram(hiroute) to run Hiroute.
done=0
Routing table
Yes
current_horizon=rolling_horizon?
No
Call copyfile() to copy routing table produced by Hiroute to be used as input to Integration.
Call mergefiles(rolling_horizon,current_horizon,highnode) to merge routing table with fixed routing table.
Call launchprogram(integration) to run Integration.
Travel time file


Store link travel time file produced by Integration.
Copy link travel time files from the current and previous iterations to be used as input to fcomp.
Call launchprogram(fcomp)to run fcomp to check whether consecutive link travel time files are identical..
Yes
link travel time files identical?
No
done:=1
Call launchprogram(fcomp)max(0,iter-2) times to check for a cycle.
Yes
Cycle?
No
delta:=delta+0.5
Call changedelta(delta) to modify Integration input file.
done:=-1
iter:=iter+1
Call copyfile() to copy link travel time file produced by Integration to be used as input to Hiroute.
Yes
done=1?
No
Yes
current_horizon=horizon
END
No
current_horizon=
current_horizon+ rolling_horizon
delta:=0.0

Conclusions

We have developed a modular, extendible framework of software tools for the construction of traffic optimization algorithms. Using these tools, we have created a rolling horizon implementation the route guidance algorithm SAVaNT. In the course of the development of this algorithm, we have identified many parameters and variations of the SAVaNT algorithm which we plan to examine both theoretically and through empirical testing in order to improve both the speed and accuracy of the algorithm.

References

[1] Kaufman, D. E., Smith, R. L., and Wunderlich, K. E., 1992. Dynamic User-Equilibrium Properties of Fixed-Points in Iterative Routing/Assignment Methods. IVHS Technical Report 92-12, University of Michigan.

[2] Wunderlich, K. E., 1994. Link Travel Time Prediction Methods and Convergence for Iterative Anticipatory Route Guidance Methods.

[3] Wunderlich, K. E., and Smith, R. L., 1992. Large Scale Traffic Modeling for Route-Guidance Evaluation: A Case Study. IVHS Technical Report 92-08, University of Michigan.