Title

Implementing a Kalman Filtering Dynamic O-D Algorithm within Paramics- Analysing Quadstone Won Efforts for the Dynamic O-D Estimation Problem

Author

Garcia, Reinaldo C.

Publication Date

2003-05-01

CALIFORNIA PATH PROGRAM
INSTITUTE OF TRANSPORTATION STUDIES
UNIVERSITY OF CALIFORNIA, BERKELEY

Implementing a Kalman Filtering Dynamic O-D Algorithm within Paramics–Analysing Quadstone Won Efforts for the Dynamic O-D Estimation Problem

Reinaldo C. Garcia

California PATH Working Paper
UCB-ITS-PWP-2003-8

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

Report for Task Order 4130

May 2003

ISSN 1055-1417

IMPLEMENTING A KALMAN FILTERING DYNAMIC O-D ESTIMATION ALGORITHM WITHIN PARAMICS — ANALYSING QUADSTONE OWN EFFORTS FOR THE DYNAMIC O-D ESTIMATION PROBLEM

Reinaldo C. Garcia
Institute of Transportation Studies
PATH Research Program
Univeristy of California at Irvine
Irvine, 94720 CA

February 19, 2002

Executive Summary

This Project moves forward from the accumulated knowledge garnered from MOU 4121. Over the last year a good deal of insights was obtained about the implementation of dynamic Origin-and-Destination (O-D) estimation. Under MOU 4121, the Kalman Filtering (KF) algorithm described in Hu et al. (2000) for dynamic O-D estimation was implemented. In this research, the implemented algorithm is incorporated to the microscopic traffic simulator Paramics.

Paramics offers important and unprecedented features, such as high performance and scalability, to handle realistic real world traffic networks under ITS (Intelligent Transportation Systems). Nevertheless, Paramics has its own limitations, particularly relating to the model s ability to interface with dynamic routing protocols, and dynamic O-D estimation.

Hu et al. (2000) implemented their KF algorithm for dynmic O-D estimation and tested it for a freeway system. Needing the KF algorithm inputs such as link traffic counts and assignment matrices to be applied, Hu et al. (2000) used the DYNASMART mesoscopic traffic simulator to obtain the necessary input data for the KF algorithm.

In this research, being Caltrans (The California Transportation Department) the main sponsor of this project, the implementation of the algorithm was made applying *solely* the Paramics traffic simulator (not using therefore the DYNASMART simulator). Doing so, it was intended to avoid future training costs to Caltrans, as it is already applying mainly Paramics through its own districts. To implement the KF algorithm described in Hu et al. (2000) within Paramics, the following API s (Advanced Programming Interfaces) were developed: net_action, vehicle_action, vehicle_link_action and net_post_action. These API s will be explained in detail in this report. Furthermore, this research will discuss the own proposed development of Quadstone (the developer of Paramics) for the dynamic O-D estimation algorithm.

Introduction

This work builds on the contribution of MOU 4121, which focus on the methodological of the dynamic O-D estimation. The important contributions of this work are not only in incorporating a dynamic O-D estimation algorithm but also in evaluating its implementation and, helping Quadstone with its own dynamic O-D estimation approach. The problem of dynamic O-D estimation and prediction of O-D has received increasing attention in recent years because its applicability to on-line traffic management systems. It is important to say that time-dependent O-D matrices are essential input for advanced traffic management and information systems. In MOU 4121 before applying any dynamic O-D algorithm, a series of different dynamic O-D approaches were studied. The chosen dynamic O-D algorithm is based on the development by Hu et al. (2000).

Several approaches have been investigated for estimation and prediction of dynamic OD matrices. They can be categorized into two families: statistical inference and state space models. A large number of statistical inference methods in this field have already been proposed (for an extensive description of them see the report for MOU 4121). The objective of state estimation is to track variables that characterize some dynamic behavior, by processing observations afflicted by errors. State estimators rely on a model to relate the state variables to each other, to the observations, and to the forcing (see Norton (1986)).The Kalman Filtering (KF) algorithm is the most typical state space model that has been intensively investigated. Previous studies applying the KF algorithm were already described in detail in MOU 4121 (see Cremer and Keller (1987), Nihan and Davis (1987), Okutani (1987), Ashok and Ben-Akiva (1993), Chang and Wu (1994), Zijpp and Hamerslag (1994), Wu and Chang (1995)). In this research a KF algorithm developed by Hu et al. (2000) is implemented.

Methodology

A KF based dynamic freeway O-D estimation and prediction algorithm with route switching and time-varying traffic characteristic considerations is presented in Hu et al.(2000) and applied in this work. The model applied extends previous work in the field (see Ashok and Ben-Akiva (1993)) by incorporating dynamic traffic characteristics and a behavioral component within the KF algorithm.

Time-dependent O-D distributions are the outcome of motorists trip plans, given network traffic conditions and users specific characteristics. For on-line traffic management these O-D matrices can be obtained by using the information contained inexisting O-D data and link traffic counts measured at each time interval. Those O-D data are estimates of travel demands during previous time intervals and, therefore, are associated with random errors in a dynamic system. Link traffic counts, on the other hand, suffer from measurement errors as a result of technological limitations, data processing errors, and so forth. A dynamic O-D estimation and prediction model that predicts time-varying O-D demands from these two sources of information should recognize the uncertainty inherent in the dynamic traffic system and provide flexibility to accommodate different degrees of error. To specify the problem the following variables are defined:

$q_k(i)$ = the number of vehicles entering the freeway from on-ramp i during time interval k, $i=1,2,\ ,N-1$;

$y_k(j)$ = the number of vehicles leaving the freeway from off-ramp j during time interval k, $j=2,3,\ N$;

$m_k(l)$ = the number of vehicles crossing the freeway mainline segment l during time interval k, $l=2,3,\ ,N-1$;

$x_k(i,j)$ = the number of vehicles crossing the entering the freeway from on-ramp i during time interval k that are destined to measurement location j;

$a_k^r(i,j,d)$ = the fraction of $x_r(i,j)$ that arrives at its destination during time interval k, $r=1,2,\ k$.

The state variables to be estimated are the time-dependent $x_k(i,j)$ O-D flows. In view of platoon dispersion, the assignment fraction variable $a_k^r(i,j,d)$ capture the temporal relationship between the time-dependent O-D flows and the observed link traffic counts. The aim of the algorithm is to find an estimate of $x_k(i,j)$ using measurement link traffic counts, represented by

$z_k(i,j)$. It can be shown that $z_k(i,j)$ depends on $q_k(i)$, $y_k(j)$ and $m_k(l)$ (see MOU 4121 and Hu et al.(2000)).

Solution of the Kalman Filtering Problem

Given the initial conditions X_0 and P_0 (the covariance of X_0), the Kalman filter recursively estimates the state variables by the following equations:

$$\Sigma_{k|k-1} = F_k \Sigma_{k-1|k-1} F_k^T + Q_k \quad (1)$$

$$K_k = \Sigma_{k|k-1} A_k^T [A_k \Sigma_{k|k-1} A_k^T + R_k]^{-1} \quad (2)$$

$$\Sigma_{k|k} = \Sigma_{k|k-1} - K_k A_k \Sigma_{k|k-1} \quad (3)$$

$$X_{k|k-1} = F_k X_{k-1|k-1} \quad (4)$$

$$X_{k|k} = X_{k|k-1} + K_k [z_k - X_{k|k-1}] \quad 5)$$

where: - $X_{k|k-1}$ is the ns-vector of augmented one-step ahead O-D flow predictions, and $X_{k|k}$ is the ns-vector of augmented filtered O-D flow estimates.

- F_k and A_k are the augmented transition and assignment coefficient matrices, respectively.
- K_k is the Kalman filtering gain matrix
- $\Sigma_{k|k}$ and $\Sigma_{k|k-1}$ are the covariance matrices of the state estimators $X_{k|k}$ and $X_{k|k-1}$ respectively and,
- Q_k and R_k are covariance matrices for the individually noise and Gaussian processes related to the variables $x_k(i,j)$ and $z_k(i,j)$ respectively.

One of the new developments exploited by this project is the implementation of the chosen algorithm for a general network. Hu et al. (2000) applied their algorithm for just a stretch (6 mile) of a freeway[1]. To understand one of the main reasons why Hu et al.(2000) applied their algorithm for just a stretch of a freeway, let s look at the assignment matrix A_k.

The matrix A_k has dimensions $ns\ X\ ns$ where n is a vector of O-D flows pairs and, $s=max\{p,u\}$, where p is the order of the transition equation and, u is the maximum number of time intervals required to travel between any O-D pair of the entire network. Therefore, if the

network has 10 origins and 10 destinations, there will be a total of 100 O-D s pairs (i.e. $n=100$). Besides that, if $p=3$ (the order of the transition equation) and, if a commuter takes 1 hour (60 minutes) to arrive at her/his destination and, the time interval applied is 5 minutes, then $u=12$ (as $u=60/5=12$ is the number of time intervals to traverse the network), making s=12 (i.e., $s=max\{u,p\}$=max{12,3}=12).

One can then see that from this simple example given above that the dimension of the matrix A_k is 1,200 (i.e., ns X ns = 1,200 X 1,200). Thus, A_k will have a total of 1,440,000 (!!) elements to be stored ! During the application of the algorithm not only the matrix A_k has to be stored, meaning that the problem can become computationally unfeasible (there won t be enough memory to store all the necessary elements to apply the implemented algorithm).

Remark: Prior to the start of this research the option considered were either apply methods (such as the Least Squares one) that could be easily implemented but would give very poor results or, implement new methods (such as the KF algorithm) and, afterwards try to come up with ways to make the KF algorithm computationally feasible. The implementation of the KF algorithm was the chosen as recent research published in the literature obtained promising results with its application.

Development of the Advanced Programming Interfaces (API s)

This section gives a brief description of the API s developed for this research (a more detailed description is given in Appendix A). To implement the chosen KF algorithm within Paramics, the following API s were developed: api_setup, net_action, vehicle_action, vehicle_link_action and, net_post_action. A brief description of them is given next.

The API api_setup is called during the initialization phase, before the simulation has started. It allows us to insert initialization routines, and basically it explains the objectives of the program.

In the API net_action some variables, such as the total number of vehicles in the network (G_n_vehicles_in_the_network), the number of origins and destinations and, the flows in the links, are defined and initialized. These variables will be in general used in the rest of the program to obtain other variables (matrices), such as the assignment matrix A_k.

---

[1] When the implementation of the algorithm was initiated, a personal communication between this author and Shou-Ren Hu was exchanged. Shou-Ren called the attention that extra care for the memory allocation issue should be given, as the algorithm was computationally very demanding.

The API vehicle_link_action is applied mainly to obtain the value of s, i.e., $s=max\{u,p\}$ where, as already explained previously, p is the order of the transition equation and, u is the number of time intervals required to travel between any O-D pair of the entire network.

The API vehicle_action is applied mainly to obtain the assignment matrix A_k. Therefore, the *origin*, *destination* and *cost* for each vehicle to traverse the network is obtained here.

The API net_post_action implements the KF algorithm. The previous API s obtained the necessary data to serve as input for the algorithm s application. At the end of this API net_post_action, the new O-D table is obtained and, written into the Paramics format.

The next section describes the results obtained when applying the KF algorithm within Paramics.

Obtaining and Analyzing the Results of the KF algorithm within Paramics

One of the main difficulties with the implementation of the KF algorithm was the management of memory allocation. Even tough extra care was given during the KF algorithm s implementation[2], the program as it stands today can handle up to 2,500 vehicles in the network. This total number of vehicles depends on the complexity of the network. Thus, if the network has more (less) pairs of O-D s or more (less) number of links, means that less (more) vehicles can be handled by the program[3]. One can argue that this short number of vehicles could make the program not very useful for real world applications. Therefore, the following idea was implemented to make the program more suitable for the real world issues.

The question to be asked is: How it is possible to make a program that can handle up to 2,500 vehicles be applied to *mirror* real world networks, which have hundreds of thousands of vehicles? Well, one can suppose that the 2,500 vehicles represent a sample of the network and, once the *new* O-D is obtained based on these 2,500 vehicles, a rescale is applied obtaining the *real new* O-D table[4].

---

[2] During the implementation of the KF algorithm, whenever variables (matrices) were created, applied and, then would not be anymore used in the simulation, they were deleted from the program to save memory.

[3] The number of vehicles handled by the program can also be increased applying a more powerful computer. The computer applied for this research was a Pentium III 350 Mhz with 128 Mb. Of RAM memory.

[4] One of form of scaling can be understood as: suppose that there are at the moment 10,000 vehicles at this moment in the network. As the program can handle up to 2,500 vehicles, ach element of the new O-D should be rescaled by a actor of 4 (i.e., 10,000/2,500=4) and the new real O-D would then have been obtained.

To test the approach developed in this research, the network showed in Figure 1 was used. This applied network has about 64 pairs of O-D s, 152 links and its initial O-D matrix is given in Table 1.

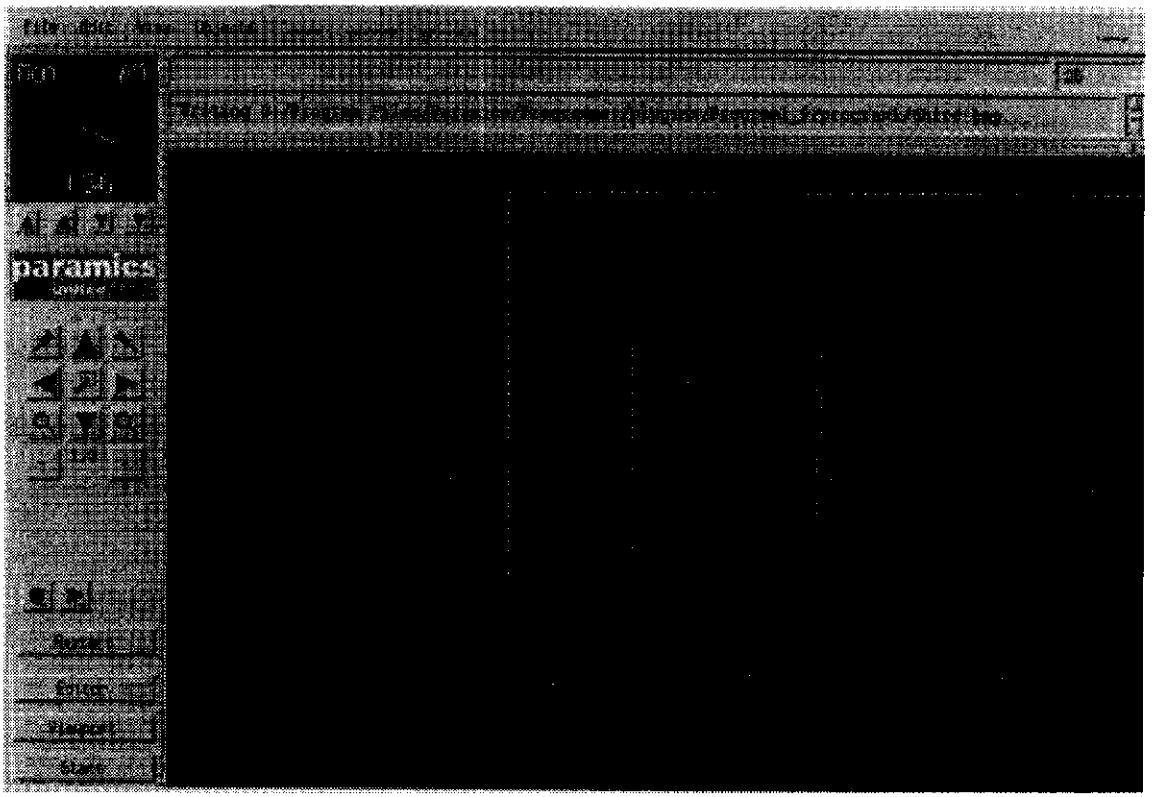

Figure 1 — Description of the network applied in this research.

from 1	**0**	**3150**	**156**	**129**	**64**	**103**	**411**	**3150**	**## 7163**
from 2	**4267**	**0**	**218**	**181**	**89**	**144**	**566**	**896**	**## 6361**
from 3	**373**	**339**	**0**	**16**	**8**	**13**	**50**	**78**	**## 877**
from 4	**320**	**291**	**16**	**0**	**7**	**11**	**42**	**67**	**## 754**
from 5	**213**	**194**	**11**	**9**	**0**	**7**	**28**	**45**	**## 507**
from 6	**133**	**121**	**7**	**6**	**3**	**0**	**18**	**28**	**## 316**
from 7	**533**	**485**	**27**	**23**	**11**	**18**	**0**	**112**	**## 1209**
from 8	**5**	**5**	**0**	**0**	**0**	**0**	**1**	**0**	**## 11**
## total	**5844**	**4585**	**435**	**364**	**182**	**296**	**1116**	**4376**	**## 17198**

Table 1 — O-D Matrix applied at the start of the simulation.

The results obtained applying the KF algorithm are given in Table 2. Comparing Tables 1 and 2, the results agree quite well in magnitude. It must be added that in previous research applying other methods (such as the Least Squares, LS, method) to estimate and predict dynamic O-D tables, the results could differ by as much as 5,000% ! Similarly, when comparing the results stated in Tables 1 and 2, one can see that the most of the cases differ by up to 25% (the exceptions are the results in the last row, when comparing 5 to 8, i.e. 60%; but as those numbers are so small, it basically does not affect the congestion of the network, for instance).

from 1	**0**	**2464**	**124**	**104**	**52**	**84**	**332**	**2338**	**## 5498**
from 2	**2708**	**0**	**160**	**132**	**63**	**103**	**629**	**1010**	**## 4805**
from 3	**272**	**411**	**0**	**36**	**18**	**27**	**94**	**127**	**## 985**
from 4	**228**	**221**	**41**	**0**	**16**	**25**	**79**	**103**	**## 713**
from 5	**150**	**148**	**30**	**26**	**0**	**0**	**54**	**68**	**## 496**
from 6	**91**	**91**	**20**	**17**	**10**	**0**	**33**	**47**	**## 309**
from 7	**607**	**529**	**52**	**44**	**20**	**134**	**0**	**173**	**## 1459**
from 8	**8**	**8**	**0**	**0**	**0**	**0**	**3**	**0**	**## 19**
## total	**4064**	**3872**	**427**	**359**	**179**	**293**	**124**	**3866**	**## 14284**

Table 2 — O-D Matrix obtained with the application of the KF algorithm within Paramics.

Finally, it can be concluded that the results obtained applying the KF algorithm look quite promising. Once real-time data is acquired through the ongoing research in the PATH program, more tests should be realized to further validate the algorithm. Nevertheless, as far as it is known, this project is the first one to apply a KF algorithm to the dynamic O-D estimation problem in a *network*. And once again, it is quite exciting the results when comparing Tables 1 and 2. Different types of networks (more complex ones) must also be studied to further understand the capacity of the algorithm. The next section discusses the own approach suggested

by Quadstone (the developer of Paramics) to implement its own dynamic O-D estimation procedure.

Quadstone Approach to Develop a Dynamic O-D Estimation Procedure

Quadstone has proposed to develop its own dynamic O-D estimation procedure called Udimode (User-Directed Interactive Method for O-D Estimation). The main idea behind the Udimode approach is that the user will change parameters/weights of the links, intersections, on-ramps, off-ramps, etc. and will then obtain a reasonable solution for the O-D estimation table.

Studying more carefully the Quadstone approach, it can be concluded that it can (will) be extremely time consuming for the user to calibrate just a small network. Among the many parameters that the user will have to deal with are the following ones:

(i) Traffic counts on links, and confidence weights for each;

(ii) Turn counts and confidence weights for each;

(iii) Pattern seed matrix — this is assumed to be unit-less, but a previous matrix can be used. A confidence interval may be supplied for each trip count !!

To better understand how time consuming it can be for the user to calibrate a simple network, let s look only at case (iii) stated above. Let s suppose that for case (iii) there is a 10 mile section of a freeway, with on-ramps and off-ramps spaced one mile apart from each other. Therefore, there will be about 10 possible origins and 10 possible destinations, for this case. Then, for this 10 by 10 O-D matrix, there will be about 100 possible weights and, the existence of any arterial was not even considered here. Continuing this analysis, if the user had been able to choose some parameters (after a lot of hard work) that could describe the demands of 60 of those O-D s pairs, there is no guarantee that once the user starts to change again the already chosen parameters (to try to match the other 40 O-D pairs of demands), repeating, there is no guarantee that the new number of good matches will become worse than 60 (once the new parameters have been chosen). Basically, the user would be walking backwards and not forward to the solution of the problem.

It can be concluded that once the whole Udimode project has been implemented, it will demand a lot of work hours just to calibrate a small network, for just one period of time. Then there will be another problem.

As this approach intends to be extended for dynamic O-D estimation, the implementation for different periods of time will probably demand new calibration. Therefore, a whole bunch of new parameters for a different period of time will have to be obtained by the user, demanding again many hours of work (calibration). Once again, there is no guarantee that the process of calibration can ever be concluded.

Quadstone has suggested that its own implementation shall be sufficiently flexible to allow extensions to provide dynamic capability. The main problem is that with dynamic O-D estimation and prediction, during different periods of time of the day, there are different patterns of the distribution of the data. Thus, early in the morning more people are going to work and, a little bit later, some of the families are taking their children to schools or go shopping, leisure and so on. Therefore, if Quadstone s approach is implemented, once the user has calibrated the model, let s suppose, for the first period of the day (working commute), and the new O-D s are being generated, when the next period arrives (shopping, leisure), the new O-D s to be generated should not be generated by the same parameters that the user obtained when calibrating the model for the 1st. period. Some other issues, as described next, can continue to challenge Quadstone own development for the dynamic O-D estimation problem.

Let s suppose now that the user calibrated the model when no incident happened in the newtork. For this case, the user would have chosen some parameters and weights that would perceive the real world behavior. What happens if some kind of incidents (accidents, close of off-ramps) occur !? By Quadstone s approach, to perceive the behavior of the real world, the user will/would have to calibrate the whole model again (!!).

Finally, it can be said that Quadstone approach of basically trying to achieve the estimation of new O-D s by allowing the user to choose weights, parameters, etc. can be extremely dangerous and, most probably, so time consuming that the real O-D s (or at least values close to the real ones) will never be achieved. As the name suggests Udimode (User-Directed Interactive Method for O-D Estimation), it should be told to Quadstone to try to come

up with a more automatic way to obtain the new O-D s matrices, and not leaving the main calibration task to the user.

CONCLUSIONS

This project has implemented an adaptive Kalman filtering algorithm for the dynamic and prediction of a network O-D matrices within the traffic simulator Paramics. As far as it is known, it is one of the first researches trying to implement a KF algorithm for a network. To make possible the implementation of the KF algorithm within Paramics, API s had to be developed.

The problem of implementing the algorithm and developing API s requires the storage of some variables (matrices) which can have huge dimensions. To make the problem computationally feasible, an approach of obtaining an O-D table through a sample of the number of vehicles present in the network was developed. The results obtained with this research look quite promising.

Further tests applying the procedure here presented should be made to validate the described procedure. Tests such as looking at different types of networks, with different complexities and, making tests such as when incidents occur or not in the network, would be helpful to further validate the procedure. Once real-time data is also obtained through the ongoing research in the PATH program, the KF algorithm here presented could then be tested with real-time data. It is hoped that this project can be a valuable tool to Caltrans and to the PATH program in their ongoing effort to expand not only the Paramics capabilities but also any other future applications .

REFERENCES

Anderson, B.D.O. and Moore, J.B. (1979) *Optimal Filtering*, Prentice Hall, New Jersey.

Ashok, K. and Ben-Akiva, M.E. (1993) *Dynamic Origin-Destination matrix estimation for real-time traffic management systems*. In Transportation and Traffic Theory (C.F. Daganzo, ed.), Elsevier Science Publishers B.V., 465-484.

Bell, M.G.H. (1991) *The real time estimation of origin-destination flows in the presence of platoon dispersion*, Transportation Research, Part B, Vol. 25, No. 2, 1157-125.

Ben-Akiva, M.E., Ashok, K. and Yang Q. (1994), *Commentaries on a statistical analysis of the reliability of using RGS vehicle probes as estimators of dynamic O-D departure rates*, IVHS Journal, Vol. 2, No. 1, 21-44.

Chang, G.L. and Wu, J. (1994) *Recursive estimation of time-varying Origin-Destination flows from traffic counts in freeway corridors*, Transportation Research, Part B, Vol. 28, No. 2, 141-160.

Cremer, M. and Keller, H. (1987) *A new class of dynamic methods for the identification of Origin-Destination flows*, Transportation Research, Part B, Vol. 21, No. 2, 117-132.

Daganzo, C. (1997), *Fundamentals of Transportation and Traffic Operations*, Elsevier Science Ltd., Great Britain.

Davis, G. A. (1993) *Estimating freeway demand patterns under impact of ncertatinty on ramp controls*, ASCE Journal of Transportation Engineering, Vol. 119, No. 4, 489-503.

Gartner, N.H. (1982) *OPAC: A demand-responsive strategy for traffic signal control*, Transportation Research Record, **906**, 75-81.

Gartner, N.H. (1983) *Simulation study of OPAC: A demand-responsive strategy for traffic signal control*. In Gartner N.H. and Wilson N.H. (eds.), Transportation and Traffic Theory (pp. 233-250). New York: Elsevier Science Publishing Company.

Hellinga, B. and Van Aerde, M. (994) *A statistical analysis of the reliability of using RGS vehicle probes as estimators of dynamic O-D departure rates*, IVHS Journal, Vol. 2, No. 1, 21-44.

Hu, S.R., Madanat, S.M., Krogmeier, J.V. and Peeta, S. (2000), *Estimation of dynamic assignment matrices and OD demands using adaptive Kalman Filtering*, Working paper, 2000 (to appear in the ITS Journal).

Maher, M. J. *Inferences on trip matrices from observations on link volumes: a Bayesian statistical approach*, Transportation Research, Part B, Vol. 17, No. 6, 435-447.

Mahmassani, H.S. and Liu, Yu-Hsiu (1999), *Dynamics of commuting decision behavior under advanced traveler information systems, Transportation. Research*, **7C**, 91-107

Norton, J.P. (1986) *An introduction to identification*, Academic Press, Inc. London, U.K..

Okutani, I. (1987) *The Kalman filtering approach in some transportation and traffic problems*. In International Symposium on Transportation and Traffic Theory (N.H. Gartner and N.H.M. Wilson, eds.), Elsevier Science Publishing Company, Inc., 397-416.

Peeta, S. (1994), *System optimal dynamic traffic assignment in congested networks with advanced information systems*, Doctoral Dissertation, The University of Texas at Austin.

Peeta, S. and Mahmassani, H.S. (1995), *Multiple user-class real-time traffic assignment for online operations: a rolling horizon solution framework*, Transportation. Research, **3C**, 83-98.

Nihan, N.L. and Davis, G.A. (1987) *Recursive estimation of Origin-Destination matrices from Input/Output counts*, Transportation Research, Part B, Vol. 21, No. 2, 149-163.

Nihan, N.L. and Davis, G.A. (1989) *Application pf prediction-error minimization and maximum likelihood to estimate intersection O-D matrices from traffic counts*, Transportation Science, Vol. 23, No. 2, 77-90.

Flannery, B.P., Press, W.H., Teukolsky, S.A. and Vettering, W.T. (1994) *Numerical recipes in C, the art of scientific computing*, Second Edition, Cambridge University Press, Cambridge, England.

Zijpp, N.J. and Hamerslag, R. (1994) *An improved Kalman filtering approach to estimate Origin-Destination matrices for freeway corridors*. Presented at 73rd. Annual Meeting of the Transportation Research Board, Washington, D.C.

Wu, J. and Chang, G.L. (1995) *Estimation of time-varying O-D matrices with dynamic screenline flows*. Presented at 74th. Annual Meeting of the Transportation Research Board, Washington, D.C.

Appendix A

This appendix describes more technical details about the programming of the KF algorithm. As it was already said, the following API s were developed during this project: api_setup, net_action, vehicle_action, vehicle_link_action and, net_post_action. After describing a little bit more in detail these API s, this Appendix gives some hints about how to extend the program in the future and (or) how to look for bugs if unfortunately those still appear.

The api_setup

As it is described in the Paramics manual, this API is called during the initialization phase, before the simulation has started. It allows us to insert initialization routines, and basically explains the objectives of the program. In our case, it is printed that the following program implements a KF algorithm for the dynamic O-D estimation problem.

The API net_action

In this API some variables, such as: the total number of vehicles in the network (*G_n_vehicles_in_the_network*), the number of origins and the number of the destinations in the network (*N_of_origins* and *N_of_destinations*), the number of links (*Number_of_links*, obtained through the *net_n_links* function of Paramics), the flows in the links (*Z_flowlinks*), are defined and initialized. These variables will be in general used in the other API s of the program to obtain other variables, such as the assignment matrix A_k (represented in the program by the variable *a_ptr* in the program). It could be also said that there is no need to have two different variables for the number of origins and the number of destinations, as usually they are identical.

Nevertheless, two variables were used in the program because if in the future it appears such a strange network where this case of equality between the origins and destinations does not hold, there will be no problem with the program implementation.

The API vehicle_link_action

This API is very important as it obtains the value of s (i.e., $s=max\{u,p\}$ where, as already explained previously, p is the order of the transition equation and, u is the number of time intervals required to travel between any O-D pair of the entire network). The way that the API works is very straightforward, as it starts with the value of 2, and whenever a vehicle has higher *cost* to traverse the network, the value of s is updated. It must be reminded that the cost of the vehicle to traverse the network can be obtained through the Paramics function *link_destination_cost*, having as inputs the current link of the vehicle, its current destination and its current route to arrive at the destination.

The API vehicle_action

This API is applied mainly as support to obtain the assignment matrix A_k. Therefore, the *origin*, *destination* and *cost* for each vehicle to traverse the network is obtained and stored here. One has to remember that the assignment matrix has dimensions of not only the number of O-D s pairs but also of the cost of each vehicle to traverse the network. Therefore, if two vehicles have the *same* origins and destinations but different costs, they can (will) be in different positions of the assignment matrix. One should imagine the assignment matrix as looking like: 00_0, 00_1, 00_2 (as being the elements representing vehicles which are going from origin 0 to destination 0 and are going to take 0 interval of time or, 1 interval of time or, 2 intervals of time, respectively, to traverse the network; as one can see, as Paramics does not allow a vehicle go from one origin to that same destination, destination 0, all these elements will be 0) , and also 01_0, 01_1, 01_2 (as being the elements representing vehicles which are going from origin 0 to destination 1 and are going to take 0 interval of time or, 1 interval of time or, 2 intervals of time, respectively, to traverse the network) and so on.

The API net_post_action

One can say that this is the main API of the Program developed. In this net_post_action API, the KF algorithm is implemented. First many variables were defined and left in the program even after its successful implementation. This decision was taken as many of those variables are applied to write the results either in the files or in the screen. One has to remember that in the future, the program can (will) be applied for more complexes networks and, as Paramics is a terrible software regarding its debugging abilities, the only way to debugger the program is printing the values of the variables.

After the definition of a whole group of variables, the assignment matrix (A_k) and the link counts (the variables z_k) are obtained. The values of those variables are found by the values of the origins, destinations and costs for each vehicle. Once the variables A_k and z_k are obtained, the application of the algorithm is a little bit straightforward, applying Eqs.(1) trough (5) of this report. Of course that before obtaining the multiplication (addition) of each equation, a new variable (matrix) having the correct dimensions has to be defined to store the result of the multiplication (addition). At the true, it can be said that the implementation is straightforward until Eq. (2), where there is the need to invert a matrix (i.e., the matrix that results from some manipulations). Before inverting this matrix, its determinant is obtained in the program and, if and only if the matrix is not singular (i.e., its determinant is different of zero), the inversion of the matrix is realized. Both functions to obtain the determinant and to invert the matrix were implemented applying the Gauss Jordan algorithm presented in the book *Numerical Recipes in C(1994).*

Finally, after the new values of the time-dependent state variables $x_k(i,j)$ O-D flows are found, the new O-D demand can be obtained. Then the rescale of the O-D matrix as already described in this report is realized and, the matrix is written in the O-D format as requested by the Paramics simulation package.

Troubleshooting for the Program

Even tough extra care was given to make the program bug free, it is impossible to say that there will be no problems in the future. Therefore, with the experience obtained during the implementation of the algorithm, some insights about where to look at possible problems are given in this section.

The first issue is the problem of: if after running the algorithm, very *strange* results are obtained to the O-D table. These strange results can be either very large numbers or even negative

numbers (!!). If that happens, almost sure the reason is that the capacity of the memory of the program has been achieved, meaning that during the creation of the pointers actually, no pointer is being created anymore. Two possible solutions can be tried for this case. The first is to increase the value of the variable time_step_matrix (defined at the beginning of the program, just over the API set_up). This time_step_matrix controls the cost for each vehicle in seconds (i.e., a time step of 10 means 10 seconds, of 300 means 5 minutes, of 600 means 10 minutes). Increasing the value of the variable time_step_matrix will decrease the dimension of the assignment matrix, but the trade off is that the results can decrease in accuracy[5]. The second possible solution is to decrease the value of the variable G_total_number_of_vehicle_in_the_network. In the beginning of the program, the value of this variable is defined as about 2,500 vehicles, as this is the total number of vehicles that could be stored in our particular example. Therefore, if we decrease the value of G_total_number_of_vehicle_in_the_network, a smaller number of variables will be created (such as the ones that store the origins and destinations of each vehicle) and more memory can be freed. But, once again, the reliability of the results will decrease, as a smaller sample of vehicles will then be used to estimate the real new O-D table.

Finally, inside the program, there are many portions of comments that were left to either help the user to print the values of specific variables to see if there are any problems with them. The comments can also be used to further understand which part of the algorithm implementation is being evaluated at that point. It is hoped that the comments here presented can be helpful for future users of the Program implementated. As it has already been said, extra care was given to make the program bug free, but in any software implementation problems can always come up when new testing or, networks or, cases are tried.

[5] The point here is that making time_step_value=600 (i.e., 10 minutes, for instance) it means for example that all the vehicles that will take between 0 and 10 minutes to arrive at their destination, will have the same cost. Basically, there is greater aggregation of the data when the value of time_step_value is increased.