Proceedings of the 1992 Winter Simulation Conference
ed. J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson

REPRESENTATION OF USER TRANSACTION PROCESSING BEHAVIOR WITH A STATE TRANSITION MATRIX

William S. Keezer
Andrew P. Fenic

Mead Data Central, Inc.
Systems Evolution and Modeling
P. O. Box 933
Dayton, Ohio 45401, U.S.A.

Barry L. Nelson

Department of Industrial and Systems Engineering
The Ohio State University
1971 Neil Avenue
Columbus, Ohio 43210-1271, U.S.A.

ABSTRACT

A new method of modeling user sessions as Markov chains using NxN transition matrices is presented. These matrices are used to provide the user transaction sequences for discrete event simulation models of an on-line transaction processing system. Methods are given for validating the matrices, and algorithms are provided for modifying the matrices. Methodology for implementation in SLAM II® is given. This method allows the realistic simulation of user behavior and can also be used as the control for load generators for system level tests. Using comparisons of two matrices, one may compare the behavior of two different user populations.

1 INTRODUCTION

We wanted to realistically and compactly characterize customer behavior for a simulation model of an on-line transaction processing (OLTP) system, which delivers full-text data to users. In this system a user session consists of signing on, defining an area of research from a set of menus, conducting the research, and eventually signing off. Users may retrieve documents, read them in any of several formats, print those of interest, obtain more documents, and read those. They may quit at any time or make other choices, such as use of other services. Since the various transaction choices have different resource requirements, changes in user behavior can make major changes in the system requirements.

We also wanted to easily modify our behavior model. The addition of new transactions, modifications to existing transitions, and the implementation of new interfaces and features are on-going processes that can cause changes in the manner that users conduct their sessions. By using a model of the predicted behavior to control the load generator for the system model, the impact of enhancements may be anticipated.

Our intent was to maintain as much of the original variability in transaction sequences at the session level as possible, while minimizing the complexity of generating transactions. The problem was compounded by the number of choices for the next transaction (over 30), and the variability in the length of the session (minutes to hours).

Several options for determining the transactions and arrival rates can be used. There are a number of advantages and disadvantages to repeated traces (Jain 1991). The biggest problem with traces, from our point of view, is that one cannot easily account for changes in user behavior as the system changes.

Transaction specific generators, each creating transactions at an average rate expected for the overall system, can overcome several of the disadvantages of scripts. They also can allow for the addition of new transactions, and changes in the relative arrival rates. However, they provide testing only at the overall average system level. This hides the variability at the session and transaction level that is of interest in OLTP systems. We also briefly considered using separate tables for each user and generating transactions in some probabilistic way. The memory and computational requirements of this made it impractical, and it was difficult to implement it in such a way as to relate it to the overall system transaction frequencies.

In our opinion, a transition matrix overcomes all of the difficulties we perceived in other approaches. In addition, we developed a set of algorithms to create modifications in an existing matrix, based on redistributing the total work done among a combination of new and old transactions or among the same set of transactions where behavior changes are anticipated. These modification methods allowed the authors to incorporate future predictions of behavior, add new transactions, and change the overall number of transactions for a session for a given sign-on arrival rate. This last capability is important in examining the impact on a system of extending session length.

One possible drawback to using a transition matrix is that, in theory, an entity could circulate for an extremely long time through the model (each entity represents a single user session). However, in the time spans being modeled, this would appear only as one of the longer sessions that did not complete during the test period. This is realistic in that there are sessions that are very long compared to the average and that do extend beyond the test period duration.

State transition matrices have been used in modeling to describe user behavior on a gross scale such as application or resource choice (Jain 1991), modeling telecommunications (Kreiger, Müller-Clostermann, and Sczittnick 1990), simulating bursts of line noise (Eier 1989), speech recognition (Vaseghi 1991) and other uses such as those listed in Kelton and Kelton (1987), but the authors' use of state transition matrices to model the details of customer sessions is a new application of the technique.

2 PRINCIPLES OF USE

The functionality of the state transition matrix is in representing each transaction as a state. Then by computing the relative probabilities of going from one transaction to any other and placing them in the cells of the matrix, state transitions will represent user choices of the next transaction. The key transactions for use of the matrix are sign-on and sign-off, which are the only two constants across all sessions. They provide known, fixed entry and exit points. A small hypothetical example of a matrix based on relative probabilities is shown in Table 1. In the examples we assume one-step dependency, which is proven valid for our application later in the paper.

The table is entered on the row that corresponds to the current state and exited on the column corresponding to the next state. If we assume that state 1 represents sign-on; state 2, fetch document; state 3, read a document's page; and state 4, sign-off, then we can see that entering the table at row 1 gives an 79% probability of fetching a document, a 1% probability of trying to read a document (an error-- there's none there at sign-on) and a 20% probability of immediately signing off. Once the user chooses to fetch a document (search), then there is a 63% probability of reading it, 17% probability of fetching a different document and a 20% probability of ending the session. The same kind of reasoning applies to state 3, reading a document page. The probabilities for state 4 are zeroes, except state 4, which terminates the modeled session. By convention this state is set equal to 1 to allow all rows to add to 1. This matrix will model varying sessions which on average will have 20% of the users signing on and immediately leaving the system, and the remainder fetching a document.

Looking at columns 2 and 3 in Table 2, the corresponding frequency matrix, the 79 transitions to the search (fetch document) state from the sign-on state will eventually create a total of 40 more entries to the search state and 125 entries to the read state. Therefore about half the users that do a search will do a second search. Since the 79 entries generate 125 total read state entries, then over half the users will read a second page. Looking at column 4, we can see that about one third sign off from the search state and the remainder sign off from the read state.

Table 1: Example of a State Transition Matrix, **Mp**, Based on Relative Transition Probabilities

Relative Transition Probabilities				
St	1	2	3	4
1	0	.79	.01	.20
2	0	.17	.63	.20
3	0	.16	.40	.44
4	0	0	0	1.0

Table 2: Example of a State Transition Frequency Matrix, **Mf**

Transition Frequencies				
State	1	2 (A)	3 (B)	4 (C)
1	0	79	1	20
2 (A)	0	20	75	24
3 (B)	0	20	50	56
4 (C)	0	0	0	100

For implementation purposes, it is difficult to deal with relative transition values. It is easier to create cumulative probabilities for a row and store them in a matrix, **Mc**. One can then determine the index of the next state by using a uniform random variate, v, between 0 and 1 according to the rule that the next state is n+1 if

$$Mc_{r,n} < v \leq Mc_{r,n+1},$$

where $Mc_{r,n}$ is the cumulative probability in row r up to column n.

The cumulative probability matrix corresponding to Table 1 is shown in Table 3. In the cumulative matrix, a uniform random variate, v, would generate a transition from state 1 as follows: If v ≤ .79, the user will fetch a document; if .79 < v ≤ .80, the user will erroneously try to read a page, and if .80 < v ≤ 1.00, the user will sign off. The other rows are handled similarly.

Table 3: State Transition Matrix, **Mc**, Based on Cumulative Transition Probabilities

Cum. Transition Probabilities				
St	1	2	3	4
1	0	.79	.80	1.0
2	0	.17	.80	1.0
3	0	.16	.40	1.0
4	0	0	0	1.0

Each transaction duration is composed of system time and think time. The system time for the transaction is determined by the simulation model. Think time is the time the user spends reading the screen received, deciding the next transaction, and keying it. Think time is calculated as discussed in Section 3.1. This leads to three matrices, one for the state transitions (transaction choices), one for the average think times associated with the transitions, and one for the corresponding standard deviations of the think time averages. The matrices are constructed from system-level transaction logs. As modeled, each session is an entity that repeatedly enters the matrix and makes a choice of next transaction until sign-off is selected, the total number and sequence of transactions for an entity being highly variable. In addition, when each entity returns to the matrix, think time is calculated and then modeled.

3 TRANSITION MATRIX DETAILS

3.1 Creation of the Matrix

The relative probabilities may be obtained by least squares estimates from the macro data of the total occurrences of each state (Kelton and Kelton, 1985, 1987, 1991) or from the micro data of a compilation of the individual transitions. The latter are ten times more precise than the former (Kelton and Kelton, 1987, 1991) and were available.

Transaction logs were analyzed for individual state transitions for approximately 2,000,000 transactions. Each transaction of interest was assigned an index into the matrix, and each transaction was paired with its following transaction. We will refer here to these as T and T′, respectively, with indices into the matrix of t and t′. For a session with S total transactions we would then have S-1 pairs (sign-off has no subsequent transaction). Each T,T′ pair could be considered to provide a row-column index into the frequency matrix, **Mf**, by using their assigned indices, t and t′. The corresponding cell, $Mf_{tt'}$, was incremented by one.

One of the considerations in building the table was to balance representation of all possible transactions against processing time and memory for building and storing the table. Careful analysis of the transactions and their frequencies allowed us to combine some which had the same system behavior and impact and eliminate others that were due to infrequent error conditions. The result was an NxN matrix, where $20<N<30$ accounted for over 90% of the system transactions.

From the transition frequency data, the probability table was generated, considering the best estimator of the probability to be the relative transition frequencies (Kelton and Kelton 1987, 1991). The total transitions for a row are summed and each cell of the row is then divided by the sum to obtain relative frequency. In our implementation this gives an NxN matrix similar to Table 1. This was then converted to a cumulative probability matrix similar to Table 3. The number of transitions for each state exceeded 1000, and for the most common transitions several thousand were counted.

To estimate think time, a constant value for communication time was subtracted from the elapsed time for the trip from the system to the user and back. Think times were modeled using a lognormal distribution. This distribution was chosen by finding the best match with the Unifit© program from Averill M. Law and Associates for the highest frequency transition for each of six transactions. In addition, frequency curves for the think time for an entire row of the table appeared to be lognormal in distribution. Since the average and standard deviation for each cell was available (see section, 4.1.3), using the assumption of lognormally distributed think times for each transition allows the dynamic generation of a random variate for think time at each state change.

3.2 Principals of Operation

Figure 1 illustrates the implementation of the transition matrix in a SLAM II® model context. As can be seen in the illustration, sign-on and sign-off provide the entry and exit points for a loop that has no set number of iterations for a session.

The state transition matrix is implemented in the cumulative probability (by row) matrix format (Table 3). Additionally, two corresponding matrices of the average think time and its standard deviation for each cell in the transition matrix are stored in the model.

When the entity returns to the transition matrix, the index of the next transaction is chosen using DPROBN, and the corresponding think time is calculated using RLOGN and the appropriate average and standard deviation values. The transaction type and the think time values are stored as attributes and the entity is scheduled. In the model context, each transaction type has its own path to set the variables for the transaction and complete its work.

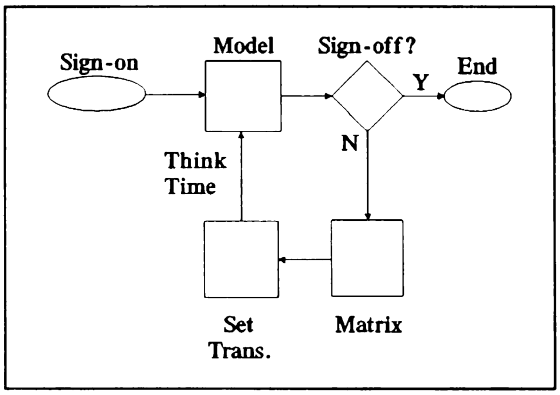

Figure 1: Use of the State Transition Matrix in a SLAM II® Model.

3.3 Algorithms to Modify the Matrix

When changes in user behavior are postulated or new transactions are added to the system or proposed to be added, the matrix must be changed accordingly. The complete set of algorithms to do this is described in the Appendix. Generally, changes would require the application of several algorithms to several rows of the matrix. Changes are made to the underlying frequency matrix, and the transition (probability) matrix is then recalculated. In these algorithms, one-step dependency is assumed. For a given state, the frequency row and the frequency column for that state must sum to the same value, and the frequency row sum for the start state must equal the frequency column sum for the stop state. This guarantees that all sign-ons will eventually have a sign-off (even if not in the span of the modeling run).

To illustrate their use, we will take a simple case in which it is postulated that customers will change behavior and 50% of the time go from states A to B to C instead of A to C as they currently do. We will use for our example the data in Table 1. In this particular example the user will do more browsing on more found documents rather than signing off after searching.

The algorithms operate on the frequency matrices and not the probability matrices. Therefore, we use Table 2, the underlying frequency matrix for the example. We will take as state A, state 2; as state B, state 3; and as state C, state 4.

Referring to the Appendix, algorithm 2.2 is the appropriate choice. It adds a number, n, of A->B->C transitions, replacing the same number of A->C transitions, using the following equations, (1-3), where $Mf_{i,j}$ are the cells of the frequency matrix.

$$Mf_{a,b} = Mf_{a,b} + n; \quad (1)$$
$$Mf_{b,c} = Mf_{b,c} + n; \quad (2)$$
$$Mf_{a,c} = Mf_{a,c} - n; \quad (3)$$

Looking in Table 2, we see that currently the user has 24 transitions, A->C. If fifty percent of them will become transitions to B, then

$$n = 12,\ Mf_{a,b} = 75,\ Mf_{b,c} = 56, \text{ and } Mf_{a,c} = 24.$$

After calculation,

$$Mf_{a,b} = 87,\ Mf_{b,c} = 68, \text{ and } Mf_{a,c} = 12.$$

The resultant frequency matrix is shown in Table 4. The modified values are indicated with an asterisk and can be compared to Table 2. The corresponding probability matrix is shown in Table 5. This can be compared to Table 1 to see the differences the change has made.

The impact is to increase the number of sign-off transactions (C) after entering the read state (B), and decrease the number of sign-offs from the search state (A). It also increases the number of entries to the read state from the search state. Since the sign-offs must equal the sign-ons, it is evident that the sign-offs from the browse state must increase in absolute numbers. However, the proportion of sign-offs does not increase as rapidly, as can be seen in Table 5, the new probability matrix.

The increased probability of signing off from the read state decreases the probability of staying there. This is somewhat offset by the greater arrival probability.

Table 4: Example of a State Transition Frequency Matrix after Modification

Modified Transition Frequencies				
St	1	2	3	4
1	0	79	1	20
2	0	20	* 87	* 12
3	0	20	50	* 68
4	0	0	0	100

Table 5: Example of a State Transition Probability Matrix Based on the Frequencies in Table 4

Relative Transition Probabilities				
St	1	2	3	4
1	0	.79	.01	.20
2	0	.17	.73	.10
3	0	.15	.36	.49
4	0	0	0	1.0

One can apply these algorithms in an iterative manner to modify more than one transaction's transitions in a matrix. Some scenarios to illustrate the use of the algorithms are:

If a subsequent state becomes mandatory for all exits from a given state (A->B->y changes to A->B->C), use algorithm A.4.1 iteratively to replace each former transition from the given state with one to the mandatory state;

If a new transaction is created, use any of the addition algorithms depending on how much is known about the preceding and subsequent states;

If executing more or fewer B transactions, use addition of x->B->y, algorithm A.2.5, or its removal analog, if nothing is known about the preceding or subsequent states, and

If a new transaction is providing partial or total replacement of an existing transaction, use algorithms A.4.1-A.4.4, depending on how much is known about the preceding and subsequent states.

4 EXPERIMENTAL RESULTS

4.1 Validation Tests

4.1.1 Matrix Representativeness and Reproducibility

A relative probability matrix was calculated for each of six days. A standard error for each cell was created for the average of corresponding cells' probabilities across the six days. The standard error was less than 1% for any cell with more than 1000 data points. We took this to indicate that we had represented the overall behavior of users on the system in a reproducible way, and that users had fairly consistent behavior from day to day.

4.1.2 One-Step Dependency Test

It was important to determine whether we had one-step or two-step dependencies. Using data from a restricted set of the most significant transactions, a 16x16 matrix representing one-step dependence and a 16x16x16 matrix representing two-step dependence were created. Using an algorithm based on Bhat (1984) and Anderson and Goodman (1957), the two matrices were compared. A Chi-squared statistic of 1800 for 16*15*15 degrees of freedom was obtained. This was compared to a Chi-squared distribution with the same degrees of freedom at the 0.95 quantile which gave a value of 3740. Since $1800 \ll 3740$, the null hypothesis of one-step dependence was accepted.

4.1.3 Think-Time Dependence

The question arose about whether think time was dependent on the preceding transaction (row effect), the subsequent transaction (column effect), the specific combinations (cell effect) or was totally random (no effect). Our desire was to minimize storage of think times, and a row or column effect would reduce by N the number of values we would have to have to calculate think times, and no effect would allow using a single distribution. ANOVA calculations indicated a definite cell effect. Therefore we provided matrices of mean think times and standard deviations for each cell in the transition matrix.

4.1.4 Comparison of Output to Input Data

A program was created to create sessions of transactions without system time but with accumulated customer think time, using the transition matrix and its associated think time mean and standard deviation tables.

Six runs of 2000 sessions each were performed. The batch means for session length exclusive of system time and frequency of a key transaction were calculated. Both were reproducible to 0.5% error. The expected values for session length and transaction frequency means were taken from actual system values and were well within the 95% confidence interval of the batch means.

Using the matrix modifying algorithms listed above, a modified matrix was produced to reflect hypothetical changes in user behavior. Tests of this matrix produced session length histograms of the same general shape as before but with a higher mean length, as was expected with the addition of new transaction choices.

4.2 Use of the Matrix in the System Model

The state transition matrix built from current user behavior was incorporated into a system model. The output from the model had session lengths and transaction frequencies that agreed well with the current system values. Output from the model for modified matrices gives values that agree well to those estimated from the theoretical behavior that created the modifications.

4.3 Comparison of Two User Populations

We would expect to see differences in behavior reflected in differences in the transition probabilities. Since we had demonstrated one-step dependence of the Markov chains representing customer sessions, we can use goodness of fit tests for the comparison of Markov chain transition probabilities (Anderson and Goodman 1957, Chatfield 1973, Bhat 1984) to determine if there are significant differences between any two NxN matrices. This test is an extension of the comparison of a given probability to a specified probability. The given probabilities, $p_{i,j}$, would be the $Mp_{i,j}$ of one relative probability matrix, and the specified probability, $p'_{i,j}$, would be the $Mp'_{i,j}$ of the second matrix. This test is planned for future work.

5 CONCLUSIONS

State transition matrices allow user transaction behavior to be characterized realistically and realistically. Their stochastic nature creates the variance in transaction sequences and session lengths seen in the actual system and still provides the correct overall averages for the system. Their ease of modification allows for testing the effects of hypothetical changes in user behavior.

Transition matrices can also be used to simulate user behavior in load generators for tests of actual systems, by creating stored scripts or generating transactions during testing.

These matrices can be used to compare the behavior patterns of two groups of users.

ACKNOWLEDGMENTS

The authors wish to acknowledge the many useful discussions during this work with the members of the Systems Evolution and Modeling Department, particularly Jack Eddington and David Withers. We also wish to thank David Villwock for his assistance in implementing the transition matrix into our SLAM II® models, and Linda Showalter for editing assistance.

APPENDIX: MODIFICATION ALGORITHMS

The governing rule in the algorithms is that the row frequency for a state must equal the column frequency for a state, that is, the exits from the state must equal the entries to the state. If an entity enters a state, it must exit, or else the system eventually freezes with all the entities in states that have exit frequencies less than entry frequencies. As will be shown below, we must back out as many starting and ending transitions on a path as we add. The number of transitions being deleted in a modification must not exceed the minimum number in the affected cells. As listed here, this check is missing in the algorithms.

A.1 Global Declarations

Mf -- NxN matrix of frequencies.
R -- NxN matrix of % based on row sums.
C -- NxN matrix of % based on column sums.
a,b,c -- indices corresponding to transactions A,B,C.
N -- Number of states in the matrix.

Calculate **R** and **C**:

$$R_{i,j} = Mf_{i,j} / \sum_{j} Mf_{i,j} .$$

$$C_{i,j} = Mf_{i,j} / \sum_{i} Mf_{i,j} .$$

A.2 Addition Algorithms

A.2.1 Add a Totally New Transaction to the Matrix; Both the Preceding and Subsequent States are Known

Add a number, n, of x->D->y transitions, where D is a totally new transaction, removing n x->y transitions. In this algorithm x represents the preceding transaction types, y represents the subsequent transaction types, and both x and y are known. This adds the x->D->y transitions in exactly the overall proportions of the various x->y transitions. Prior to executing these algorithms, the matrix, **Mf**, must have been expanded to (N+1)x(N+1) and the index for D assigned, as well as the remaining indices reassigned as necessary for the original N transactions.

```
Declarations:
  P -- Vector of P preceding transactions;
  S -- Vector of S subsequent transactions;
  Rf -- NxN relative frequency matrix;
  p -- index of P vector;
  s -- index of S vector;
  D -- the new transaction;
  d -- the index of D in Mf.

Algorithm:
  Calculate relative frequency of each preceding and
  subsequent transaction.

    Sum = 0;
    FOR p = 1 to P
      FOR s = 1 to S
        $Sum = Sum + Mf(P_p,S_s)$ ,                    (A1)

  and

    FOR p = 1 to P
      FOR s = 1 to S
        $Rf(P_p,S_s) = Mf(P_p,S_s) / Sum$ .            (A2)

  Calculate the row-column entries for B.

    FOR p = 1 to P
      FOR s = 1 to S
      { $trn = n * Rf(P_p,S_s)$;                       (A3)
        $Mf(P_p,S_s) = Mf(P_p,S_s) - trn$;             (A4)
        $Mf(P_p,d) = Mf(P_p,d) + trn$;                 (A5)
        $Mf(d,S_s) = Mf(d,S_s) + trn$; },              (A6)

  where $1 \leq n \leq Sum$.
```

Equation (A2) determines the relative ratios for each $M_{p,s}$ to the sum of all $Mf_{p,s}$ determined by (A1) and populates the **R** matrix, which generally is sparse (P<N and S<N). The double loop, (A3) through (A6), calculates the number to be changed for each $Mf_{p,s}$ (A3), subtracts it from the $Mf_{p,s}$, (A4), and adds it to the preceding, $Mf_{p,d}$, and subsequent, $Mf_{d,s}$, states, (A5) and (A6), respectively.

A.2.2 Both Prior and Subsequent States Are Known

Add a number, n, of A->B->C transitions, replacing the same number of A->C transitions. This requires adding two state transitions where one existed before. This is a special case of the preceding algorithm where **P** and **S** both have a single state.

```
  $Mf_{a,b} = Mf_{a,b} + n$;        (A7)
  $Mf_{b,c} = Mf_{b,c} + n$;        (A8)
  $Mf_{a,c} = Mf_{a,c} - n$;        (A9)
```

Equation (A7) adds the transitions from A to B; (A8) adds the transitions from B to C, and (A9) removes the A->C transitions.

A.2.3 Prior State is Unknown

Add a number, n, of x->B->C transitions, replacing the same number of x->C transitions, where x represents all the transactions that have B as a subsequent state. This distributes the removal and addition proportionately across all x's. Since B is a subsequent state, we use the **C** matrix to calculate the proportion of change for each x->B transition.

```
$Mf_{b,c} = Mf_{b,c} + n$;                (A10)
FOR x = 1 TO N
{ $n' = n * C_{x,b}$;                     (A11)
  $Mf_{x,b} = Mf_{x,b} + n'$;             (A12)
  $Mf_{x,c} = Mf_{x,c} - n'$; }.          (A13)
```

Equation (A10) adds the B->C transitions; (A11) calculates the proportion of change for state x; (A12) adds the proportion of x->B for each state x, and (A13) removes the x->C transitions for each state, x.

This algorithm and the following two algorithms are not equivalent to the first. We have no knowledge of **P** in this case, and no knowledge of **S** in the next case. In the last addition algorithm, we know neither **P** or **S**.

A.2.4 Subsequent State is Unknown

Add a number, n, of A->B->y transitions, replacing the same number of A->y transitions, where y represents all the transactions that have B as a preceding state. This distributes the removal and addition proportionately across all y's. Since B is a preceding state, we use the **R** matrix to calculate the proportion of change for each B->y transition.

```
$Mf_{a,b} = Mf_{a,b} + n;$                 (A14)
FOR y = 1 TO N
{ $n' = n * R_{b,y};$                      (A15)
  $Mf_{b,y} = Mf_{b,y} + n';$              (A16)
  $Mf_{a,y} = Mf_{a,y} - n';$  }           (A17)
```

Equation (A14) adds the A->B transitions; (A15) calculates the proportion of change for state y; (A16) adds the proportion of B->y for each state y, and (A17) removes the A->y transitions for each state, y.

A.2.5 Both Prior and Subsequent States are Unknown

Add a number, n, of x->B->y transitions, replacing them with the same number of x->y transitions, retaining the same overall table structure and proportions. This process distributes the additions evenly over the preceding x states and over the subsequent y states in proportion to their transitions to or from B respectively.

```
FOR x = 1 TO N
{ $n' = n * C_{x,b};$                      (A18)
  $Mf_{x,b} = Mf_{x,b} + n';$              (A19)
  FOR y = 1 TO N
  { $n'' = n * R_{b,y};$                   (A20)
    $Mf_{b,y} = Mf_{b,y} + n'';$           (A21)
    $Mf_{x,y} = Mf_{x,y} - n'';$  } }      (A22)
```

The first loop handles the states that precede B. Equation (A18) calculates the number of the transitions to B that each preceding transaction contributes to the total being added of all transactions that precede B. (A19) then adds the transitions to B. For each preceding state, the second loop then prorates the transitions to the subsequent state based on the number of B transitions being added, and the proportion of transactions subsequent to B for each. Equation (A20) calculates the proportion for each preceding/subsequent combination; (A21) adds it to the B->y count and (A22) causes it to be removed from the x->y transition count. The total number of transactions does not change; but the proportion that follow x->y increases and the proportion for x->B->y decreases.

A.3 Removal Algorithms

There are five algorithms that have a 1:1 correspondence with the addition algorithms except that in the equations, + becomes - and - becomes +. Since they are exact opposites they will not be discussed.

A.4 Replacement Algorithms

These algorithms are analogous to the addition algorithms A.2.1.2-A.2.1.5. The difference is that there is one more transition to be maintained. In these algorithms one is replacing a two-transition path with another two-transition path, instead of replacing a single transition path with a two transition path. They are presented without discussion as the logic follows previously explained lines.

A.4.1 Both Prior and Subsequent States are Known

Remove a number, n, of A->B->C transitions and replace them with the same number of A->D->C transitions.

```
$Mf_{a,b} = Mf_{a,b} - n;$
$Mf_{b,c} = Mf_{b,c} - n;$
$Mf_{a,d} = Mf_{a,d} + n;$
$Mf_{d,a} = Mf_{d,a} + n;$
```

A.4.2 Only the Subsequent State is Known

Remove a number, n, of x->B->C transitions, and replace them with n x->D->C transitions, where x represents all the preceding states to B. This will retain the same relative proportions of transitions in the rest of the table. It distributes the x->D transitions in the same proportions as the original x->B transitions being replaced. This assumes that the matrix has already been adjusted to include the replacing transaction.

```
$Mf_{b,c} = Mf_{b,c} - n;$
$Mf_{d,c} = Mf_{d,c} + n;$
FOR x = 1 TO N
{ $n' = n * C(x,b);$
  $Mf_{x,b} = Mf_{x,b} - n';$
  $Mf_{x,d} = Mf_{x,d} + n';$  }.
```

A.4.3 Only the Prior State is Known

Remove a number, n, of A->B->y transitions and replace them with the same number of A->D->y transitions, where y represents all transactions that have B as a preceding state. This distributes the removal and addition proportionately across all y's.

```
$Mf_{a,b} = Mf_{a,b} - n$;
$Mf_{a,d} = Mf_{a,d} + n$;

FOR y = 1 TO N
{   $n'' = n * R(b,y)$;
    $Mf_{b,y} = Mf_{b,y} - n''$;
    $Mf_{d,y} = Mf_{d,y} + n''$;  }
```

A.4.4 Both the Prior and the Subsequent States are Unknown

Remove a number, n, of x->B->y transitions, and replace them with the same number of x->D->y transitions. The changes are distributed across the x's and y's in the same proportion as their occurrences in the matrix.

```
FOR x = 1 TO N
{ $n' = n * C_{x,b}$;
  $Mf_{x,b} = Mf_{x,b} - n'$;
  $Mf_{x,d} = Mf_{x,d} + n'$;

  FOR y = 1 TO N
  {   $n'' = n * R_{b,y}$;
      $Mf_{b,y} = Mf_{b,y} - n''$;
      $Mf_{d,y} = Mf_{d,y} + n''$;   } }
```

REFERENCES

Anderson, T.W., and Goodman, L.A. 1957. Statistical Inference about Markov Chains. *Annals of Mathematical Statistics* 28: 89-110.

Bhat, U.N., 1984. *Elements of Applied Stochastic Processes*, Second Edition. New York: John Wiley and Sons.

Chatfield, C. 1973. Statistical Inference Regarding Markov Chain Models. *Applied Statistics* 22: 7-20.

Eier, R., 1989. Discrete Stochastic models by means of Markov Chains. *ITG-Fachbericht* 107: 167-173.

Jain, R. 1991. *The Art of Computer Systems Performance Analysis*. New York: John Wiley and Sons.

Kelton, C.M.L., and Kelton, W.D. 1985. Development of Specific Hypothesis Tests for Estimated Markov Chains. *Journal of Statistical Computation and Simulation* 23: 15-23.

Kelton, C.M.L., and Kelton, W.D. 1987. Comparison of Hypothesis testing techniques for Markov processes Estimated from Micro versus Macro data. *Annals of Operations Research* 8: 175-194.

Kelton, C.M.L., and Kelton, W.D 1991. A Comparison of Micro versus Macro Point Estimators for Markov-Process Models. *Journal of Statistical Computation and Simulation* 38: 201-210

Krieger, U.R., Müller-Clostermann, B., and Sczittnick, M. 1990. Modeling and Analysis of Communications Systems Based on Computational Methods for Markov Chains. *IEEE Journal on Selected Areas in Communications* 8: 1630-1648.

Vasegji, S.V. 1991. Hidden Markov Models with Duration-Dependent State Transition Probabilities. *Electronics Letters* 27: 625-626.

AUTHOR BIOGRAPHIES

WILLIAM S. KEEZER has been with Mead Data Central (MDC) for over five years and is currently a Staff Analyst focusing on system and communications performance issues. Prior to coming to MDC, he was an in-house consultant on OLTP system performance problems for the Data Pathing Division of NCR. He holds B.S. and Ph.D. degrees from the University of Oklahoma, and is a member of the ACM.

ANDREW P. FENIC is a Basic Software Engineer with MDC in the Systems Evolution and Modeling Department. His work has included user interface design and prototyping, system load characterization, and performance data retrieval and analysis. He holds a B.S. degree in Computer Science from Sycracuse University.

BARRY L. NELSON is an Associate Professor in the Department of Industrial and Systems Engineering at The Ohio State University. His research interests are design and analysis of computer simulation experiments. He is an Associate Editor for *Operations Research* and President of the TIMS College on Simulation.