Time Dependent Rate-Based Modeling of Trace Data

Anup Mathur Marc Abrams

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061-0106
{mathur,abrams}@cs.vt.edu

October 14, 1995

Abstract

This paper introduces a novel technique to construct an empirical model fitting time-varying (transient) trace data. The trace is a categorical or numerical time-series. The modeling technique described in this paper first builds a Mass Evolution Graph (MEG) from the trace data. Linear regression is then used to construct a time-dependent probability mass function (pmf) for each state in the trace data.

1 Motivation

Building models of system behavior is key to the advancement of science. This is certainly the case for computer science, where modeling is used to analyze and debug computer system performance, to fine tune the performance of network protocols, and to design memory components such as caches.

A classical approach towards modeling the behavior of a system is to hypothesize a model with known properties. In contrast, there is the data-driven approach which builds empirical models by statistically analyzing the *trace data* representing the observed behavior of a system. This paper focuses on the latter approach. *Trace data* collected from systems typically records the *state* of the system along with a *timestamp* or *event number*. The state is a snapshot of system parameters at the corresponding timestamp (event number). An example is a program execution trace output from a computer software profiler; the state can be the name of a function or procedure and the timestamp can be the time the program entered that function or procedure.

While the trace data collected from many systems exhibits time-dependent characteristics, most modeling research assumes that the underlying system is described by time-invariant probability distributions.

In this paper we describe a modeling approach that can model categorical time-series data and delivers a time-dependent probability mass function as the final model.

1.1 General Problem Statement

Given trace data from a system we want to:

- generate a model that outputs a time-dependent probability function for each unique state in the trace,
- generate synthesized traces from our model, and
- validate our model by testing how closely the synthesized trace follows the original trace.

1.2 Why Transient Behavior Occurs

Many systems display time-varying behavior. An example is systems that have behavioral phases. Informally a phase is a time-duration during which a system's state can be represented by a time-invariant probability distribution. This distribution, may be represented with a time-homogeneous transition probability matrix, as is used in a Markov Chain model. In most systems a change in phase can be attributed to some cause. For example let us consider the system under analysis to be the pollution level in a certain U.S city measured yearly. In this case the state of the system is some indicator of pollution denoted I. If the city management authority enforced a cleaner environment policy in a certain year T we may see a phase change in the system at time T. The above is a simple example where the cause for a phase change is known. In other cases a phase change can reveal a cause-effect relationship unknown hitherto.

1.3 Our Approach

As mentioned in section 1.1 our goal is to produce a time-dependent probability mass function $f_i(t)$ for each state i recorded in the trace-data from a system. The first step towards recovering $f_i(t)$ is to plot a curve with the $NumberofOccurrences$ of state i as the abscissa and $Time$ or $Eventnumber$ as the ordinate. The slope of this curve at any instant t is directly proportional to $f_i(t)$. We denote the set of such curves for all unique states in the trace-data as the Mass Evolution Graph (MEG) for the system. The next step in our approach is to fit a curve to each path in the MEG using linear regression techniques and compute $s_i(t)$, the time-dependent slope function for each fitted curve. Since $s_i(t)$ represents the proportion of time state i occurs, instead of $s_i(t)$ we use the normalized $s_i(t)$, $s_i^{'}(t)$, defined as follows:

$$s_i^{'}(t) = \frac{s_i(t)}{\sum_i s_i(t)}$$

The result is that now $\sum_i s_i^{'}(t) = 1$.

1.4 Example: A Tollbooth

A bored toll collector in a toll booth on a highway entertains herself by trying to estimate, based on one shift, the probability that the i-th vehicle to arrive during her shift is a truck rather than a car. She recalls that an estimate of the probability of an event, based on a set of trials, is the frequency of the event: the ratio of the number of times the event occurs to the number of trials. Thus a simple approach to estimate the car arrival probability is to merely count the number of cars that arrive in a shift, as well as the number of vehicles that arrive, and use the ratio of the first to the second as the estimate. However, this estimate is too crude because the probability is time dependent: the arrival rate of cars and trucks fluctuates during the shift; for example, more cars arrive during rush hour.

If we know the *instantaneous* arrival rate of cars and trucks, we can use the ratio of one to the sum of the two as a probability estimate. For example, if the instantaneous arrival rate of cars and trucks when the i-th vehicle arrives is 31 and 11 per hour, respectively, then the probability that the i-th vehicle is a car is 31/42.

To calculate the instantaneous arrival rate of cars, the toll collector writes down the sequence of vehicle types that arrive, as shown in the left and middle columns of Table 1. At the end of the shift, the collector plots the number of cars that have arrived as a function of the number of vehicles that arrived, and fits a path to the points, resulting in the path marked "Cars" in Fig. 1. Based on the slope of the line segments comprising the path, the effect of rush hour is evident, because the path can be partitioned into three line segments or *phases* ($1 \leq i \leq 6$, $6 < i \leq 14$, and $14 < i \leq 20$). *The derivative of the path*

i	S_i	$g(i)$
0	C	1
1	T	1
2	C	2
3	T	2
4	C	3
5	T	3
6	C	4
7	C	5
8	C	6
9	C	7
10	C	8
11	C	9
12	C	10
13	C	11
14	C	12
15	T	4
16	C	13
17	T	5
18	C	14
19	T	6
20	C	15

Table 1: Vehicle arrivals as recorded by the toll collector.

at x coordinate i is the instantaneous arrival rate of cars when the i-th car arrives, thus solving the toll collector's problem. For example, after rush hour ($i > 14$) the path has slope 0.5, because one out of every two arriving vehicles is a car. Thus the estimated probability of a car arrival is 0.5.

The graph of truck arrivals is shown in the path marked "Trucks" in Fig. 1. Truck arrivals also experiences three phases, but the boundary points of the phase intervals (at event numbers 6 and 16) are not the same as those for cars. Thus the graph can capture a realistic feature of many systems: that different states can change phase at different times.

Fig. 1 is a Mass Evolution Graph (MEG) of the trace in Table 1. A MEG is a two dimensional Cartesian graph whose x axis represents an event number. Each possible value of the state vector is represented by a distinct path in the MEG. The y axis represents the number of occurrences of a given state vector value. The following points are plotted. Let g be a function such that $g(i)$ is the number of times that state value v, where $S_i = v$, occurs in states $S_1, S_2, \ldots, S_i$ (see right column in Table 1).

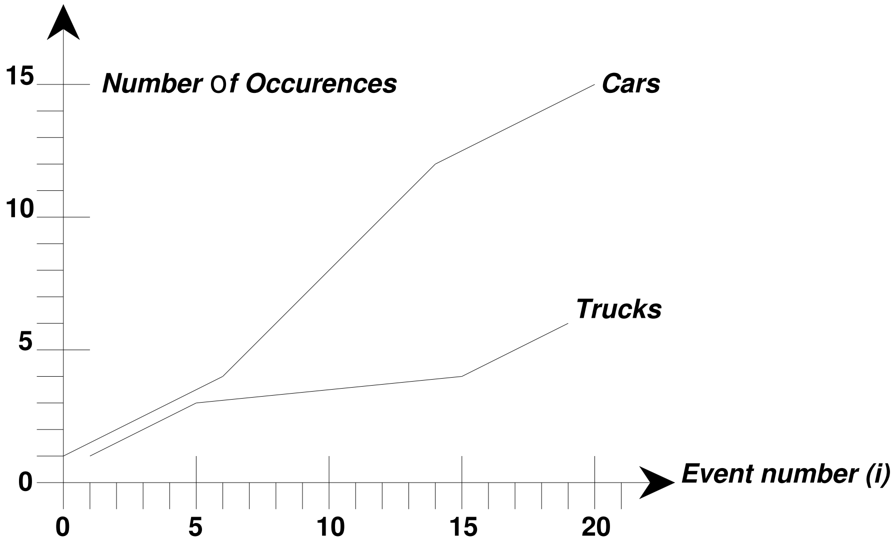

Figure 1: Graph of arrival rate of cars and trucks to a toll both, equivalent to a MEG.

Formally, $g(i) = ||\{S_j \mid S_j = S_i \wedge 1 \leq j \leq i\}||$. The MEG for state sequence $S_1, S_2, \ldots, S_L$ consists of the points $(1, g(1)), (2, g(2)), \ldots, (L, g(L))$ and, for each state value v, a path connecting all points $\{(i, g(i)) \mid S_i = v\}$ in order of increasing i.

The analogy of the toll booth problem to modeling systems is that the sequence of vehicle types recorded can be considered a trace. The vehicle is a component which we want to model, and the vehicle types (car or truck) are the two possible states of the component. Table 1 might represent a trace listing the sequence of reads and writes arriving to a memory, instead of cars and trucks.

In general a trace can have more than two states, in which case we just add more paths to the MEG. For example, the MEG of an execution sequence trace from a dining philosophers program with five diners program on a multiprocessor is shown in Fig. 2. The state vector has five components, each representing a philosopher. Each philosopher has four states (acquiring forks, eating, releasing forks, and thinking). Only some of the 5^4 possible state vector values appear in the trace, and each is plotted as a distinct path in Fig. 2. This figure shows several distinct points when the slopes of lines suddenly change, indicating phase changes. There are nine phases in all, and the changes occur whenever a philosopher starts execution or terminates execution.

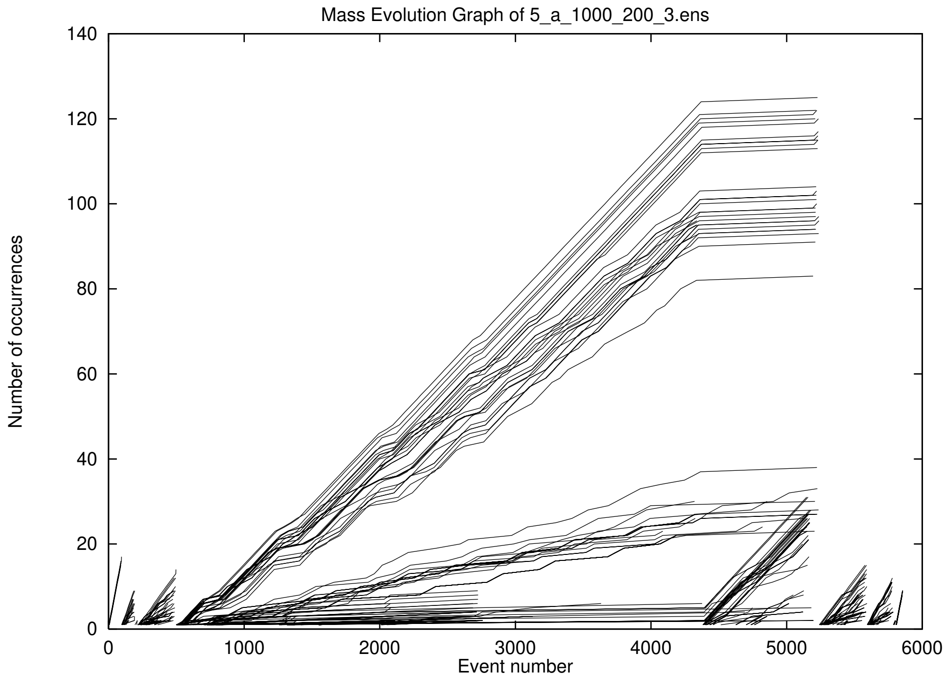

Figure 2: MEG of a five dining philosopher program with five philosophers.

The MEG visualizes categorical traces without imposing an ordering on the component state values by plotting an attribute of the state sequence (i.e., function g, denoting number of occurrences) on the y axis, rather than the states directly. The MEG illustrates which states occur at which points in the trace, whether there are phases in behavior, and how the rate of occurrence of each state changes with time.

1.4.1 Model for the Tollbooth

Tables 2 and 3 show the probability estimates for the Tollbooth example as generated by our model. To arrive at these estimates the path corresponding to state 0 (*Cars*) was partitioned into segments of nearly-constant slope viz $(0,7),(7,15),(15,16)$, and $(16,18)$. Straight lines were fitted to each segment using simple linear regression. The slope of each fitted line is the estimated probability for the corresponding segment. The same process was repeated for state 1 (*Trucks*). It must be noted that probability for a state at a *transition point* (i.e., a point where one segment ends and the next begins) is computed as the maximum of the estimated probability for the two segments that the transition point separates. The algorithm that generates the probability estimates allows other options: *average, left,* and *right.* These and other options are detailed in the Section 2.6.4. The probability estimates shown in tables 2 and 3 do not represent a true probability mass function (pmf) because the probability estimates corresponding to a segment for state 0 and state 1 do not add up to 1. To get a pmf, the probability measures are normalized so that they add up to 1.

Table 2: Probability estimates for State 0 (Cars)

Estimated probabilities $P_0(t)$				
Event beginning the interval I	Event ending the interval I	$P_0(I)$	Transition Event X	$P_0(X)$
0	7	0.50	7	1.00
7	15	1.00	15	1.00
15	18	0.50	18	0.50

Table 3: Probability estimates for State 1 (Trucks)

Estimated $P_1(t)$				
Event beginning the interval I	Event ending the interval I	$P_1(I)$	Transition Event X	$P_1(X)$
0	1	0.00	1	0.50
1	6	0.50	6	0.50
6	15	0.10	15	0.50
15	19	0.50	19	0.50

2 Introduction to MEG Modeling

2.1 Modeling Problem Statement

We view a trace as a realization of a stochastic process. Given a trace, we wish to build a model that

- does not impose a total ordering amongst model states, and
- infers the underlying stochastic process.

2.2 Terminology

Some terminology related to stochastic processes is reviewed next. "A variable X is a *discrete random variable* if it takes one of a specified set of values with a specified probability." [Jai91, p. 179]. Suppose that X takes n distinct values $\{x_1, x_2, \ldots, x_n\}$ with probabilities $\{p_1, p_2, \ldots, p_n\}$. A *probability mass function* for X maps x_i to p_i, for $1 \leq i \leq n$. A *stochastic process* is a collection of random variables indexed by an integer parameter i representing time: $\{X_i\}$. We view a trace $\{S_i \mid 0 \leq i < L\}$ as a realization of a stochastic process $\{X_i \mid 0 \leq i < L\}$. Therefore the pmf for X_i gives the probability that variable X_i equals state component value S_i.

In general the pmf of X_i will be different for each event number i in the trace. Therefore the pmf is time-dependent, and we say that the system from which the trace data was collected displays *transient behavior*. Therefore our problem is to obtain the pmf for each variable X_i.

2.3 Mass Evolution Graphs

The mass evolution graph (MEG) is a visualization of trace data that does not impose a total ordering on the data. Hence MEG is suitable for visualization of ordinal as well as categorical data.

The mass evolution graph (MEG) is so named because one can calculate the time dependent pmf of the random variable X_i forming the stochastic process as follows. Let m_i^s denote the derivative of the path for state s at $x = i$. Draw the line $x = i$ on the MEG. Let P_i denote the set of states corresponding to a path that the line intersects. For each state not in P_i, the estimated probability of the state occurring at event i is zero. For each state $s \in P_i$, the estimated probability of s is

$$\frac{m_i^s}{\sum_{s' \in P} m_i^{s'}}.$$

Another way to view this is geometrically: We compute the slope of a tangent line to each path intersecting line $x = i$, and use the ratio of each tangent line slope to the sum of the slopes of all tangent lines as estimated probabilities.

2.4 Uses of the MEG

The MEG has two primary uses: as a visualization tool and as an aid for modeling time-dependent data. The MEG is unique from other views of time-series data because it visualizes categorical as well as ordinal trace data.

2.5 MEG Used to Build Models

The focus of this paper is to illustrate how the MEG is used to build accurate models for time-dependent trace data. The MEG contains a path for each state in the trace. The probability that the stochastic process underlying the trace is in a particular state at a certain instant (event number) is given by the slope of the path corresponding to the state at that instant (event number). The model is built by fitting a straight line segment (or series of line segments) to each path in the MEG. The slope of each fitted line segment estimates the probability of the process being in the corresponding state for the event duration implied by the fitted line segment.

2.6 Process Used to Build Models

The process of building a model from trace data can be decomposed into the following steps:

1. Generate a MEG from the trace data.
2. Partition each path in the MEG into segments of nearly-constant slope. We present two approaches to partition each path in the MEG namely *slope-based partitioning* and *incremental regression partitioning* in section 2.6.2.
3. Fit line segments to each path in the MEG.
4. Generate the final model by inspecting the slopes of each fitted line segment.

The figure 3 shows the stepwise process in flowchart form.

2.6.1 Generating the MEG

The first step towards building a MEG model of a system is to generate a MEG from trace data collected by executing the system. The trace data records system-states in the order they occur during execution. The Table 1 shows the trace collected from our example Tollbooth *system*. The states in this example system are: the arrival of a car (C) and the arrival of a truck (T). The MEG is generated as a collection of paths. Each path corresponds to a unique state in the trace. A path is a collection

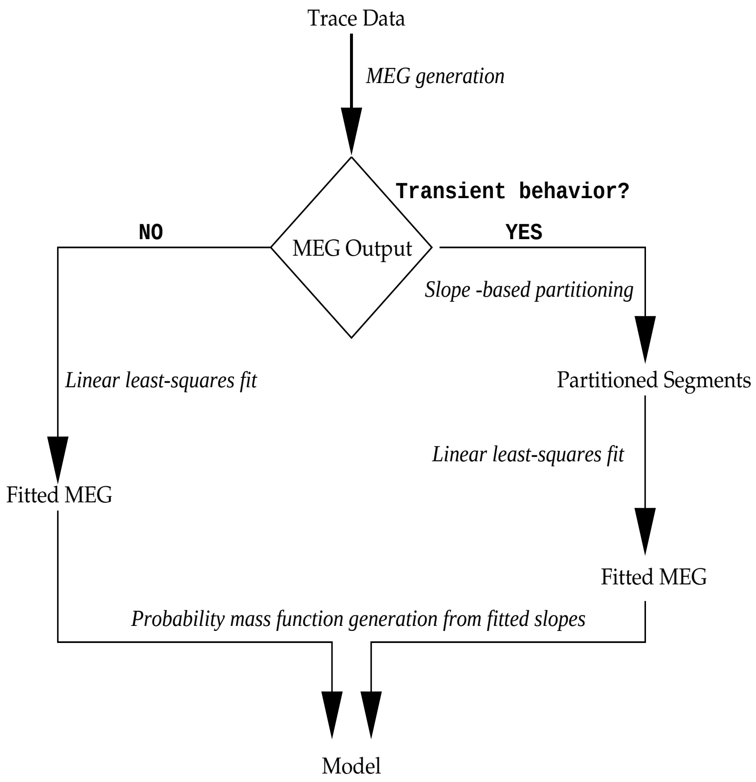

Figure 3: Process of building a MEG model

of $\langle Event, NumOccurrences\rangle$ tuples describing a curve in the $X - Y$ plane. $NumOccurrences$ represents the number of occurrences of a state at the event number $Event$. Recall that the MEG records a $\langle Event, NumOccurrences\rangle$ tuple every time NumOccurrences increases by one.

2.6.2 Partitioning Paths in the MEG

A path i in the MEG is a realization of the time-dependent pmf, $f_i(t)$, for the corresponding state. Our goal is to obtain an estimate of this pmf. As discussed in section 1.3 above, the pmf for a state i at event t is directly proportional to the slope of the corresponding MEG path $s_i(t)$ at that event. This holds true for all events except where the time-dependent pmf $f_i(t)$ is not differentiable. Denote such events as *partitioning events*. Partitioning events define the boundaries of *event-intervals* such that:

- during each event-interval the function $f_i(t)$ has a constant value, and
- for each consecutive event-interval the function $f_i(t)$ has a distinct value.

To estimate the function $f_i(t)$ we must determine all the partitioning events for it. In a MEG path, events where the slope of the path changes significantly are strong indicators of a partitioning event for the corresponding state. In this step we determine the partitioning events for the pmf of a state by partitioning the corresponding MEG path into segments of nearly-constant slope. We describe two approaches to partition MEG paths here:

Slope-based partitioning This approach relies on two algorithms that form a UNIX-style pipeline to partition a path in the MEG into segments having a constant slope. The first algorithm computes the slope at every event for each path in the MEG. The input to this algorithm is the MEG. The second algorithm partitions each path in the MEG into segments of constant slope. The input to this algorithm are the MEG and the output from the first algorithm.

The partitioning algorithm inspects the slope at each consecutive pair of events in a MEG path. If the slope at event E_i differs from the slope at the immediately previous event E_{i-1} by more than the user defined parameter *threshold*, then the event E_i is a partitioning event. Another user defined parameter *runlength* enforces a lower bound on the number of events in a segment (e.g., if $runlength = 2$ then a segment must have at least two points).

These algorithms are detailed in appendix A.1.1.

Incremental regression partitioning This technique is based on incrementally regressing the independent variable E_i, which denotes the event number in a MEG path, to predict the dependent variable O_i which denotes the corresponding number of occurrences. We construct a simple linear regression function, denoted $\hat{O}$ based on the first k $\langle EventNumber, NumOccurences\rangle$ tuples, $\langle E_0, O_0\rangle, \ldots, \langle E_{k-1}, O_{k-1}\rangle$, in the MEG path. Using the regression function $\hat{O}$ we predict $\hat{O_k}$, the number of occurrences at event number E_k. We also determine the standard error for the prediction, $s\{\hat{O_k}\}$, and consequently a confidence interval around $\hat{O_k}$. Denote the confidence interval around $\hat{O_k}$ as $Conf\{\hat{O_k}\}$.

If the actual observed number of occurrences at event E_k, denoted O_k, falls outside the confidence interval $Conf\{\hat{O_k}\}$ then E_k is a partitioning event. The search for the next partitioning event starts at the current partitioning event. The process is repeated until the entire MEG path has been partitioned.

This approach restricts partitions to be at least three events wide. This is because at least three points are needed to compute a regression function. An algorithm based on this approach is presented in appendix ??.

2.6.3 Fitting straight line segments to MEG paths

This step can be described by the following simple algorithm:

```
for every path i in the MEG {
   for every partitioned segment j in path i {
      fit a straight line to segment j using linear regression;
   } /* endfor */
} /* end for */
```

A more detailed version of this algorithm is described in Appendix ??.

Figure 4 shows the MEG for the Gaussian Elimination trace after each path in it has been partitioned (using the approach described in section 2.6.2) and each partitioned segment has been fitted with a straight line.

Note that a single straight line can be fitted to the entire MEG path. This saves us from the partitioning step described in the previous section (section 2.6.2). However, this usually leads to a less accurate model.

2.6.4 Generating the Model

Generating the model after fitting straight lines to segments involves inferring an empirical probability mass function using the fitted lines. To illustrate the mechanism for generating the model we start with a simpler scenario, namely a single straight line is fitted to every path in the MEG output. Recall that every path in the MEG output corresponds to a state in the trace. Also the slope of a MEG path is

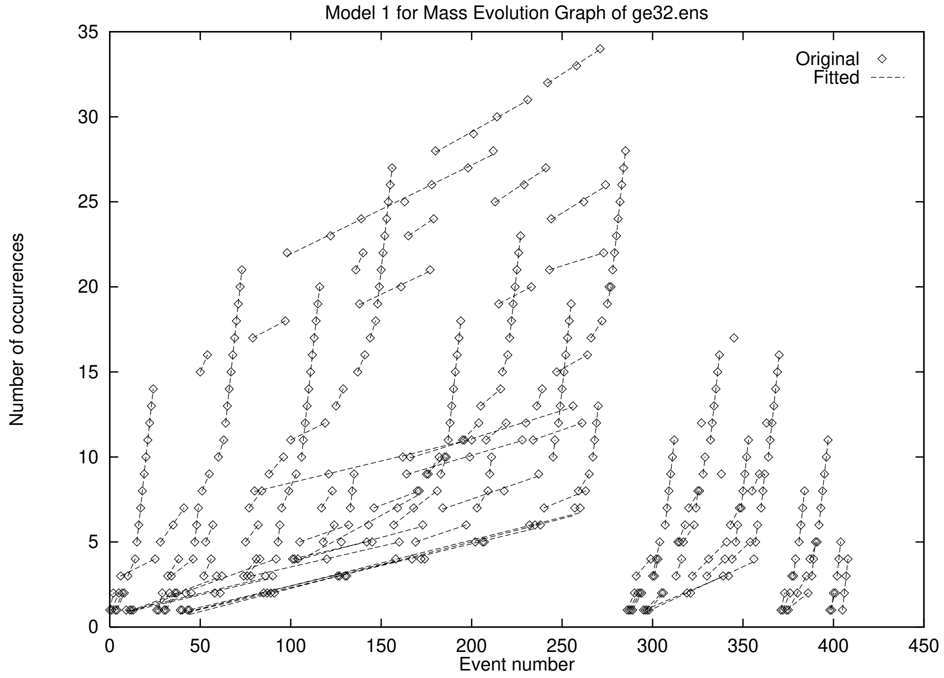

Figure 4: MEG of the Gaussian Elimination trace after fitting straight lines to partitioned segments

proportional to the pmf of the corresponding state. Using the above we can derive the probabilities for this simple scenario by using the slopes of the straight lines fitting each path in the MEG output. Recall that we already have these $FittedSlopes$ from the previous step described in section 2.6.3.

The same idea is extended to MEG paths that are partitioned into segments. Consider the scenario where a MEG path i (corresponding to state i) is partitioned into k segments and a straight line is fitted to each of these k segments. From the $FittedSlopes$ of each of the k segments we can build a time-dependent estimated pmf for state i. Applied to all paths (and hence all states) in the MEG output this generates an estimated time-dependent probability mass function for all states.

Our model generation algorithm also addresses the following issues:

Slope of a partition with missing data The MEG does not record the cumulative number of occurrences of a state at every event; instead it only records events where the cumulative number of occurrences of a state increases. Therefore, what appears as a continuous path in the MEG, upon partitioning, may be a series of discontinuous segments with areas devoid of data points separating them. Consider two such segments S_i and S_{i+1}. Denote the last $\langle event, numberofoccurrences \rangle$ tuple for S_i as $\langle E_{t_0}, O_{t_0} \rangle$ and the first such tuple for S_{i+1} as $\langle E_{t_1}, O_{t_1} \rangle$. While $O_{t_1} = O_{t_0} + 1$, $E_{t_1} = E_{t_0} + k$. There is therefore a gap k events wide for which we have no data. We can choose to :

1. assign a gap such as the one above a probability which is equal to the slope of the line joining the endpoints $\langle E_{t_0}, O_{t_0} \rangle, \langle E_{t_1}, O_{t_1} \rangle$ or
2. assign the probability 0 to such a gap.

The above choices are admissible options in our model generation algorithm.

Slope of transition points Recall from section 2.6.2 that partitioning a path in the MEG into segments is done based on the slope of the MEG path. A new segment starts at the *event* where the slope of the MEG path changes. Consider two consecutive segments S_i, S_{i+1} in the MEG path M (corresponding to state M). Denote the event that starts S_{i+1} as E_s. When generating the pmf for state M we must decide whether it gets its value from the pmf of M for segment S_i, the pmf of M for segment S_{i+1} or a function of both. In our algorithm we allow the following four options:

1. pmf at E_s = pmf for S_i.
2. pmf at E_s = pmf for S_{i+1}.
3. pmf at $E_s = Maximum(S_{i+1}, S_i)$.
4. pmf at $E_s = Average(S_{i+1}, S_i)$.

The model generation algorithm detailed in appendix ?? can be used to output the pmf in a formatted human-readable form or generate a *synthetic* trace from the model using a random number generator.

We used a real trace (the Gaussian Elimination trace described in section 3.1) to compare the model generation options discussed above. MEG models were generated using permutations of different options. The results illustrated in Table ?? in the appendix show that the best results were achieved by choosing option 1 for the *slope of a partition with missing data* and option 3 for *slope of transition points*.

3 Model Validation

This section describes how we validate the MEG model. Also in this section we compare how well our model represents the behavior of a system compared to a model that does not take into account the time-varying behavior in systems. This section is organized as follows:

1. First we describe the trace data taken from a real system.
2. We then describe the tests we use to validate our model for the trace data.
3. Finally we describe the results we achieved. We also present the results of the tests in (2) on a *time-homogeneous* semi-Markov model generated from the same trace data.

3.1 Trace Data from Real Systems

3.1.1 Gaussian Elimination

The subject of this case-study is a Gaussian elimination algorithm solving a 64×64 system of equations using 32 processors of an NCUBE multicomputer [RW93]. The trace data obtained by executing this algorithm was filtered to retain time-stamped events which represent the receiving of a message from a processor. The state of the system recorded in the trace is the processor number sending a message.

3.1.2 Word Wide Web Cache Hit Rate

The WWW Cache Hit Rate trace records a "hit" when a requested file is available locally, and records a "miss" when a file must be accessed from the World Wide Web.

3.2 Tests for Model Validation

3.2.1 Autocorrelation

An important guide to the properties of time-dependent data is provided by a series of quantities called sample autocorrelation coefficients, which measure the correlation between observations of a system at different instances of time. Denote the state of a system S at time t to be X_t, and the state of the same system at time $t+k$ to be X_{t+k}. Then the autocorrelation of S at lag k, denoted $r(k)$, is defined in terms of all recorded $X_i s$ which are k time units distant from each other. See Box and Jenkins [BJ76] or any other standard book in time series analysis for details about autorrelation. By varying the lag k we can plot the autocorrelation function $r(k)$ versus k to generate a *correlogram*. The correlogram provides an insight into the probability model which generated the data. We use the correlogram as a test by first generating synthesized traces from our model, and then comparing correlograms obtained from our traces to the correlogram obtained from the original trace.

3.2.2 Categorical Correlation

Categorical correlation measures how closely categorical time series resemble one another. This is a novel concept and is discussed in [RMA95]. We use categorical correlation plots to gauge the accuracy of our model by measuring the correlation between a trace synthesized from our model and the original trace.

3.2.3 Test of Homogeneity

Another validation test we use is the test of homogeneity. It is a statistical test which tests the following null hypothesis:

- H_0 : A trace synthesized from our model is generated from the same distribution that generated the original trace.

The particular test we use is called the *Kruskal-Wallis rank sum test* [Ott88]. The Kruskal-Wallis test is a non-parametric test and thus it does not make any distributional assumptions about the trace data.

To validate our model using the Kruskal-Wallis test we first synthesize traces from our model. A synthesized trace is then compared with the original trace using the Kruskal-Wallis test. To make the test more rigorous we partition the synthesized and the original traces into ten partitions and then apply the Kruskal-Wallis test to each such partition in the synthesized trace and the corresponding partition in the original trace. Denote by n the number of partition pairs for which H_0 is not rejected. If $n = 10$ then all ten partitions of the synthesized trace match their corresponding partitions in the original trace.

3.3 Validation Results for the Gaussian Elimination Trace

The results from the validation tests for the model generated for the Gaussian Elimination trace are described below:

3.3.1 Plot of the sample autocorrelation

The sample autocorrelation plot for the original Gaussian Elimination trace (*ge32.original*), a trace synthesized from the MEG model (*ge32.MEG*), and a trace synthesized from the semi-markov model (*ge32.semi-markov*) is shown in figure 5. Figure 5 clearly shows that the trace ge32.MEG follows ge32.original very closely (especially when compared to the trace ge32.semi-markov).

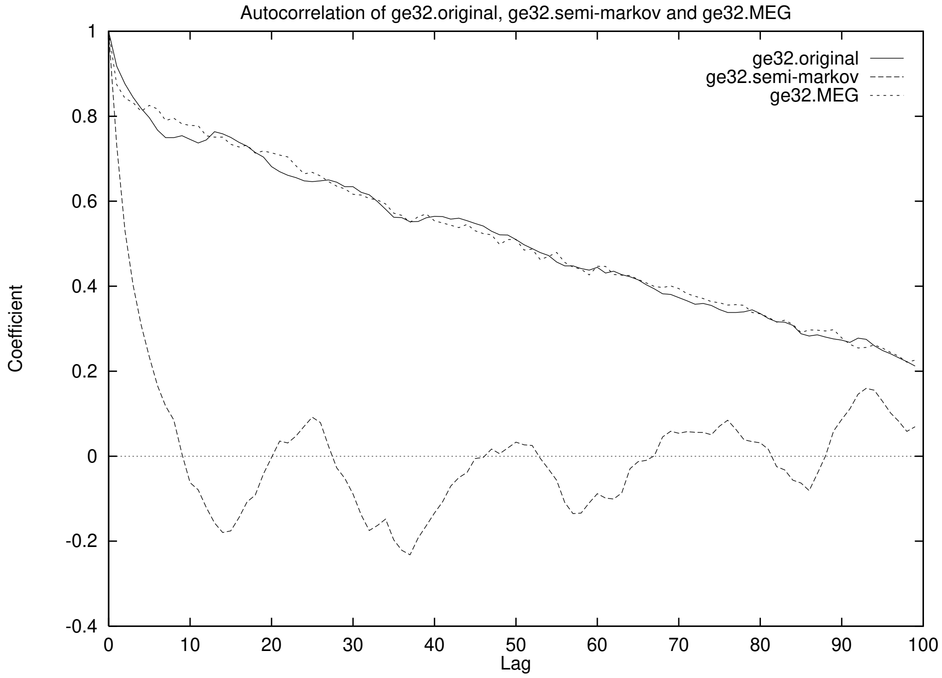

Figure 5: AutoCorrelation Plot for the Gaussian Elimination Trace

3.3.2 Categorical Correlation Plots

The categorical correlation plots are constructed using a moving window 32 events wide. The ordinate in these plots shows the categorical correlation component as a fraction that represents the degree of correlation between traces. The plots in Figures 6 and 7 display the categorical correlation between the original Gaussian Elimination trace (*ge32.original*) and a trace synthesized from the MEG model (*ge32.MEG*). Also shown on each plot is the categorical correlation between the original Gaussian Elimination trace (*ge32.original*) and a trace synthesized from a semi-markov Model (*ge32.semi-markov*). Figures 6 and 7 illustrate that, for each of the categorical correlation components displayed, the trace synthesized from the MEG model follows the original trace more accurately than the trace synthesized from the semi-markov model.

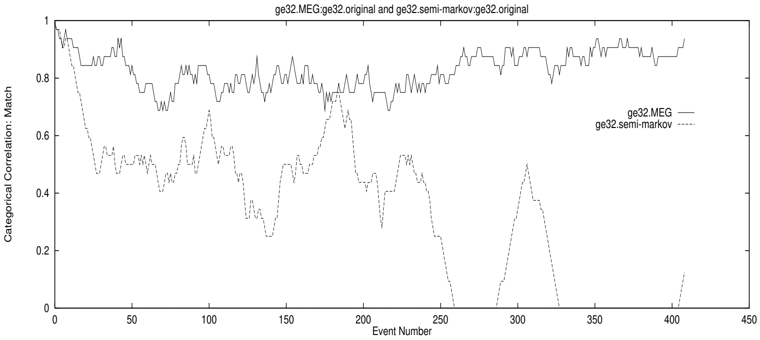

Figure 6: Categorical Correlation Plot: Match Component

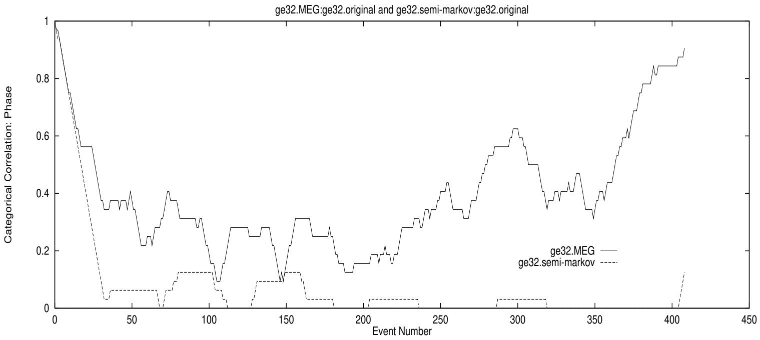

Figure 7: Categorical Correlation Plot: Phase Component

3.3.3 Results of the Kruskal-Wallis test

Table 4 shows the results of the Kruskal-Wallis test on the original Gaussian Elimination trace (*ge32.original*), a trace synthesized from the MEG model (*ge32.MEG*), and a trace synthesized from the semi-markov model (*ge32.semi-markov*). As shown in Table 4, only 5 (out of 10) of the trace ge32.semi-markov pass the Kruskal-Wallis test when compared to ge32.original. In a similar test, all 10 segments of the trace ge32.MEG passed.

Table 4: Results of the Kruskal-Wallis test for the Gaussian Elimination Trace

Kruskal-Wallis test results ($\alpha = 0.05, \chi_2 = 3.84146$)			
Model	Segments passing the test	$\mu(H^{'})$	$\sigma(H^{'})$
Semi-Markov	5 of 10	25.12	27.67
MEG	10 of 10	0.34	0.38

3.4 Validation Results for the WWW Cache Hit Rate Trace

The results from the validation tests for the model generated for the WWW hit rate trace are described below.

3.4.1 Plot of the sample autocorrelation

The sample autocorrelation plot for the original WWW Cache Hit Rate trace (*www.hitrate.original*), a trace synthesized from the MEG model (*www.hitrate.MEG*), and a trace synthesized from the semi-markov model (*www.hitrate.semi-markov*) is shown in figure 8. Fig 8 shows that for this case-study both www.hitrate.semi-markov and www.hitrate.MEG follow the original trace www.hitrate.original. Although the difference between www.hitrate.semi-markov and www.hitrate.MEG is not as drastic as in figure 5, the latter does follow the original trace more closely.

3.4.2 Results of the Kruskal-Wallis test

The Kruskal-Wallis test on the original WWW Cache Hit Rate trace (*www.hitrate.original*), a trace synthesized from the MEG model (*www.hitrate.MEG*), and a trace synthesized from the semi-markov model (*www.hitrate.semi-markov*) confirm the results we drew from the correlogram in figure 8. The trace synthesized from the MEG model still outperforms the trace synthesized from the semi-markov model but not by a large margin. The Kruskal-Wallis test results are shown in Table 5.

Table 5: Results of the Kruskal-Wallis test for the WWW Cache Hit Rate Trace

Kruskal-Wallis test results ($\alpha = 0.05, \chi_2 = 3.84146$)			
Model	Segments passing the test	$\mu(H^{'})$	$\sigma(H^{'})$
Semi-Markov	9 of 10	1.07	1.69
MEG	10 of 10	0.49	0.65

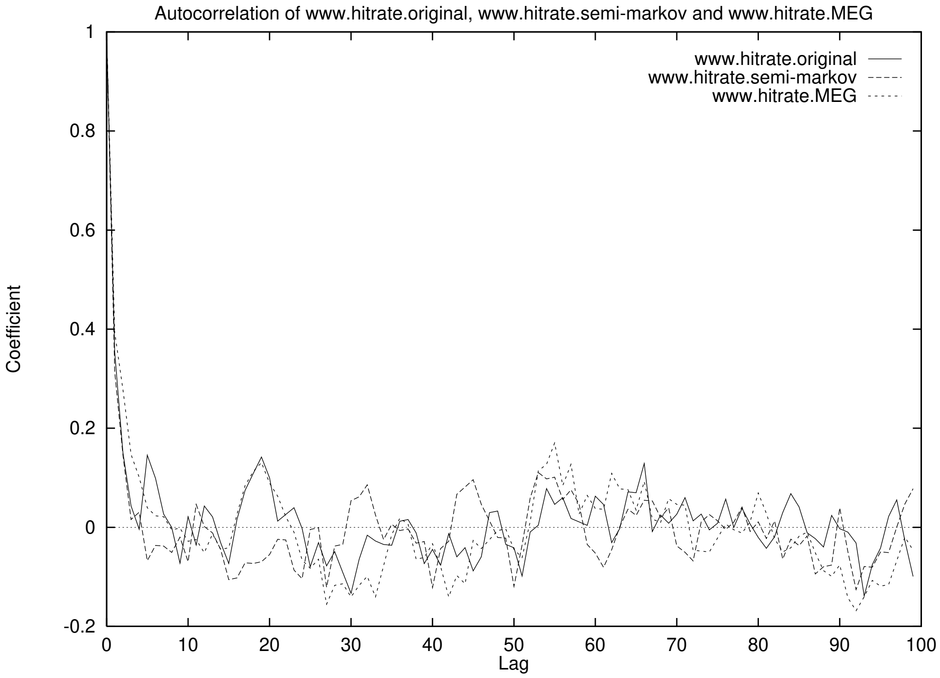

Figure 8: AutoCorrelation Plot for the WWW Cache Hit Rate Trace

4 Conclusions

In this paper we have demonstrated a modeling approach (MEG modeling) that can build accurate models of time-dependent data. The MEG modeling is being refined to construct a model from an ensemble of traces instead of a single trace. The probability mass function from the MEG model is expensive in terms of storage. Work continues towards the development of accurate models which are more compact. The research described in this paper is a part of the Chitra project at Virginia Polytechnic Institute and State University. The algorithms described in the appendix are implemented as a part of Chitra95 the third generation visualization and modeling tool being developed as a part of the Chitra project.

5 Acknowledgments

The authors thank Abdul Waheed [RW93] for providing the Gaussian Elimination trace and details about the instrumentation used to create the trace.

References

[BJ76] G. Box and G. Jenkins. *Time Series Analysis: Forecasting and Control.* Holden-Day, San Fransisco, 1976.

[Jai91] R. Jain. *The Art of Computer Sytems Performance Analysis.* John Wiley & sons,Inc., New York, 1991.

[KMN89] D. Kahaner, C. Moler, and S. Nash. *Numerical methods and software.* Prentice Hall, Englewoods Cliff, N.J., 1989.

[NWK90] J. Neter, W. Wasserman, and M. H. Kutner. *Applied Linear Statistical Models THIRD EDITION.* IRWIN, 1990.

[Ott88] L. Ott. *An Introduction to Statistical Methods and Data Analysis.* PWS-Kent Publishing Company, Boston, 1988.

[RMA95] R. Ribler, A. Mathur, and M. Abrams. Visualizing and Modeling Categorical Time Series Data. *ICASE/LaRC Symposium on Visualizing Time-Varying Data*, September 1995.

[Ros87] S. M. Ross. *INTRODUCTION TO PROBABILITY AND STATISTICS FOR ENGINEERS AND SCIENTISTS.* John Wiley & Sons, New York, 1987.

[RW93] D. T. Rover and A. Waheed. Multiple-Domain Analysis Methods. In *Third ACM/ONR Workshop on Parallel and Distributed Debugging, San Diego*, May 17-18 1993.

A Algorithms

A.1 Partitioning paths in the MEG

A.1.1 Slope-based partitioning

Slope based partitioning is a two step process. The first step generates slopes for every event in the MEG. The final step uses the slopes to generate partitions.

Algorithm 1.0 is a *C-Style* code that describes the generation of slopes.

Input(s): Input to this algorithm is a file that contains the MEG. Denote this file as MEGFILE.

Output(s): Output from this algorithm is a file that contains the slopes. Denote this file as SLOPEFILE.

Description of terms: The following list describes, amongst other terms, the variables and constants used in the algorithm.

- The variable $ThisEv$ is the current event number.
- The variable $PrevEv$ is the event recorded before the current event.
- The variable $ThisNumOccur$ is the cumulative number of occurrences of the state at event = $ThisEv$.
- The variable $PrevNumOccur = ThisNumOccur - 1$.
- The variable $ThisSlope$ = The slope of the path at $event = ThisEv$.
- N is the number of unique states in the trace, or alternatively, the number of paths in the MEG.
- $T[i]$ is the number of $< Event, Slope >$ tuples $\forall$ states $i, 1 \leq i \leq N$ in the file $SLOPEFILE$.
- The constant $threshold$ is the threshold slope that determines if a new segment must be created.
- The constant $runlength$ enforces a lower bound for the number of points in a segment. (e.g if $runlength = 2$ a segment must have at least 2 points.)

Algorithm 1.1 is a *C-Style* code that describes the generation of partitions.

Input(s): Input to this algorithm are two files namely MEGFILE and SLOPEFILE. The contents of these files has been described in section A.1.1.

Output(s): Output from this algorithm is a file that contains the MEG as well as the partitioning events for each MEG path. Denote this file as PARTFILE.

Description of terms: The new terms used in this algorithm are defined below:

- The variable $ThisEv$ is the current event number.
- The variable $ThisNumOccur$ is the cumulative number of occurrences of the state at event $=$ $ThisEv$.
- The variable *ThisSlope* = The slope of the path at *event* = $ThisEv$.
- The variable *PrevSlope* = The slope of the path at event before $ThisEv$ in SLOPEFILE.

A.1.2 Incremental regression partitioning

Algorithm 1.2 is a *C-Style* code that describes the generation of partitions using the incremental regression approach.

Input(s): Input to this algorithm is the file MEGFILE.

Output(s): Output from this algorithm is a file that contains the MEG as well as the partitioning events for each MEG path. Denote this file as PARTFILE.

Description of terms: The incremental regression partitioning described in algorithm 1.2 is based on the following regression function.

$$\hat{Y} = b_0 + b_1 X$$

where

$$b_1 = \frac{\sum_i X_i.Y_i - n\overline{X}.\overline{Y}}{\sum_i X_i^2 - n\overline{X}^2}$$

and

$$b_0 = \overline{Y} - b_1\overline{X}$$

The variables X,Y,b_0, and b_1 correspond to *ThisEv*, *ThisNumOccur*, *intercept*, and *slope* respectively in the algorithm. Also, the standard deviation of the predicted value for *ThisNumOccur* (*stdev* in the algorithm) is given by:

$$s\{\hat{Y}_{new}\} = \sqrt{MSE.(1 + \frac{1}{n} + \frac{(X_{new} - \overline{X})^2}{\sum_i X_i^2 - n\overline{X}^2})}$$

where

$$MSE = \frac{SSE}{n-2}$$

and

$$SSE = \sum_i Y_i^2 - b_0 \sum_i Y_i - b_1 \sum_i X_i.Y_i$$

The above mathematical results have been taken from [NWK90, Ros87].

A.2 Fitting line segments to paths in the MEG

Algorithm 1.3 is a *C-Style* code that describes the linear least-squares fitting of every partitioned segment in a MEG path.

Input(s): Input to this algorithm is the file PARTFILE.

Output(s): Output from this algorithm is a file that contains the MEG as well as the *fitted slope* and *fitted Y-intercept* for every partitioned segment in a MEG path. Denote this file as FITFILE.

Description of terms: The new terms used in this algorithm are defined below.

- $fortfit$ is a fortran routine that does linear least-squares data fitting. This is used without modification from [KMN89].

- The variable $ThisEv$ is the current event number.
- The variable $ThisNumOccur$ is the cumulative number of occurrences of the state at event = $ThisEv$.
- The variable $FittedSlope$ is the slope of the straight line fitting a segment. It is returned from the routine $fortfit$.
- The variable $YIntercept$ is the y-intercept of straight line fitting a segment. It is returned from the routine $fortfit$.

A.3 Generating the model

Algorithm 1.3 is a *C-Style* code that describes the generation of a probability estimate for every partitioned segment in a MEG path.

Input(s): Input to this algorithm is the file FITFILE.

Output(s): Output from this algorithm is a file that contains all the partitioned segments and their corresponding slopes for every MEG path. Denote this file as MODELFILE.

Description of terms: The new terms used in this algorithm are defined below.

- The constant ZERO is user specified. It determines the estimated probability of a partition with missing data (see section 2.6.4).

B Comparison of options for Model Generation

In this section we use a trace that we have introduced before, namely the Gaussian Elimination trace, to compare and evaluate which combination of the model building options provide us with a better MEG models. For model validation we use one of the tools introduced in section 3, namely the Kruskal-Wallis test.

B.1 Results of the Kruskal-Wallis test

Table ?? shows the results of the Kruskal-Wallis test on the Gaussian Elimination trace.

Table 6: Results of the Kruskal-Wallis test on the MEG model for the Gaussian Elimination Trace

Kruskal-Wallis test results($\alpha = 0.05, \chi^2 = 3.84146$)					
Seed	Inter Segment slope	Transition Point	Segments passing test	$\mu(H')$	$\sigma(H')$
1	zero	left	9 of 10	1.13	1.87
1	zero	right	9 of 10	2.23	3.51
1	zero	max	9 of 10	0.99	1.78
1	zero	avg	9 of 10	1.10	1.91
1	megslope	left	10 of 10	0.45	0.50
1	megslope	right	10 of 10	0.84	0.97
1	megslope	max	10 of 10	0.77	0.79
1	megslope	avg	10 of 10	0.56	0.66
2	zero	left	9 of 10	0.78	1.37
2	zero	right	9 of 10	0.86	1.24
2	zero	max	10 of 10	0.59	0.95
2	zero	avg	10 of 10	0.68	1.13
2	megslope	left	10 of 10	0.60	0.94
2	megslope	right	10 of 10	0.48	0.74
2	megslope	max	10 of 10	0.46	0.76
2	megslope	avg	10 of 10	0.48	0.73
99	zero	left	10 of 10	0.93	0.86
99	zero	right	10 of 10	0.46	0.74
99	zero	max	10 of 10	0.34	0.38
99	zero	avg	9 of 10	0.54	0.77
99	megslope	left	10 of 10	0.68	1.12
99	megslope	right	10 of 10	0.58	1.14
99	megslope	max	10 of 10	0.56	0.93
99	megslope	avg	10 of 10	0.58	0.96

```
for each path  i in the MEG output
{
  n = 0;
  PrevEv = 0;
  PrevNumOccur = 0;
  while (path not finished) {
    read (ThisEv, ThisNumOccur) from MEGFILE;
    ThisSlope = (ThisNumOccur - PrevNumOccur)/(ThisEv - PrevOccur);
    if (n != 0) /* ignore the first slope */
    {
       if (n == 1)
       {
          write (PrevEv, ThisSlope) to SLOPEFILE;
          write (ThisEv, ThisSlope) to SLOPEFILE;
       }
       else
       {
          write (ThisEv, ThisSlope) to SLOPEFILE;
       } /* end if  n == 1 */
    } /* end if n != 0 */
  } /* end while */
} /* end for */
```

Figure 9: Algorithm 1.0: Generation of slopes

```
for every path i in the MEG output
{
   j = 0;
   run = 0;
   PrevSlope = 0;
   for  every tuple k in T[i]
   {
      read (ThisEv, ThisSlope) from SLOPEFILE
      read (ThisEv, ThisNumOccur) from MEGFILE
      /* no new partition after the first tuple */
      /* also enforce runlength */
      if ( ( j != 0 ) and (run >= runlength) )
      {
                 if ( |PrevSlope - ThisSlope| > threshold )
                 {
                          put a partition point in PARTFILE;
                          run = 0;
                 } /* end if */
      } /* end if j != 0 */
      PrevSlope = Slope;
      run ++;
      j++;
      } /* end for every tuple */
} /* end for every path */
```

Figure 10: Algorithm 1.1:Generation of Partitions

```
for every path i in the MEG output
{
  read (ThisEv, ThisNumOccur) from MEGFILE;
  xbar = sumx = ThisEv;
  ybar = sumy = ThisNumOccur;
  sumprodxy = ThisEv * ThisEv;
  sumsquarex = ThisEv * ThisEv;
  sumsquarey = ThisNumOccur * ThisNumOccur;
  j = 1;
  count = 1;
  run = 1;

  while ( j < T[i] )
  {
    prevx = ThisEv;
    prevy = ThisNumOccur;
    read (ThisEv, ThisNumOccur) from MEGFILE;
    if ( (j != (T[i]-1)) and (run >= runlength) )
    {
      slope = (sumprodxy - (run*xbar*ybar))/
              (sumsquarex - (run*xbar*xbar));
      intercept = ybar - slope * xbar;
      SSE = sumsquarey - (intercept * sumy)
                       - (slope * sumprody);
      MSE = SSE/(run - 2);
      stdev = sqrt(MSE * (1 + (1/run) +
                          (((ThisEv - xbar)*(ThisEv - xbar))/
                          (sumsquarex - run * xbar * xbar)))
                  );
      ypredicted = intercept + slope * ThisEv;
      ypredlow = ypredicted - t-STAT * ThisEv;
      ypredhigh = ypredicted + t-STAT * ThisEv;
      if ( ThisNumOccur > ypredhigh ||
           ThisNumOccur < ypredlow ))
      {
         run = 0;
         sumx = sumsquarex = sumy = sumprodxy =0;
         xbar = ybar = sumsquarey = 0;
         mark the beginning of a NEW PARTITION in PARTFILE;
      } /* end if */
    } /* end if j */
    run++;
    sumprodxy = sumprodxy + ThisEv * ThisNumOccur;
    sumx = sumx + ThisEv;
    sumsquarex = sumsquarex + (ThisEv * ThisEv);
    sumsquarey = sumsquarey + (ThisNumOccur * ThisNumOccur);
    if (run > 0) xbar = xbar + (ThisEv - xbar)/run;
    else xbar = ThisEv;

    if (run > 0) ybar = ybar + (ThisNumOccur - ybar)/run;
    else ybar = ThisNumOccur;
    j++;
  } /* end while */
} /* end for every path */
```

Figure 11: Algorithm 1.2:Generation of partitions

```
for every path i in the MEG output
{
   for every segment j in path i
   {
     invoke fortfit for this segment;
     write (FittedSlope, YIntercept) for this segment into FITFILE;
   } /* end for every segment */
} /* end for every path */

for every path i in the MEG output
{
   for every segment j in path i
   {
     write (ThisEv, ThisNumOccur) into FITFILE;
   } /* end for every segment */
} /* end for every path */
```

Figure 12: Algorithm 1.3:Fitting lines to MEG paths

```
for every path i in the MEG output
{
   for every segment j in path i
   {
      read (FittedSlope, YIntercept) for this segment from FITFILE;
      while (not end of segment j)
      {
         read (ThisEventNum, ThisNumOccur) from FITFILE;
         store the first such tuple as (FirstEventNum, FirstNumOccur);
      } /* end while */
      if ( j is not the first segment in the path i)
      {
        if (FirstEventNum != PrevLastEventNum + 1)
        {
           if (ZERO) slope = 0;
           else
           {
             slope = (FirstNumOccur - PrevLastNumOccur)/
                          (FirstEventNum - PrevLastEventNum);
           } /* end else */
           if (slope = FittedSlope) {
              write (PrevLastEventNum+1, FittedSlope) to MODELFILE;
           }
           else {
              write (PrevLastEventNum+1, slope) to MODELFILE;
              write (FirstEventNum, FittedSlope) to MODELFILE;
           } /* end if */
        } /* end if */
      }
      else {
        write (FirstEventNum , FittedSlope) to MODELFILE;
      } /* end if */
      PrevLastEventNum = ThisEventNum;
      PrevLastNumOccur = ThisNumOccur;
   } /* end for every segment */
} /* end for every path */
```

Figure 13: Algorithm 1.4:Generating the MEG model