A Stochastic Process Model

for

Transient Trace Data

by

Anup Mathur

Dissertation submitted to the faculty of the

Virginia Polytechnic Institute and State University

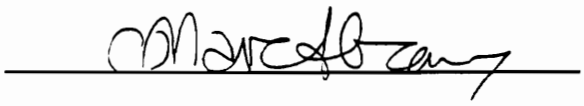
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

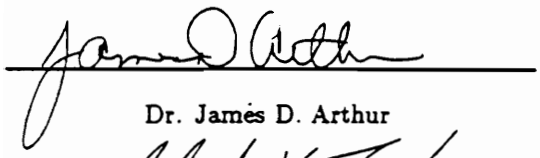
in

Computer Science

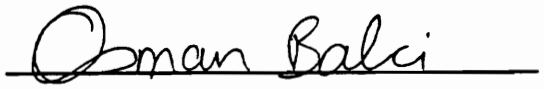
APPROVED:



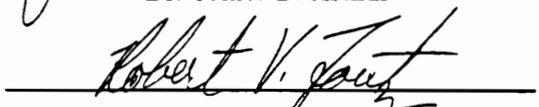
Dr. Marc Abrams, Chairman



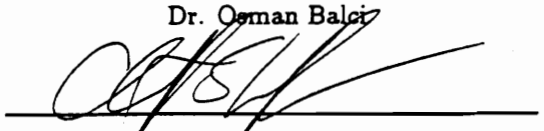
Dr. James D. Arthur



Dr. Osman Balci



Dr. Robert V. Fontz



Dr. Clifford A. Shaffer

October 1996

Blacksburg, Virginia

Keywords: Workload Models, Program Behavior, Categorical Data

A Stochastic Process Model for Transient Trace Data

by

Anup Mathur

Committee Chairman: Dr. Marc Abrams

Computer Science

(ABSTRACT)

Creation of sufficiently accurate workload models of computer systems is a key step in evaluating and tuning these systems. Workload models for an observable system can be built from traces collected by observing the system.

This dissertation presents a novel technique to construct non-executable, artificial workload models fitting transient trace data. The trace can be a categorical or numerical time-series. The trace is considered a sample realization of a non-stationary stochastic process, $\{X_t\}$, such that random variables X_t follow different probability distributions. To estimate the parameters for the model a *Rate Evolution Graph* (REG) is built from the trace data. The REG is a two-dimensional Cartesian graph which plots the number of occurrences of each unique state in the trace on the ordinate and time on the abscissa. The REG contains one path for all instances of each unique state in the trace. The derivative of a REG path at time t is used as an estimate of the probability of occurrence of the corresponding state at t. We use piecewise linear regression to fit straight line segments to each REG path. The slopes of the line segments that fit a REG path estimate the time dependent probability of occurrence of the corresponding state. The estimates of occurrence probabilities of all unique states in the trace are used to construct a time-dependent joint probability mass function. The joint probability mass function is the representation of the *Piecewise Independent Stochastic Process* model for the trace. Two methods that assist to compact the model, while retaining accuracy, are also discussed.

The modeling approach is evaluated by the application of the model to twelve test case traces. Traces with a small number of unique states, low degree of autocorrelation, and

REG paths that can be fitted by a small number of straight line segments are the best candidates for the model. The model is adequately accurate and parsimonious for test case traces such as a periodic trace, a trace of World Wide Web traffic, and a trace from a transaction processing system. For traces with high autocorrelation, such as traces of *Variable Bit Rate* encoded video, the model is not adequately accurate.

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Marc Abrams for his support throughout my tenure as his advisee at Virginia Tech. I would like to thank Dr. James .D. Arthur, Dr. Osman Balci, Dr. Robert V. Foutz, and Dr. Clifford A. Shaffer for being in my committee and providing input and support towards the completion of my dissertation. I would also like to thank Dr. Verna Schuetz for providing me with sound academic advice during my tenure at the Computer Science department. The faculty and staff of the Computer Science department made my stay at Virginia Tech a pleasant one, for this I thank them.

I would like to gratefully acknowledge the constant support, advice, and love I received from my parents. It is due to their encouragement that I have been able to achieve this academic milestone.

Finally, I thank my wife Alexandra Kapatou who has stuck with me through periods of anxiety and exaltation. She has provided emotional support and loving companionship throughout our life together. Her help during the final months of my life as a doctoral candidate has been crucial to my success.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

Formation of mathematical models to explain the behavior of physical systems is critical to science. Some physical systems are governed by laws that deterministically explain the behavior of such systems. In contrast many other systems depend on factors that are unknown and can best be probabilistically estimated from data collected by observing such systems. Observed data from a system can be a sequence of values collected over a period of time. The terms *trace data* and *time series data* are often used to describe such data. In this document we use the terms trace data and time series interchangeably to describe data collected from observable systems.

Stochastic processes represent an established class of analytic tools for building probabilistic models of time series data. The work described in this dissertation is focused on building a stochastic process model applicable to trace data collected from computer systems and communication networks. Such models capture the behavior of the system generating the trace and can consequently be used to evaluate and tune the performance of the system. In the field of computer performance analysis the term *non-executable artificial workload model* [Ferrari et al., 1983] is often used for models of computer performance trace data.

A *stationary* stochastic process models time series data such that the probability laws governing the process do not change with time. Although the assumption of stationarity provides us with powerful analytical tools to build a model, this assumption is almost always violated by real trace data. A large segment of computer performance trace data can aptly be described as following probability laws that vary with time. In our exploratory study of several traces from the computer performance analysis domain we have observed traces

that follow different probability distributions in disjoint *intervals* of time. As an example consider the 2 dimensional X-Y plot of the GE Message trace (Section 6.1) in Figure 1.1. The time varying behavior exhibited by traces such as the one shown in Figure 1.1 is often

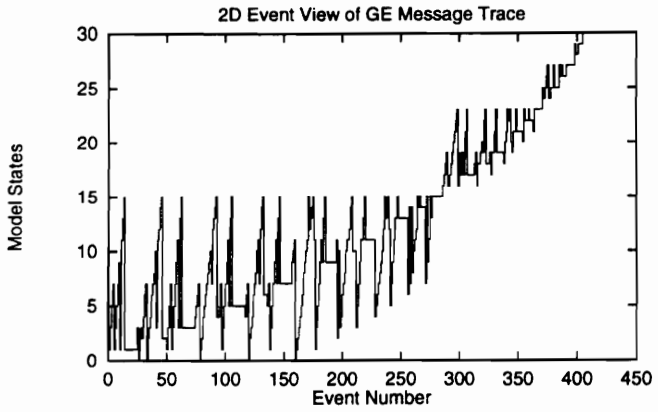


Figure 1.1: 2D View of the GE Message Trace

called *transient behavior*. Various traces of program behavior such as traces of memory references and traces of messages routed to a processor show transient behavior. Other examples of transient traces are compressed video packet size traces and traces of Ethernet traffic. Twelve such traces are introduced in Section 6.1.

1.1 Problem Statement

The problem investigated by this dissertation is to develop a model for transient trace data, which includes developing methods to estimate the parameters of the model and evaluating the quality of the model by its application to a variety of test case traces.

1.2 Summary of the Solution Approach

We introduce the *Piecewise Independent Stochastic Process* (Section 3.4) as a model for transient trace data. Our model describes the trace data as a family of independent probability mass functions. We partition the time axis into intervals such that each probability

mass function accurately represents the behavior of the trace within an interval. The probability mass function corresponding to an interval describes the probability of occurrence of each unique state in the trace within that interval. We estimate the probability of occurrence of a state by the rate of occurrence of the state. Intervals are also defined in terms of the rate of occurrence of states. The starting point of an interval is the time when the rate of occurrence of any unique state in the trace changes significantly from its previous value. Thus, the determination of rate of occurrence of states is central to the estimation of the joint probability mass function for the trace. To estimate the rate of occurrence of a state we utilize a visualization of trace data known as the Rate Evolution Graph (REG). The REG, which is described in Section 3.2, is a two dimensional plot. The ordinate in a REG is the number of occurrences of unique states in the trace. The abscissa is time. The set of plotted points for each unique state form a path in the REG. The rate of occurrence of a state can be obtained by computing the derivative of the REG path corresponding to that state. We estimate the rate of occurrence of a state within an interval by the slope of the straight line fitting the segment of the REG path, corresponding to that state, that is enclosed within the interval. A collection of the rates of occurrence of all the unique states in the trace, within an interval, is the estimated probability mass function of the states for that interval. A collection of the probability mass functions for all intervals is the estimated joint probability mass function for the entire trace.

1.3 Original Contributions

We present a list of the original contributions that follow from this research below.

- We introduce the *Piecewise Independent Stochastic Process* (Section 3.4) as a model for transient trace data. Our model describes the trace data as a family of independent probability mass functions. Each probability mass function represents the behavior of the trace within an interval of time.

- Our model can be used to generate synthetic traces that are equal in length to the trace being modeled. Our model captures the behavior of the original trace in any distinct interval of the index parameter space of the original trace.

- We also present a procedure to estimate the collection of intervals and corresponding probability mass functions that completely describe a piecewise independent stochastic process model for a trace (Chapter 4). We present two approaches, slope-based partitioning and incremental regression partitioning, to estimate the intervals.

- The parameters of the piecewise independent stochastic process model are estimated based on the REG which is a visualization of categorical trace data. This allows us to model categorical as well as numerical trace data.

- The procedure to estimate the parameters of the piecewise independent stochastic process allows automated model generation with minimum user interaction. The implementation of the procedure to estimate the model parameters as CHITRA utilities provides an automated modeling tool.

1.4 Organization of the Dissertation

This rest of this dissertation is organized as described below. In Chapter 2 we discuss the role of modeling in the domain of performance analysis. We also discuss current research in performance and workload modeling. In Chapters 3 and 4 we define the piecewise independent stochastic process as a model for transient trace data and propose a method to estimate the parameters of this model. Chapter 5 introduces two techniques that assist in building a parsimonious model. In Chapter 6 we introduce 12 traces representing real and simulated workloads that are then used to evaluate our modeling approach. The Appendix B describes the tools for building a piecewise independent stochastic process model from trace data. These tools are implemented within the CHITRA [Abrams et al., 1992] performance analysis and visualization system.

Chapter 2

LITERATURE SURVEY

In this chapter we describe the role that modeling plays in the field of performance analysis. We describe contemporary tools that are used to build models from software performance data. We discuss the evolution of CHITRA a software performance modeling and visualization tool developed at Virginia Tech. Finally, we describe some important stochastic process models that are extensively used in the field of computer performance analysis.

2.1 Modeling as a Tool in Performance Analysis

Historically, model building or analysis has been one of the cornerstones of performance evaluation research. Abrams [Abrams, 1986] categorizes analytical software performance evaluation literature into several areas, such as queuing network models, complexity analysis, and state space analysis. We shall limit our discussion to state space analysis, because it is most relevant to the research proposed in this document. Hereafter, the word modeling has been used instead of state space analysis for brevity. State space analysis characterizes program behavior by the concept of a state and feasible transitions between states.

2.1.1 Modeling used for Performance Diagnosis

Analytical techniques formulate a model of program behavior. Such a model describes program performance metrics in terms of certain program parameters. The objective is to obtain functional dependence of performance metrics on parameter values. The term *parameterized model* has been used in this document for such a model. Two approaches to

this end have been used in literature namely: *forward analysis* and *backward analysis.*

Forward Analysis: Forward analysis techniques construct a model of program behavior based on program code before the program is executed. In [Berry and Jefferson, 1985] the authors describe a forward analysis technique that can provide lower bounds on the execution times of object-oriented distributed simulations. The technique involves the construction of a graph. The nodes in the graph represent events in the simulation and the edges represent the time dependency (ordering) of the events required to execute the simulation correctly. Nodes in the graph are weighted with estimates of the computing time required to execute the event(s) they represent. Edges are weighted with estimates of the communication time between the nodes they connect. The authors choose the weights for nodes and edges based on prior knowledge about the hardware platform on which the simulation is expected to run. A critical path algorithm is applied to the graph to find the longest weighted path in the graph. The sum of weights on the critical path represents a lower bound on the execution time of the simulation program on the chosen platform. Other examples of forward analysis are the geometric concurrency model [Abrams, 1986], stochastic automata [Plateau and Atif, 1991], and stochastic petri nets [Molloy, 1982].

Backward Analysis: Backward analysis techniques, on the other hand, construct models of program behavior from trace data collected after executing a program. An example of backward analysis is demonstrated by the assortment of performance tools that accompany the Time Warp Operating System (TWOS) [JPL, 1991]. TWOS is a special purpose operating system designed to support parallel discrete event simulations. TWOS runs on platforms such as the BBN GP-1000 multiprocessor on top of the MACH operating system. After every execution run of a simulation program, TWOS writes "raw" statistics to a file called XL_STATS. Tools called *check* and *measure* can be used to produce summary statistics for many runs of the same simulation program for different values of parameters such as number of processors. Furthermore, summary statistics can be used to compute perfor-

mance metrics such as speedup curves. In order for a simulation program to achieve good performance running on multiple processors, it is necessary that the computational load be distributed evenly over all the processors. One way of achieving this goal is to initially assign simulation objects to processors in an "optimal" way, so that the computational load over all the processors remains balanced during the execution of the simulation. Such an optimal initial assignment of simulation objects to processors is known as a *balanced configuration.* TWOS offers tools called *reduce* and *balt* that use the information in XL_STATS from multiple runs of a simulation program to obtain a balanced configuration. Other examples of backward analysis include CHITRA [Abrams et al., 1992], a knowledge based monitoring and control tool for distributed systems [Hitson, 1990], and IPS-2 [Miller et al., 1990]. IPS-2 is described below.

IPS-2 IPS-2 [Miller et al., 1990] is a performance analysis system for parallel and distributed programs from the Computer Science Department at the University of Wisconsin-Madison. The IPS-2 system offers the following services to its users:

- It provides user-guided instrumentation of source code.
- It displays performance metrics chosen by the user.
- It employs analytical techniques to guide the user in the identification of bottlenecks.

IPS-2 presents the parallel or distributed program to the user at various levels of abstraction. At one end of this spectrum is the *program level*, where the entire program is viewed as a black box, while at the other end is the *primitive activity level* which includes low level events such as process creation and message passing. The user can therefore monitor the program at the level of granularity that he(she) prefers by instructing the tool to instrument the source code at the desired level and display appropriate performance metrics. IPS-2 needs compiler, and operating system level support to automatically generate trace data. The trace data is analyzed off-line to produce the performance metrics selected

previously. Trace data is also utilized to construct the *program activity graph (PAG)*. Analytical techniques such as *critical path analysis*, and *phase behavior analysis* subsequently use the *PAG* to identify performance bottlenecks in the source code.

2.1.2 Workload Modeling

Another important application of modeling within the domain of performance analysis is workload modeling. In contrast to models used for performance diagnosis, models of a workload may not yield performance metrics directly. Instead they provide an analytical tool that can generate *synthetic* workloads on demand.

Jain [Jain, 1991] describes a *workload* as *a list of service requests to a system.* To measure and analyze the performance of a system, workloads that characterize the usage of the system are used as an input to the system. A *real workload* is one that is observed during the operation of a system. It is usually not possible to replicate real workloads. This limits their usage in performance analysis. In practice a model of the real workload is constructed. Such a model is then used to generate *synthetic workloads* that have characteristics similar to the real workload. Replications of synthetic workloads can be generated on demand. Consequently, synthetic workloads form the basis of many performance analysis experiments. The art and science of constructing models of real workloads therefore constitutes a significant part of research in computer performance analysis. In this dissertation we are concerned with building stochastic models of workloads of program behavior and network traffic.

Workload models for a system can be built from traces collected by observing the system. Such a trace represents the workload. For the purpose of this document we use the term trace to imply a workload.

2.2 Stochastic Process Models

Bhat [Bhat, 1984] defines a stochastic process, $\{X_t\}$, as a family of random variables $\{X_t,\ t \geq 0\}$ indexed by the *parameter* (usually time) t. The set of all possible values that can be assumed by a stochastic process is called its *state space*. Similarly, the set of all the possible values of the indexing parameter is called the *parameter space* of the stochastic process. The symbols **S** and **T** are often used to denote the state and parameter spaces respectively of a stochastic process. The state space of a stochastic process may be discrete or continuous. Accordingly, the stochastic process is termed *discrete-state* or *continuous-state*. Similarly, a stochastic process can also be *discrete-time* or *continuous-time*. A set of instances of the stochastic process $\{X_t\}$ is called the *realization* or *sample realization* of the stochastic process.

Stochastic processes play a major role in performance modeling of computer hardware and software. In the sections below we describe some commonly used stochastic processes that are used to model trace data.

2.2.1 Markov Chains and Processes

A Markov process is a continuous-time stochastic process $\{X_t\}$ which displays the Markov dependence [Bhat, 1984] property described below.

Definition 1 *A stochastic process, $\{X_t\}$, obeys the* Markov Property *if the conditional distribution of X_t for given values of $X_{t_1}, X_{t_2}, \ldots, X_{t_n}$ depends only on X_{t_n} which is the most recent known value of the process, that is*

$$P[X_t \leq x \mid X_{t_n} = x_n, \ldots, X_{t_1} = x_1] = P[X_t \leq x \mid X_{t_n} = x_n]$$

Markov processes form a large family of interesting stochastic processes.

A discrete-time Markov process is called a *Markov chain*. An important feature of a Markov chain is its *transition probability matrix*. The *k-step* transition probability matrix $\mathbf{P^k}$ is defined as the matrix of conditional probabilities $[P[X_{t_m+k} = i \mid X_{t_m} = j],\ \forall i, j \in \mathbf{S}]$.

Markov process models are used by Heyman [Heyman and Lakshman, 1996] to model compressed video and Abrams [Abrams et al., 1994] to model program behavior.

2.2.2 Poisson Process

The Poisson process is a discrete state, continuous time stochastic process that obeys the Markov dependence property. It therefore belongs to the family of Markov processes. The Poisson process is often described as a *counting process* that counts the number of times a specific randomly occurring event occurs in a given interval of time. Such a specific event, for instance, can be the arrival of a person to join a queue of waiting people. The times between such events often called the *inter-arrival* times are independent, and identically distributed as exponential random variables. Poisson processes are commonly used to model bursty traffic such as the traffic generated by phone calls on a telephone network.

2.2.3 Self Similar Stochastic Process

Self-similar stochastic process models were introduced to the field of statistics by Mandelbrot [Beran, 1994]. The self-similar models address two characteristics of observed time series data which is not explained by more conventional models namely:

1. Certain observed time series such as the *Yearly minimum water levels of the Nile river* [Beran, 1994] show that the autocorrelation of the time series does not exponentially decay to zero. This phenomenon of *long range dependence* (LRD) is contrary to the assumptions of many stochastic process models such as the Markov model.

2. Observed time series such as the one mentioned above also display the *self similarity* characteristic when the parameter space of the time series is rescaled.

We discuss the second property in more detail as it is the basis of the definition of self-similar stochastic processes. Let us consider a time series $\{X_t, t = 1, 2, .., k\}$. First we divide this time series into m non-overlapping blocks denoted, X_t^1, X_t^2, .. X_t^m, such that each such block is a also a time series. Next, we sum the terms in each block and thus derive an

aggregate time series denoted $\{X_T, T = 1, 2, .., m\}$. If the original time series $\{X_t\}$ was a sample realization of a self-similar stochastic process then the distribution of $\{X_t\}$ must be equal to the distribution of $\{X_T\}$.

Beran [Beran, 1994] provides the following rigorous definition: *Let Y_t be a stochastic process with continuous time parameter t. Y_t is called self-similar with self-similarity parameter H, if for any positive stretching factor c, the rescaled process with time scale ct, $c^{-H}Y_{ct}$, is equal in distribution to the original process Y_t.*

Recent research has rejected well known and understood stationary stochastic process models such as the Poisson model. In [Paxson and Floyd, 1994] Paxson and Floyd show that the Poisson model does not describe LAN traffic adequately. Leland and others [Leland et al., 1994] have shown that ethernet traffic shows *long range dependence* which make Poisson and Markov models inappropriate for it. Such traffic can be modeled by *self-similar* stochastic processes [Beran, 1994]. Self-similar stochastic processes are stationary stochastic processes which exhibit long range dependence. This is in contrast to stationary stochastic processes such as Markov processes which are *memoryless.*

Garrett [Garrett and Willinger, 1994] has used a *Gamma/Pareto* self-similar stochastic process to model variable bit rate (VBR) video packet traffic on a LAN. Crovella [Crovella and Bestavros, 1995] has shown some workloads of WWW traffic to be self-similar also.

2.3 Modeling Trace Data

In this section we discuss contemporary research towards modeling trace data. The research presented in this section is related to modeling trace data that represents workloads of network traffic, compressed video, and program behavior.

2.3.1 Modeling Program Behavior

With the introduction of multiprogrammed computer systems in the 1960's, memory management for multiprogrammed operating systems emerged as a field of intense research. The objective of this research was to develop memory management techniques that maximized the performance of programs running on the multiprogrammed computer. To develop general purpose memory management techniques it became necessary to study the memory reference behavior of real programs running on the computer system. Thus the term *program behavior* was coined. Informally, the term program behavior represents the demand for computer system resources made by a program over time. Program behavior is often represented by a trace that represents the demand for a particular computer resource over time. An example is the trace of memory references made by a program.

Initial models of program behavior were motivated by the need to understand and identify patterns of demand for virtual memory made by a running program and to evaluate techniques such as page replacement algorithms. Hence most studies of program behavior model memory reference traces. An early outcome of studying program behavior was the discovery of the *working set model* of a program by Denning [Denning, 1968].

Almost all models of program behavior assume the memory reference trace collected from a program to be the sample realization of a stochastic process. Earlier models of program behavior (1965-1974) used simple stationary stochastic processes to model program behavior. Denning [Denning, 1980] describes the *Simple Renewal Model* (SRM) and *Independent Reference Model* (IRM) as examples of such models. SRM treats successive references to the same memory segment in a memory reference trace as recurrent events of a renewal process. Thus each memory segment referenced in the memory reference trace corresponds to a renewal process. Also, all renewal processes thus defined are uncorrelated with each other. IRM regards the memory reference trace as a sequence of independent random variables with a common stationary distribution.

Simple models such as SRM and IRM described above do not account for the fact

that real programs have different phases of behavior. The memory reference behavior of a program may differ substantially from one phase to another. Denning proposed a two layered *phase transition model* in [Denning, 1980]. The *macro model* in this two layered model was a semi-Markov chain whose states were mutually disjoint locality sets. Each locality set corresponded to a phase. The *micro model* described the behavior of the program within a phase. Simple stochastic processes such as SRM or IRM were proposed as micro models. The PISP model described in this dissertation is similar to the idea of the two layered scheme proposed by Denning. The phases in Denning's proposed model correspond to the intervals in the PISP model. Also, instead of using a semi-Markov process as a macro model we use an independent stochastic process (Section 3.3).

2.3.2 Modeling Traces of Program Behavior using CHITRA

The CHITRA performance visualization and analysis system being developed at Virginia Tech is an attempt towards visual presentation and manipulation of performance data with the objective of constructing models to describe program behavior. Among contemporary performance analysis systems [Reed et al., 1992, Goldberg and Hennessy, 1993, Martonosi et al., 1992, Miller et al., 1990], it is the only system to do so.

CHITRA takes as input a collection of traces derived from trace files produced by running instrumented source code. A trace records the state of a system as a function of time. Examples of "state" are the memory addresses referenced, the control point of processes, the values of data structures, or a combination of these. Current literature includes several examples of memory-oriented performance tuning [Martonosi et al., 1992, Goldberg and Hennessy, 1993], code-oriented performance tuning [Graham et al., 1982], and performance tuning of interconnection networks [Ahluwalia and Singhal, 1992]. CHITRA subsumes the function of such tools because of its ability to analyze an arbitrary state representation. CHITRA contains a module that generates an empirical model from a set of traces. However, building a model of the raw traces typically results in a model with a large, complex state space. Therefore CHITRA permits the reduction of the state space size

by providing a set of transforms. These transforms map a trace into a simpler trace, leading to a simpler model with a smaller state space size. Each transform has a visual analog, and is selected by a human through the visualization component of CHITRA. The transforms in CHITRA work with *any* type of software. In fact, CHITRA analyzes categorical time series data, and therefore could be used with non-software applications.

The evolution of CHITRA has proceeded hand-in-hand with its application to actual software problems. Published case studies using CHITRA include: the dining philosophers problem [Abrams et al., 1992] and a commercial implementation of the TCP/IP protocol for the MS-DOS [Abrams et al., 1992].

The following description overviews the four successive generations of the CHITRA performance analysis system. The first two generation tools, CHITRA91 and CHITRA92, use a homogeneous semi-Markov process model to describe program behavior. Third generation tool CHITRA93 adds the CHAID based model. CHITRA94 and CHITRA95 add new visualizations for categorical data and also add modeling for transient trace data.

CHITRA91 is a prototype and is described in [Abrams et al., 1992]. The input to CHITRA91 is a textual trace file that contains the trace [Doraswamy, 1991]. The trace, once loaded, can then be displayed by CHITRA91 for visual inspection and *visual editing* by the user. Visual editing is the user directed transformation of the trace. It allows the user to edit the graphical views displayed by CHITRA using a mouse. Visual editing reduces the size of the state space and helps the user build an accurate model of the trace. CHITRA91 supports four *transforms* for visual editing of the trace, defined in [Abrams et al., 1992]. *Clipping* is useful in eliminating the initial and final transient portions of the trace. *Aggregation* is useful in capturing deterministic patterns in a trace. CHITRA91 aggregates the states that form a deterministic pattern into one aggregate state. A semi-Markov model thusly formed describes the stochastic and deterministic behavior of the program. *Projection* is a transform that allows the user to manually aggregate states. It is specified textually rather than visually. It also reduces the state space size. Finally, *filtering* is useful in lumping states that occur infrequently into one state, thereby reducing the state space size. The

final output from CHITRA91 is a homogeneous semi-Markov model that describes program behavior shown by the trace.

A unique aspect of CHITRA91 is its view of the trace as a *signal* as used in electrical engineering. Operations such as filtering which are often applied to signals are supported by CHITRA91. CHITRA91 provides three *views* of the trace represented as a signal: event domain, time domain, and frequency domain. These views permit the human user to look at the trace as time-series data.

CHITRA92

Application of CHITRA91 to the case-studies published in [Abrams et al., 1992] prompted the evolution of CHITRA92. Unlike CHITRA91, which allowed only one trace to be viewed at a time, CHITRA92 allows viewing of multiple traces at a time. It also allows multiple views of the same trace at a time. CHITRA92 provides two new views: *text* and *model*. The model view displays the semi-Markov model for a trace and the text view displays the trace textually. CHITRA92 keeps all displayed views of the same trace instance consistent. In other words if a transform is applied to one displayed view of a trace instance, all other displayed views of the same trace instance are updated to reflect the trace transformation.

CHITRA93

The third generation tool, CHITRA93, analyzes an ensemble of traces from the same program simultaneously. Whereas CHITRA92 generates one model per trace, CHITRA93 generates one parameterized model for an ensemble of traces. Here, parameter refers to variables that the program depends on, such as input(s) to the program and defined constants in the program. The approach is rooted in the idea that by actually analyzing the behavior of the program for a set of program parameters we should be able to predict the behavior of the program for a range of program parameter values.

CHITRA94

Our past experience in using CHITRA93 as well as other visualization and statistical analysis tools (chiefly Pablo, SAS, S-Plus, and UniFit II) suggested several problems in the way people designed such tools. A tool should be user extensible in a way that does not require one to understand, recompile, or link existing source code or libraries. It should be possible to add a new module, such as a model, in a few evenings of work. A graphical user interface (GUI) for a user added module should automatically be generated (unless the user adds a complex visualization). Any crash of user written code should not be propagated to the rest of the tool (in Contract, a bug in the Motif or X toolkit code of a newly added module can crash a monolithic X-windows performance tool). It should be possible to incorporate the tool into another tool (e.g., incorporate CHITRA94 into an existing CASE or simulation model development tool). Experience from older performance analysis tool projects show that integration is very clumsy with a monolithic tool with its own custom GUI [ICASE, 1993]. Finally, the tool should be portable across hardware, operating systems, and GUI's (especially Unix/Motif and Windows for PC's). We envision a performance analyst or student bringing home an ensemble on a disk to analyze on a IBM PC-compatible computer, or analyzing data in the field with a laptop, rather than being tied to a Unix workstation in an office or lab.

CHITRA94 is a collection of small programs, following the Unix model of a library of programs with simple functions that read standard input and write standard output. The system documentation includes a set of Unix-style manual pages (stored as text files on a non-Unix system).

Data is passed between programs in the a Canonical Format (CF) file. A CF file represents data as space separated columns of data augmented with header lines that contain a pound sign in column one. Each header line lists one attribute of the data. We chose to define a new format (i.e., CF) over an existing one, such as Pablo's SDDF [Reed et al., 1992], specifically to use a format that can be directly usable by Unix utilities (e.g., `sort` and

`grep`) or exported into other performance analysis tools. For example, one could save the programming effort of writing a function to sort data in a CF file by using the standard Unix command `sort` as follows: `grep -v '^#' <filename> | sort`. (`grep` removes the header lines because they begin with a pound sign.)

CHITRA94 is user extensible in two ways. First, a user is free to write new commands to add to CHITRA94 using any programming language. The user program simply reads and writes CF files, which are used to communicate data between all CHITRA94 commands (functions to simplify CF file I/O are provided). Second, most operating systems provide a programmable command shell, and this provides an easy way for a user to write analysis procedures that invoke CHITRA94's commands. Furthermore, command shell scripts are usually interpreted (as opposed to compiled), and this allows a user to quickly try out a small algorithm and quickly debug it. We encourage the use of the Bourne shell (*sh*), because it is available for DOS as well as provided with all Unix systems, and thus increases portability of user written shell scripts.

CHITRA94 has eight families of commands. Each command name consists of two to four terms connected by dots. The first term is always a verb. The last term is always a noun. The other terms, if any, always contain the object of a prepositional phrase. For example, *a.c.ens* means "assist [verb] by closing [object] an ensemble [noun]." Under DOS, the dots are dropped when typing the command names. The purpose of each command family is described below.

Assistants: Provide miscellaneous functions (e.g., *a.c.ens*), and utilities to help write shell scripts (e.g., *a.mod.set*, assist by modifying a set).

Descriptors: Describe various objects in CHITRA.

Extractors: Instantiates an ensemble, and extracts information or answers a query about the ensemble. Writes a CF file. Example: Output all state occupancy time samples of state S in the ensemble (*e.all*, extract all).

Generators: A collection of programs that factor out algorithms common to two or more modelers, synthesizers, and viewers. For example, all modelers have a state transition matrix of some form. One generator (*g.amOrStm* - generate an adjacency or state transition matrix) does the job for all.

Furthermore, there is a separation between the algorithm that computes a result for the user (i.e., the points defining an autocorrelation graph) and the module that actually displays the result (i.e., a graph in a GUI window of size chosen by the user). The former is a generator, and the later is a viewer.

Modelers: Each model program executes one or more generators to actually compute a model, and then executes one or more viewers to display the resultant model. For example, to generate the semi-Markov model (*m.sm.ens* command), the following modules are called: *e.all* to extract the raw data, *g.amOrStm* to generate the transition matrix, *v.matrix* to display it, *g.ots* to generate occupancy time statistics, and *v.ots* to display the statistics.

Testers: Reads the output of an extractor, and performs a statistical test to accept or reject a hypothesis.

Transformers: Instantiates an ensemble; modifies ensemble by implementing a transform.

Viewers: Viewers provide some type of view or visualization of data. A viewer attractively formats and displays (i.e., as text in a window or as a graph) the output of a generator. For example, *v.matrix* lays out a matrix and its headers to fit a GUI window whenever it is resized. All current visualization viewers display a set of points written by a generator by calling *gnuplot*. *gnuplot* has been ported to almost all known operating systems and platforms, and thus provides portable visualization.

CHITRA95 and Beyond

CHITRA95 and newer evolutions of Chitra will add many new commands to CHITRA94s

family of commands. New visualizations of trace data will form a large addition to CHITRA95. The work described in this dissertation has added new *Transformers* , *Synthesizers*, *Viewers*, *Transformers*, and *Modelers* to CHITRA95. A Java based GUI generation tool (under implementation) will provide user defined and configured interfaces for the new evolutions of CHITRA96.

2.3.3 Modeling Traces of Network Traffic and Compressed Video

The rapid growth of the Internet, advent of new technologies such as Asynchronous Transfer Mode (ATM) networks and the World Wide Web (WWW) have brought traffic modeling research to the forefront of research activity.

In a recent paper [Heyman and Lakshman, 1996], Heyman and Lakshman model twelve video traffic traces by using a variety of stationary distributions. No single model is used for all the traces. The *Gamma*, *Paretto*, and *Weibull* distributions are used to model traffic sequences that not autocorrelated. *Markov Chain* and *Discrete Auto Regressive* models are used to model autocorrelated traffic sequences.

In [D.Geist and B.Melamed, 1992] Geist discusses the TES (Transform-Expand-Sample) modeling methodology for trace data. The authors use stationary stochastic processes called TES processes to model trace data as a time series. The marginal distribution of the TES process model matches the empirical distribution of the trace data. Simultaneously the autocorrelation function of the TES process approximates the empirical autocorrelation function of the trace data. The authors apply their model on traces of VBR (variable bit rate) compressed video.

2.4 A Survey of Model Validation Techniques

One of the challenging problems faced by modelers is to establish that a model is valid and credible. The case of validating models of trace data differs from that of validating simulation models. While the objective of a simulation model is to accurately represent

the system being simulated, the objective of a trace model is to represent the trace with satisfactory accuracy.

Validation techniques for trace models are a subset of those used for simulation models. In the case where only a trace is available and the actual system generating the trace as well as expert users of the system are unavailable, many validation techniques, such as *Face Validation* [Balci, 1994], cannot be used. In such a case the validation of a trace model focuses on a comparison of the original trace with synthetic traces generated from the model. Validation techniques for such trace models fall into two broad categories namely: *graphical comparison techniques* and *statistical techniques*.

2.4.1 Graphical Comparison Techniques

Balci [Balci, 1994] describes graphical comparison techniques as "subjective, inelegant and heuristic, yet quite practical." Law and Kelton [Law and Kelton, 1991] refer to them as *heuristic procedures*. Though not as credible as statistical validation techniques graphical comparison techniques can help rule out clearly inappropriate models.

Graphical comparison techniques usually entail the application of a function to the trace data. The function is applied simultaneously to the original as well as the synthetic traces generated from the model. Graphs of the function applied to the original trace and those of the function applied to the synthetic traces are then compared visually.

One of the simplest graphical comparison techniques is the 2-dimensional graph (X-Y plot) of the trace data. The term *2D View* is used henceforth in this dissertation to refer to the 2-dimensional graph of a trace. Other commonly used graphical comparison techniques are histograms, Q-Q plots, P-P plots, and plots of the cumulative distribution function.

2.4.2 Statistical Techniques

Application of statistical techniques forms the bulk of model validation research. In [Balci, 1994] Balci presents a detailed classification of statistical techniques that have been used to validate simulation models of observable systems. Some of the commonly used

statistical validation techniques for trace models are described below:

Chi-square Goodness of Fit Test: The Chi-square goodness of fit test is described in [Law and Kelton, 1991]. It is a non-parametric statistical hypothesis test. It is used to test whether a given random sample follows a known hypothesized probability distribution. The hypothesized distribution may be discrete or continuous.

Komolgorov-Smirnov Test: Another non-parametric statistical hypothesis test, the Komolgorov-Smirnov test [Law and Kelton, 1991] is used to compare an empirical distribution function to a known hypothesized probability distribution. The hypothesized distribution must be continuous.

Wilcoxon Rank Sum Test: The Wilcoxon rank sum test [McClave and Dietrich, 1985] is a non-parametric statistical hypothesis test. It is used to test whether the two given random samples have identical probability distributions. The probability distributions may be discrete or continuous.

Kruskall-Wallis Test: The Kruskall-Wallis test [Ott, 1988] is a non-parametric statistical hypothesis test. It is used to test whether k given random samples have identical probability distributions. The probability distributions may be discrete or continuous.

Spectral Analysis: This approach is based on the spectral analysis of time-series data. It involves the computation of the logarithmic difference of the sample spectra of two given traces. The logarithmic difference of the spectra is used as an estimate of the difference between the autocorrelation functions corresponding to the two traces. This technique is described in detail in [Law and Kelton, 1991].

Correlation Analysis: Correlation analysis is based on the time-domain analysis of time-series data. In its simplest form it consists of graphical comparison of the sample autocorrelation functions computed from the two (or more) given traces. However, this simple

technique is not applicable to traces that contain categorical data. In [Ribler et al., 1995] the authors describe *categorical correlation functions* that can be used instead to compute the degree of correlation between two given categorical traces.

Chapter 3

A STOCHASTIC PROCESS MODEL OF TRANSIENT TRACE DATA

In this chapter we explain the term transient behavior (as exhibited in a time series). We also present an overview of our solution approach and a formal definition of the stochastic process that we use to model the transient behavior in a trace.

3.1 Transient behavior

To explain *transient behavior* we first introduce some concepts about stochastic processes. Stochastic processes have been discussed before in Section 2. Our interest in understanding stochastic processes stems from our assumption that trace data is a sample realization of a stochastic process. An important classification of stochastic processes is that of *stationary* and *non-stationary* stochastic processes. Stationary stochastic processes are governed by probability laws that are invariant to time (index parameter). Two definitions of stationarity, namely *strong stationarity* and *weak stationarity* are described in most literature related to stochastic processes. For the purpose of this dissertation we discuss the definition of weak stationarity [Box and Jenkins, 1976] below.

Definition 2 *A stochastic process* $\{X_t\}$ *is considered weakly stationary if*

1. *the mean of the process is constant,*
2. *the variance of the process is constant, and*
3. *the autocovariance of the process at lag* h, $\gamma_{t,t+h}$, *is independent of* t.

Deviance from these properties leads to non-stationary stochastic processes. Consider the class of non-stationary stochastic processes where the distribution of the random variables X_t changes with time. In particular, let us consider non-stationary stochastic processes where the random variables X_t follow different probability distributions within disjoint *intervals* of time. A trivial case of this class of stochastic processes is one where each random variable X_t follows a different probability distribution. The 2D view of a sample realization from a stochastic process that belongs to the class described above may display the time varying nature of the stochastic process. For the purpose of this dissertation we define *transient behavior* as follows:

Definition 3 *A trace shows* transient behavior *if it can be considered a sample realization of a non-stationary stochastic process* $\{X_t\}$ *such that the random variables* X_t *follow different probability distributions.*

The trace of processor numbers obtained by running a distributed implementation of the Gaussian Elimination algorithm (Section 6.1), shown in the Figure 1.1, clearly shows transient behavior. The mean of the stochastic process underlying this trace varies with time.

3.2 Overview of the Modeling Approach

The discussion of our solution approach is based on the following terms: A *state* is a discrete value or a vector of discrete values that uniquely describes the system under study. For the purpose of the discussion in this dissertation a state is always a single value (as shown in Appendix D, a unique vector of discrete values can be mapped to a single unique discrete value).

A *trace* is a sequence of states that records the behavior of the system over a period of time. Generally a trace can have time stamps associated with every occurrence of a state. Our analysis is limited to traces which can be considered as *time series* [Box and Jenkins, 1976] with an index parameter space that is discrete, equally spaced, and monotonically increasing. The term *event numbers* has been used throughout this dissertation to refer to the

index parameter space for a trace. The state space for the trace is discrete as well.

Rates and frequencies of occurrence are often used as estimates of probabilities (for an example see [Abrams et al., 1992]). We estimate the probability that the state x occurs at event number t, in the trace, by the rate of occurrence of state x at t. This leads to a simple modeling approach: for every event t in the trace, we estimate the probabilities of occurrence of all the unique states in the trace by their corresponding rates of occurrence at t. Therefore we can determine the probability mass function of all the unique states at every event number t in the trace. Finally, the joint probability mass function for the entire trace is the collection of probability mass functions at all events in the trace.

The modeling approach just discussed estimates the joint probability mass function using the rate of occurrence of each state at all values of the event number t. Consequently, we still need a way to determine rate of occurrence. To solve this problem, consider a two-dimensional Cartesian graph with the number of occurrences of states as the ordinate and event numbers as abscissa. This graph is known as the *Rate Evolution Graph* (REG). The set of points plotted for each unique state form a path. The slope of the REG path for a state x at t has units of *number of occurrences / event duration* which is thus the rate of occurrence of x at t. If we fit a curve to each path in the REG then the rate of occurrence of each state at t is estimated by the derivative of the curve at t. As an example consider the REG of the first 150 events for the WWW Trace (Section 6.1) in Figure 3.1. The 3 REG paths in this figure have been fitted with $4th$-degree polynomials:

$$\begin{aligned}
&\textit{Directory} && 1.15 + 0.03\,x + 1.18 \times 10^{-7}\,x^4, \\
&\textit{Graphics} && -0.84 + 0.97\,x - 1.55 \times 10^{-7}\,x^4, \text{ and} \\
&\textit{Text_or_HTML} && 1.92 - 0.04\,x + 9.17 \times 10^{-8}\,x^4.
\end{aligned}$$

A polynomial can be fitted to almost any REG path, however, this technique has the following problems:

1. Fitting a polynomial is an iterative process that requires human intervention. Consequently, this process cannot be automated. For the example in Figure 3.1 polynomials

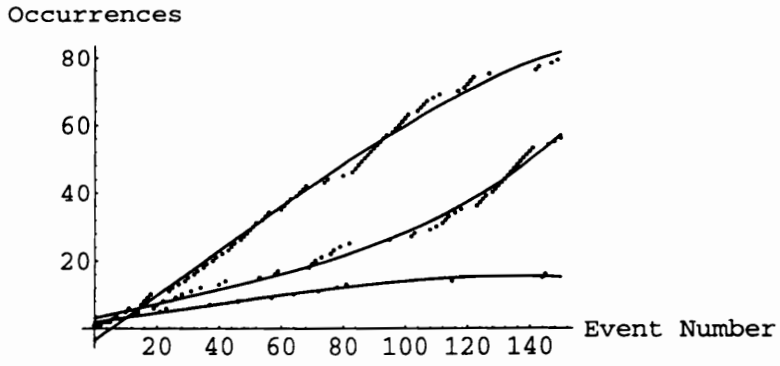


Figure 3.1: Polynomial Fit for the WWW Trace REG

of various degrees were tried before deciding upon a $4th$ degree polynomial as a suitable fit.

2. Polynomial functions that exactly fit a REG path may need the estimation of a large number of parameters. In general, if we have k data points, we may need a $k-1$ degree polynomial to fit the k data points [Wolfram, 1992].

A central observation upon which this dissertation is based is the following: From exploratory analysis of several traces we have discovered that for each state in the trace, the event number axis can be partitioned into *intervals* such that the rate of occurrence of the state is nearly constant within each interval. The number of such intervals is a small fraction of the length of the trace. Thus, we can find the joint probability mass function

for the trace by finding:

1. the collection of intervals such that the rates of occurrence of all states remains nearly constant within an interval, and

2. the probability mass function within each interval.

The REGs of such traces consist of REG paths that are made up of straight line segments. The REG for the IBM CICS Trace (Section 6.1) shown in Figure 3.2 illustrates our argument well. Thus for such traces we can fit a collection of straight line segments to each REG path.

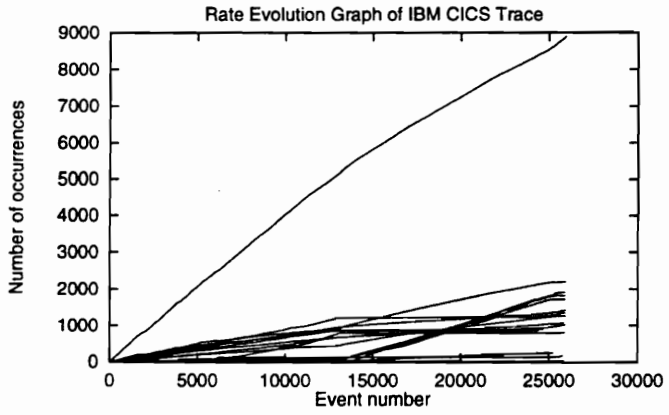


Figure 3.2: REG for the IBM CICS Trace

Consequently, the rate of occurrence of a state within an interval can be estimated by the slope of the straight line segment fitting the corresponding REG path. Fitting a collection of straight lines per REG path has the following advantages over fitting a polynomial function:

1. Fitting a collection of straight lines to a REG path is a mechanical process which can be easily automated.

2. A straight line has a trivial derivative (its slope). Thus the estimation of the rate of occurrence of a state is a simple computation.

3. If the number of intervals per REG path is small then we have few parameters to estimate (a slope per interval).

The joint probability mass function for the entire trace is consequently expressed as a collection of probability mass functions such that each probability mass function holds over an interval.

Figure 3.3 shows that our solution approach, the *Piecewise Independent Stochastic Process* (PISP) and its general form the *Independent Stochastic Process* (ISP), form a subset of REG based stochastic processes that can be used to model transient trace data. The independent stochastic process and the piecewise independent stochastic process are defined in Sections 3.3 and 3.4 respectively.

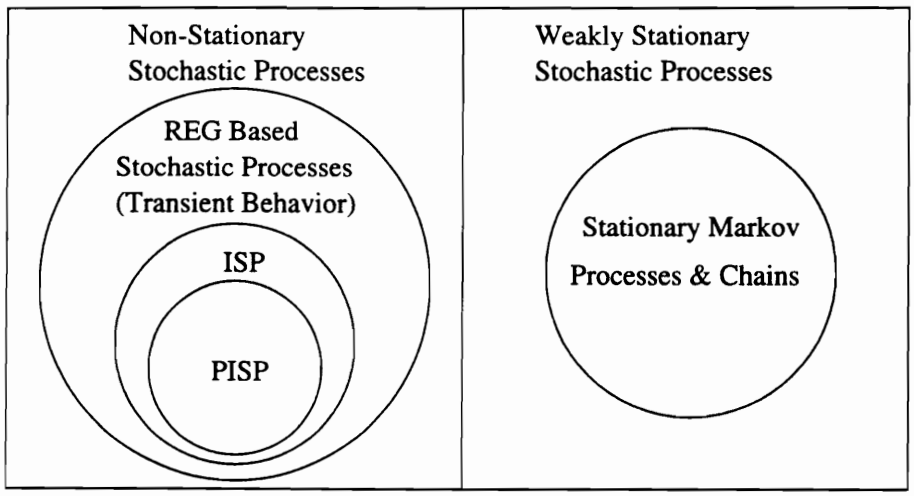


Figure 3.3: Stochastic Process Models Based on the REG

3.3 The Independent Stochastic Process

Consider a stochastic process $\{X_t \mid t \in \mathbf{T}\}$ with a discrete and finite index parameter space $\mathbf{T} = \{t_1, t_2, t_3, \ldots, t_n\}$ and discrete state space $\mathbf{S} = \{x_1, x_2, x_3, \ldots, x_m\}$. Let us now introduce notation which will be used in defining terms through out this document. For X_i in $\{X_t \mid t \in \mathbf{T}\}$ and x_i in $\mathbf{S}$, denote the probability $P\{X_i = x_i\}$ as $p_{x_i}^{X_i}$. For all $x_1, x_2, \ldots . x_n$ in $\mathbf{S}$, denote the set $\{x_1, \ldots, x_n\}$ as $\mathbf{x}$. Also, for all $X_1, X_2, \ldots . X_n$ in $\{X_t \mid t \in \mathbf{T}\}$, denote the set $\{X_1, \ldots, X_n\}$ as $\mathbf{X}$. A stochastic process $\{X_t\}$ is usually described using the joint

distribution of its member random variables X_t. The joint probability mass function of the stochastic process $\{X_t\}$ denoted $p_{\mathbf{x}}^{\mathbf{X}}$, is given by

$$p_{\mathbf{x}}^{\mathbf{X}} = p_{\{x_1,x_2,\ldots,x_n\}}^{\{X_1,X_2,\ldots,X_n\}} = P\{X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n\}. \tag{3.1}$$

In the absence of simplifying assumptions the computation of $p_{\mathbf{x}}^{\mathbf{X}}$ as described in equation (3.1) would require us to determine $p_{\mathbf{x}}^{\mathbf{X}}$ for all values in the index parameter space $\mathbf{T}$.

The *independent stochastic process* is derived from equation (3.1) by making the simplifying assumption that all member random variables of the stochastic process $\{X_t\}$ are independent of each other. Thus, the joint probability mass function, $p_{\mathbf{x}}^{\mathbf{X}}$, factors into the product

$$p_{\mathbf{x}}^{\mathbf{X}} = p_{x_1}^{X_1}.p_{x_2}^{X_2}\ldots p_{x_n}^{X_n} = \prod_{1\leq t\leq n} p_{x_t}^{X_t}. \tag{3.2}$$

To model a trace as an independent stochastic process we must obtain all marginal distributions $p_{x_t}^{X_t}$, $1 \leq t \leq n$. Obtaining all the marginal distributions is a formidable task. Also, the storage for such a model will exceed the storage for the trace.

3.4 The Piecewise Independent Stochastic Process

We introduce the piecewise independent stochastic process as a variant of the independent stochastic process. Consider partitioning $\mathbf{T}$ into r non-overlapping *intervals* $[t_{a_1}, t_{a_2})$, $[t_{a_2}, t_{a_3}), \ldots, [t_{a_r}, t_{a_{r+1}})$, where $0 = t_{a_1} < t_{a_2} < \ldots < t_{a_r} < t_{a_{r+1}} = \infty, 1 \leq r \leq n$, and $\{t_{a_1}, t_{a_2}, \ldots, t_{a_r}\} \in \mathbf{T}$. We shall refer to the set $\{t_{a_1}, t_{a_2}, \ldots, t_{a_r}\}$ as the set of *partitioning events* and denote this set as $\mathbf{R}$. The set $\{X_t \mid t \in \mathbf{R}\}$ is a proper subset of the stochastic process $\{X_t \mid t \in \mathbf{T}\}$ and consists of the family of random variables $\{X_{a_1}, \ldots, X_{a_r}\}$.

Definition 4 *A piecewise independent stochastic process is a discrete state and discrete index parameter stochastic process $\{X_t\}$ whose joint probability mass function $p_{\mathbf{x}}^{\mathbf{X}}$ is given by*

$$\forall x_{a_j} \in \mathbf{S},\ p_{\mathbf{x}}^{\mathbf{X}} = \prod_{1\leq j\leq r} p_{x_{a_j}}^{X_{a_j}}. \tag{3.3}$$

Consequently, the joint probability mass functions of $\{X_t\}$ within two distinct intervals $[t_{a_i}, t_{a_{i+1}})$ and $[t_{a_j}, t_{a_{j+1}})$, denoted $p_{x_{a_i}}^{X_{a_i}}$ and $p_{x_{a_j}}^{X_{a_j}}$ respectively, are independent of each other. Denote the joint probability mass function of X_{a_i} and X_{a_j} as $p_{\{x_{a_i}, x_{a_j}\}}^{\{X_{a_i}, X_{a_j}\}}$. Then,

$$\forall x_{a_i}, x_{a_j} \in \mathbf{S} \wedge \forall t_{a_i}, t_{a_{i+1}}, t_{a_j}, t_{a_{j+1}} \in \mathbf{R} \wedge i \neq j, \quad p_{\{x_{a_i}, x_{a_j}\}}^{\{X_{a_i}, X_{a_j}\}} = p_{x_{a_j}}^{X_{a_i}} . p_{x_{a_j}}^{X_{a_j}}.$$

Also, the joint probability mass function of $\{X_t\}$ within an interval $[t_{a_i}, t_{a_{i+1}})$, denoted $p_{x_{a_i}}^{X_{a_i}}$, is invariant to t. Formally,

$$\forall x_{a_i}, x_u, x_v \in \mathbf{S} \wedge \forall t_u, t_v \in [t_{a_i}, t_{a_{i+1}}) \wedge \forall t_{a_i}, t_{a_{i+1}} \in \mathbf{R}, \quad p_{x_u}^{X_u} = p_{x_v}^{X_v} = p_{x_{a_i}}^{X_{a_i}}.$$

To model a trace as a piecewise independent stochastic process we need only to estimate the following parameters:

1. the set of partitioning events $\mathbf{R}$, and
2. the joint probability mass function $p_{\mathbf{x}}^{\mathbf{X}}$ expressed as a set of probability mass functions $\{p_{x_t}^{X_t} \mid t \in \mathbf{R}\}$.

Chapter 4

ESTIMATING MODEL PARAMETERS

In this section we describe the steps to estimate the parameters of the piecewise independent stochastic process model described in section 3.4. To completely describe this stochastic process we must estimate the set of partitioning events $\mathbf{R}$ and the joint probability mass function $\{p_{x_t}^{X_t} \mid t \in \mathbf{R}\}$.

The first step in our solution approach is to estimate $\mathbf{R}$. In Section 3.2 we introduced the REG as a tool that plots the rate of occurrence of the states in a trace. A formal definition of the REG is discussed now. Let the function $g(x_i, t)$ be the number of times that the state x_i occurs up until and including the event number t. Formally, $g(x_i, t) = \|\{t' \mid X_{t'} = x_i \wedge 1 \leq t' \leq t\}\|$. Also, we define $\mathbf{T_i}$ as a proper subset of $\mathbf{T}$ such that the state x_i occurs at each event number t in $\mathbf{T_i}$. Formally, $\mathbf{T_i} = \{t \mid X_t = x_i \wedge x_i \in \mathbf{S}\}$. The REG path for state x_i, denoted Ψ_i, consists of the points $(t, g(x_i, t))$ such that t is in $\mathbf{T_i}$.

Definition 5 *The* REG path *for state x_i in* $\mathbf{S}$ *is given by*

$$\Psi_i = \{(t, g(x_i, t)) \mid x_i \in \mathbf{S} \wedge t \in \mathbf{T_i}\}.$$

A REG is a collection of all REG paths. A REG Ψ is the set $\{\Psi_i \mid \forall x_i \in \mathbf{S}\}$.

Example 1 *Consider a trace of hits and misses from a cache in Table 4.1. Our example system has two states namely a hit in the cache, denoted by H, and a miss in the cache, denoted by M. The state space* $\mathbf{S}$ *for this trace is given by $\{H, T\}$. The integer i indexes the state space. The domain of i is given by the set $\{0, 1\}$. Therefore, for x_i in* $\mathbf{S}$, *$x_0 = H$ and $x_1 = M$. The set $\mathbf{T_0} = \{0, 1, 2, 3, 8, 9, 10, 11, 13, 15\}$ and the set $\mathbf{T_1} = \{4, 5, 6, 7, 12, 14, 16\}$. The REG for the states H and M is displayed as the two paths* Hits *and* Misses, *respectively, in Figure 4.1.*

Table 4.1: Cache Hit Trace.

Cache Hit Rate																	
t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
X_t	H	H	H	H	M	M	M	M	H	H	H	H	M	H	M	H	M

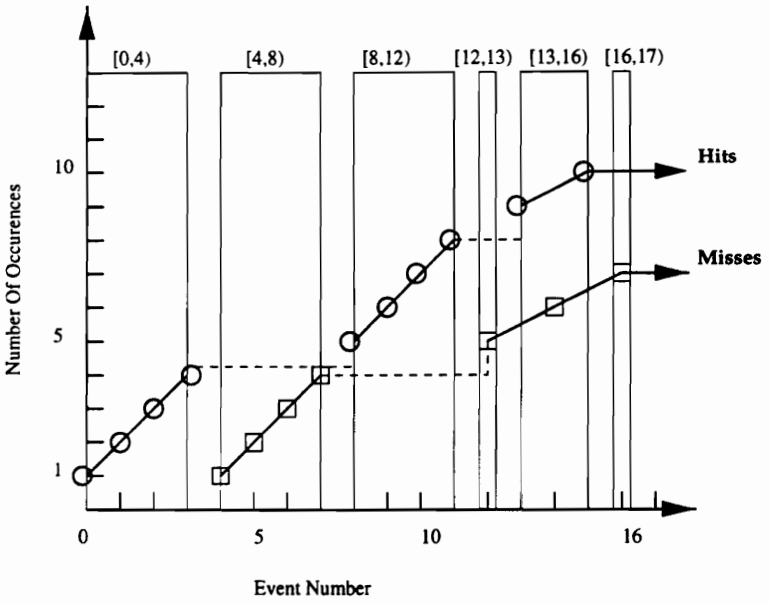


Figure 4.1: REG for the Cache Trace

Recall from Section 3.2 that within an interval the rate of occurrence of all states remains nearly constant. Also, recall from Section 3.2 that we define a partitioning event as that event number where the rate of occurrence of any state changes (by more than some constant value ϵ) from its previous value. The rate of occurrence of a state x_i at event number t is the slope of the REG path, Ψ_i, at t. Consequently the event number 0 along with any event number at which the slope of any REG path changes significantly from its previous value is a partitioning event.

Example 2 *Figure 4.1 shows the set of partitioning events for the example trace in Table 4.1 (see details in Section 4.1). The rate of occurrence of the state* Hits *changes at event*

numbers $0, 4, 8, 12, 13, 16$. *Similarly, the rate of occurrence of the state* Misses *changes at event numbers* $4, 8, 12, 17$. *Therefore, the set* $\mathbf{R} = \{0, 4, 8, 12, 13, 16, 17\}$ *for this trace.*

The computation of $\mathbf{R}$ is detailed in Section 4.1. Once, we have $\mathbf{R}$, we can determine the joint probability mass functions $\{p_{x_i}^{X_t} \mid t \in \mathbf{R}\}$. Recall from Section 3.2 that the rate of occurrence of state x_i at event number t can be used as an estimate of the probability of occurrence of x_i at t. Denote by $\phi_{i,t}$ the rate of occurrence of x_i at t. Let us examine the segment of the REG path Ψ_i within an interval $[t_{a_1}, t_{a_2})$. Recall from Section 3.2 that $\phi_{i,t}$ is nearly constant within an interval. Consequently, we can estimate $\phi_{i,t}$ $(t_{a_1} \leq t < t_{a_2})$ by the *constant* slope of the straight line fitting the REG path Ψ_i within the interval $[t_{a_1}, t_{a_2})$. The *rate vector*, Φ_t, defined at partitioning event t, is a collection of rates $\phi_{i,t}$ for all states x_i in $\mathbf{S}$.

Definition 6 *Consider an event* $t \in \mathbf{R}$. *The* rate vector Φ_t *is defined as the set*

$$\Phi_t = \{\phi_{i,t} \mid x_i \in \mathbf{R}\}$$

.

Example 3 *For the example trace in Table 4.1 the rate vectors* Φ_t, *for all* t *in* $\mathbf{R}$, *are displayed in Table 4.2.*

Table 4.2: Rate Vectors Φ_t

Rate Vectors Φ_t $(t \in \mathbf{R})$		
Φ_t	$x_i = H$	$x_i = M$
Φ_0	1	0
Φ_4	0	1
Φ_8	1	0
Φ_{12}	0	1
Φ_{13}	0.5	0.5
Φ_{16}	0	1
Φ_{17}	0	0

We normalize the rate vector such that all its member rates, $\phi_{i,t}$, satisfy the property $\sum_{x_i \in \mathbf{S}} \phi_{i,t} = 1$. Denote the normalized $\phi_{i,t}$ as $\phi'_{i,t}$. The normalized rate vector, $\phi'_{i,t}$, is the estimate of the probability mass function of the interval starting at the partitioning event t, $p_{x_t}^{X_t}$. Finally, because $\mathbf{R}$ has been already determined, we can compute the set $\mathbf{\Phi} = \{\Phi'_t \mid t \in \mathbf{R}\}$. The set $\mathbf{\Phi}$ estimates the joint probability mass function $\{p_{x_t}^{X_t} \mid t \in \mathbf{R}\}$.

Example 4 *For the example trace in Table 4.1, the estimated joint probability mass function* $\{p_{x_t}^{X_t} \mid t \in \mathbf{R}\}$ *is displayed in Table 4.3.*

Table 4.3: Joint Probability Mass Function

Time dependent *Pmf* ($\mathbf{\Phi}$)		
Interval	$P(x_i = H)$	$P(x_i = M)$
[0,4)	1	0
[4,8)	0	1
[8,12)	1	0
[12,13)	0	1
[13,16)	0.5	0.5
[16,17)	0	1
[17,∞)	0	0

4.1 Estimation of Partitioning Events

In this section we describe the computation of the set $\mathbf{R}$. In order to explain the computational details we define some terms next. The *point slope* $s_i(t_u, t_v)$, of the REG path Ψ_i is given by the expression

$$\forall\ (t_u, g(x_i, t_u)), (t_v, g(x_i, t_v)) \in \Psi_i \wedge t_u < t_v, \quad s_i(t_u, t_v) = \frac{g(x_i, t_v) - g(x_i, t_u)}{t_v - t_u}.$$

A *partition beginning event* is the event number t ($t \in \mathbf{T_i}$) immediately following the event number where the point slope of the state x_i changes more than a tolerance from its previous value. Consider the consecutive members $(x_i, g(x_i, t_u))$, $(x_i, g(x_i, t_{u+1}))$, and $(x_i, g(x_i, t_{u+2}))$

of the REG path Ψ_i. If the absolute difference between the point slopes $s_i(t_u, t_{u+1})$ and $s_i(t_{u+1}, t_{u+2})$ exceeds a constant ϵ then t_{u+2} is a partition beginning event for x_i.

Definition 7 *The set of* partition beginning events *for state x_i, denoted $\mathbf{B_i}$, is defined as follows:*

$$\mathbf{B_i} = \{t_u \mid t_{u-1} \notin \mathbf{B_i} \wedge |s_i(t_{u-2}, t_{u-1}) - s_i(t_{u-1}, t_u)| > \epsilon \wedge t_{u-2}, t_{u-1}, t_u \in \mathbf{T_i}\}.$$

For the special case where $u = 0$, t_u is in $\mathbf{B_i}$. Similarly, for the special case where $u = 1$, t_u is not in $\mathbf{B_i}$.

A *partition ending event* is the event number t ($t \in \mathbf{T_i}$) that immediately precedes the next partition beginning event (with the exception of the first partition beginning event for a state).

Definition 8 *The set of* partition ending events *for a state x_i, denoted $\mathbf{E_i}$, is defined as follows:*

$$\mathbf{E_i} = \{t_u \mid t_{u+1} \in \mathbf{B_i} \wedge u > 0 \wedge t_u, t_{u+1} \in \mathbf{T_i}\}.$$

An *intervening partition beginning event* is the event number t ($t \in \mathbf{T}$) that begins an interval where the state x_i does not occur.

Definition 9 *The set of* intervening partition beginning events *for a state x_i, denoted $\mathbf{A_i}$, is defined as follows:*

$$\mathbf{A_i} = \{t \mid t - 1 \in \mathbf{E_i} \wedge t \in \mathbf{T} - \mathbf{T_i}\}.$$

A *state partitioning event* is the event number t ($t \in \mathbf{T}$) that marks the start of each interval during which the rate of occurrence of the state x_i remains nearly constant. Let us denote the collection of all state partitioning events for x_i as $\mathbf{R_i}$.

Definition 10 *For a state x_i, and its corresponding REG path Ψ_i, the set of state partitioning events, $\mathbf{R_i}$, is given by:*

$$\mathbf{R_i} = \mathbf{B_i} \cup \mathbf{A_i}.$$

Example 5 *For the example trace in Table 4.1, the sets* $\mathbf{B_0}$ *and* $\mathbf{B_1}$ *are given by* $\{0, 8, 13\}$ *and* $\{4, 12\}$ *respectively. The sets* $\mathbf{E_0}$ *and* $\mathbf{E_1}$ *are given by* $\{3, 11, 15\}$ *and* $\{7, 16\}$ *respectively. The sets* $\mathbf{A_0}$ *and* $\mathbf{A_1}$ *are given by* $\{4, 12, 16\}$ *and* $\{8, 17\}$ *respectively. Finally, the sets* $\mathbf{R_0}$ *and* $\mathbf{R_1}$ *are given by* $\{0, 4, 8, 12, 13, 16\}$ *and* $\{4, 8, 12, 17\}$ *respectively.*

We call this method of finding state partitioning events by the name *slope-based partitioning.* Another method for finding state partitioning events, *incremental-regression partitioning* is presented later in this document. We have found that *slope-based partitioning* is a more sensitive method because it can find consecutive partitioning events even when the partitioning events are only one event apart.

Once we have the set $\mathbf{R_i}$ for all unique states in the trace, the set of partitioning events $\mathbf{R}$ is determined by computing a union of all $\mathbf{R_i}$. Formally, $\mathbf{R} = \bigcup_{x_i \in \mathbf{S}} \mathbf{R_i}$.

4.2 Estimation of the Joint Probability Mass Function

To estimate the joint probability mass function $\{p_{x_t}^{X_t} | t \in \mathbf{R}\}$ we need the rate vectors Φ_t, for all t in $\mathbf{R}$. Therefore we describe a method to estimate Φ_t first. Recall from Section 3.2 that Φ_t is a set of rates $\phi_{i,t}$ (x_i in $\mathbf{S}$ and t in $\mathbf{R}$) for the interval that starts at partitioning event t. Consequently, to determine Φ_t we must compute its members $\phi_{i,t}$, for all $1 \leq i \leq m$. Consider an interval $[t_{a_1}, t_{a_2})$. The REG path Ψ_i corresponding to this interval is the set of points $(t_{a_1}, g(x_i, t_{a_1})), \ldots, (t_{a_2} - 1, g(x_i, t_{a_2} - 1))$. We compute the rate $\phi_{i,t_{a_1}}$ as the slope parameter of the linear regression function [Ott, 1988] that fits the points $(t_{a_1}, g(x_i, t_{a_1})), \ldots, (t_{a_2} - 1, g(x_i, t_{a_2} - 1))$.

Example 6 *For the example trace in Table 4.1, the rates* $\phi_{0,t}$ *(*$t \in \mathbf{R_0}$*) are shown in Table 4.4. Similarly, the rates* $\phi_{1,t}$ *(*$t \in \mathbf{R_1}$*) are shown in Table 4.5.*

We can compute the rate vector $\Phi_{t_{a_1}}$ by computing $\phi_{i,t_{a_1}}$, for all $1 \leq i \leq m$. Let Φ^i denote the subset of Φ such that the members of Φ^i are the rate vectors Φ_t corresponding to the state partitioning events in $\mathbf{R_i}$. Formally, $\Phi^i = \{\Phi_t \mid \forall t \in \mathbf{R_i}\}$.

Table 4.4: Partitioning Events and Corresponding Rates for State H

Partitioning Events and Rates						
Partitioning Event ($t \in \mathbf{R_0}$)	0	4	8	12	13	16
Rate ($\phi_{0,t}$)	1	0	1	0	0.5	0

Table 4.5: Partitioning Events and Corresponding Rates for State M

Partitioning Events and Rates					
Partitioning Event ($t \in \mathbf{R_1}$)	0	4	8	12	17
Rate ($\phi_{1,t}$)	0	1	0	0.5	0

Example 7 *For the example trace in Table 4.1 the sets Φ^0 and Φ^1 are displayed in Table 4.6 and Table 4.7 respectively.*

Table 4.6: Rate Vectors Φ^0

Rate Vectors Φ_t ($t \in \mathbf{R_0}$)		
Φ_t	$\Phi_{0,t}$	$\Phi_{1,t}$
Φ_0	1	0
Φ_4	0	1
Φ_8	1	0
Φ_{12}	0	1
Φ_{13}	0.5	0.5
Φ_{16}	0	1

The set $\mathbf{\Phi}$ can now be computed as a union of all Φ^i. Formally, $\mathbf{\Phi} = \bigcup_{x_i \in \mathbf{S}} \Phi^i$.

Example 8 *For the example trace in Table 4.1 the set $\mathbf{\Phi}$ is displayed in Table 4.2.*

Finally, each rate vector Φ_t in the set $\mathbf{\Phi}$ is normalized as described below. Recall that $\Phi_t = \{\phi_{i,t} \mid t \in \mathbf{R}\}$. Each $\phi_{i,t}$ in Φ_t is replaced by the value $\frac{\phi_{i,t}}{\sum_{1 \le i \le m} \phi_{i,t}}$. The set of rate vectors $\mathbf{\Phi}$ can now be used as an estimate of the joint probability mass function $\{p_{x_t}^{X_t} | t \in \mathbf{R}\}$.

Table 4.7: Rate Vectors Φ^1

Rate Vectors Φ_t ($t \in \mathbf{R_1}$)		
Φ_t	$\Phi_{0,t}$	$\Phi_{1,t}$
Φ_4	0	1
Φ_8	1	0
Φ_{12}	0	1
Φ_{17}	0	0

Example 9 *For the example in Table 4.1 the estimated joint probability mass function is displayed in Table 4.3.*

4.3 Model Storage

We store the model as the sets $\mathbf{R_i}$ and $\{\Phi_{i,t} \mid t \in \mathbf{R_i}\}$ for each state x_i in the trace. The expression for the size of piecewise independent stochastic process model, is a function of two factors, namely the number of states in the trace and the number of partitioning events to be stored for each state. As defined earlier the cardinality, m, of the state space $\mathbf{S}$ is the number of states to be stored for the model. Denote as k_i the number of partitioning events to be stored for the state x_i (the cardinality of the set $\mathbf{R_i}$). The storage for the model is therefore given by:

$$C \sum_{i=1}^{m} k_i \tag{4.1}$$

The constant C in the above expression denotes the constant storage needed for each $\langle partitioning\ event, rate \rangle$ pair.

Example 10 *For the example trace in Table 4.1 our model is stored as Tables 4.4 and 4.5. According to equation (4.1) above the storage requirement for our model is* $(5+6)C = 11C$ *units.*

In general the storage required to store our model is not significantly different from that of the raw trace.

Chapter 5

CONSTRUCTING A PARSIMONIUS MODEL

One of the desirable qualities of a model is parsimony. Specifically, we want the model to be as compact (in terms of storage) as possible without excessive loss of accuracy.

Recall from Section 4.3, that the size of the piecewise independent stochastic process model is given by the expression $C\sum_{i=1}^{m} k_i$. The only variables in this expression are k_i and m. Consequently, two approaches to reduce the model storage are to *reduce the number of partitioning events* that need to be stored for each state and to *reduce the number of states* in the model. The following sections discuss both approaches.

5.1 Reducing the Number of Partitioning Events

The key idea in this approach is to iteratively reduce the size of $\mathbf{R_i}$ for each state x_i. The iterations converge when $\mathbf{R_i}$ is a singleton set. However, with only one partitioning event per state stored, the model obtained may be extremely inaccurate. Therefore, each iteration to reduce $\mathbf{R_i}$ must be followed by an evaluation of the quality of the model obtained. This insures that we have a compact model without significantly compromising the quality of the model.

5.1.1 Compact REG

To reduce the size of $\mathbf{R_i}$, we start by generating a compact REG path for each state x_i.

Definition 11 *We define f as a compaction function that maps each point $(t, g(x_i, t)) \in \Psi_i$ to the set $f(\Psi_i) \subset \Psi_i$. To define the membership of $f(\Psi_i)$ we introduce t_l and t_k as*

$\{t \mid t \geq t' \ \wedge \ \forall \ (t', g(x_i, t')) \in \Psi_i\}$ *and* $\{t \mid t \geq t' \ \wedge \ \forall \ t' \in \mathbf{R_i}\}$ *respectively. Thus,* $(t, g(x_i, t)) \in f(\Psi_i)$ *iff* $(t \neq t_k \ \wedge t \in \mathbf{R}) \ \vee (t = t_l)$.

Example 11 *For the example in Table 4.1, the compact REG paths* $f(\Psi_H)$ *and* $f(\Psi_M)$ *are displayed in Tables 5.1 and 5.2 respectively. The paths* $f(\Psi_H)$ *and* $f(\Psi_M)$ *are also displayed graphically in Figure 5.1.*

Table 5.1: Compact REG for State H

Compact REG for *Hits* $(x_i = H)$						
Event Number (t)	0	4	8	12	13	15
Number of Occurrences $(g(x_i, t))$	1	4	4	8	9	10

Table 5.2: Compact REG for State M

Compact REG for *Misses* $(x_i = M)$				
Event Number (t)	4	8	12	16
Number of Occurrences$(g(x_i, t))$	1	4	5	7

5.1.2 Estimating Partitioning Events from the Compact REG

The compact REG $f(\Psi_i)$ is now utilized to generate a new set of state partitioning events $\mathbf{R_i}$. Instead of using the slope-based partitioning technique (Section 4.1), we now use an alternative technique called *incremental regression partitioning.*

Incremental regression partitioning Recall that a REG is a set of points $\{(t, g(x_i, t))\}$. The incremental regression partitioning approach regresses the independent variable t to predict the dependent variable $g(x_i, t)$. We construct a simple linear regression function, denoted G, based on the first k $(k \geq 3)$ points in $f(\Psi_i)$. Using the regression function G we predict the ordinate of the $k + 1^{st}$ point, $\hat{g}(x_i, t_{k+1})$, with a corresponding prediction interval, denoted I_{k+1}. If the actual $k + 1^{st}$ ordinate $g(x_i, t_{k+1})$ falls outside I_{k+1} then t_{k+1}

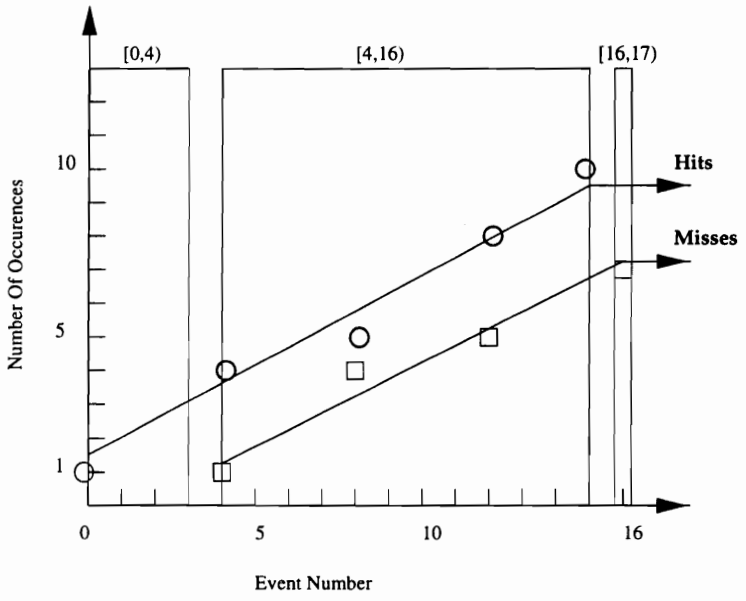


Figure 5.1: The Compact REG for Cache Trace

is a partitioning event. The search for the next partitioning event starts at the current partitioning event. The process is repeated until all points in $f(\Psi_i)$ have been considered.

5.1.3 Estimating the Joint Probability Distribution

Once the reduced state partitioning event sets $\mathbf{R_i}$ have been determined, we compute $\mathbf{R}$ and $\mathbf{\Phi}$ as detailed in Sections 4.1 and 4.2.

Example 12 *For the example trace in Table 4.1, the reduced set* $\mathbf{R} = \{0, 4, 16, 17\}$. *The sets* $\{\Phi_{0,t} \mid t \in \mathbf{R_0}\}$ *and* $\{\Phi_{1,t} \mid t \in \mathbf{R_1}\}$ *are displayed in the second row of Tables 5.3 and 5.4 respectively. The first row of Tables 5.3 and 5.4 display the sets* $\mathbf{R_0}$ *and* $\mathbf{R_1}$ *respectively. Finally, the joint probability mass function,* $\{p_{x_t}^{X_t} \mid t \in \mathbf{R}\}$, *is displayed in Table 5.5. The storage for the compact model is* $4C$ *units; a reduction of more than* 50% *from the original model. Notice however, that the compact model is not accurate as the original model.*

Table 5.3: Partitioning Events and Corresponding Rates for State H

Partitioning Events and Rates		
$Partitioning\ Event(t \in \mathbf{R_0})$	0	16
$Rate(\Phi_{H,t})$	0.59	0

Table 5.4: Partitioning Events and Corresponding Rates for State M

Partitioning Events and Rates		
$Partitioning\ Event(t \in \mathbf{R_1})$	4	17
$Rate(\Phi_{M,t})$	0.48	0

5.2 Reducing the Number of States

The goal of this approach is to reduce the state space of the model. This approach is only applicable to traces of numerical data. Consider the trace as a time series $\{X_t\}$. Denote as M the largest integer in the time series $\{X_t\}$. Let us denote as $\mathbf{S_k}$ a state space with cardinality k ($k < M$). The defintion of $\mathbf{S_k}$ is given below:

$$\mathbf{S_k} = \{\ y_j \mid y_j = j\lfloor\frac{M}{k}\rfloor \wedge\ 0 \leq j < k\ \}.$$

The cardinality of $\mathbf{S_k}$, $\|\mathbf{S_k}\|$, can be made significantly smaller than the cardinality of $\mathbf{S}$, $\|\mathbf{S}\|$. Let U_t denote a Uniform$(0,1)$ random variable. Also, let us define an indicator function I_t

Table 5.5: Joint Probability Mass Function (Reduced)

Time dependent $Pmf\ (\Phi)$		
Interval	$P(x_i = H)$	$P(x_i = M)$
[0 - 4)	1	0
[4 - 16)	0.55	0.45
[16 - 16]	0	1
[17 - ∞)	0	0

as

$$\forall\, t \in \mathbf{T}, \quad I_t = \begin{cases} 0 & \text{if } U_t < 0.5 \\ 1 & \text{if } U_t \geq 0.5. \end{cases}$$

Next, let x_t denote the value of the random variable X_t. An original trace $\{X_t\}$ is transformed into a reduced trace (with domain $\mathbf{S_k}$), denoted $\{Y_t\}$, as follows:

$$\forall\, t \in \mathbf{T}, \quad Y_t = \begin{cases} x_t & \text{if } x_t \in \mathbf{S_k}. \\ y_j + I_t(y_{j+1} - y_j) & \text{if } \exists j\ (0 \leq j < k) \mid y_j < x_t < y_{j+1} \wedge y_j, y_{j+1} \in \mathbf{S_k} \end{cases}$$

We construct a piecewise independent stochastic process model for the trace $\{Y_t\}$. We expect that such a model of $\{Y_t\}$ will require less storage than a similar model of the original trace $\{X_t\}$. Finally, $\{X_t\}$ is modeled in terms of $\{Y_t\}$ by adding a noise component $\{\xi_t\}$. The noise $\{\xi_t\}$, $t = 1, 2, \ldots, n$, are IID random variables that follow a Normal$(0, \lfloor \frac{M}{2k} \rfloor)$ distribution. Formally,

$$X_t = Y_t + \xi_t.$$

Adding the noise $\{\xi_t\}$ re-introduces the states in $\mathbf{S} - \mathbf{S_k}$ back into $\{Y_t\}$.

Example 13 *To show the application of this technique we use the Ingalls trace (Section 6.1) as an example. The 2D view of this trace is shown in Figure 5.2. After reducing the state space ($k = 20$) the 2D view of the transformed trace is shown in Figure 5.3. Next, Figure 5.4 shows the 2D view of the synthetic trace generated from the piecewise independent stochastic process model of the transformed Ingalls trace. Finally, we add the noise to the synthesized transformed trace so that the states which had been filtered out are added back. The final synthesized trace is displayed in Figure 5.5.*

A side effect of this technique is that the partitioning events of a filtered state (a state in the set $\mathbf{S} - \mathbf{S_k}$) causes an increase in the partitioning events of the states in $\mathbf{S_k}$. This can cause the number of partitioning events for the states in $\mathbf{S_k}$ to increase by large amounts.

This approach can be used in combination with the model size reduction approach (Section 5.1).

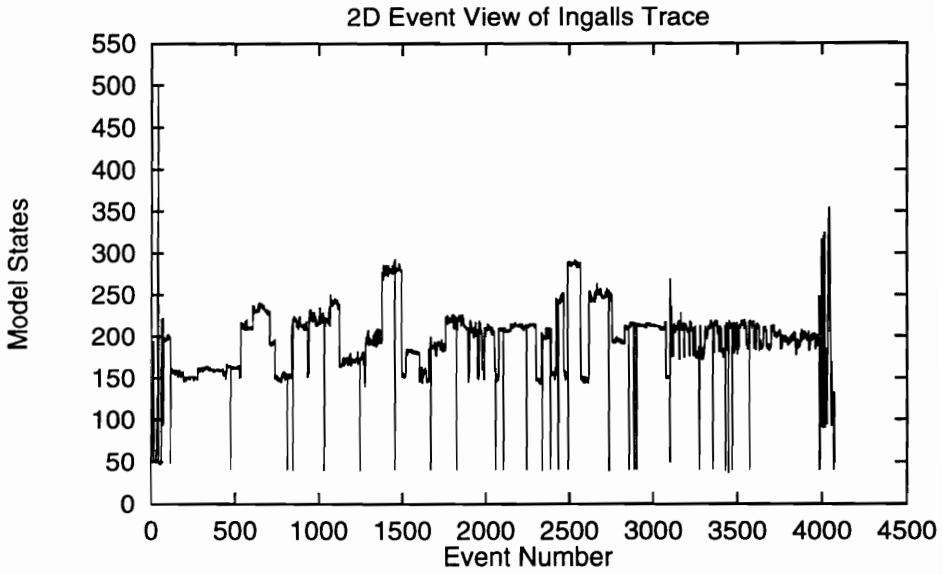


Figure 5.2: 2D View for the Ingalls Trace

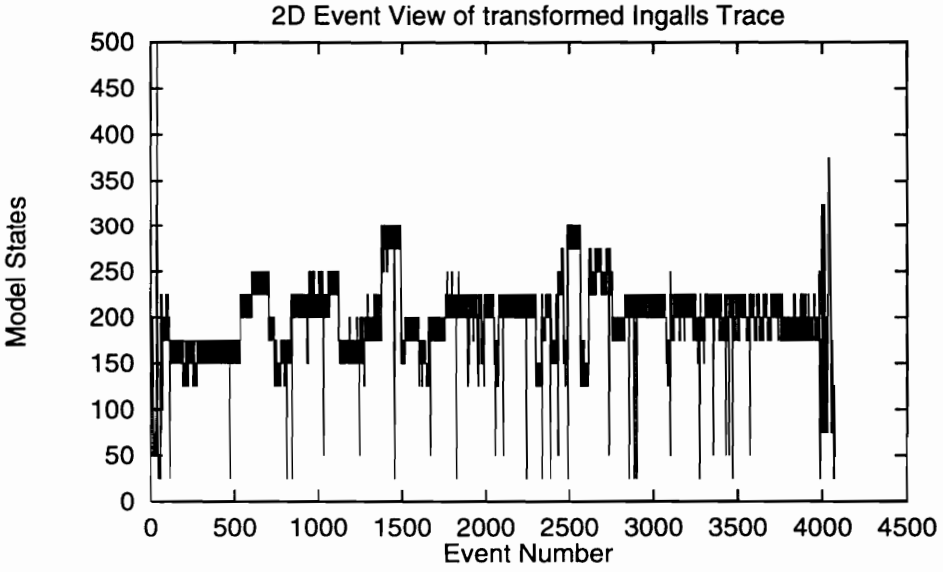


Figure 5.3: 2D View for the Transformed Ingalls Trace (k=20)

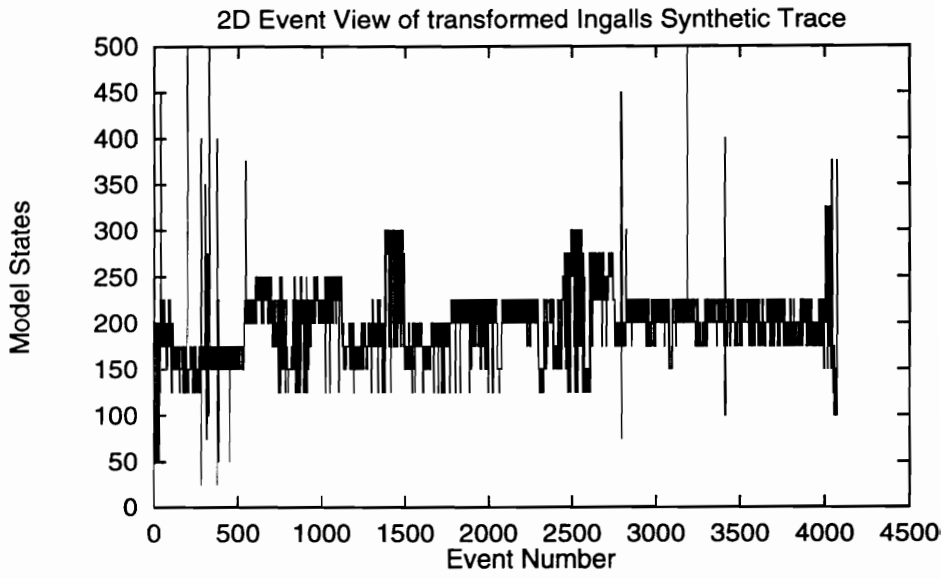


Figure 5.4: 2D View for the Transformed Ingalls Synthetic Trace

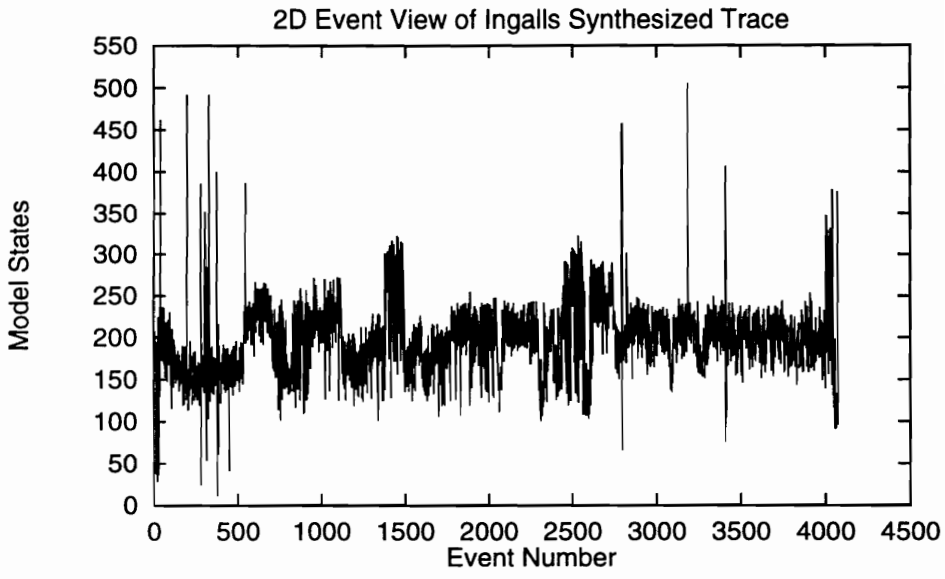


Figure 5.5: 2D View for the Ingalls Synthetic Trace (with noise added)

Chapter 6

MODEL VALIDATION

In this chapter we demonstrate the application of the piecewise independent stochastic process model to twelve test case traces. We describe three techniques that were used to validate our model. Finally, we also present the results of the validation techniques on piecewise independent stochastic process models of the test case traces.

6.1 Test Case Traces

We chose traces from real applications as well as traces simulated from known distributions to evaluate our modeling approach. The traces were chosen such that they cover a wide spectrum of performance data observed in real systems. The following different classes of traces were selected:

- Traces that contain categorical data as well as traces that contain numerical data.
- Traces from real systems as well as traces synthesized from known probability distributions.
- Traces with a large number of unique states as well as traces with a small number of unique states.
- Traces varying in length by several orders of magnitude (i.e., 100 to 170,998).
- Traces that represent current areas of interest in computer performance analysis such as memory reference behavior of parallel and distributed programs, behavior of transaction processing systems, packetized traffic on Ethernet and WWW, and VBR encoded video traffic.

6.1.1 Description of Traces

Table 6.1 describes various aspects of interest about the traces.

Table 6.1: Description of Test Case Traces

Name	State	Data Type	Number of States	Trace Length (events)
GE Message Trace	Processor Number	Categorical	31	409
Dining Philosophers Trace	Philosopher Numbers	Categorical	597	5857
UoS Ethernet Trace	Host Name	Categorical	11	3772
IBM CICS Trace	Transaction Type	Categorical	24	25953
GE Memory Trace	Memory Address	Categorical	48	376
WWW Trace	File Type	Categorical	7	9139
Ingalls Trace	Packet Size	Numerical	199	4076
ACM Trace	Packet Size	Numerical	18	1739
Star Wars Trace	Packet Size	Numerical	236	170998
Periodic Trace	Numerical State	Numerical	2	100
Time-trend Trace	Numerical State	Numerical	2	999
Mean-change Trace	Numerical State	Numerical	4	999

GE Message Trace: This trace was obtained from an instrumented implementation of the Gaussian elimination algorithm, which solves a 64×64 system of equations using 32 processors on an NCUBE multicomputer [Rover and Waheed, 1993]. 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 number of the processor that last sent a message.

Dining Philosophers Trace: This trace was obtained by executing a parallel implementation of Dijksta's dining philosophers problem (for five philosophers) on a Sequent Symmetry multiprocessor. Each state in the trace is a five tuple (one tuple component per philosopher). Each tuple component records whether the corresponding philosopher is

Thinking, Acquiring fork 1, *Acquiring fork* 2, *Eating, Releasing fork* 1, or, *Releasing fork* 2. The original trace was transformed to a trace of 1−tuples as detailed in Appendix D.

UoS Ethernet Trace: This is a trace of Ethernet traffic from the University of Saskatchewan. The trace records the addresses of machines generating traffic to a certain host on the Ethernet.

IBM CICS Trace: This trace was obtained from a CICS transaction system running on an IBM 3090. The trace records calls made to various CICS functions during a 19 minute interval of system operation.

GE Memory Trace: This trace was obtained by executing a sequential implementation of the Gaussian elimination algorithm. The trace records the memory addresses referenced during the execution.

WWW Trace: This is trace of WWW traffic originating from a class of undergraduate students at Virginia Tech. The trace was obtained as a part of on-going WWW research [Abrams et al., 1995] at Virginia Tech. The trace records the types of files (*Graphics, Directory, Audio, Text_or_HTML, Script, Video*, or *Other*) which were accessed.

Ingalls Trace: This is a trace of frame sizes from the MPEG movie of a videotape presentation titled "Object-Oriented Programming" by Dan Ingalls. The trace was recorded at 30 frames per second using a display window size of 320 lines × 240 pixels. [Vazirani, 1996]. The original trace was transformed, as detailed in Appendix C.

ACM Trace: This is a trace of frame sizes from the MPEG movie of a videotape presentation titled "An introduction to interactive digital video". The trace was recorded at 10 frames per second using a display window size of 320 lines × 240 pixels. For more details about this trace see [Vazirani, 1996]. The original trace was transformed, as detailed in Appendix C.

Star Wars Trace: This is a trace of frame sizes of the Star Wars movie [Garrett, 1993] [1]. The Star Wars trace is a sequence of frame sizes from the movie *Star Wars* (running time 2 hours). The original trace was transformed, as detailed in Appendix C.

Periodic Trace: This trace is generated from a known synthetic distribution. The two unique states in this trace occur with identical probability (0.5).

Time-trend Trace: This trace is generated from a known synthetic distribution. The trace has two unique states (0 and 1). The probability of the first state increases proportional to the length of the trace. The probability of the second state decreases proportional to the length of the trace. Denote the trace as a time series $\{X_t\}$ where $1 \leq t \leq N$ and the random variables X_t are IID. The probability mass for the random variables X_t, denoted $P\{X_t\}$ is given by:

$$P\{X_t = x\} = \begin{cases} \frac{t}{N} & \text{if } x = 0 \\ 1 - \frac{t}{N} & \text{if } x = 1. \end{cases}$$

Mean-change Trace: This trace is generated from a known synthetic distribution. The trace has four unique states. The main behavioral characteristic of this trace is that it shows three intervals of distinct behavior. Within each interval only two out of the four states occur, each with probability 0.5.

6.1.2 REG Plots of Test Case Traces

We introduced the REG as a tool for estimating the parameters of the piecewise stochastic process. The REG also provides insight into the stochastic process generating the trace. In this section we present the REG plots for the test case traces (Section 6.1).

[1] The trace is obtainable to the public via anonymous ftp from thumper.bellcore.com under the directory /pub/vbr.video.traces

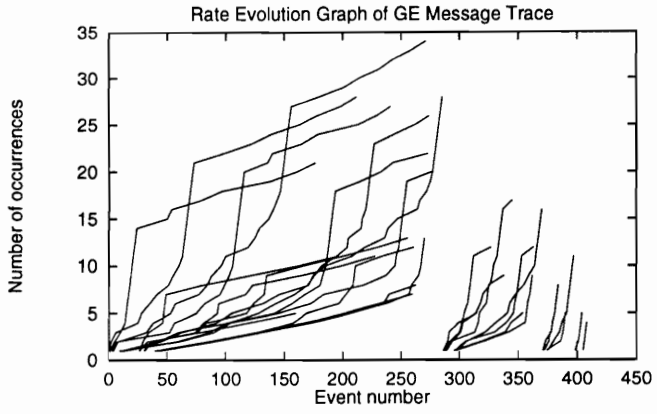


Figure 6.1: Rate Evolution Graph for the GE Message Trace

GE Message Trace: The REG for the GE Message trace is shown in Figure 6.1. The REG paths in this trace are made up of segments of straight lines. Thus this trace is a good test cases for our model. Another interesting feature of the trace highlighted by the REG is that the trace has four distinct group of states. Each group occupies a disjoint section of the event number axis. This suggests that the underlying stochastic process does not return to a state belonging to an already visited group once it has left that group. This interesting behavior is similar to that of certain finite Markov chains [Bhat, 1984]. If each of the four group of states in the REG can be considered as an equivalence class of a finite Markov chain. The first three equivalence classes are *transient* while the last one is *recurrent.*

Dining Philosophers Trace: The REG for the Dining Philosophers trace is shown in Figure 6.2. The REG of the Dining Philosophers trace has many features similar to the GE message trace. The REG paths are made up of straight lines. Also, as in the GE message trace there exist distinct groups of states such that each group occupies a disjoint section of the event number axis. Although the number of unique states in this state is large, the REG indicates that this trace is a good test case for our model.

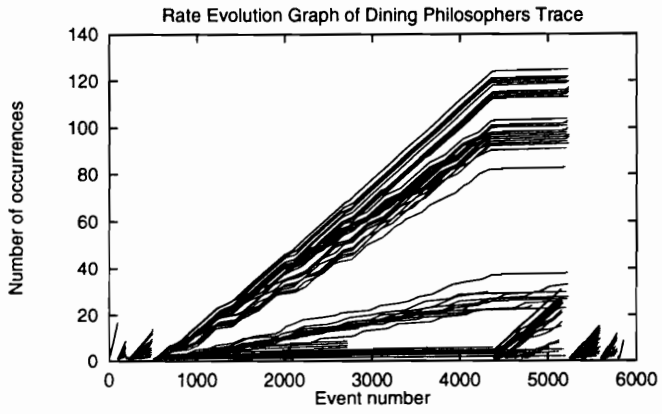


Figure 6.2: Rate Evolution Graph for the Dining Philosophers Trace

UoS Ethernet Trace: The REG for the UoS Ethernet trace is shown in Figure 6.3. The number of states in this trace is small which helps our modeling approach, however, many paths in the REG consist of a large number of small straight line segments. This will add to the number of partitioning events needed for the model.

IBM CICS Trace: The REG for the IBM CICS trace is shown in Figure 6.4. The REG for this trace shows that all the REG paths consist of a few straight line segments. Our modeling approach is ideally suited to a trace such as this one.

GE Memory Trace: The REG for the GE Memory trace is shown in Figure 6.5. The REG for this trace is dense due to a large number of unique states in the trace. While the REG paths have few segments many of the segments are smooth curves and not straight lines. These two factors will limit the effectiveness of our model.

WWW Trace: The REG for the WWW trace is shown in Figure 6.6. This is a trace with very few states. Unlike the REG plots of other traces considered so far the REG paths for this trace can be fitted by single straight lines. This indicates that the probabilities of occurrence of the states in this trace remain almost constant. Thus this trace shows

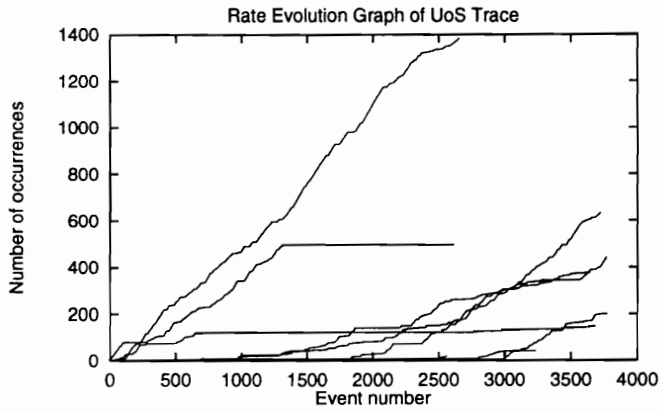


Figure 6.3: Rate Evolution Graph for the UoS Ethernet Trace

stationary (instead of transient) behavior. Our model for this trace should be compact and accurate.

Ingalls Trace: The REG for the Ingalls trace is shown in Figure 6.7. The REG for this trace has two main features. The REG is dense because the trace has a large number of states. This feature limits the effectiveness of our model because the size of the model increases. A large portion of the REG paths consist of a few straight line segments. This feature helps our model to estimate model parameters accurately.

ACM Trace: The REG for the ACM trace is shown in Figure 6.8. The ACM trace does not have a large number of states. This will help in reducing the model size. However, many REG paths in this trace consist of several small straight line segments. This means that an accurate model will require more parameters to be estimated.

Star Wars Trace: The REG for the Star Wars trace is shown in Figure 6.9. The REG for the Star Wars trace demonstrates that this trace is a bad case for our model. The trace has an extremely large number of states. Also, each REG path consists of several small straight line segments. These two factors will severely limit the effectiveness of our model

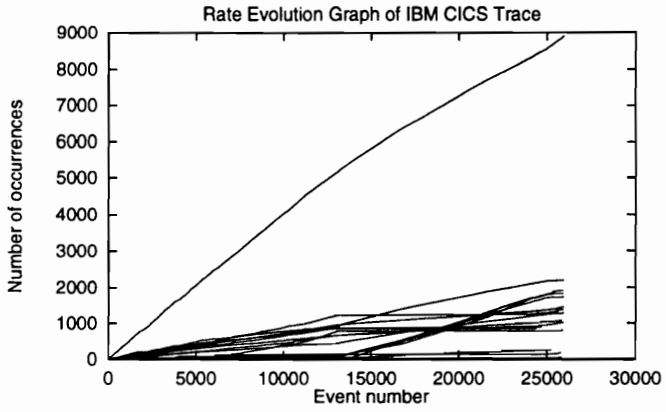


Figure 6.4: Rate Evolution Graph for the IBM CICS Trace

for this trace.

Periodic Trace: The REG for the Periodic trace is shown in Figure 6.10. The Periodic trace has only two states. Also, its REG paths can be fitted by a small number of straight line segments. Thus, our model for this trace should be accurate and compact.

Time-trend Trace: The REG for the Time-trend trace is shown in Figure 6.11. The REG for the Time-trend trace represents another case which exposes a weakness in our modeling approach. The REG paths for this trace are smooth curves. Consequently it takes many straight lines to fit a REG path. However, the fact that this trace has only two unique states and the underlying stochastic process is simple (Section 6.1) helps our modeling approach.

Mean-change Trace: The REG for the Mean-change trace is shown in Figure 6.12. The trace has a small number of states. Also, the REG paths for this trace can be can be fitted by very few straight line segments. Thus this trace should be a good test case for model.

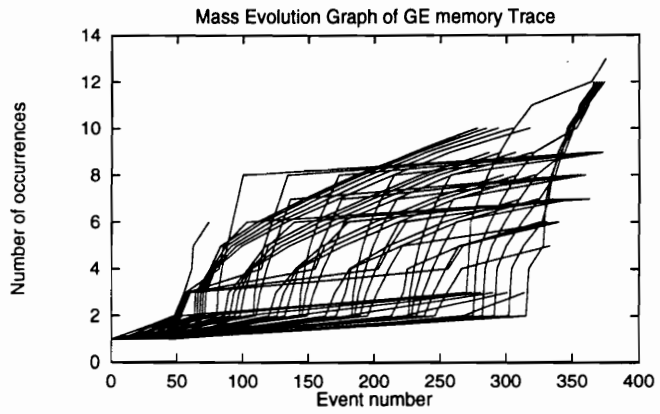


Figure 6.5: Rate Evolution Graph for the GE Memory Trace

6.2 Validation Techniques

In this section we describe the techniques used to evaluate the application of our modeling approach on the test case traces described in Section 6.1. Our choice of validation techniques is based on the appropriate validation techniques available in CHITRA (Section 2.3.2). We chose the *2D view*, as the *graphical comparison* validation technique. As statistical validation techniques we chose the *Kruskall-Wallis* test and *Correlation Analysis*. The Kruskall-Wallis test is an appropriate test in our case because it is a hypothesis test that makes no assumptions about the distribution of the sample data being tested and it is suitable for discrete probability distributions. For the correlation analysis of traces containing categorical data we used the *categorical correlation plots* described later in this section. For the correlation analysis of traces containing numerical data we employed *autocorrelation plots* as well as categorical correlation plots.

6.2.1 2D Views

We use 2D views of traces for the validation of our model. We visually compare the 2D view of a test case trace with the 2D views of traces synthesized from our model of the test case trace.

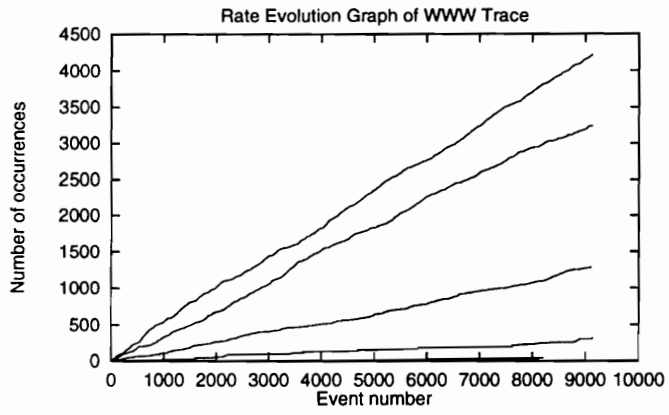


Figure 6.6: Rate Evolution Graph for the WWW Trace

6.2.2 Kruskall-Wallis Test

The KW test 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 *Kruskall-Wallis rank sum test* [Ott, 1988]. The Kruskall-Wallis test is a non-parametric test and thus it does not make any assumptions about the shape of the distribution of the trace data.

To validate our model using the Kruskall-Wallis test we first synthesize traces from our model. A synthesized trace is then compared with the original trace using the Kruskall-Wallis test. To appropriately test the time dependent nature of the trace data we divide the synthesized and the original traces into ten segments and then apply the Kruskall-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.

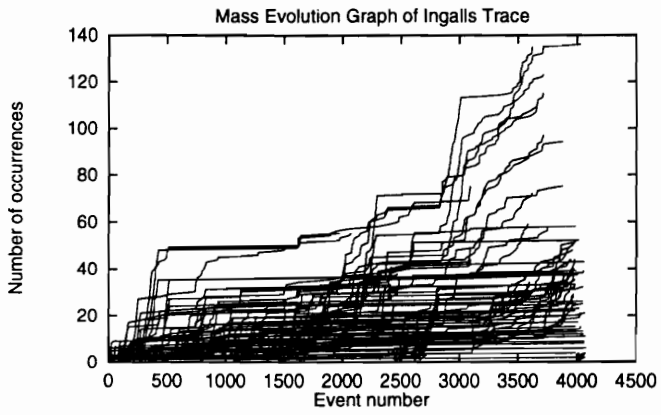


Figure 6.7: Rate Evolution Graph for the Ingalls Trace

6.2.3 Autocorrelation Plots

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 [Box and Jenkins, 1976] or any other standard book in time series analysis for details about autocorrelation. By varying the lag k we can plot the autocorrelation function $r(k)$ versus k to generate a *autocorrelation plot.* The autocorrelation plot provides an insight into the probability model which generated the data. We use the autocorrelation plot as a test by first generating synthesized traces from our model, and then comparing autocorrelation plots obtained from the synthesized traces to the autocorrelation plot obtained from the original test case trace.

6.2.4 Categorical Correlation Plots

The categorical correlation plots [Ribler et al., 1995] measure the degree of correlation between two traces. To validate our model we measure the categorical correlation between

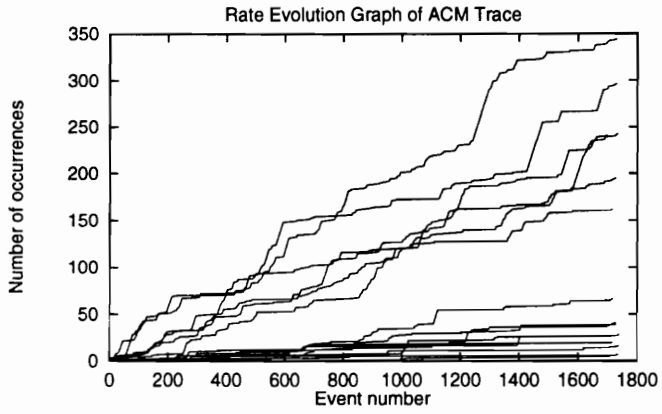


Figure 6.8: Rate Evolution Graph for the ACM Trace

a test case trace and a synthetic trace generated from our model of the test case trace. The categorical correlation plots are constructed using a moving window, denoted W, of size 32 events. The categorical correlation is plotted as the fraction of states (x_i), within W, that are common to the test case trace as well as the synthetic trace. The categorical correlation is plotted on the ordinate and is always in the range $[0-1]$. A value of 0 indicates the two traces have no states in common within the window W. A value of 1 indicates that two traces contain 32 identical states (not necessarily in the same order) within the window W.

6.3 Validation Results

We begin this section by describing the steps taken to arrive at a compact piecewise independent stochastic process model for the test case traces (Section 6.1). The compact model for a trace is a result of applying the model size reduction techniques described in Chapter 5 to the original piecewise independent stochastic process model for that trace. Next, we present the results of the Kruskall-Wallis test, Correlation plots, and 2D views when applied to the compact model of the test case traces. Finally, we also present the size of the compact models for the test case traces.

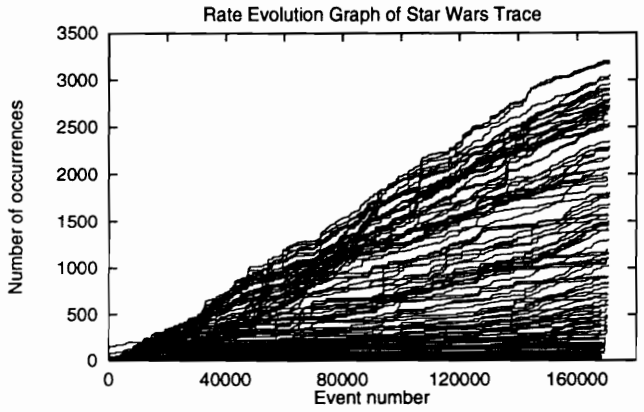


Figure 6.9: Rate Evolution Graph for the Star Wars Trace

GE Message Trace: The size of the model for the GE Message trace was reduced by reducing the number of partitioning events in the model. The model was reduced iteratively until only one partitioning event was stored for each state in the model. The results of the Kruskall-Wallis test (Table 6.2) show that reduction in model size did not result in loss of model accuracy. This required 2 iterations of the model reduction algorithm. Table 6.3 shows the size of the final model.

Dining Philosophers Trace: The model for the Dining Philosophers Trace has only one partitioning event for each state. Consequently, this model cannot be reduced any further. Table 6.3 shows the size of the final model.

UoS Ethernet Trace: The size of the model for the UoS Ethernet trace was reduced by reducing the number of partitioning events in the model. The model was reduced iteratively until further reduction in model size reduced model accuracy significantly. The Kruskall-Wallis test was used to evaluate model accuracy. Figure 6.13 plots the number of iterations of the model size reduction algorithm as the abscissa. The ordinate in this figure shows the percentage of segments of a trace synthesized from the reduced model (corresponding to the number of iterations of the model size reduction algorithm) that pass the Kruskall-Wallis

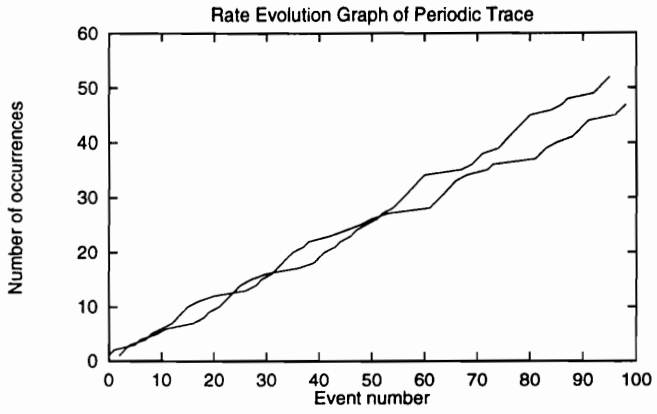


Figure 6.10: Rate Evolution Graph for the Periodic Trace

test (Section 6.2.2). Inspection of Figure 6.13 shows that 2 iterations of the model reduction algorithm serves as a good cut off point up-till which the Kruskall-Wallis test shows good results. Consequently, we apply 2 iterations of the model size reduction algorithm to our model. Table 6.3 shows the size of the final model.

IBM CICS Trace: The size of the model for the IBM CICS trace was reduced by reducing the number of partitioning events in the model. The model was reduced iteratively until further reduction in model size reduced model accuracy significantly. The Kruskall-Wallis test was used to evaluate model accuracy. Figure 6.14 plots the number of iterations of the model size reduction algorithm as the abscissa. The ordinate in this figure shows the percentage of segments of a trace synthesized from the reduced model (corresponding to the number of iterations of the model size reduction algorithm) that pass the Kruskall-Wallis test (Section 6.2.2). Inspection of Figure 6.14 shows that 2 iterations of the model reduction algorithm serves as a good cut off point up-till which the Kruskall-Wallis test shows good results. Consequently, we apply 2 iterations of the model size reduction algorithm to our model. Table 6.3 shows the size of the final model.

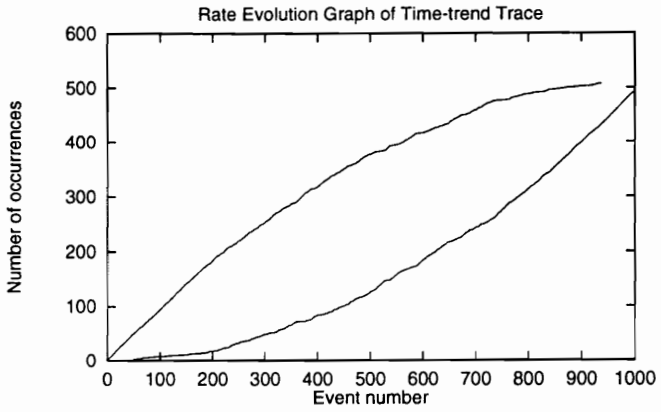


Figure 6.11: Rate Evolution Graph for the Time-trend Trace

GE Memory Trace: The size of the model for the GE Memory trace was reduced by reducing the number of partitioning events in the model. The model was reduced iteratively until only one partitioning event was stored for each state in the model. The results of the Kruskall-Wallis test (Table 6.2) show that reduction in model size did not result in loss of model accuracy. This required 1 iteration of the model reduction algorithm. Table 6.3 shows the size of the final model.

WWW Trace: The size of the model for the WWW Trace was reduced by reducing the number of partitioning events in the model. The model was reduced iteratively until only one partitioning event was stored for each state in the model. The results of the Kruskall-Wallis test (Table 6.2) show that reduction in model size did not result in loss of model accuracy. This required 8 iterations of the model reduction algorithm. Table 6.3 shows the size of the final model.

Ingalls Trace: The size of the model for the Ingalls trace was reduced by first reducing the number of unique states in the trace by 90%. This was followed by reducing the number of partitioning events in the model. Reducing the number of states is detailed in Section 5.2. To reduce the number of partitions the model was reduced iteratively until

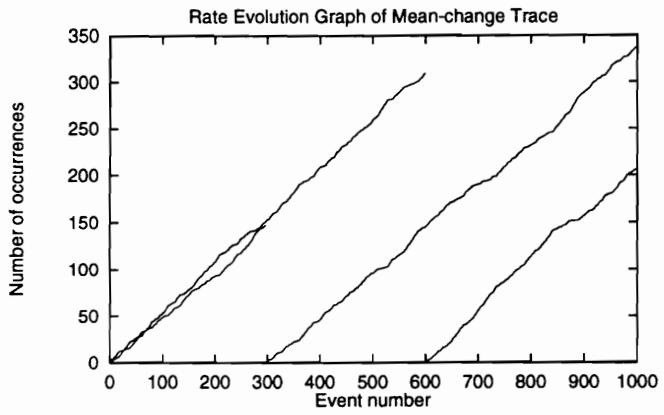


Figure 6.12: Rate Evolution Graph for the Mean-change Trace

further reduction in model size reduced model accuracy significantly. The Kruskall-Wallis test was used to evaluate model accuracy. Figure 6.15 plots the number of iterations of the model size reduction algorithm as the abscissa. The ordinate in this figure shows the percentage of segments of a trace synthesized from the reduced model (corresponding to the number of iterations of the model size reduction algorithm) that pass the Kruskall-Wallis test (Section 6.2.2). Inspection of Figure 6.15 shows that 1 iteration of the model reduction algorithm serves as a good cut off point up-till which the Kruskall-Wallis test shows good results. Consequently, we apply 1 iteration of the model size reduction algorithm to our model. Table 6.3 shows the size of the final model.

ACM Trace: The size of the model for the ACM trace was reduced by reducing the number of partitioning events in the model. The model was reduced iteratively until further reduction in model size reduced model accuracy significantly. The Kruskall-Wallis test was used to evaluate model accuracy. Figure 6.16 plots the number of iterations of the model size reduction algorithm as the abscissa. The ordinate in this figure shows the percentage of segments of a trace synthesized from the reduced model (corresponding to the number of iterations of the model size reduction algorithm) that pass the Kruskall-Wallis test (Section 6.2.2). Inspection of Figure 6.16 shows that 1 iteration of the model reduction

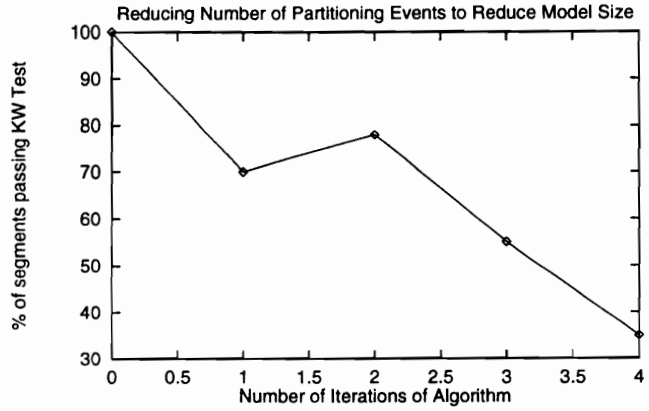


Figure 6.13: Model Size Reduction for the UoS Ethernet Trace

algorithm serves as a good cut off point up-till which the Kruskall-Wallis test shows good results. Consequently, we apply 1 iteration of the model size reduction algorithm to our model. Table 6.3 shows the size of the final model.

Star Wars Trace: The size of the model for the Star Wars trace was reduced by first reducing the number of unique states in the trace by 83%. This was followed by 1 iteration of the algorithm that reduces model size by reducing the number of partitioning events in the model. The original as well as reduced piecewise independent stochastic process models for the Star Wars trace show poor results when evaluated using the Kruskall-Wallis test. Only 40% of the segments of a trace synthesized from the original as well as the reduced models matched their counterparts in the original Star Wars trace. Table 6.3 shows the size of the final model for the Star Wars trace.

Periodic Trace: The size of the model for the Periodic trace was reduced by reducing the number of partitioning events in the model. The model was reduced iteratively until only one partitioning event was stored for each state in the model. The results of the Kruskall-Wallis test (Table 6.2) show that reduction in model size did not result in loss of model accuracy. This required 3 iterations of the model reduction algorithm. Table 6.3 shows the

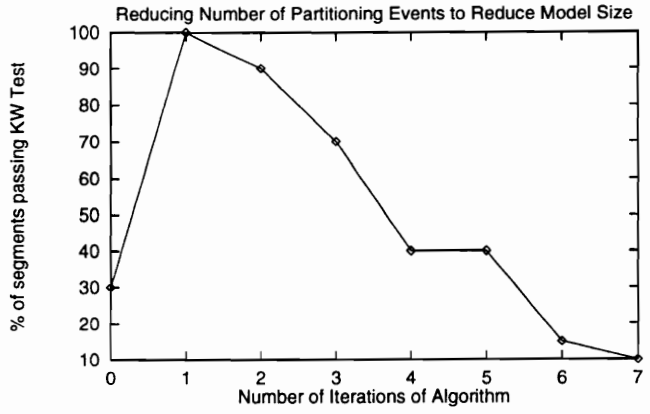


Figure 6.14: Model Size Reduction for the IBM CICS Trace

size of the final model.

Time-trend Trace: The size of the model for the Time-trend trace was reduced by reducing the number of partitioning events in the model. The model was reduced iteratively until further reduction in model size reduced model accuracy significantly. The Kruskall-Wallis test was used to evaluate model accuracy. Figure 6.17 plots the number of iterations of the model size reduction algorithm as the abscissa. The ordinate in this figure shows the percentage of segments of a trace synthesized from the reduced model (corresponding to the number of iterations of the model size reduction algorithm) that pass the Kruskall-Wallis test (Section 6.2.2). Inspection of Figure 6.17 shows that 3 iterations of the model reduction algorithm serves as a good cut off point up-till which the Kruskall-Wallis test shows good results. Consequently, we apply 3 iterations of the model size reduction algorithm to our model. Table 6.3 shows the size of the final model.

Mean-change Trace: The size of the model for the Mean-change trace was reduced by reducing the number of partitioning events in the model. The model was reduced iteratively until further reduction in model size reduced model accuracy significantly. The Kruskall-Wallis test was used to evaluate model accuracy. Figure 6.18 plots the number of iterations

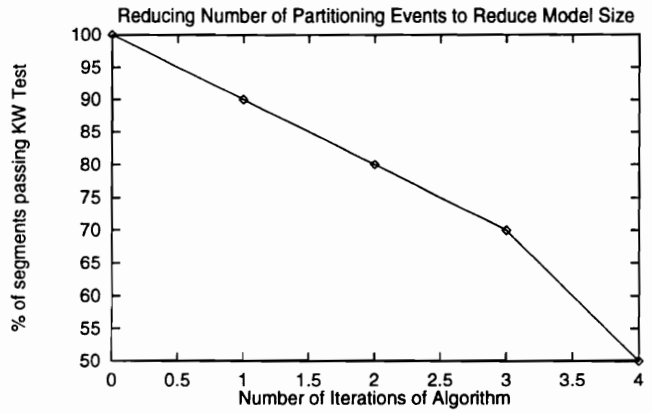


Figure 6.15: Model Size Reduction for the Ingalls Trace

of the model size reduction algorithm as the abscissa. The ordinate in this figure shows the percentage of segments of a trace synthesized from the reduced model (corresponding to the number of iterations of the model size reduction algorithm) that pass the Kruskall-Wallis test (Section 6.2.2). Inspection of Figure 6.18 shows that 2 iterations of the model reduction algorithm serves as a good cut off point up-till which the Kruskall-Wallis test shows good results. Consequently, we apply 2 iterations of the model size reduction algorithm to our model. Table 6.3 shows the size of the final model.

6.3.1 Kruskall-Wallis Test

Table 6.2 shows the results of the Kruskall-Wallis test on the test case traces (Section 6.1). With the exception of the Dining Philosophers trace and the Star Wars trace, all the other traces synthesized from our model match their corresponding original traces in 7 segments out of 10 or more. The model is exceptionally accurate for the GE Message trace, IBM CICS trace, Mean-change trace, and Periodic trace. The Kruskall-Wallis test does not show our model to be good fit for the Dining Philosophers trace. However, this result is contradicted by other two validation tests for this trace, namely the categorical correlation plot and 2D views. A possible explanation for the Kruskall-Wallis test results for the Dining

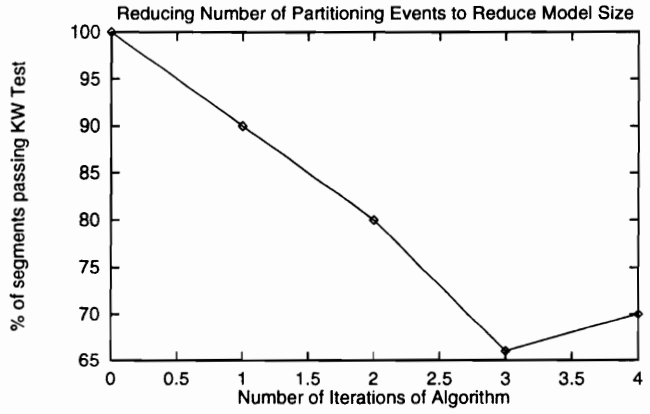


Figure 6.16: Model Size Reduction for the ACM Trace

Philosophers trace is the fact that this trace has a very large number of unique states (more than 10% of the trace length).

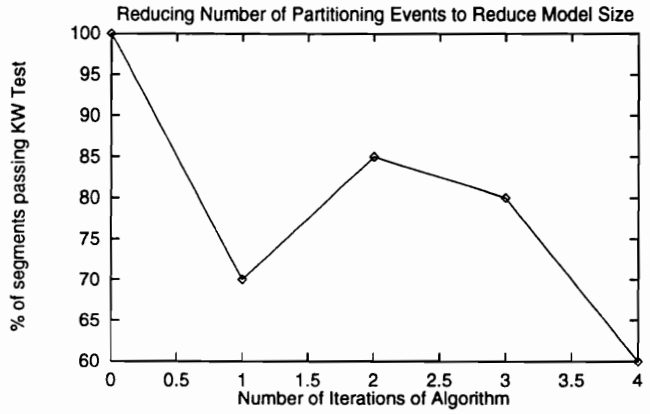


Figure 6.17: Model Size Reduction for the Time-trend Trace

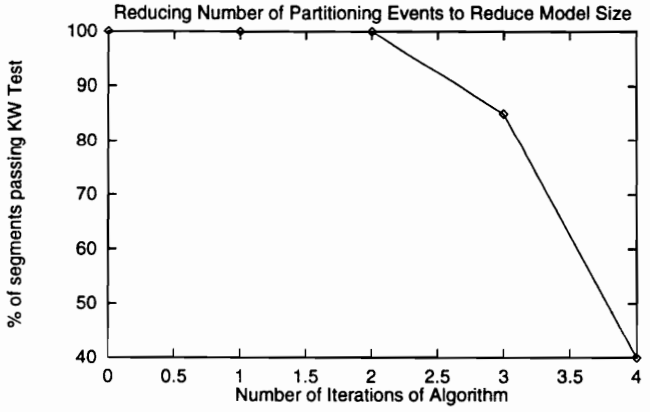


Figure 6.18: Model Size Reduction for the Mean-change Trace

Table 6.2: Results of the Kruskall-Wallis Test (Ordered by % of Segments that Pass the Test)

$\alpha = 0.01$, $\chi^2 = 6.6349$					
% of Segments that Pass the KW Test					
Name	Seed Index				
	1	*11*	*21*	*53*	*99*
Mean-change Trace	100	100	100	100	100
Periodic Trace	100	100	100	100	100
GE Message Trace	100	90	90	100	90
ACM Trace	90	100	90	80	90
Ingalls Trace	80	90	90	90	90
IBM CICS Trace	90	80	80	90	90
WWW Trace	70	90	70	90	90
UoS Ethernet Trace	80	80	80	80	70
Time-trend Trace	80	80	70	80	70
GE Memory Trace	70	80	60	60	80
Dining Philosophers Trace	40	40	40	40	40
Star Wars Trace	40	30	40	30	40

6.3.2 Correlation Plots and 2D Views

The correlation plots and 2D views of the test case traces are displayed in Figures 6.19-6.89.

The 2D views of the test case trace compare very favorably with their synthetic counterparts. The 2D views of the GE Message trace, IBM CICS trace, Dining Philosophers trace, Mean-change trace match their synthetic counterparts exceptionally well. The notable exception is the 2D view of the GE memory trace and its synthetic counterpart.

All categorical test case traces, with the exception of the Dining Philosophers trace and the GE Memory trace display an adequate degree of correlation with their synthetic counterparts (the correlation coefficient is greater than 0.5). The IBM CICS trace, WWW trace, Mean-change trace, and Periodic trace show notably high degree of correlation with their synthetic counterparts. This indicates that our model is a good fit for these traces.

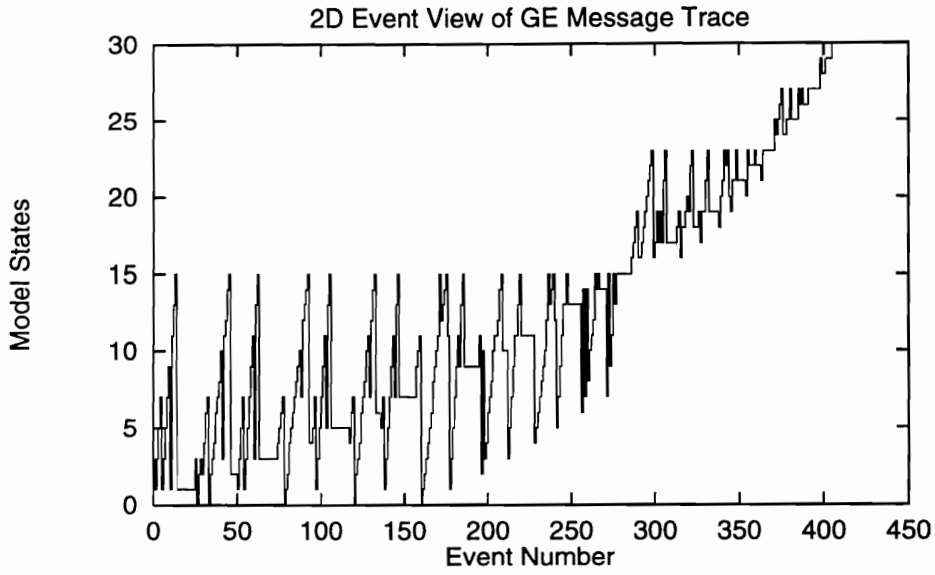


Figure 6.19: 2D View of the GE Message Trace

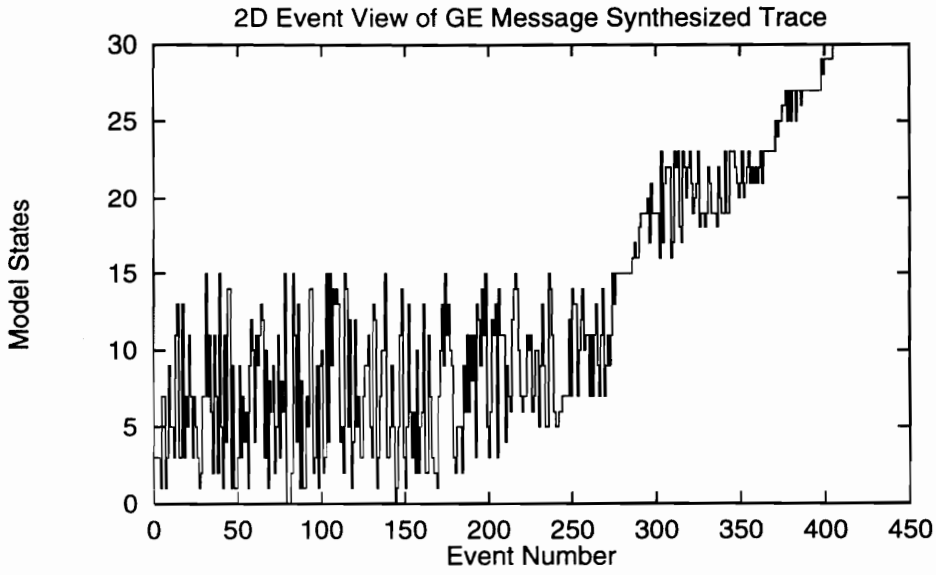


Figure 6.20: 2D View of the GE Message Synthesized Trace

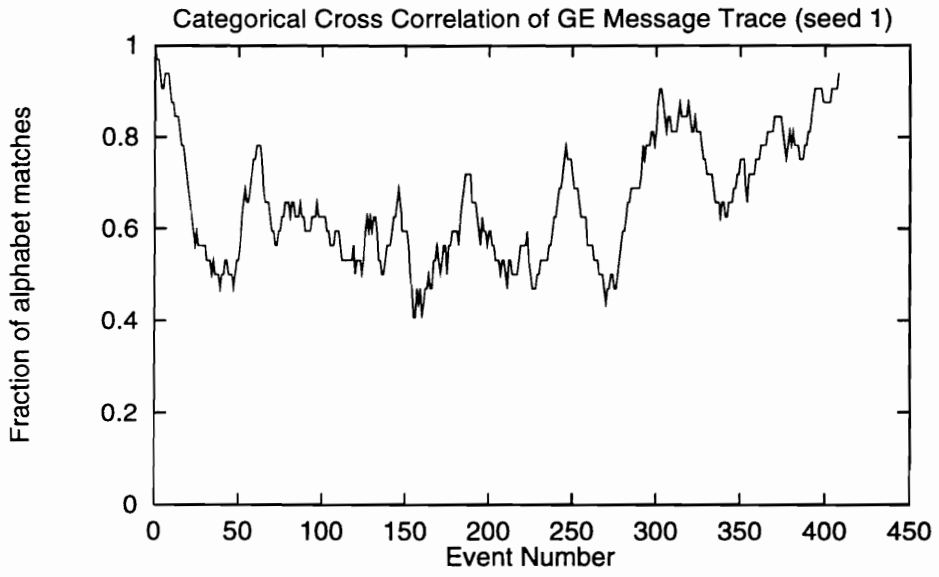


Figure 6.21: Categorical Alphabet Match Cross Correlation Plot: GE Message Original versus Synthesized Trace (seed 1)

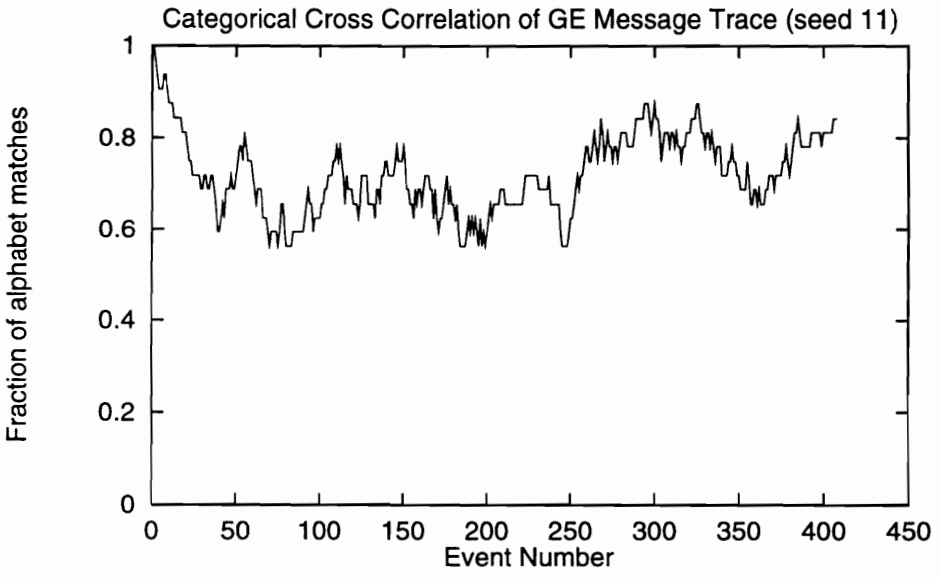


Figure 6.22: Categorical Alphabet Match Cross Correlation Plot: GE Message Original versus Synthesized Trace (seed 11)

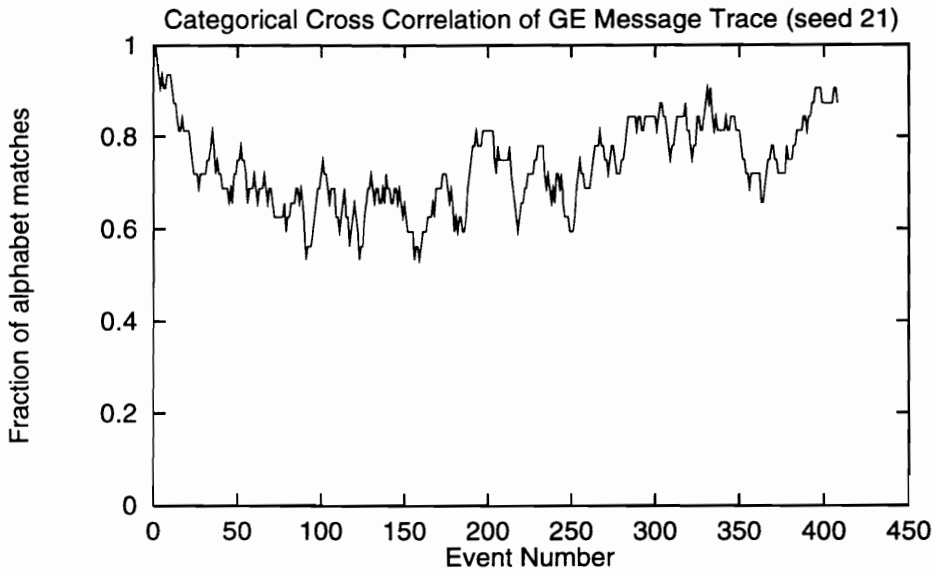


Figure 6.23: Categorical Alphabet Match Cross Correlation Plot: GE Message Original versus Synthesized Trace (seed 21)

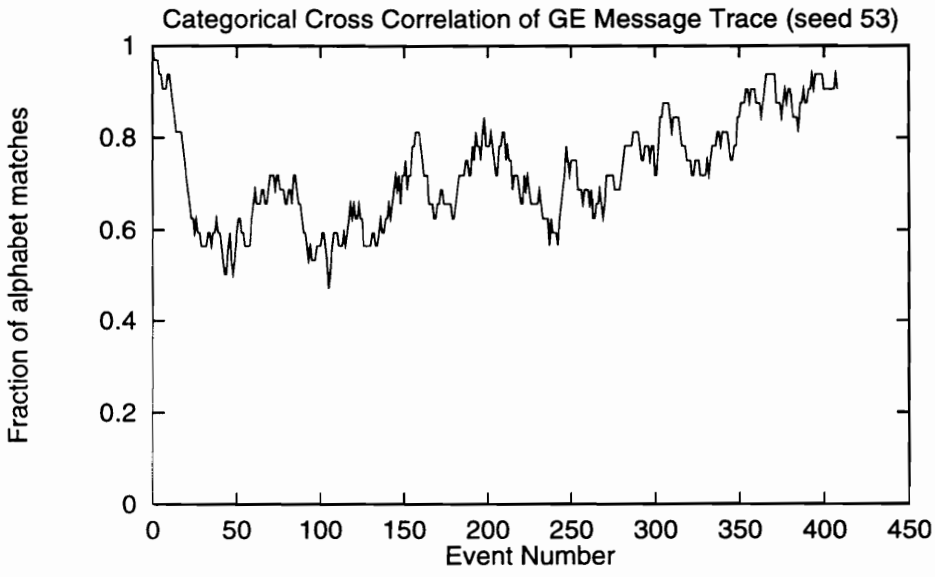


Figure 6.24: Categorical Alphabet Match Cross Correlation Plot: GE Message Original versus Synthesized Trace (seed 53)

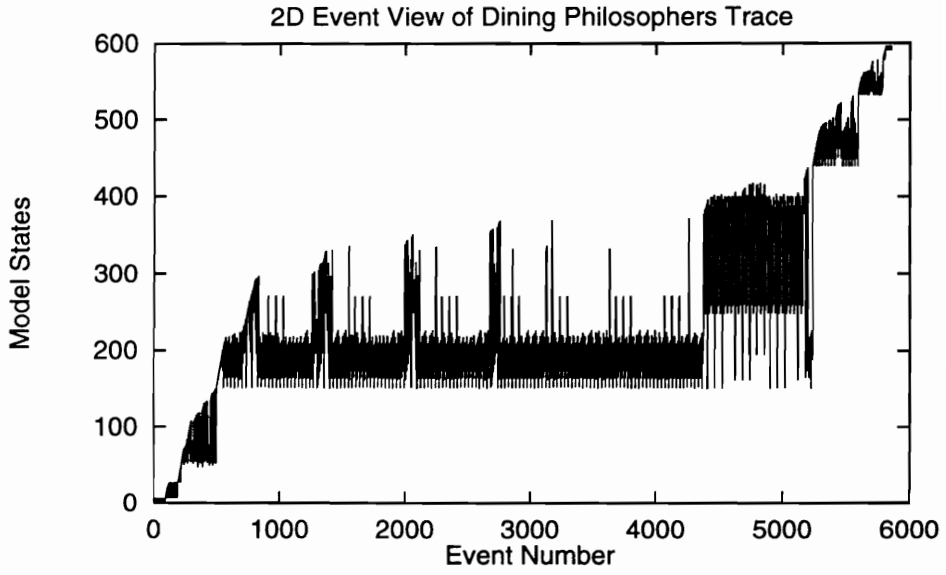


Figure 6.25: 2D View of the Dining Philosophers Trace

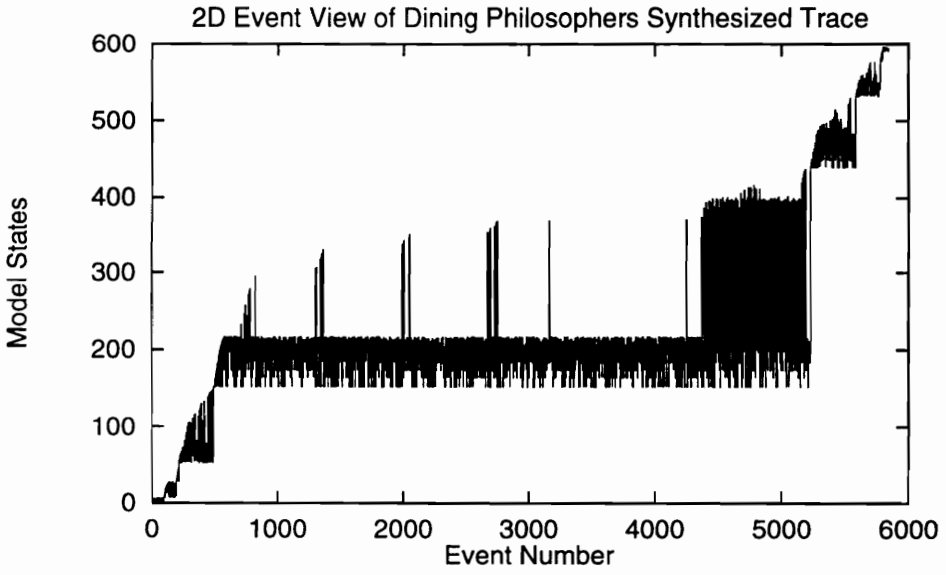


Figure 6.26: 2D View of the Dining Philosophers Synthesized Trace

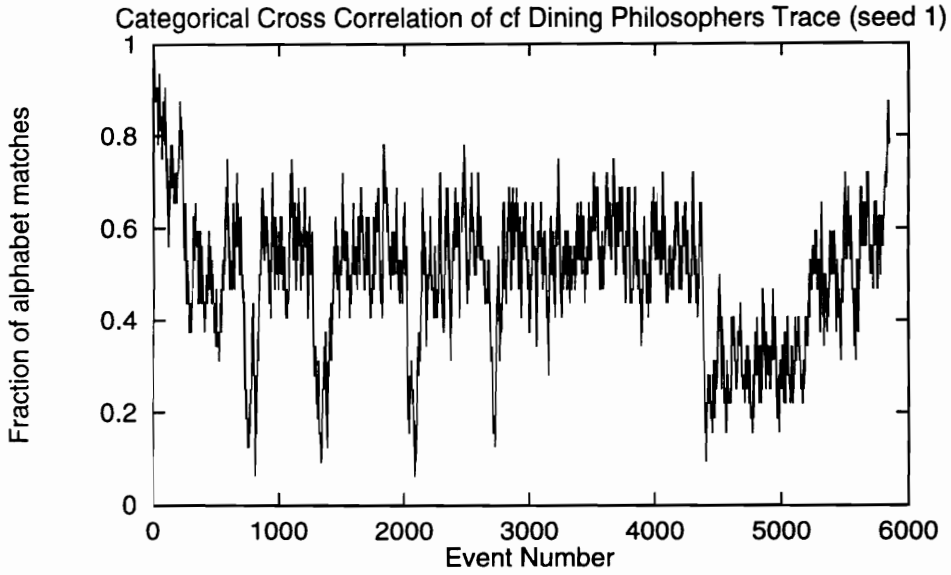


Figure 6.27: Categorical Alphabet Match Cross Correlation Plot: Dining Philosophers Original versus Synthesized Trace (seed 1)

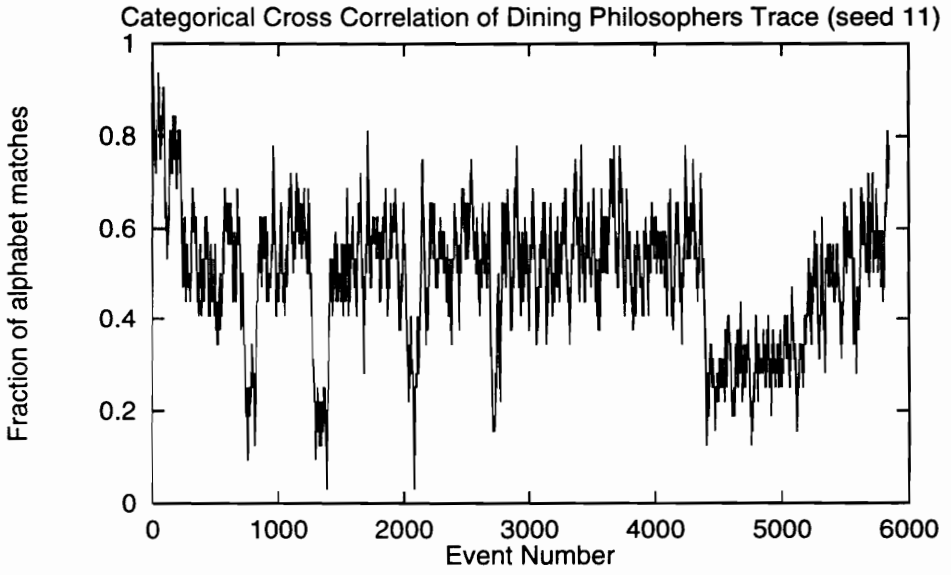


Figure 6.28: Categorical Alphabet Match Cross Correlation Plot: Dining Philosophers Original versus Synthesized Trace (seed 11)

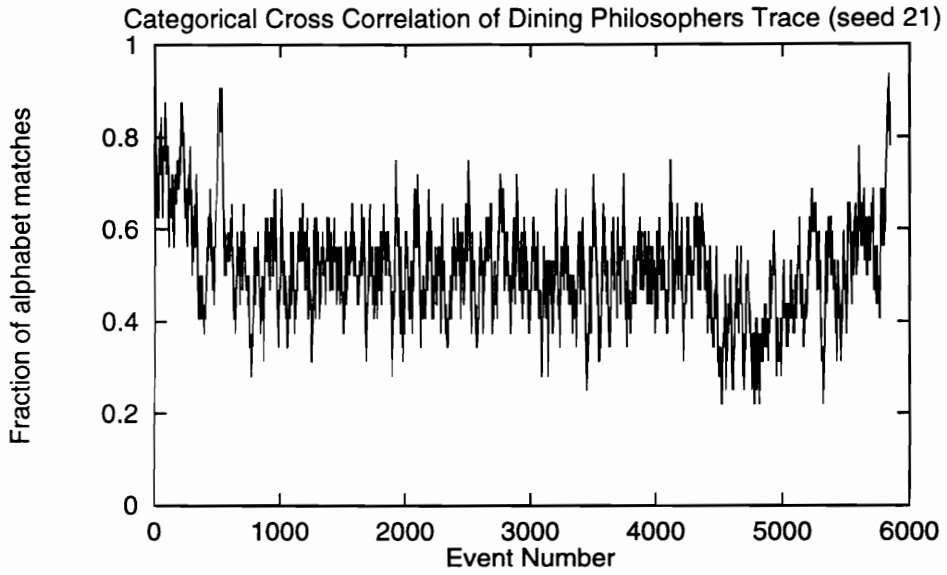


Figure 6.29: Categorical Alphabet Match Cross Correlation Plot: Dining Philosophers Original versus Synthesized Trace (seed 21)

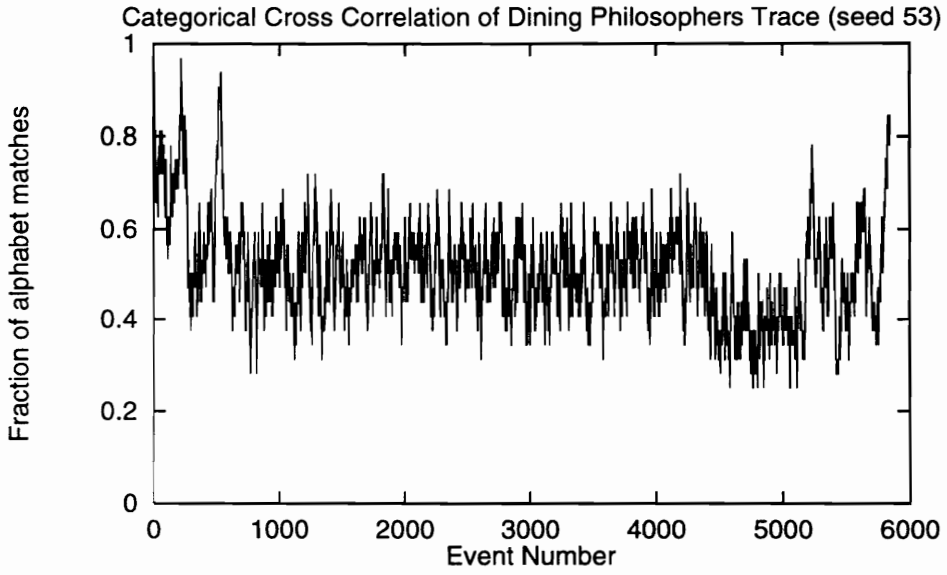


Figure 6.30: Categorical Alphabet Match Cross Correlation Plot: Dining Philosophers Original versus Synthesized Trace (seed 53)

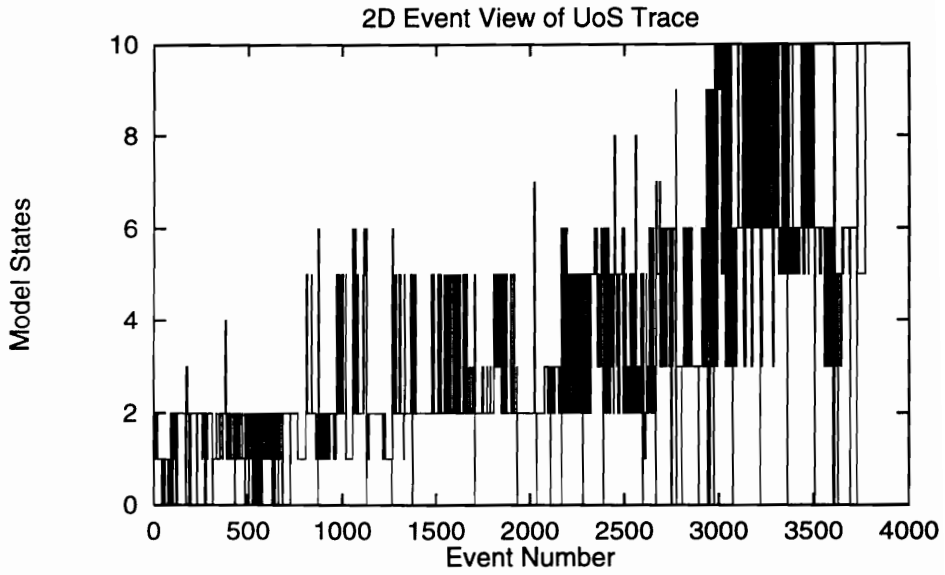


Figure 6.31: 2D View of the UoS Ethernet Trace

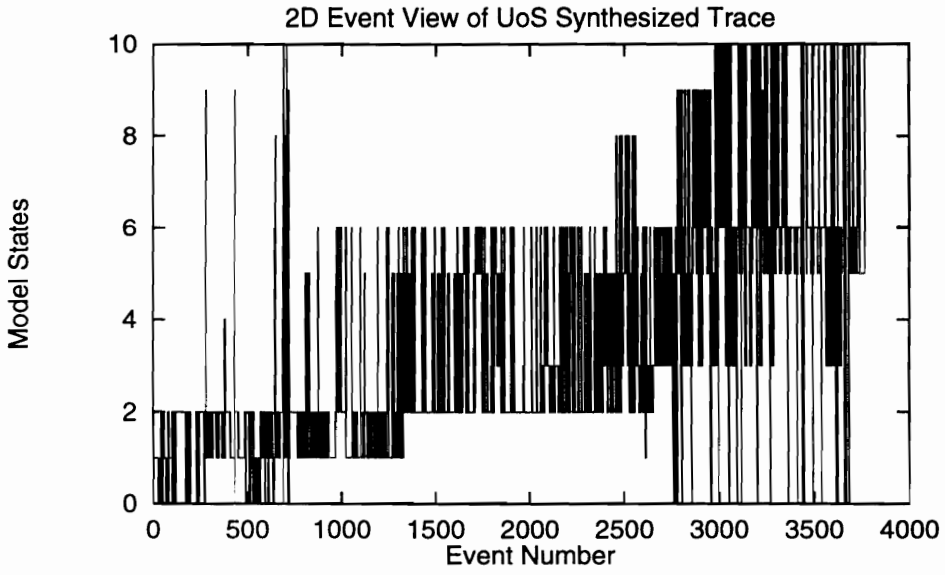


Figure 6.32: 2D View of the UoS Ethernet Synthesized Trace

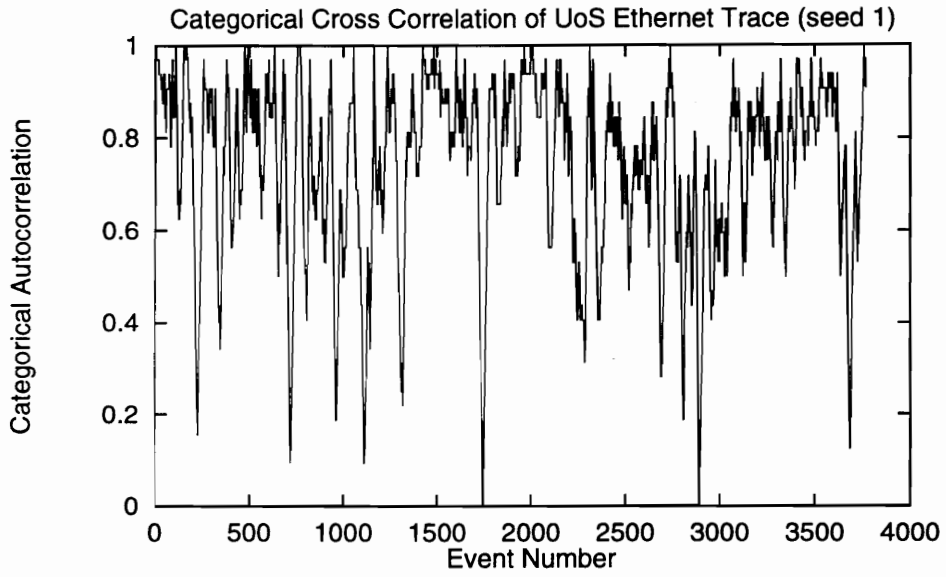


Figure 6.33: Categorical Alphabet Match Cross Correlation Plot: UoS Ethernet Original versus Synthesized Trace (seed 1)

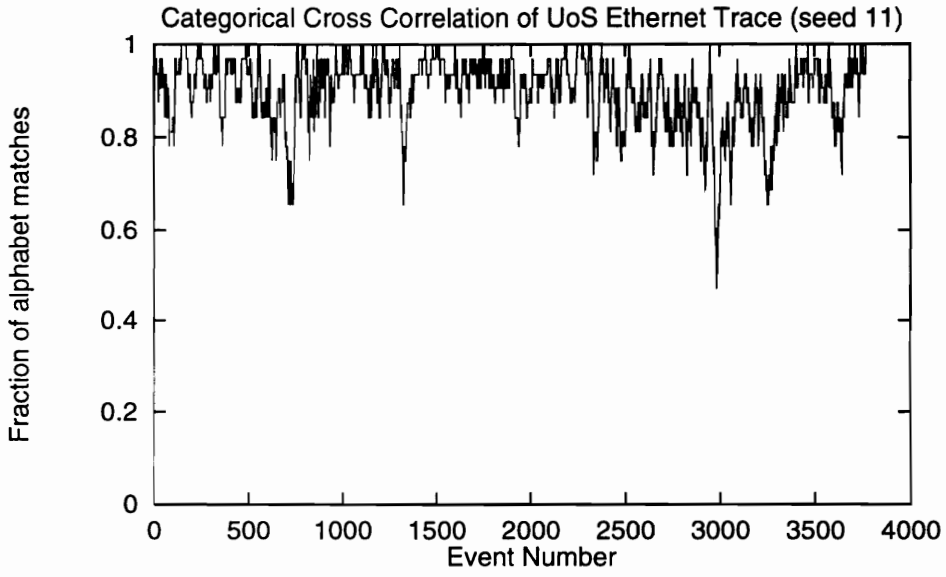


Figure 6.34: Categorical Alphabet Match Cross Correlation Plot: UoS Ethernet Original versus Synthesized Trace (seed 11)

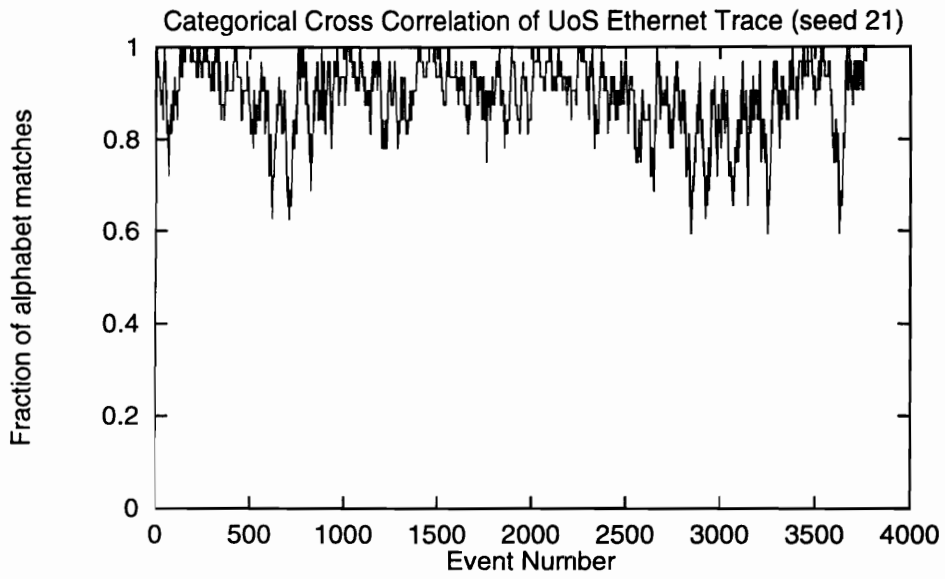


Figure 6.35: Categorical Alphabet Match Cross Correlation Plot: UoS Ethernet Original versus Synthesized Trace (seed 21)

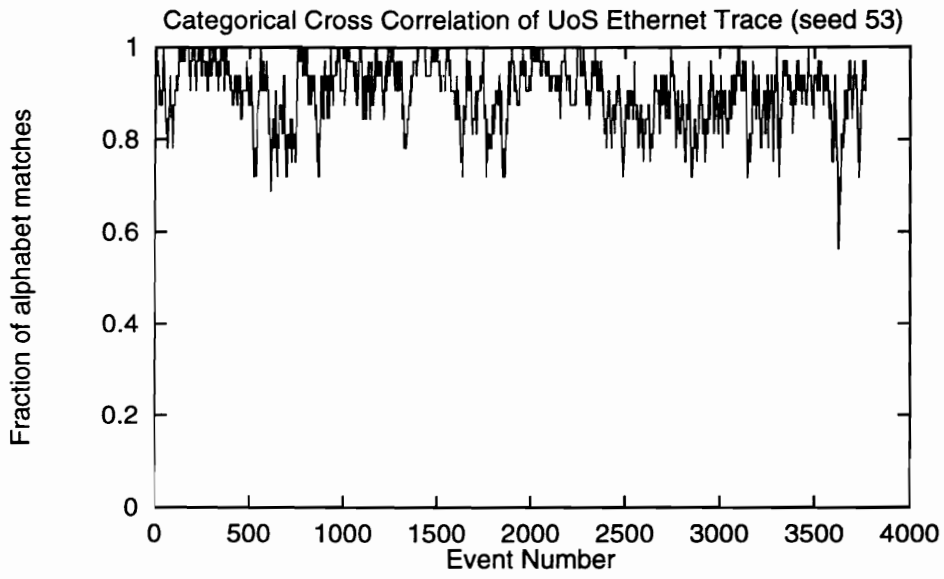


Figure 6.36: Categorical Alphabet Match Cross Correlation Plot: UoS Ethernet Original versus Synthesized Trace (seed 53)

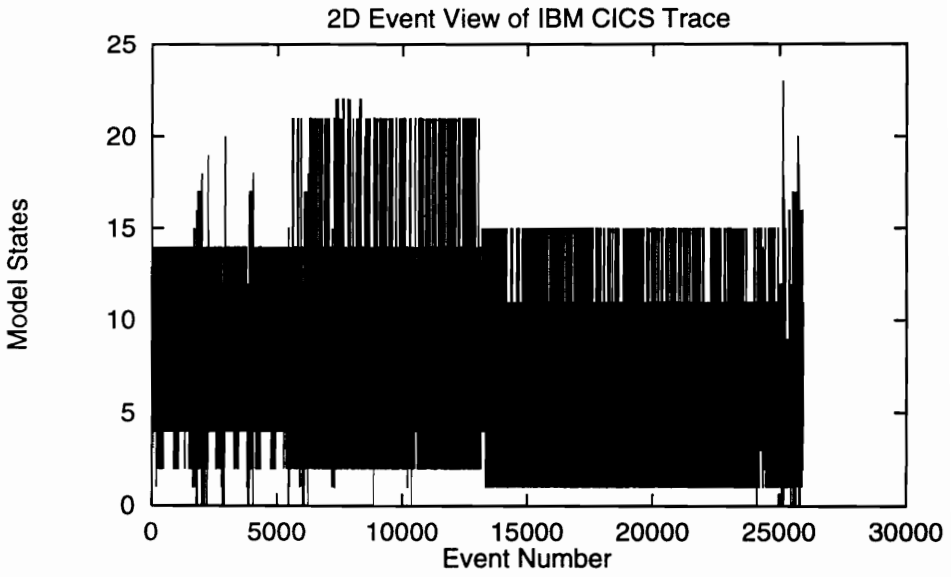


Figure 6.37: 2D View of the IBM CICS Trace

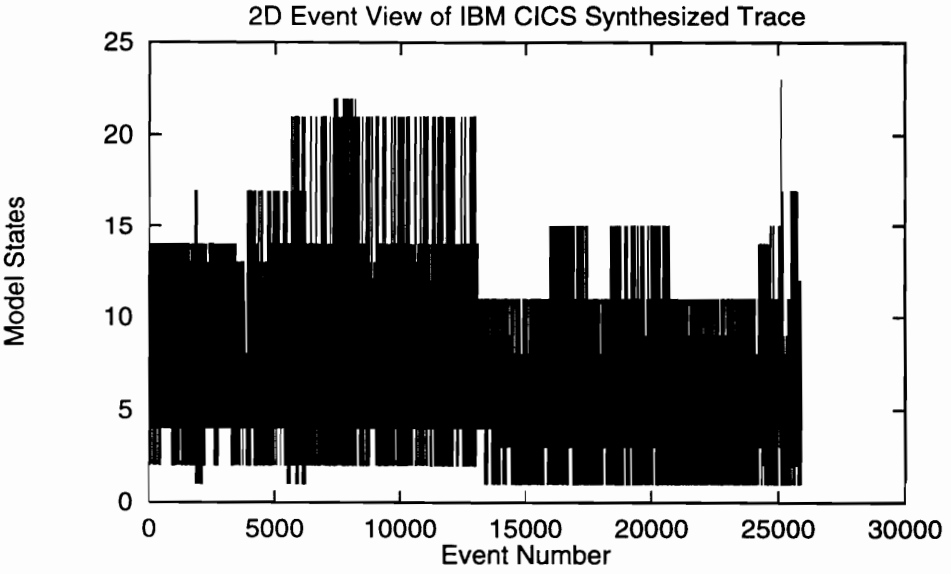


Figure 6.38: 2D View of the IBM CICS Synthesized Trace

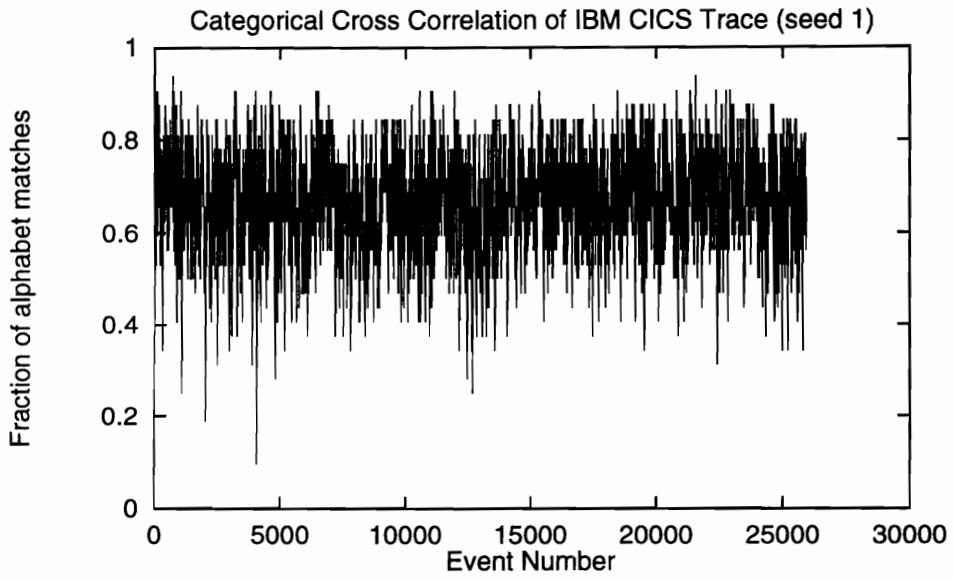


Figure 6.39: Categorical Alphabet Match Cross Correlation Plot: IBM CICS Original versus Synthesized Trace (seed 1)

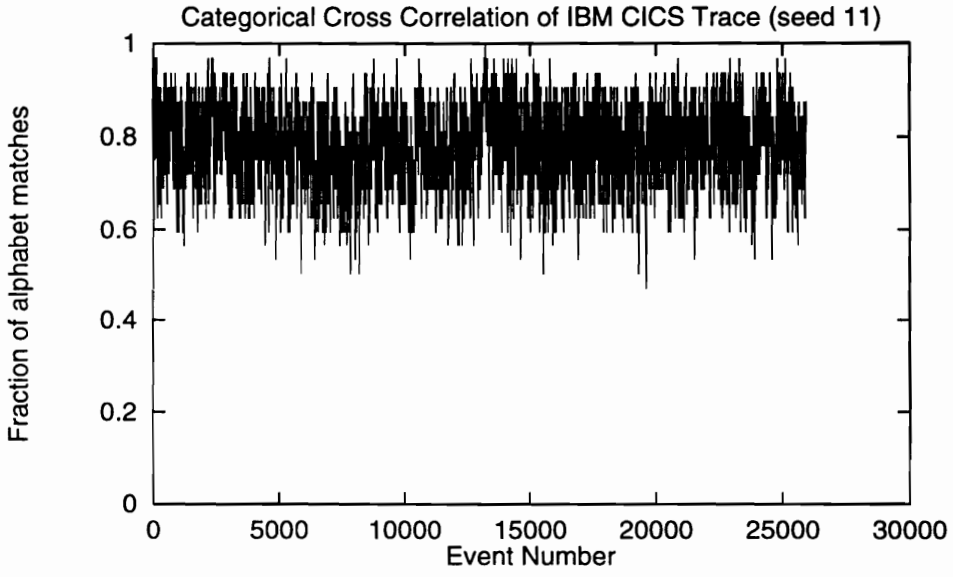


Figure 6.40: Categorical Alphabet Match Cross Correlation Plot: IBM CICS Original versus Synthesized Trace (seed 11)

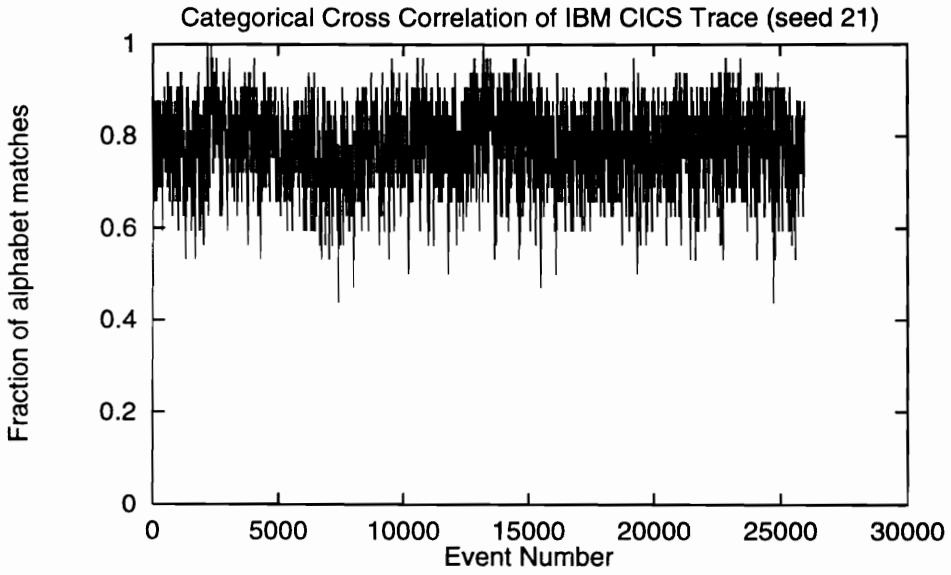


Figure 6.41: Categorical Alphabet Match Cross Correlation Plot: IBM CICS Original versus Synthesized Trace (seed 21)

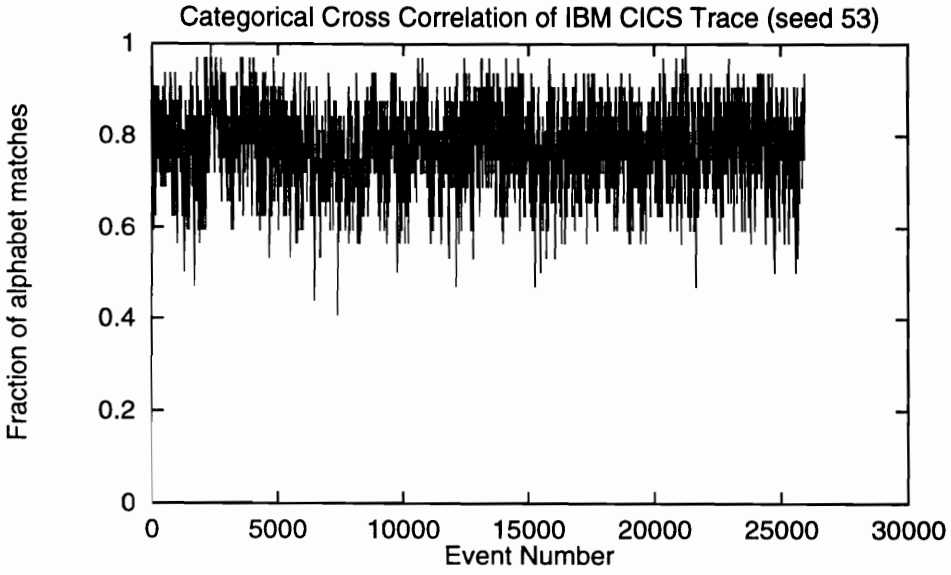


Figure 6.42: Categorical Alphabet Match Cross Correlation Plot: IBM CICS Original versus Synthesized Trace (seed 53)

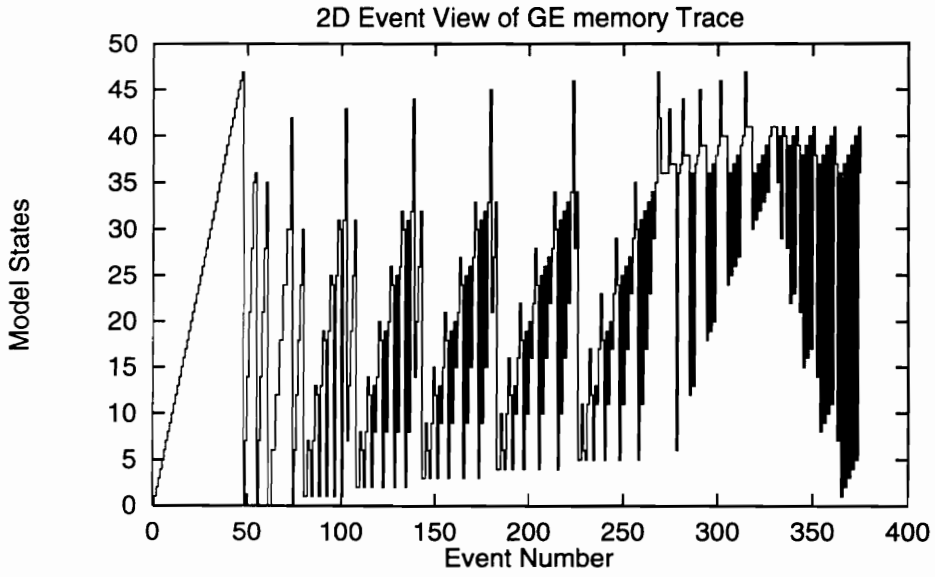


Figure 6.43: 2D View of the GE Memory Trace

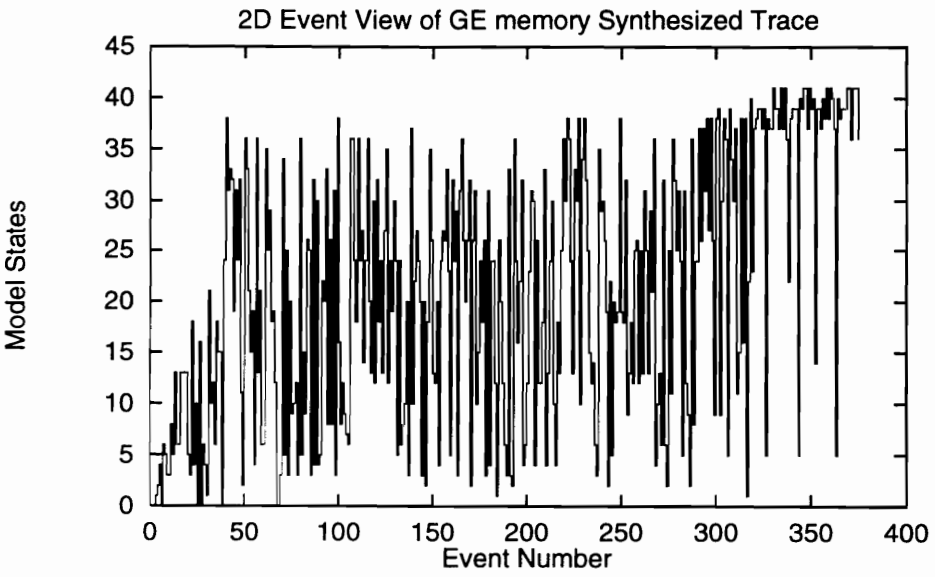


Figure 6.44: 2D View of the GE Memory Synthesized Trace

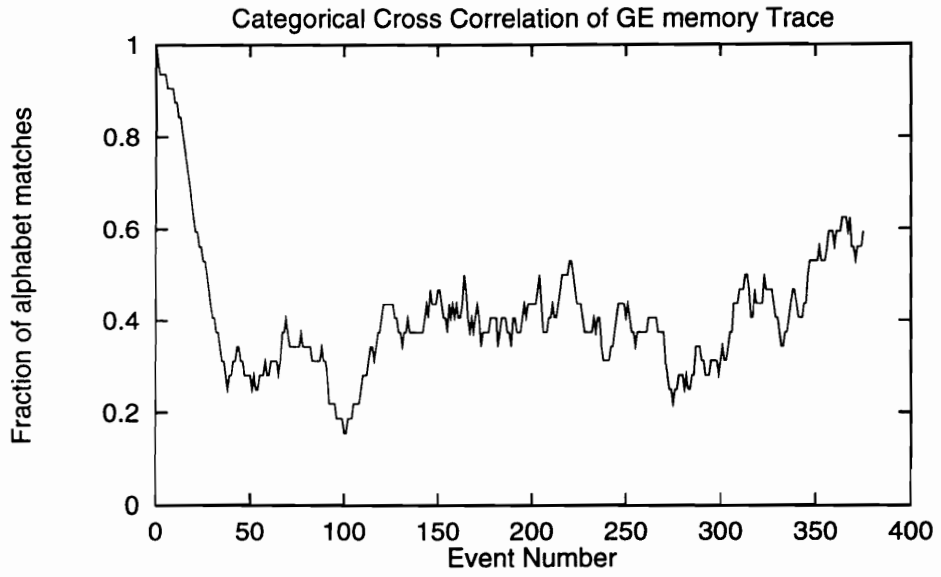


Figure 6.45: Categorical Alphabet Match Cross Correlation Plot: GE Memory Original versus Synthesized Trace (seed 1)

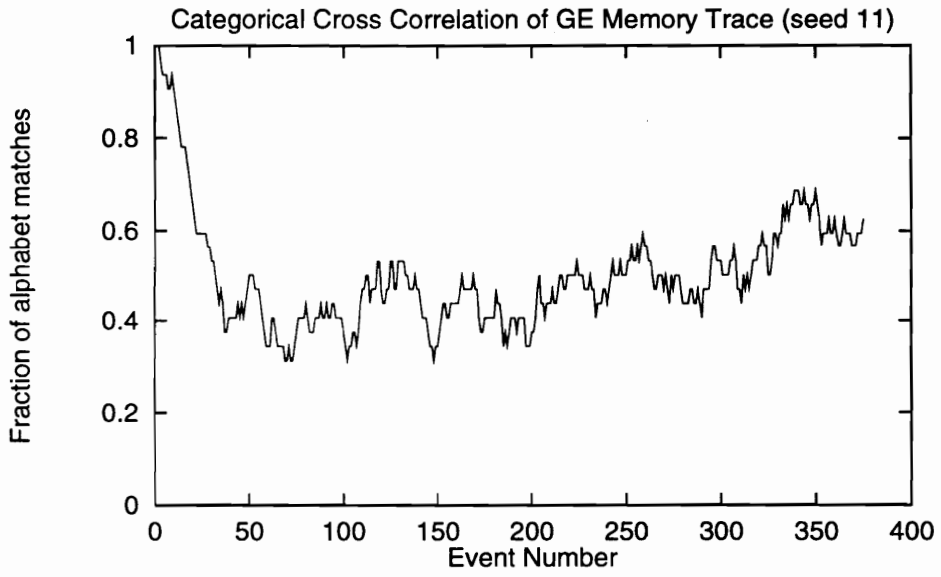


Figure 6.46: Categorical Alphabet Match Cross Correlation Plot: GE Memory Original versus Synthesized Trace (seed 11)

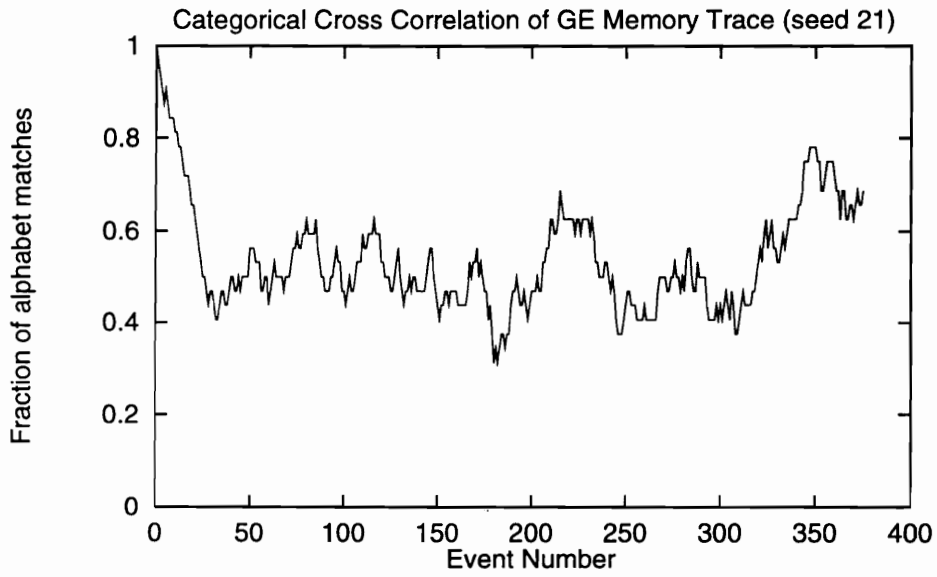


Figure 6.47: Categorical Alphabet Match Cross Correlation Plot: GE Memory Original versus Synthesized Trace (seed 21)

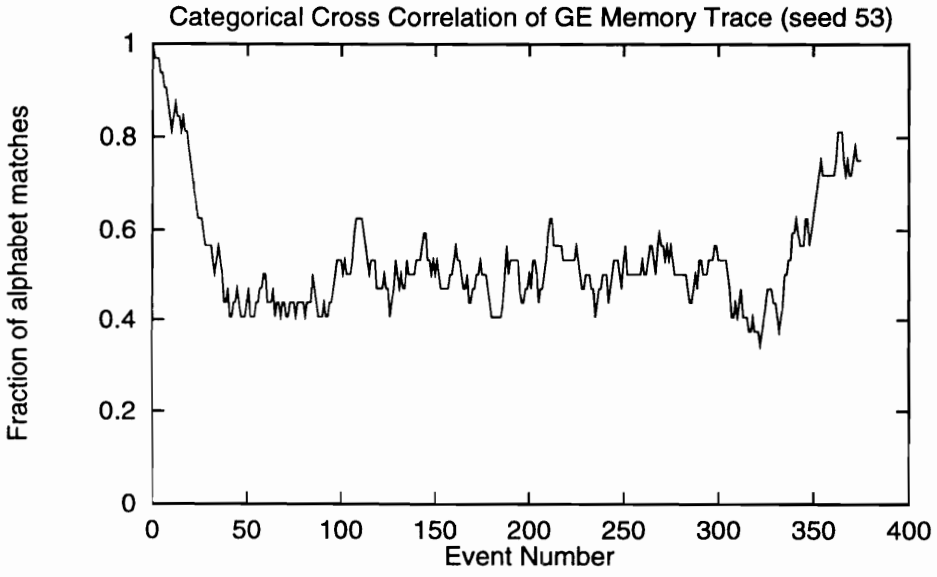


Figure 6.48: Categorical Alphabet Match Cross Correlation Plot: GE Memory Original versus Synthesized Trace (seed 53)

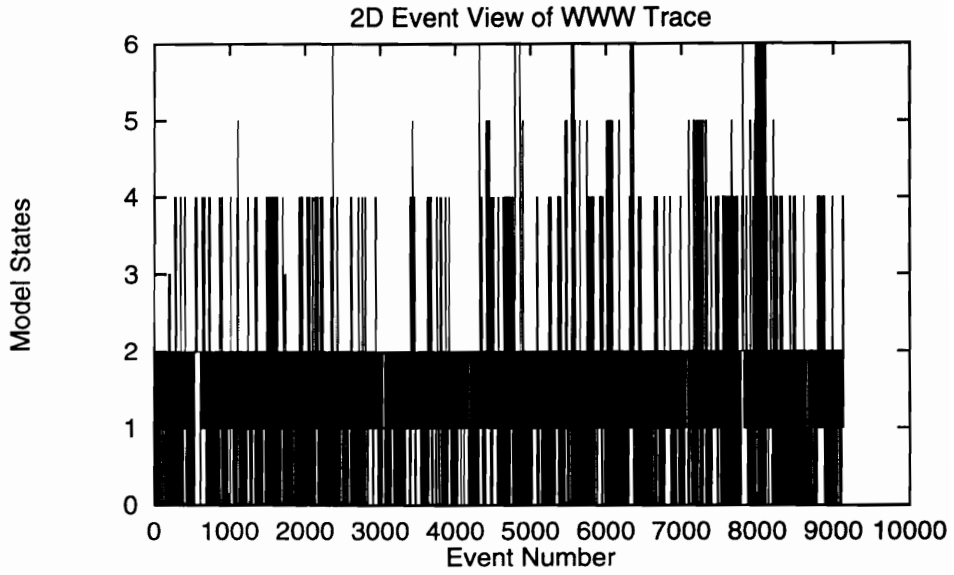


Figure 6.49: 2D View of the WWW Trace

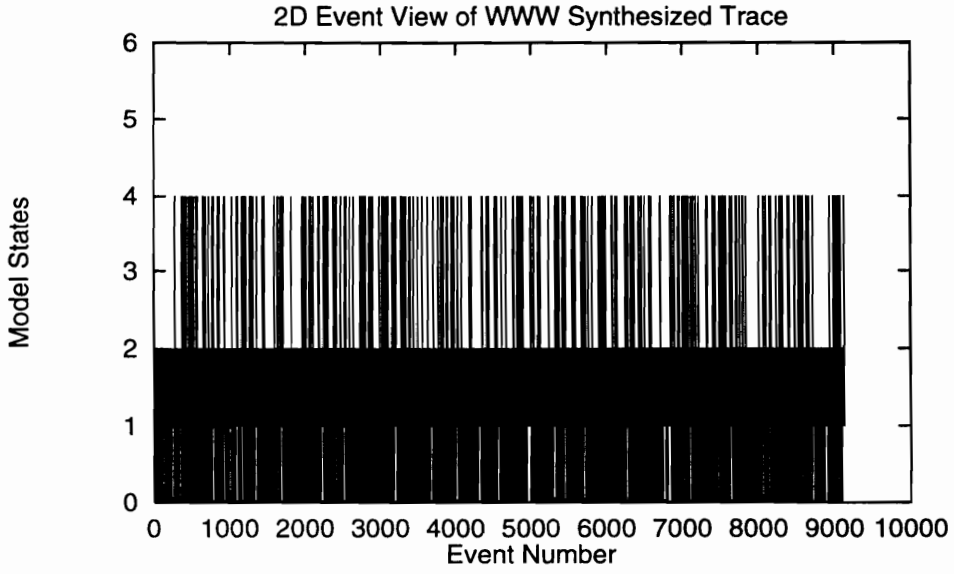


Figure 6.50: 2D View of the WWW Synthesized Trace

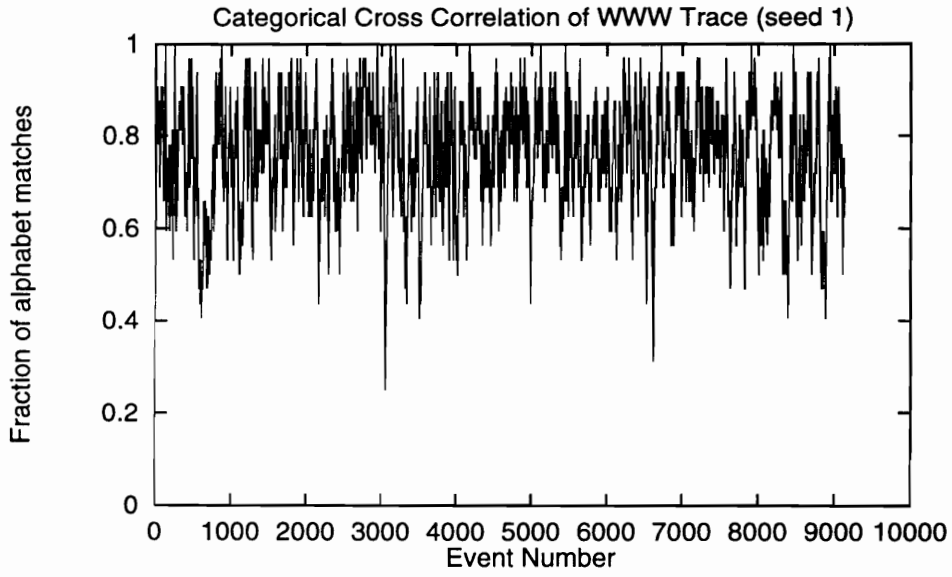


Figure 6.51: Categorical Alphabet Match Cross Correlation Plot: WWW Original versus Synthesized Trace (seed 1)

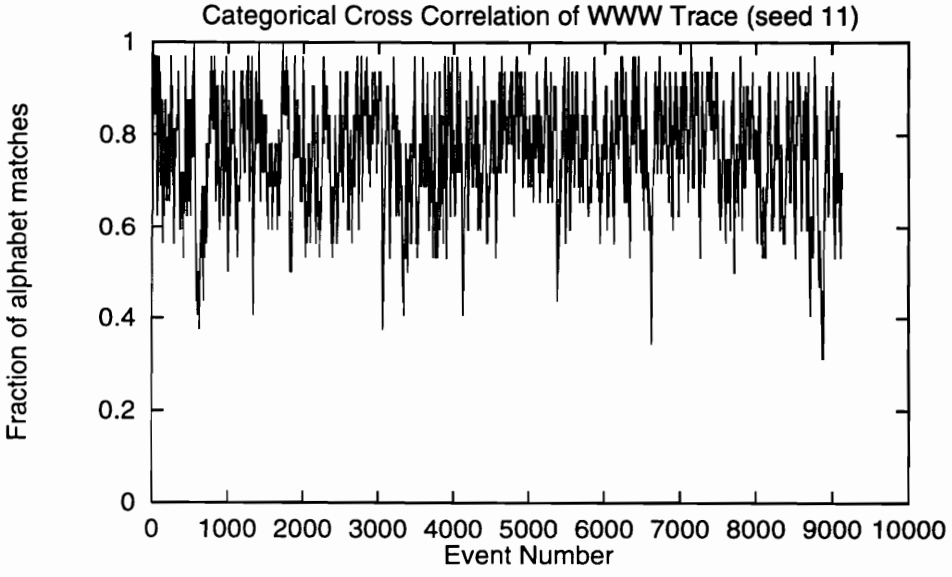


Figure 6.52: Categorical Alphabet Match Cross Correlation Plot: WWW Original versus Synthesized Trace (seed 11)

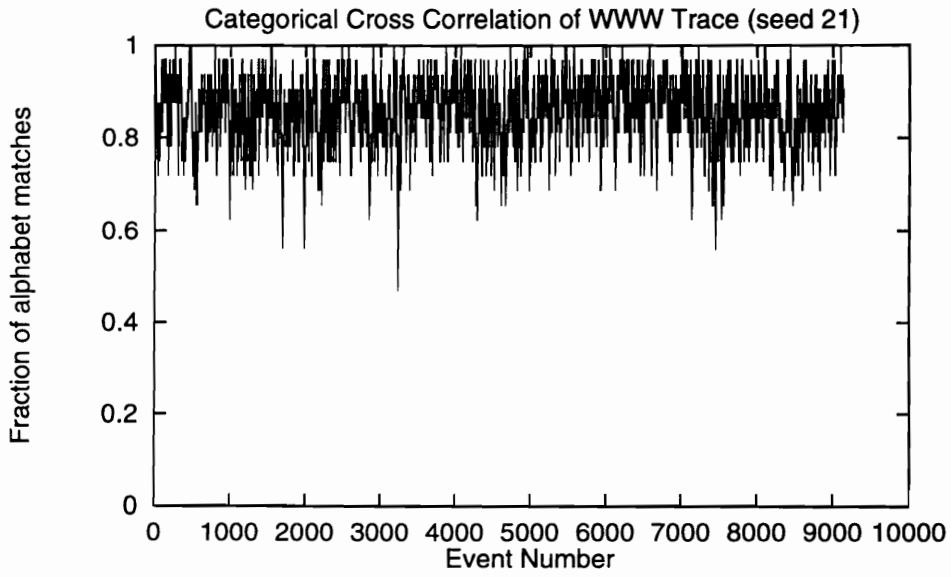


Figure 6.53: Categorical Alphabet Match Cross Correlation Plot: WWW Original versus Synthesized Trace (seed 21)

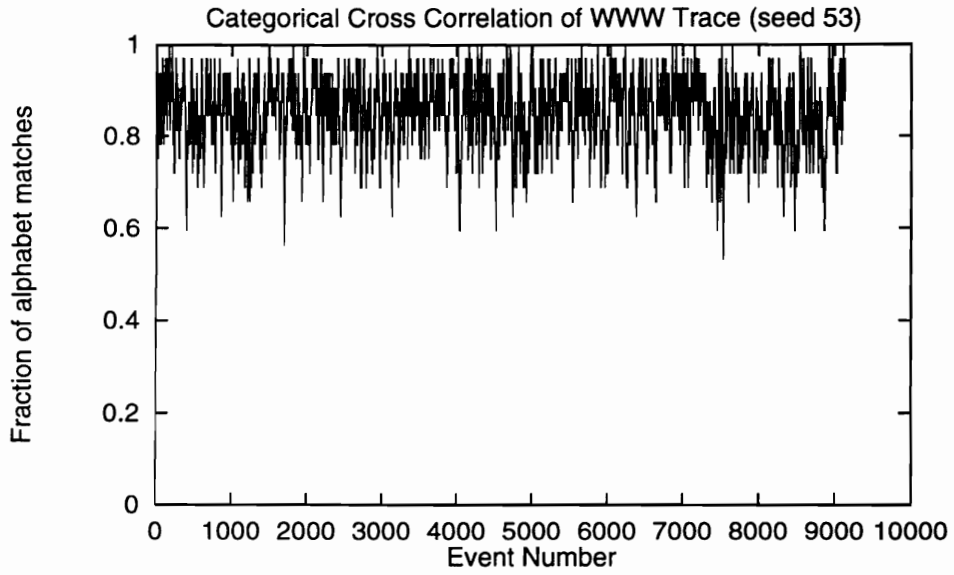


Figure 6.54: Categorical Alphabet Match Cross Correlation Plot: WWW Original versus Synthesized Trace (seed 53)

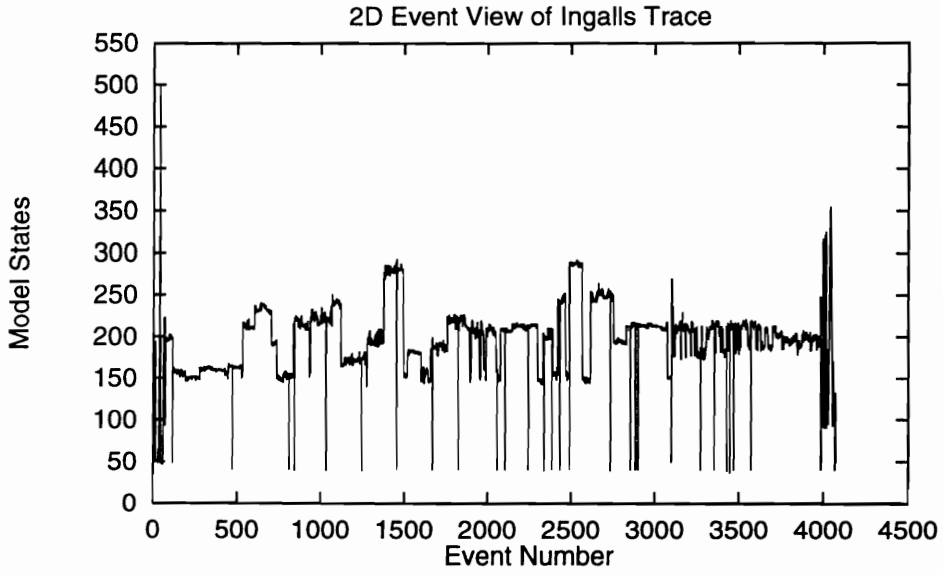


Figure 6.55: 2D View of the Ingalls Trace

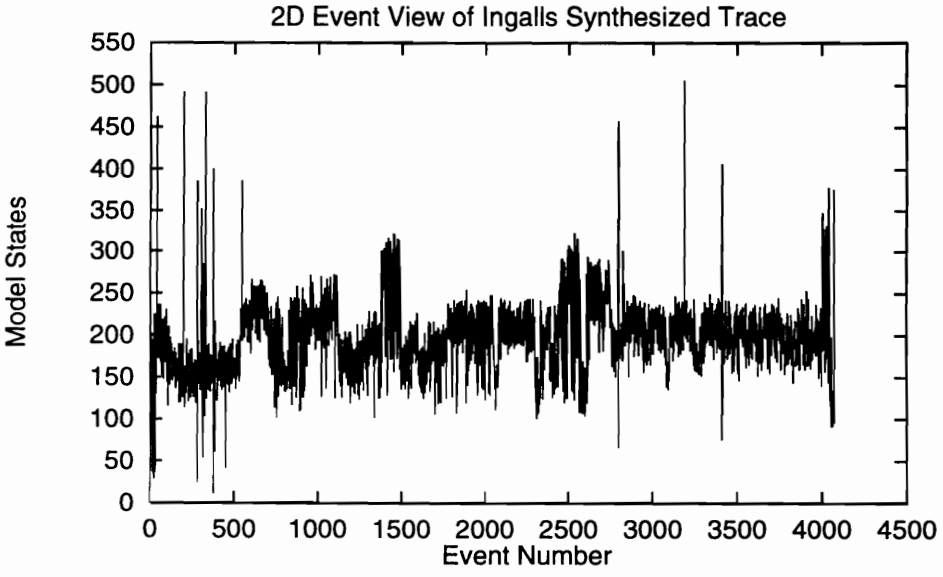


Figure 6.56: 2D View of the Ingalls Synthesized Trace

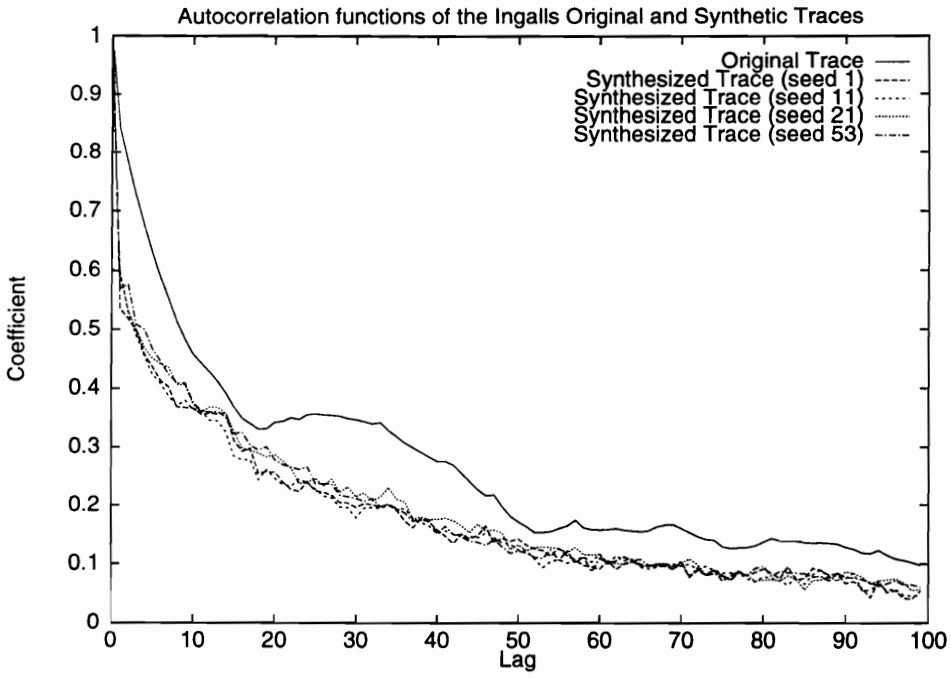


Figure 6.57: Autocorrelation Plot: Ingalls Original versus Synthesized Traces

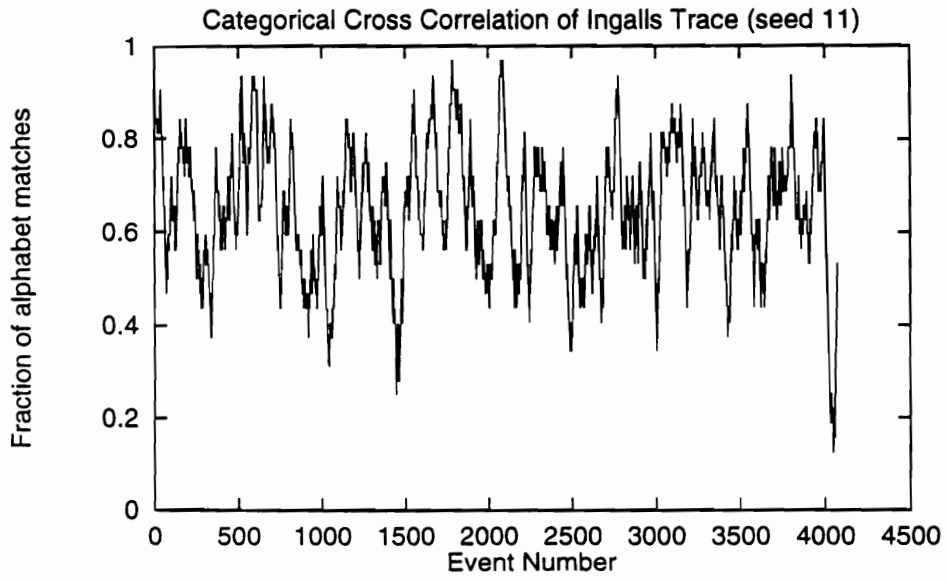


Figure 6.58: Categorical Alphabet Match Cross Correlation Plot: Ingalls Original versus Synthesized Trace (seed 11)

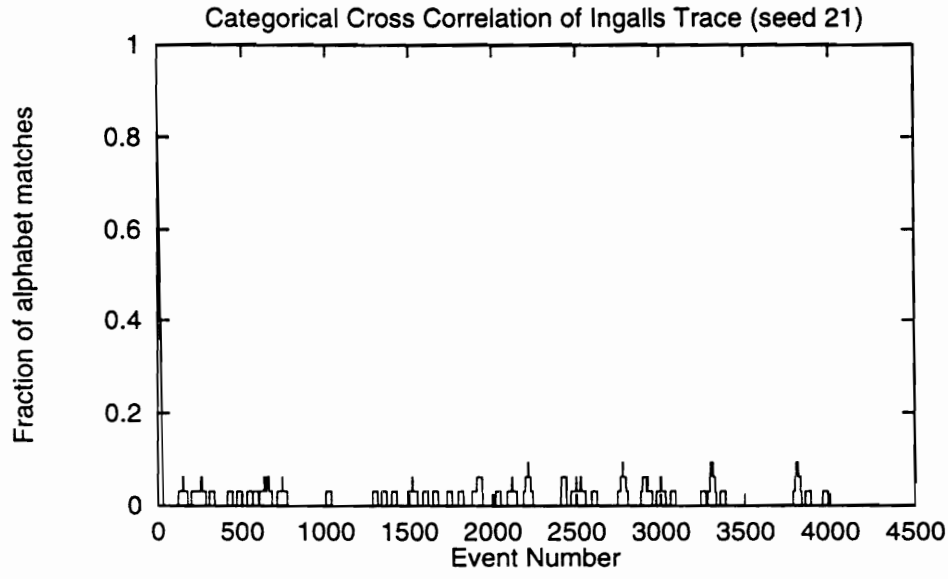


Figure 6.59: Categorical Alphabet Match Cross Correlation Plot: Ingalls Original versus Synthesized Trace (seed 21)

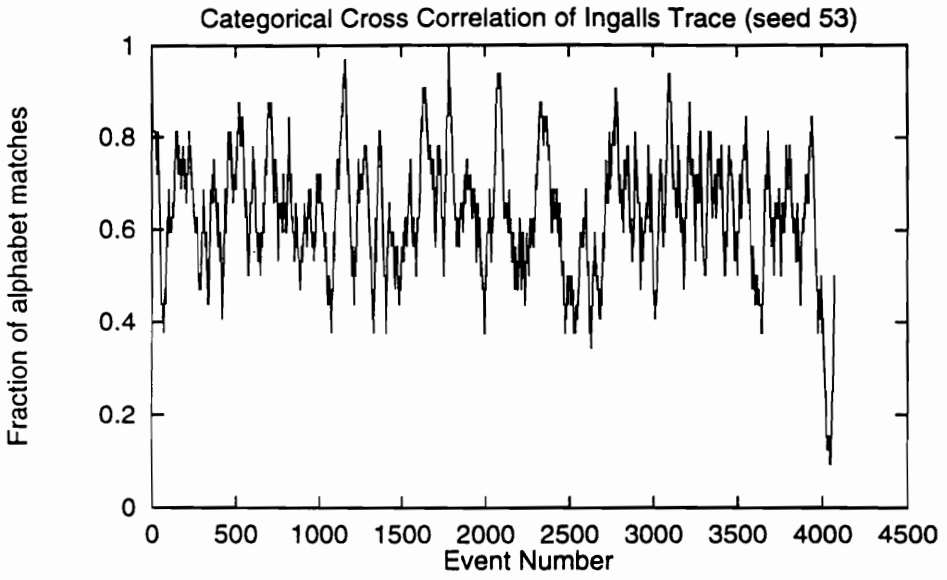


Figure 6.60: Categorical Alphabet Match Cross Correlation Plot: Ingalls Original versus Synthesized Trace (seed 53)

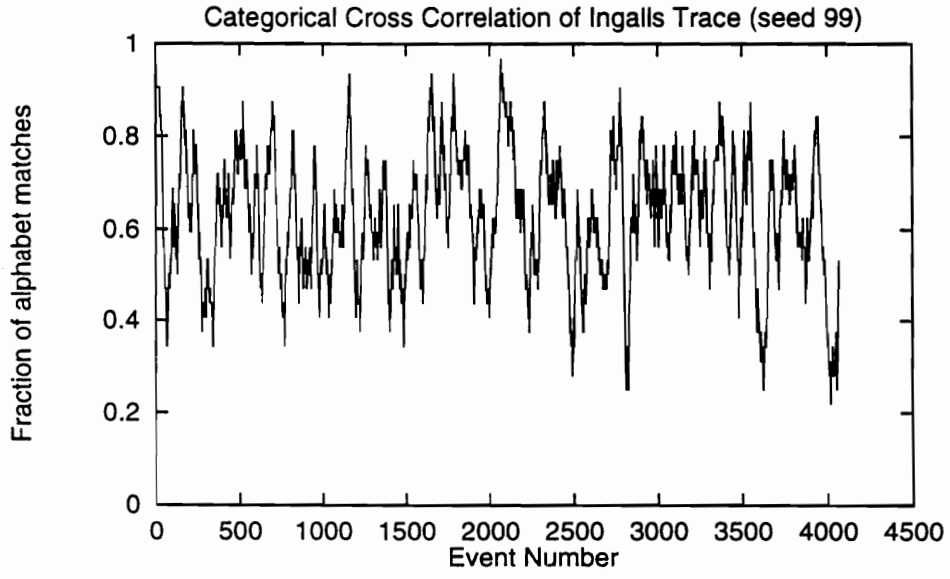


Figure 6.61: Categorical Alphabet Match Cross Correlation Plot: Ingalls Original versus Synthesized Trace (seed 99)

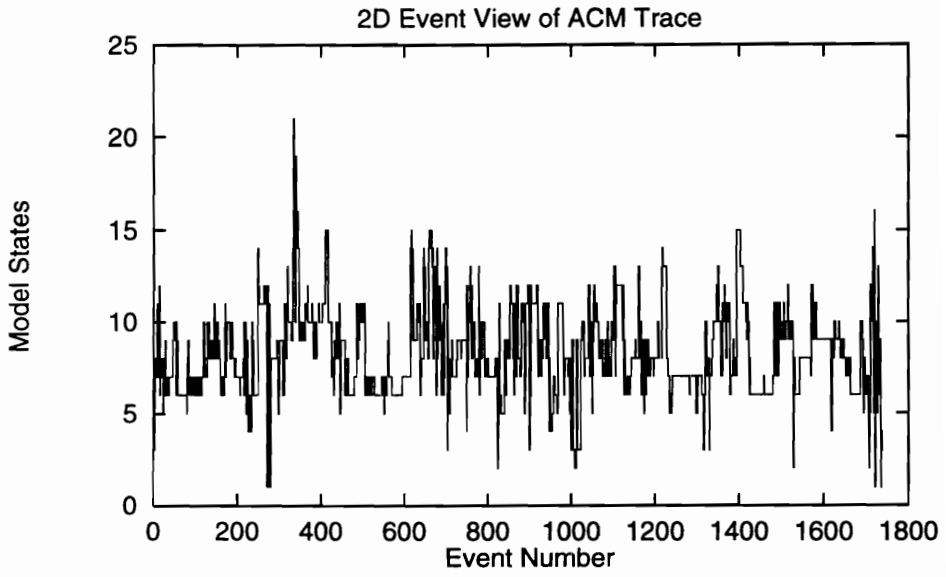


Figure 6.62: 2D View of the ACM Trace

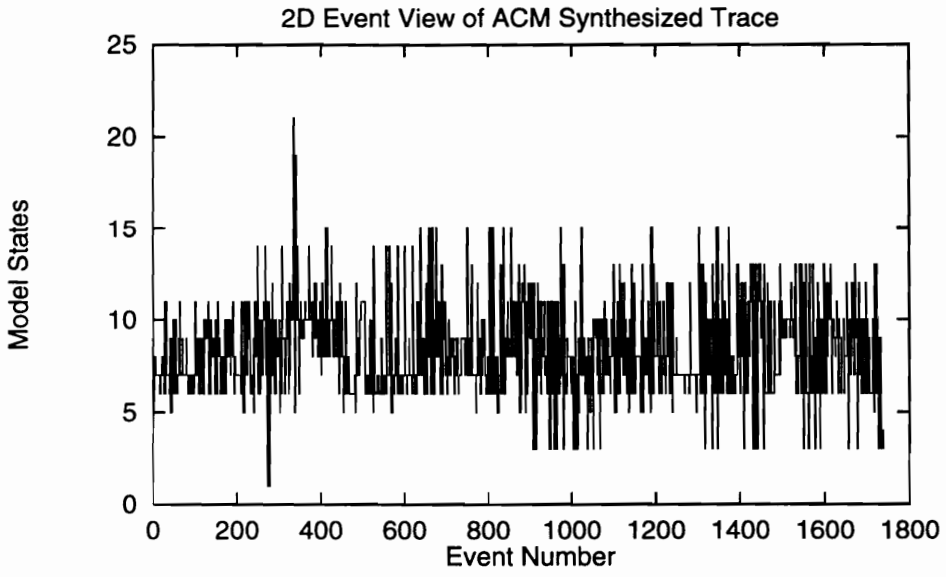


Figure 6.63: 2D View of the ACM Synthesized Trace

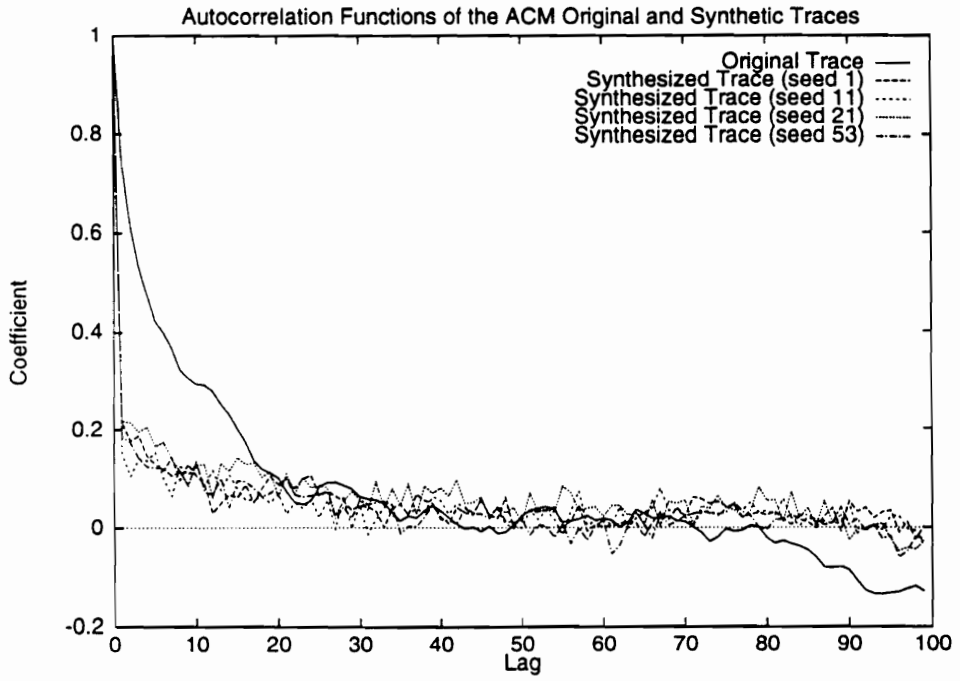


Figure 6.64: Autocorrelation Plot: ACM Original versus Synthesized Traces

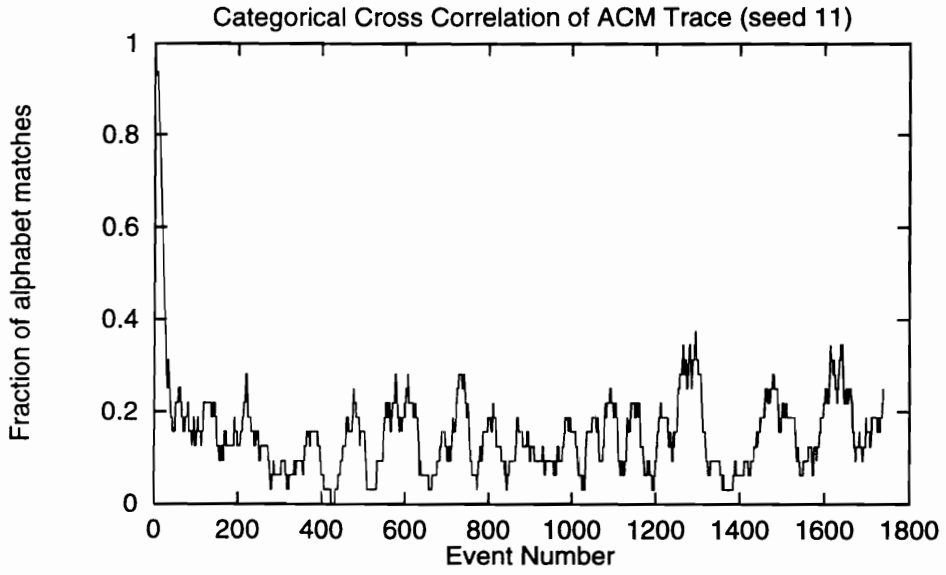


Figure 6.65: Categorical Alphabet Match Cross Correlation Plot: ACM Original versus Synthesized Trace (seed 11)

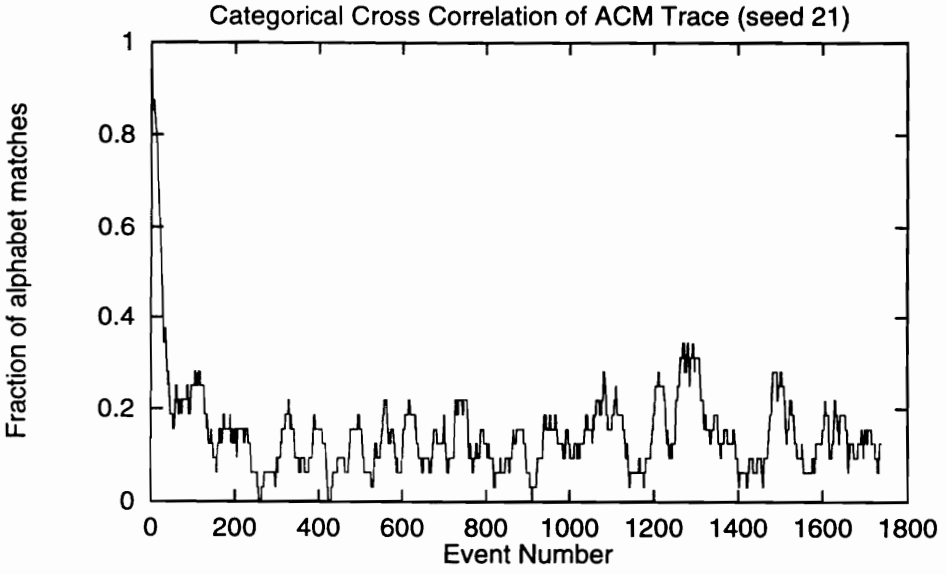


Figure 6.66: Categorical Alphabet Match Cross Correlation Plot: ACM Original versus Synthesized Trace (seed 21)

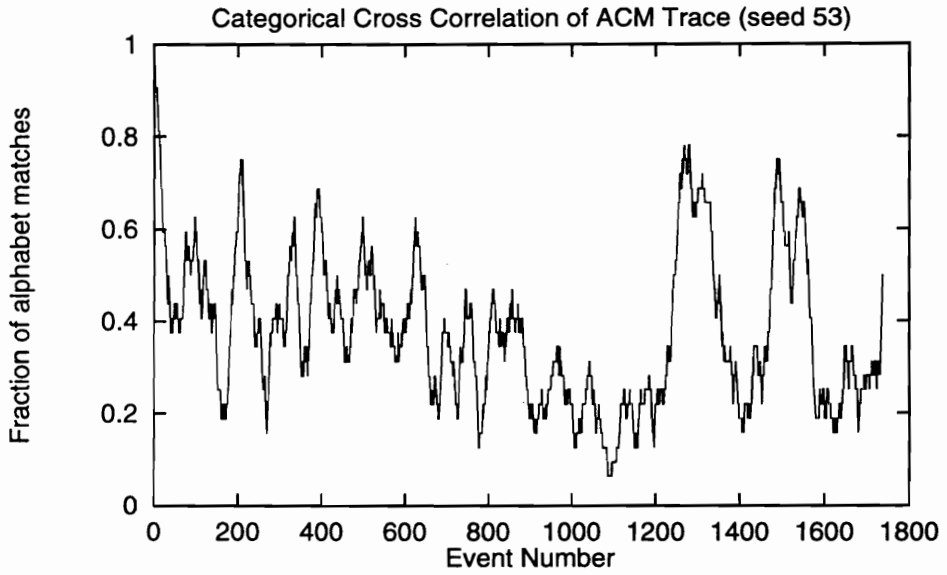


Figure 6.67: Categorical Alphabet Match Cross Correlation Plot: ACM Original versus Synthesized Trace (seed 53)

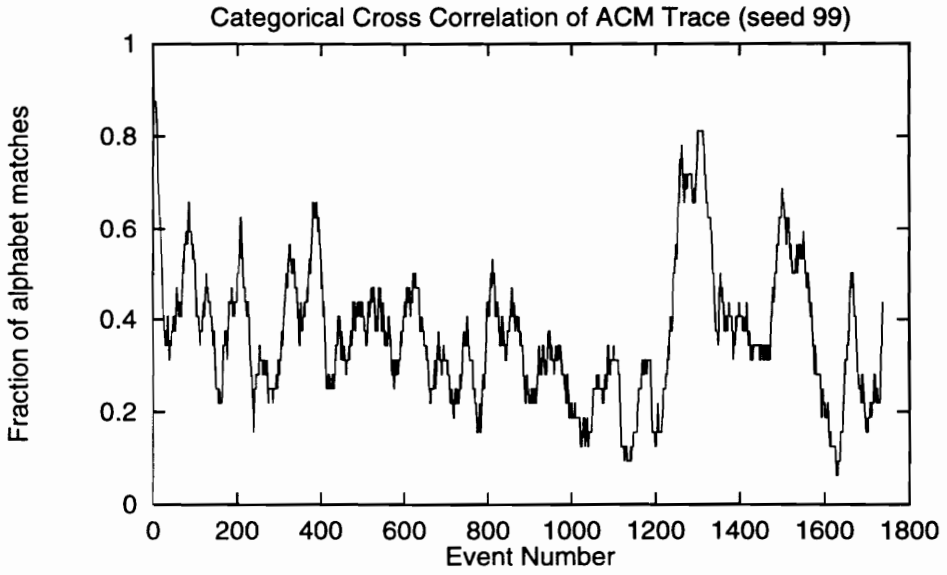


Figure 6.68: Categorical Alphabet Match Cross Correlation Plot: ACM Original versus Synthesized Trace (seed 99)

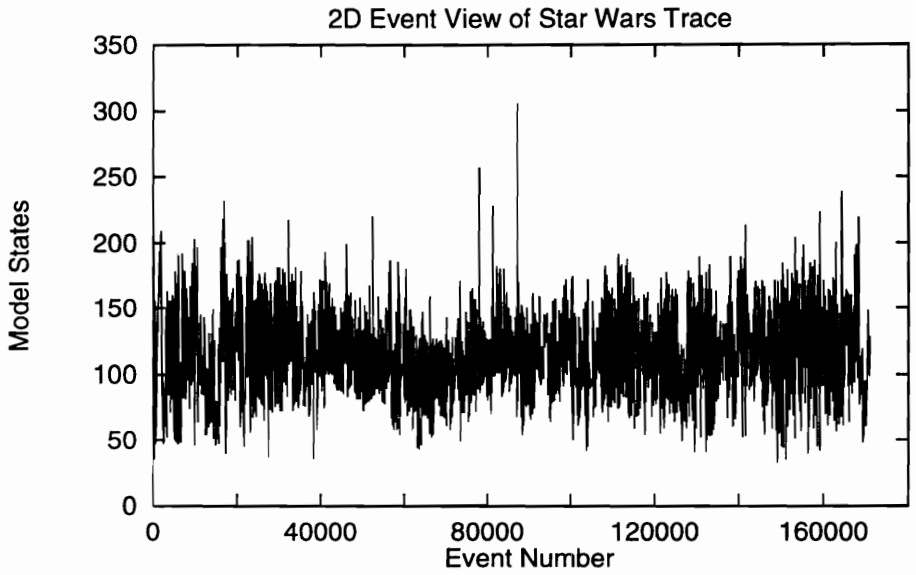


Figure 6.69: 2D View of the Star Wars Trace

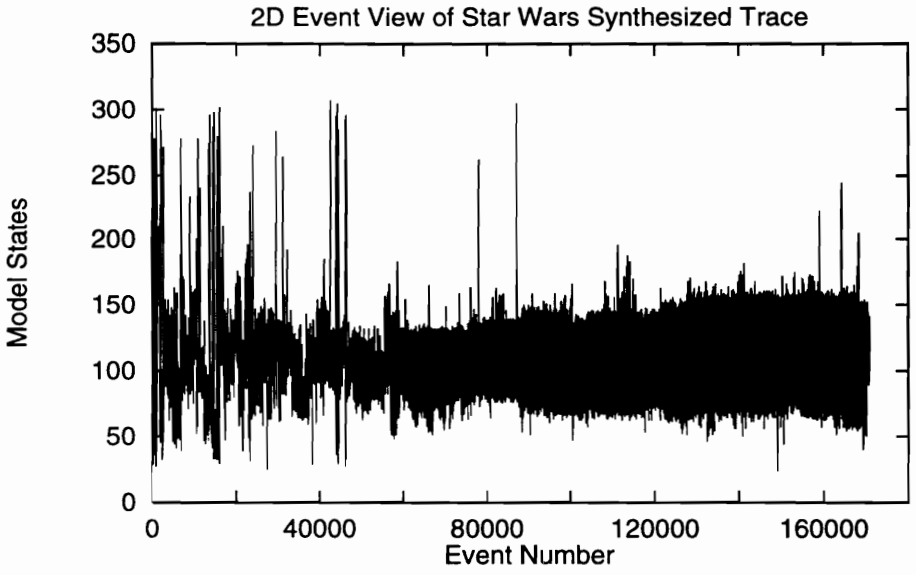


Figure 6.70: 2D View of the Star Wars Synthesized Trace

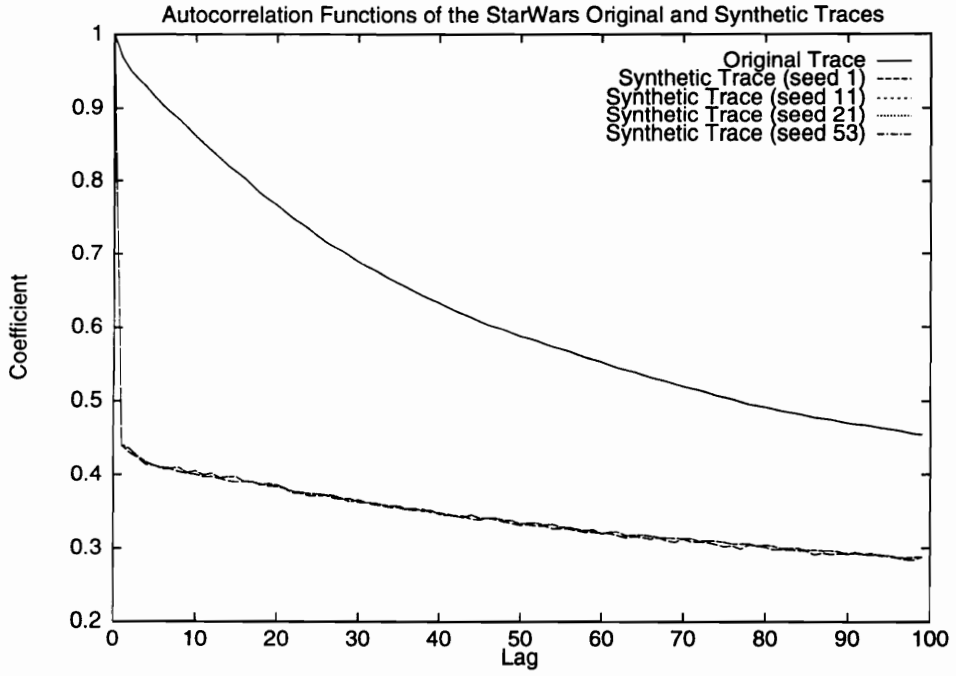


Figure 6.71: Autocorrelation Plot: StarWars Original versus Synthesized Traces

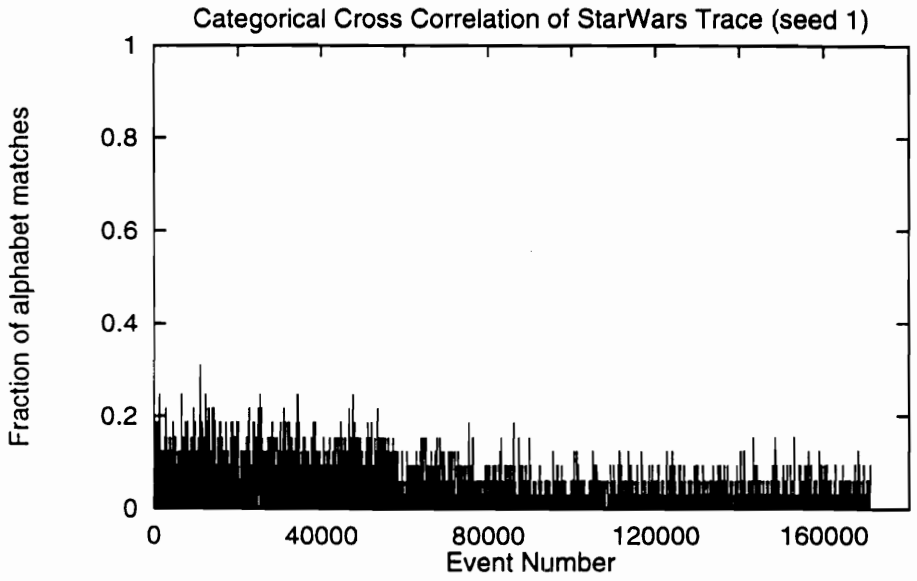


Figure 6.72: Categorical Alphabet Match Cross Correlation Plot: Star Wars Original versus Synthesized Trace (seed 1)

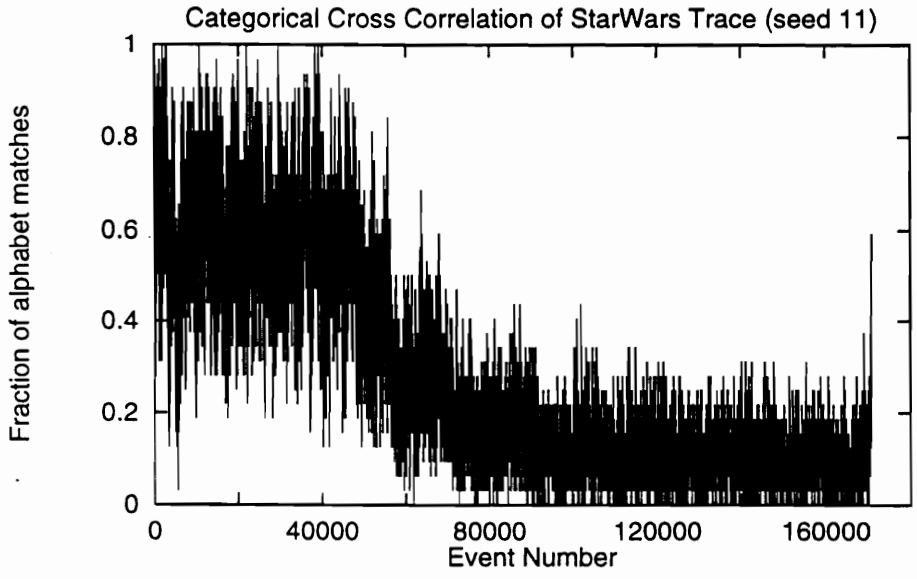


Figure 6.73: Categorical Alphabet Match Cross Correlation Plot: Star Wars Original versus Synthesized Trace (seed 11)

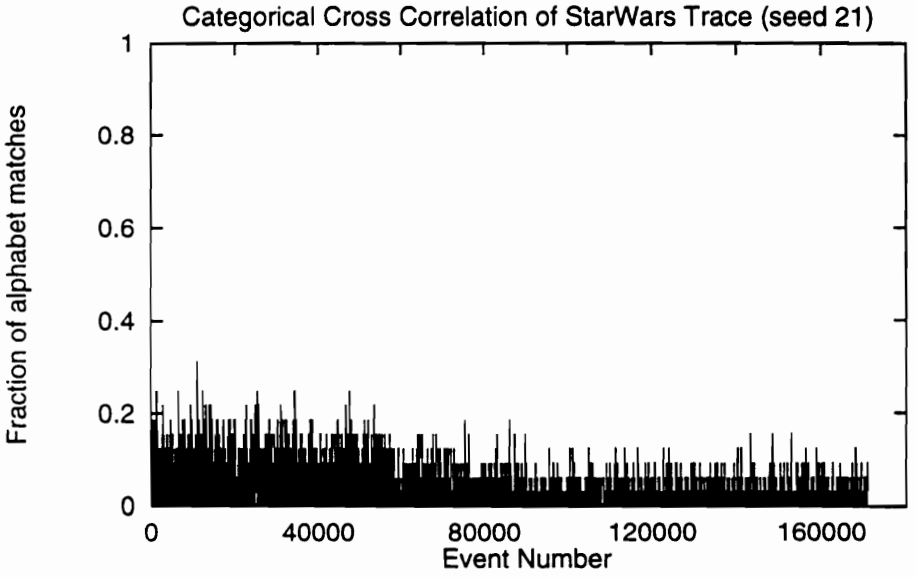


Figure 6.74: Categorical Alphabet Match Cross Correlation Plot: Star Wars Original versus Synthesized Trace (seed 21)

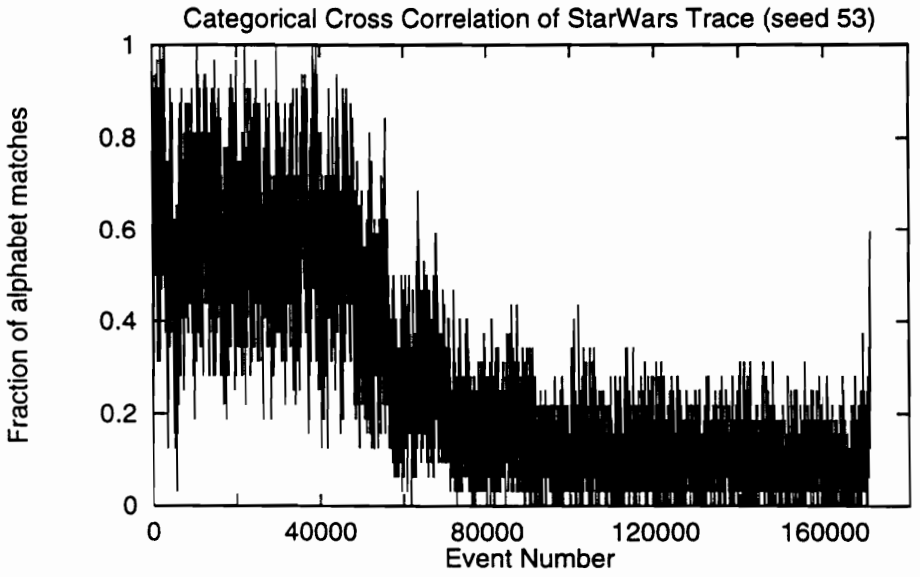


Figure 6.75: Categorical Alphabet Match Cross Correlation Plot: Star Wars Original versus Synthesized Trace (seed 53)

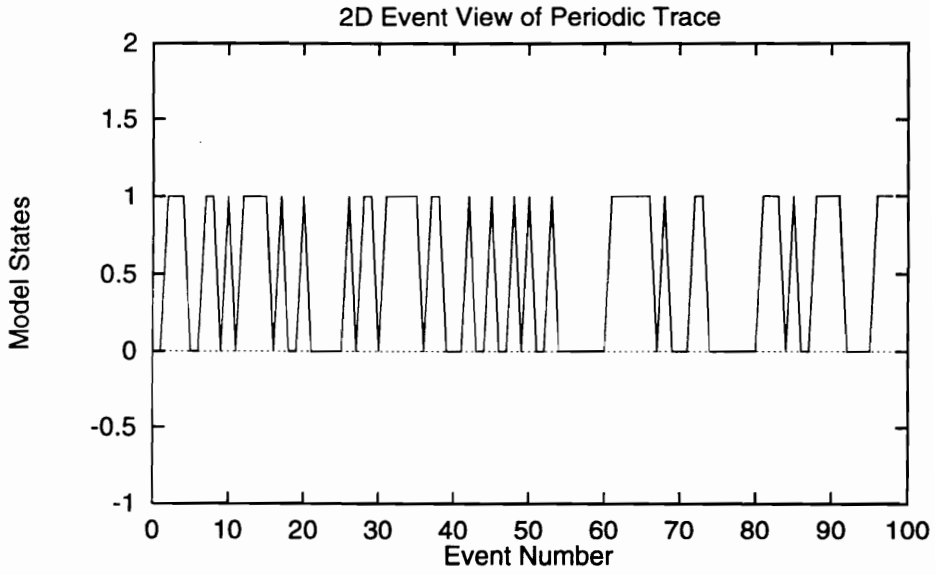


Figure 6.76: 2D View of the Periodic Trace

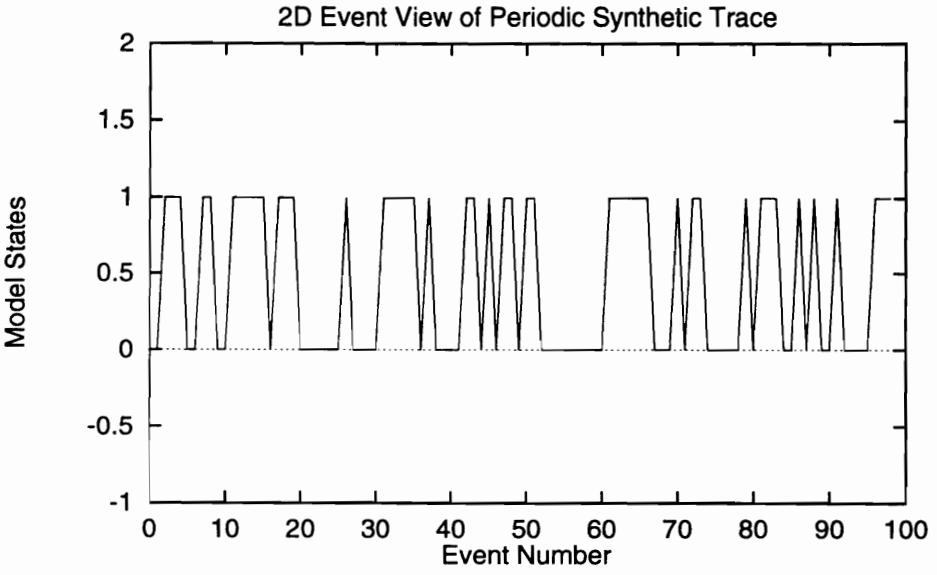


Figure 6.77: 2D View of the Periodic Synthesized Trace

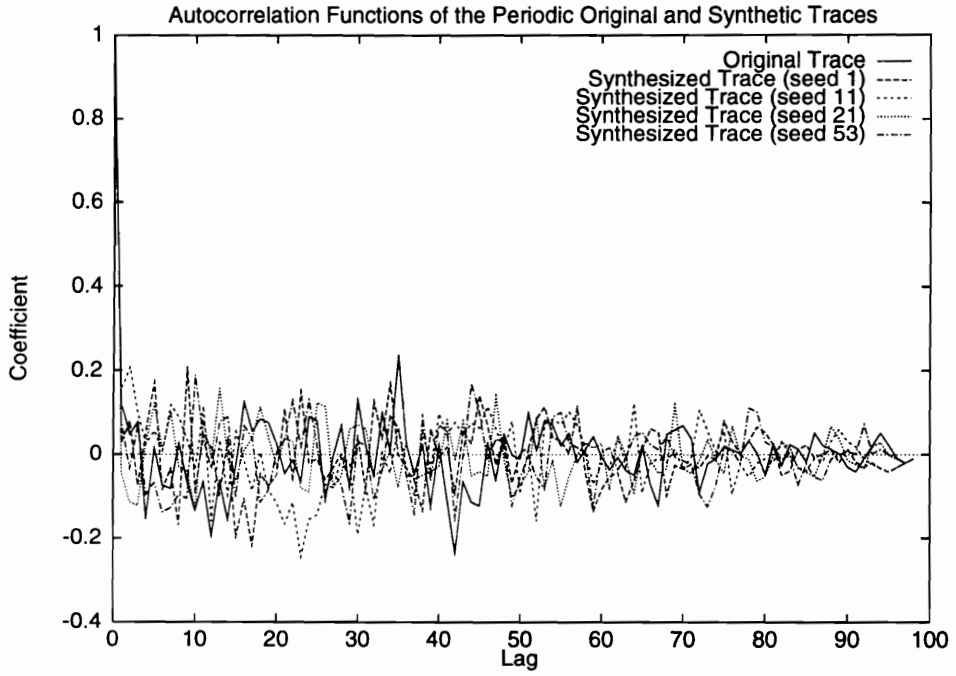


Figure 6.78: Autocorrelation Plot: Periodic Original versus Synthesized Traces

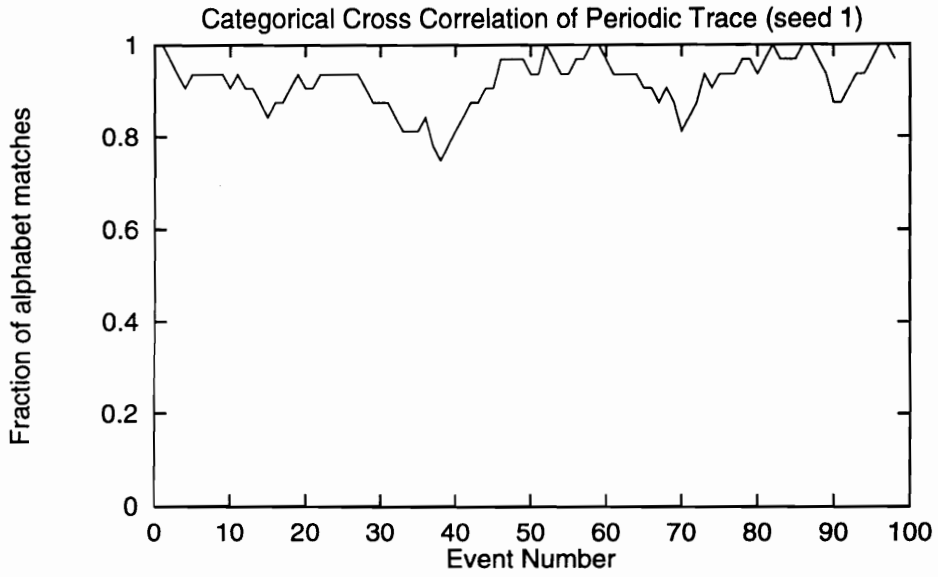


Figure 6.79: Categorical Alphabet Match Cross Correlation Plot: Periodic Original versus Synthesized Trace (seed 1)

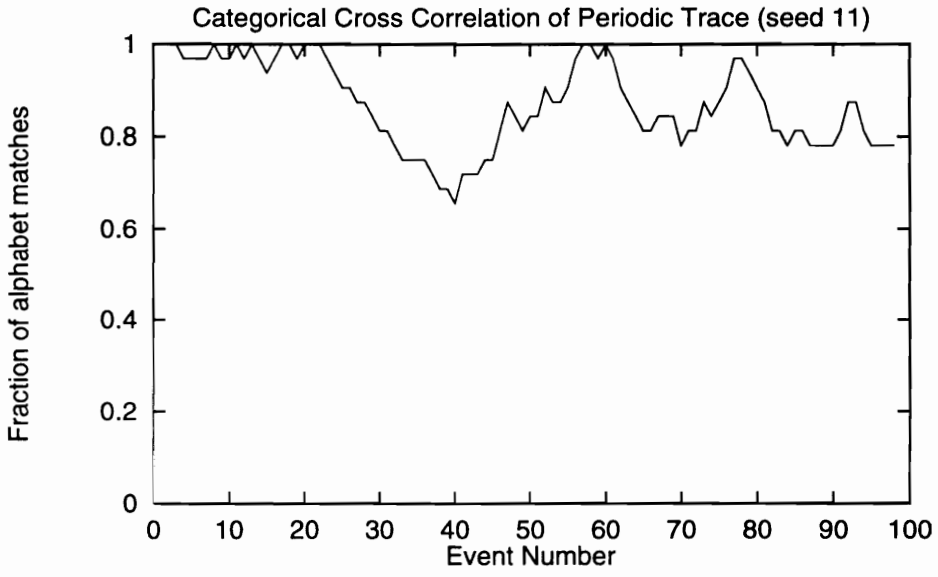


Figure 6.80: Categorical Alphabet Match Cross Correlation Plot: Periodic Original versus Synthesized Trace (seed 11)

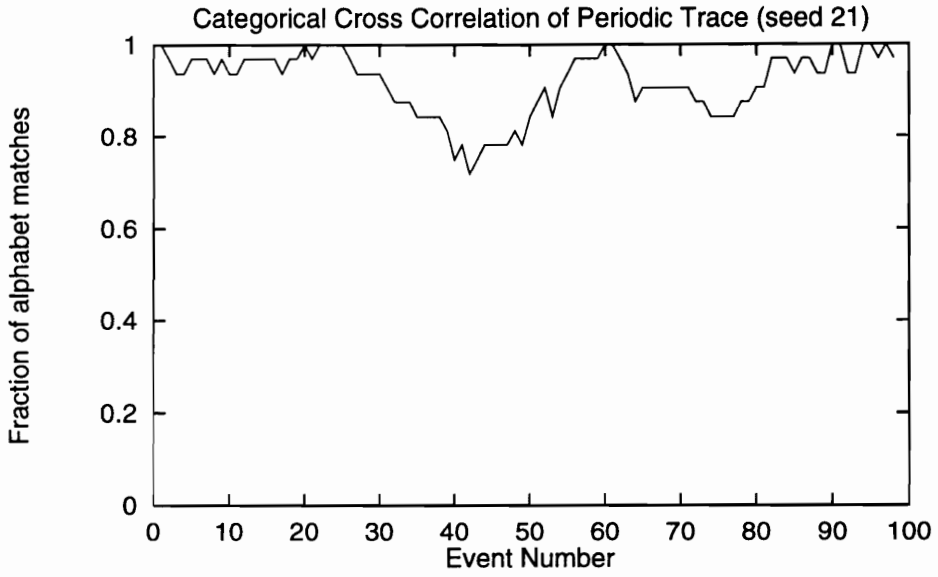


Figure 6.81: Categorical Alphabet Match Cross Correlation Plot: Periodic Original versus Synthesized Trace (seed 21)

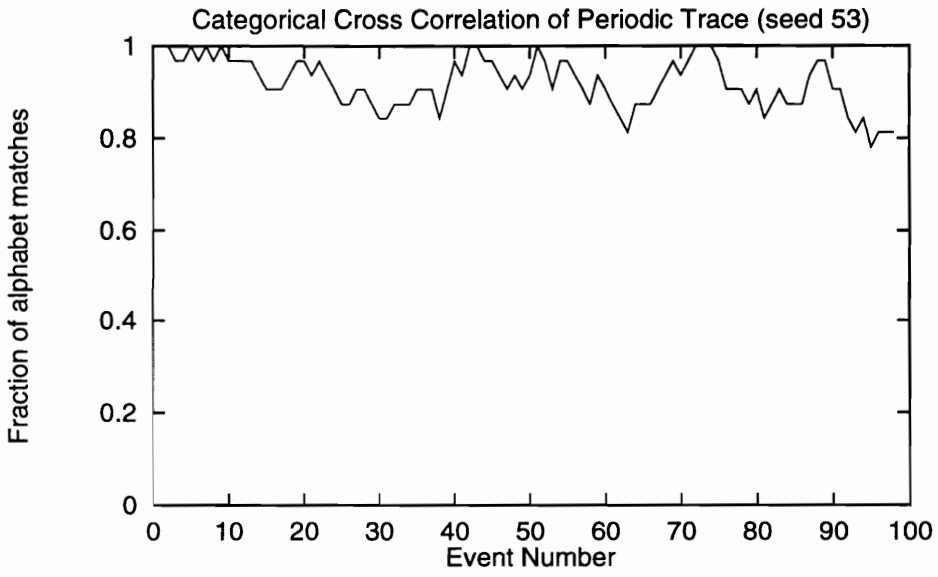


Figure 6.82: Categorical Alphabet Match Cross Correlation Plot: Periodic Original versus Synthesized Trace (seed 53)

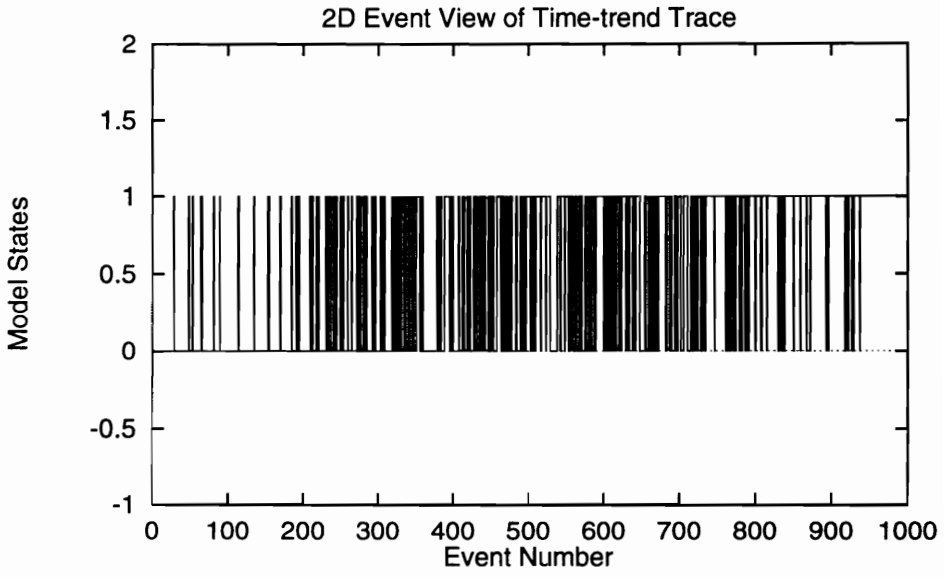


Figure 6.83: 2D View of the Time-trend Trace

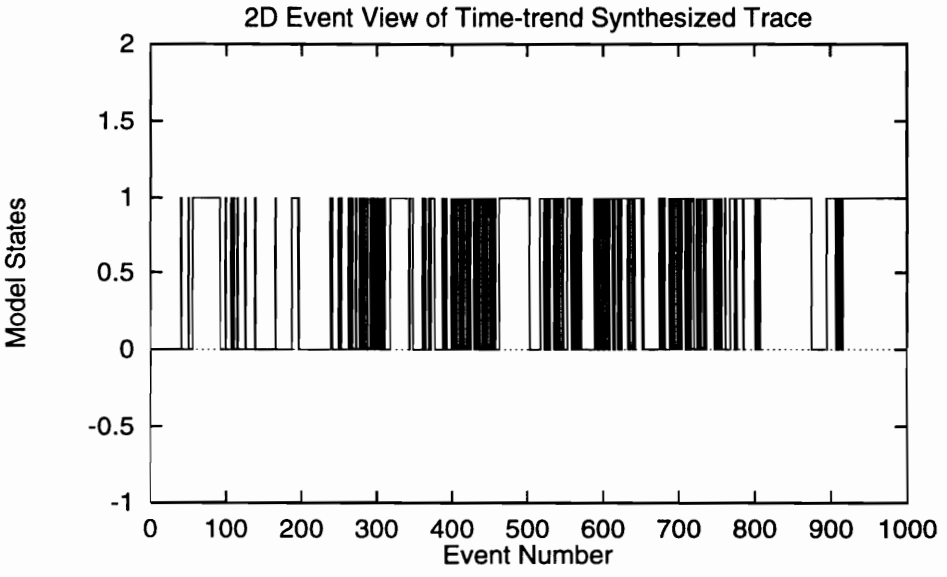


Figure 6.84: 2D View of the Time-trend Synthesized Trace

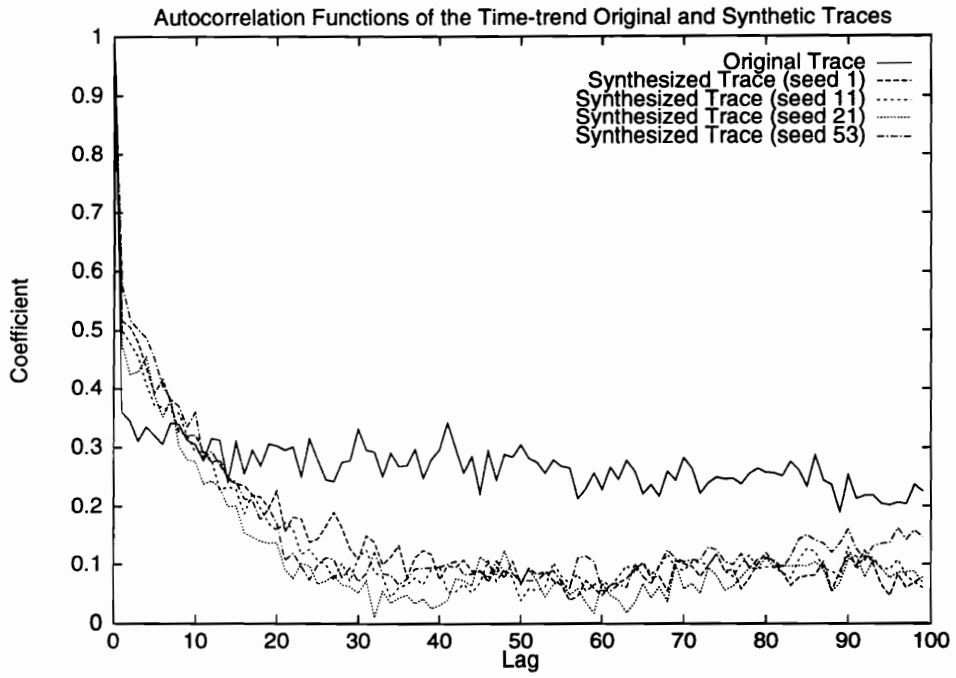


Figure 6.85: Autocorrelation Plot: Time-trend Original versus Synthesized Traces

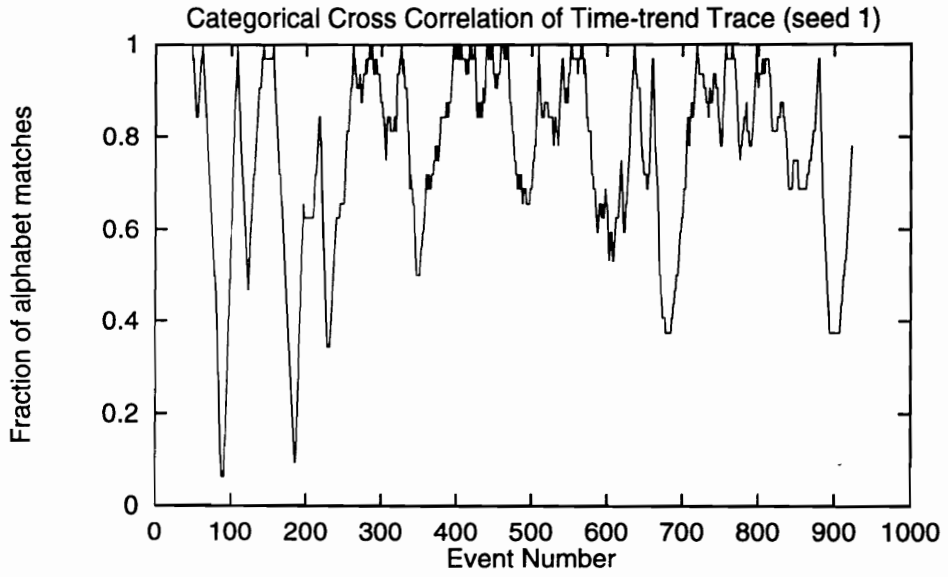


Figure 6.86: Categorical Alphabet Match Cross Correlation Plot: Time-trend Original versus Synthesized Trace (seed 1)

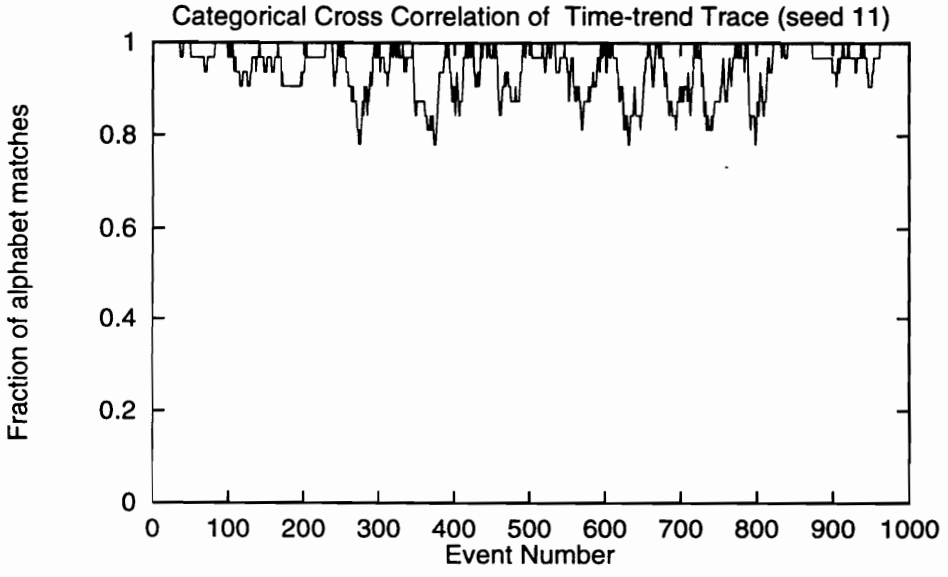


Figure 6.87: Categorical Alphabet Match Cross Correlation Plot: Time-trend Original versus Synthesized Trace (seed 11)

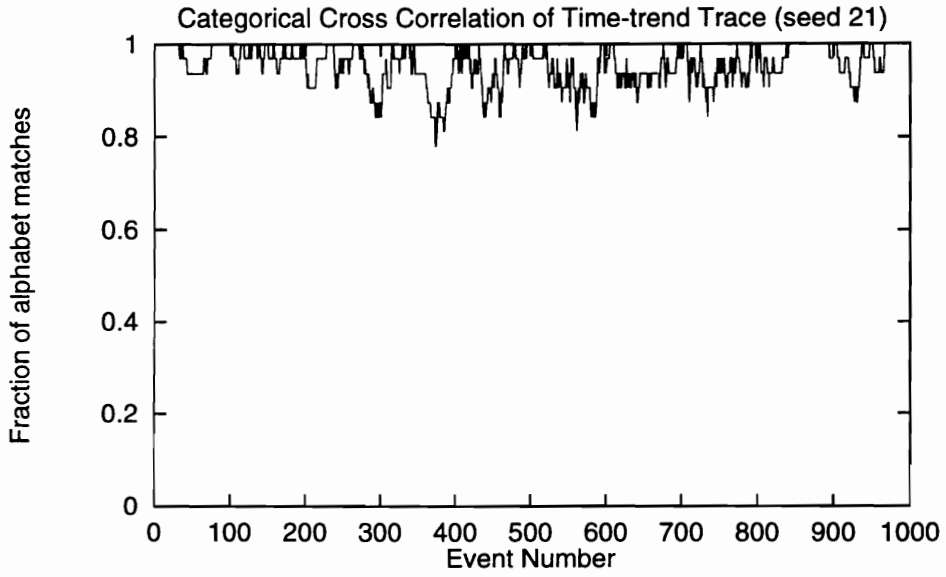


Figure 6.88: Categorical Alphabet Match Cross Correlation Plot: Time-trend Original versus Synthesized Trace (seed 21)

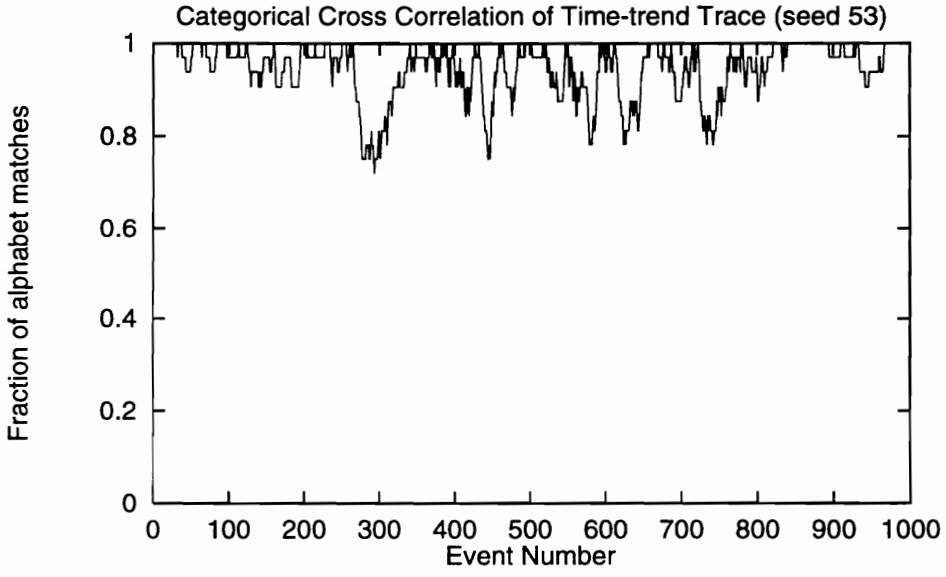


Figure 6.89: Categorical Alphabet Match Cross Correlation Plot: Time-trend Original versus Synthesized Trace (seed 53)

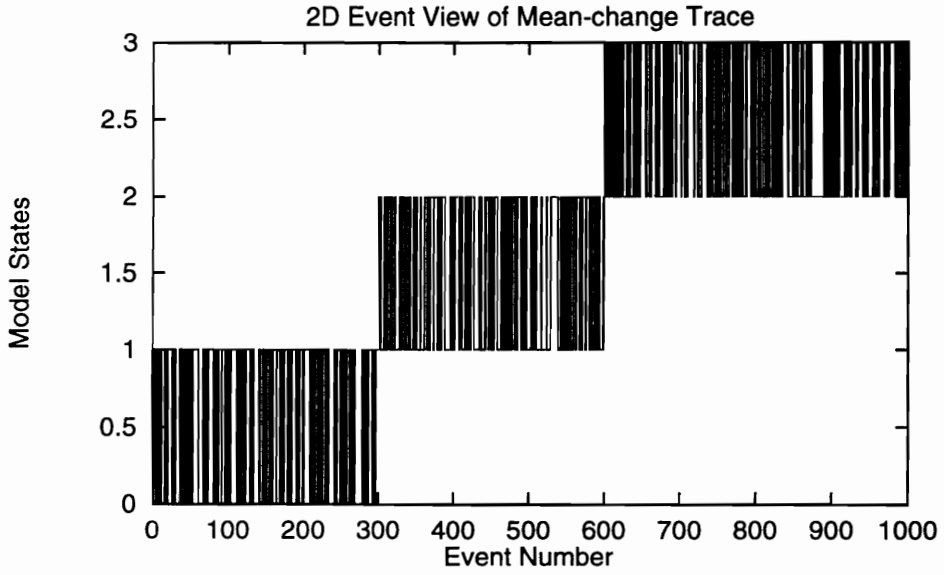


Figure 6.90: 2D View of the Mean-change Trace

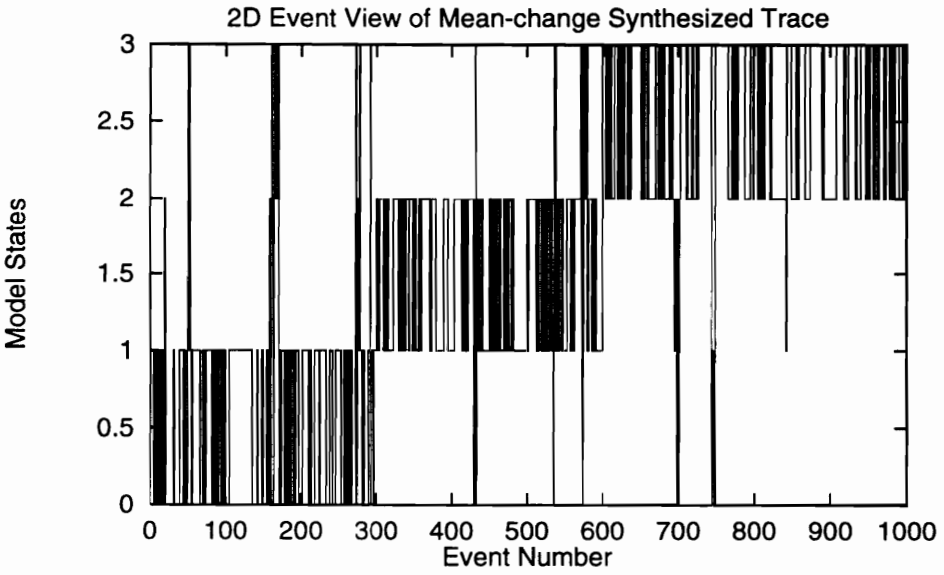


Figure 6.91: 2D View of the Mean-change Synthesized Trace

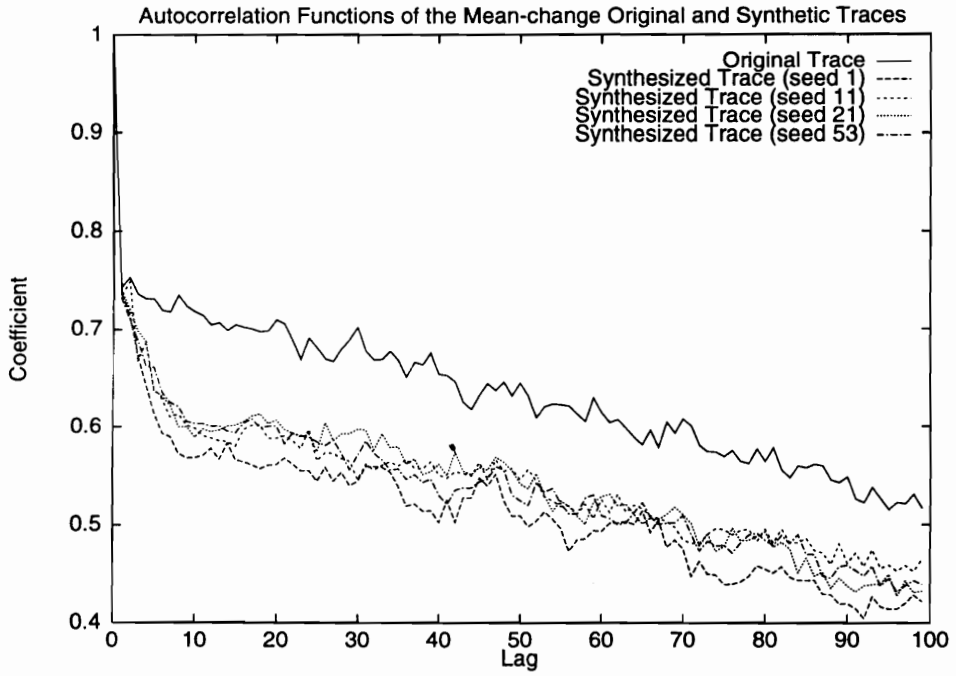


Figure 6.92: Autocorrelation Plot: Mean-change Original versus Synthesized Traces

6.4 Model Storage

We reduced the number of partitioning events (Section 5.2) in the models of all the test case traces. We also reduced the number of states in the Star Wars trace and the Ingalls trace by half as described in Section 5.1.

Table 6.3 displays the size of the model for every test case trace. In most cases the model is approximately 40% the trace size. Three notable exceptions to the above are the Periodic trace, Time-trend trace, and the GE memory trace. The models for the Periodic and Time-trend are extremely small because these traces are generated from very simple distributions (distributions with very few behavioral changes). The GE memory trace is an extremely short trace (376 *events*) with an extremely large number of states (48). The GE memory trace also has a high degree of autocorrelation. All these factors require our model

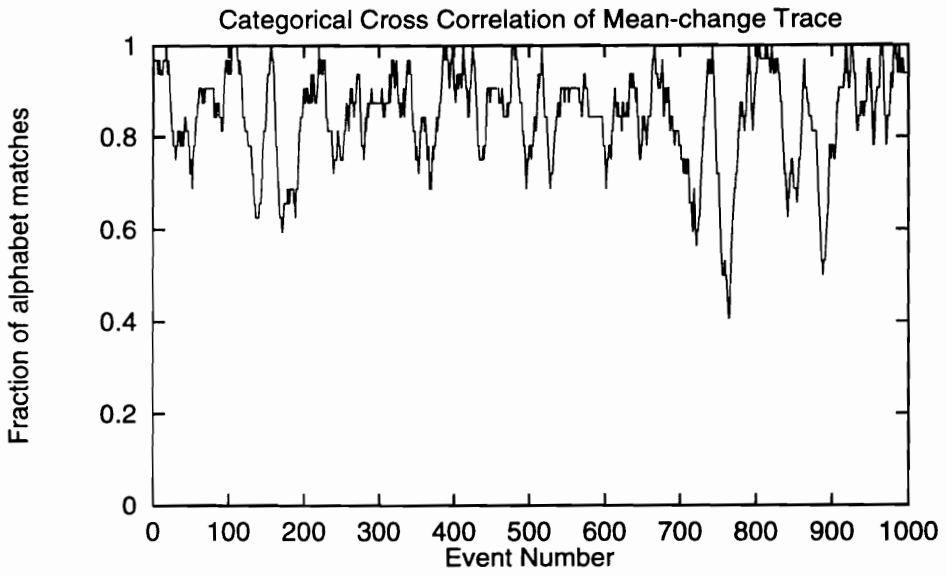


Figure 6.93: Categorical Alphabet Match Cross Correlation Plot: Mean-change Original versus Synthesized Trace (seed 1)

to store a lot of information about it and consequently the model of the GE memory trace is not very compact.

6.5 Sources of Error

The application of our model to the test case traces (Section 6.1) demonstrates that a group of traces, such as the IBM CICS trace, GE Message trace, and the Mean-change trace, are modeled exceptionally well. For the Star Wars trace, on the contrary, our model does poorly. In this section we discuss various shortcomings in the model and the techniques we use to estimate and reduce the model parameters that introduce error in our modeling approach.

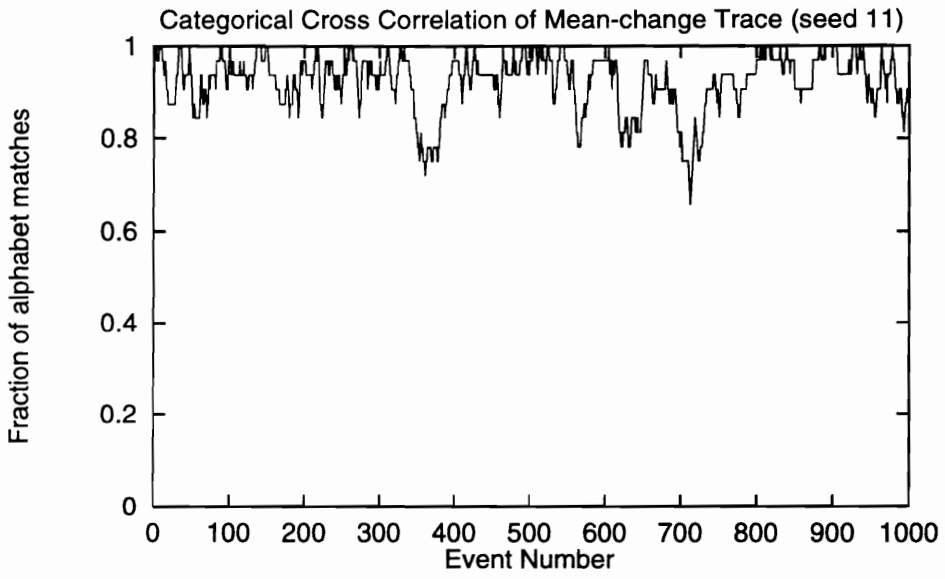


Figure 6.94: Categorical Alphabet Match Cross Correlation Plot: Mean-change Original versus Synthesized Trace (seed 11)

6.5.1 Error in Model

We have chosen a probabilistic approach to represent a trace. Consequently, a certain amount of error is attributable to the variance of the stochastic process that we use to model the trace.

The definition of the piecewise stochastic process (Section 3.4) assumes that the joint probability mass function for the trace in one interval is independent of the joint probability mass function of the trace in any other interval. This assumption is violated in many traces that are highly autocorrelated. A problem similar to that of having autocorrelated intervals also occurs in the domain of discrete event simulation while estimating the mean and the confidence interval around the mean of a stochastic process as described in [Law, 1977]. The highly autocorrelated sample realization of the stochastic process is partitioned into *batches* such that the sample means of the batches are uncorrelated. The process of batch means is described in [Law and Carson, 1978]. In our case the model has a large number of intervals

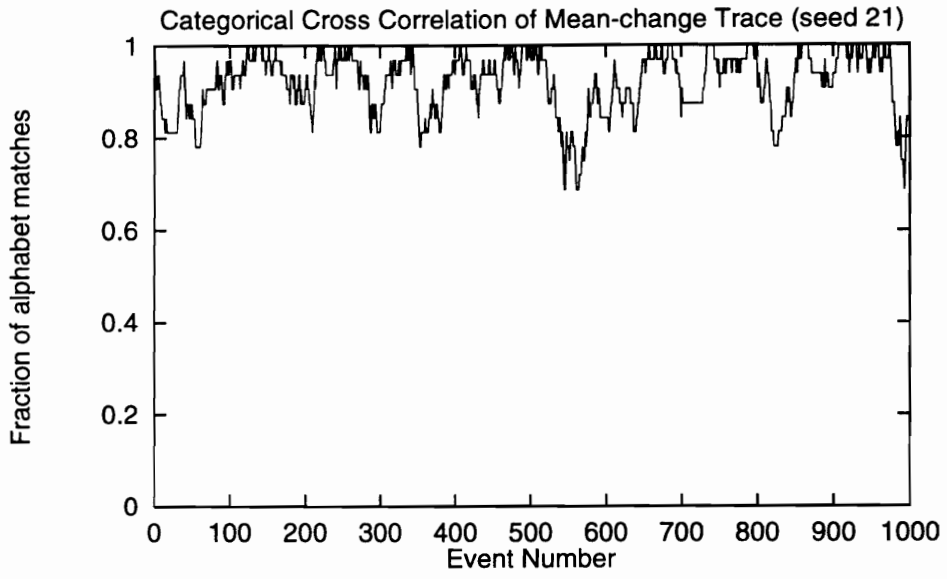


Figure 6.95: Categorical Alphabet Match Cross Correlation Plot: Mean-change Original versus Synthesized Trace (seed 21)

per state. Testing for pairwise independence of k intervals will require $\binom{k}{2}$ comparisons. Even if we did determine the size of the intervals based on a test of pairwise independence we may easily violate the other tenet of our model which states that the probability mass function of the stochastic process within an interval remains the same.

6.5.2 Error in Parameter Estimation

We estimate the probability of occurrence of a state in the trace by the rate of occurrence of that state. The rate of occurrences of a state are estimated as the slopes of a piecewise linear regression fitted to the REG path for the state (see Chapter 4). The estimates of probabilities are consequently only as good as the functions fitting the REG path. Thus, if the REG paths are smooth curves instead of straight lines our estimation procedure will contribute to the error in the model.

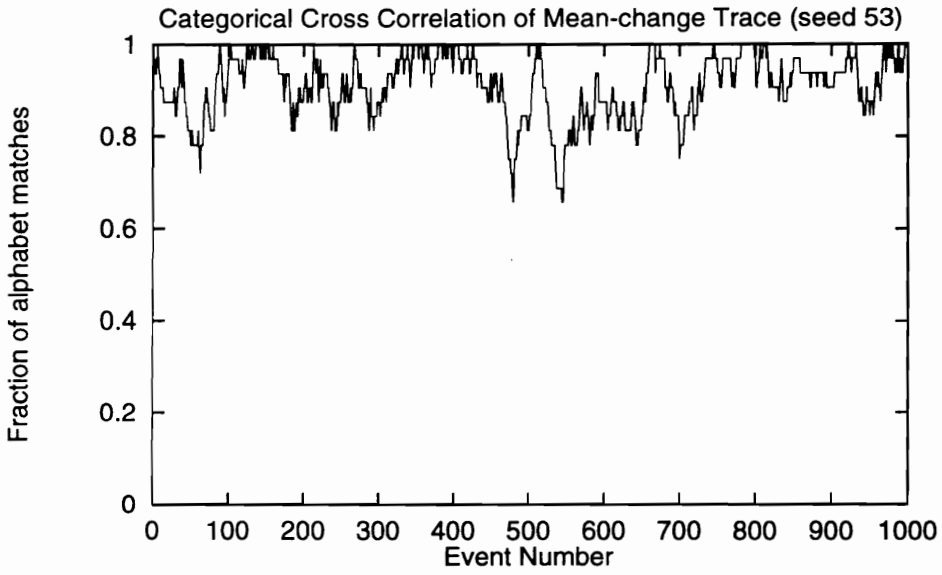


Figure 6.96: Categorical Alphabet Match Cross Correlation Plot: Mean-change Original versus Synthesized Trace (seed 53)

6.5.3 Error Due to Model Size Reduction

Model reduction is accomplished by reducing the number of unique states as well as by reducing the number of partitioning events per state. Consider the former first. Removing a state from the trace causes the probability of occurrence of this state to be shared between neighboring states. If the state being removed happens to be an frequently occurring state this can contribute significantly towards the error in the final model.

Reducing the number of partitions implies fitting the REG path for a state with a piecewise linear regression that has fewer straight lines. Clearly, the probability estimates thusly obtained are approximate. This contributes towards the error in the final model.

6.5.4 Sources of Error in Models of Test Case Traces

In the paragraphs that follow we discuss the sources of error that contribute to the loss of accuracy in the piecewise independent stochastic process models for test case traces

Table 6.3: Model Size for Test Case Traces (Ordered by Model Size as a % of Trace Size)

Storage in bytes			
Name	Trace Size	Model Size	Model Size as % of Trace Size
WWW Trace	18281	120	0.7
Time-trend Trace	2001	229	11.4
Periodic Trace	201	30	14.9
Star Wars Trace	620325	10309	16.6
UoS Ethernet Trace	7740	1840	23.7
IBM CICS Trace	57279	16656	29.1
Mean-change Trace	2001	773	38.6
Ingalls Trace	16221	6480	39.9
Dining Philosophers Trace	22890	9477	41.4
GE Message Trace	1051	562	53.5
ACM Trace	3997	2178	54.0
GE Memory Trace	1047	887	84.7

(Section 6.1).

GE Message Trace: The transient behavior of the GE Message trace is captured very accurately by our model. However, the regular periodic transitions in the trace suggest that there may be a temporal dependence between the states in this trace. This feature of the trace is not captured by our model.

Dining Philosophers Trace: The Dining Philosophers trace is very similar to the GE Message trace. It shows an underlying transient trend along with periodic transitions between states. This suggests that there may be a temporal dependence between the states. While our model captures the transient behavior quite well the dependence between the states in the trace is not captured by our model.

UoS Ethernet Trace: The REG paths in the UoS Ethernet trace (Figure 6.3) can be fitted with polynomial functions instead of straight line segments. A large portion of the error

in the model for this trace is therefore attributable to the estimation of model parameters as well as the application of model size reduction techniques.

IBM CICS Trace: The REG for the IBM CICS trace (Figure 6.4) shows simple REG paths such that each REG path can be fitted with a small number of straight line segments. This trace is therefore ideally suited for our model. Our model for this trace is accurate and parsimonious. Most of the error in the model for this trace is attributable to the application of model size reduction techniques.

GE Memory Trace: The case of the GE Memory trace is similar to the Dining Philosophers trace. This is a highly autocorrelated trace and our model for this trace fails to capture the temporal dependence between the large number of states (more than 10% of trace length) in this trace.

WWW Trace: The REG of the WWW trace (Figure 6.6) shows that each REG path in this trace can be fitted by a single straight line. Thus our model for this trace is extremely accurate and parsimonious.

Ingalls Trace: The Ingalls trace has a large state space. Our model for this trace was reduced in size by the application of two model size reduction techniques namely: reducing the number of states and reducing the number of partitioning events. Once we apply the model size reduction technique to reduce the number of partitioning events in the model the autocorrelation structure of the trace is not preserved (Figure 6.57). Most of the loss of accuracy in the model of this trace is attributable to the application of model size reduction techniques.

ACM Trace: The REG paths in the ACM trace (Figure 6.8) can be fitted with polynomial functions instead of straight line segments. The ACM trace is also autocorrelated. Once we apply the model size technique to reduce the number of partitioning events in the model

the autocorrelation structure of the trace is not preserved (Figure 6.64). A large portion of the error in the model for this trace is therefore attributable to the estimation of model parameters as well as the application of model size reduction techniques.

Star Wars Trace: The Star Wars trace has a large state space. Our model for this trace was reduced in size by the application of two model reduction techniques namely: reducing the number of states and reducing the number of partitioning events. Once we apply the model size technique to reduce the number of partitioning events in the model the autocorrelation structure of the trace is not preserved (Figure 6.71). The loss of accuracy in the model of this trace is attributable to the error in estimating the parameters for our model as well as the application of model size reduction techniques.

Periodic Trace: The Periodic trace has a simple REG (Figure 6.10) such that each REG path can be fitted with a small number of straight line segments. Our model is therefore extremely suitable for this trace. Most of the error in our model for this trace is attributable to the application of model size reduction techniques.

Time-trend Trace: The REG of the Time-trend trace (Figure 6.11) shows that the REG paths corresponding to the two states in this trace are good test cases for a polynomial fit. However, we have used straight line segments to fit these REG paths. Thus the loss of accuracy in our model of this trace is attributable to the error in estimating the parameters for the model.

Mean-change Trace: The Mean-change trace has a simple REG (Figure 6.12) such that each REG path can be fitted with a small number of straight line segments. Our model is therefore extremely suitable for this trace. Most of the error in our model for this trace is attributable to the application of model size reduction techniques.

6.6 Recommendations for Building a Model

Based on the evaluation of the PISP model using the 12 test case traces described in Section 6.1, we make the following recommendations for the use of the model:

- The appearance of the REG of a trace provides key insight into the applicability of the PISP model to the trace. A REG with few REG paths implies few unique states. This means that the resulting PISP model will be parsimonious. Also, if the paths in the REG are such that each path seems a good fit for a few straight line segments then the resulting PISP model will fit the trace data well and will be parsimonious.

- A key insight into the applicability of the PISP model to a trace containing numerical data is provided by the autocorrelation plot of the trace. Traces where the autocorrelation is small and reduces to zero within a few lags are more likely to have an adequately accurate PISP model. Therefore the PISP model will perform poorly on traces with features such as a trend, cyclic or seasonal behavior (periodicity is an exception).

- When generating the PISP model of a numerical trace with a large number of unique states, the state space may be reduced using the *Reducing the Number of States* technique described in Section 5.2, in a stepwise manner. Each step taken to the decrease the state space of the trace must be followed by a validation step to see if the resulting model is adequate.

- The size of a PISP model that has a large number of intervals may be reduced by using the em Reducing the Number of Partitioning Events technique described in Section 5.1, in a stepwise manner. Each step taken to the decrease the number of partitioning events in the model must be followed by a validation step to see if the resulting model is adequate.

Chapter 7

INFERRING AN ENSEMBLE MODEL FROM ONE TRACE

The primary focus of the work described in this dissertation is to build a model of trace data from a single trace. However, for many systems it may be possible to obtain a collection of traces representing the behavior of such systems under identical constraints. Such a collection of traces could then be considered as sample outcomes of the stochastic process underlying the system. For the discussion in this chapter we shall use the term *ensemble* to describe such a collection of traces from a system. In general if an ensemble of traces is available then to infer the behavior of the system we must construct a model of the system from all the traces in the ensemble. The construction of a model from an ensemble of traces is a step beyond the scope of this dissertation. Instead, in this chapter we present a discussion on the appropriateness of a piecewise independent process model of one trace in an ensemble for other traces in the same ensemble. We want to determine if a piecewise independent process model from only one, arbitrarily chosen, trace in the ensemble is sufficient to represent the entire ensemble.

As described in Chapter 4 the parameters of the piecewise independent process model of a trace are estimated based on the REG generated from the trace. Consequently, if two traces have REGs that differ significantly then the corresponding piecewise independent process models will also differ significantly. Recall from the definition of the piecewise independent process (Section 3.4) that the joint probability mass function of a piecewise independent process is identical within an *interval.* If the intervals of the piecewise independent process model for a trace represent actual behavioral changes in the stochastic process underlying the ensemble then a piecewise independent process model from one trace in an

ensemble will represent all the traces in the ensemble well.

7.1 Comparing Models for Similarity

Our objective is to determine if a model generated from one trace in an ensemble represents other traces in the ensemble well. One way of testing this hypothesis is to generate a model from each trace in an ensemble and then determine how similar these models are to each other. The more similar the models are to each other the stronger the evidence that one model represents all traces adequately. A model is expressed as a joint probability mass function, consequently, the problem of comparing models for similarity is the same as comparing probability distributions for similarity. Probability distributions can be directly compared by using measures of distance such as the Kullback Liebler distance and Hellinger distance [Torgersen, 1991]. However, none of these distances can be measured conveniently for a time dependent discrete probability mass function that describes our model. Therefore, instead of comparing models we compare traces synthesized from the models. We use categorical correlation plots (Section 6.2.4) to measure the strength of the correlation between pair of traces (each trace synthesized from a different model). Strong correlation between traces is an indicator of similarity between corresponding models. We also use the Kruskall Wallis test of homogeneity (Section 6.2.2) to determine whether synthesized traces from different models are homogeneous. Strong indications of homogeneity imply that all the synthesized traces may be sample realizations of from the same model.

7.2 Experimental Results from Test Case Ensembles

We use the known distributions used earlier to generate the Mean-change trace, the Time-trend trace, and the Periodic trace (Section 6.1) to generate ensembles of traces. We shall name these ensembles *Mean-change ensemble*, *Time-trend ensemble*, and *Periodic ensemble* respectively. Each ensemble consists of three traces. In each ensemble the first trace was generated with the random seed 10000001, the second trace was generated with the

random seed 230523271, and the third trace was generated with the random seed 999999999. We constructed a piecewise independent process model from every trace in an ensemble. Consequently, 3 models were constructed for an ensemble. Finally, we synthesized 5 traces with seed indexes 1,11,21,53, and 99 respectively from each model. Thus each trace in an ensemble was used to synthesize 5 synthetic traces. Each ensemble was therefore used to generate 15 synthetic traces. We use the Kruskall-Wallis test to determine if all 15 synthesized traces generated from the same ensemble are homogeneous. Recall that each ensemble has 3 traces and each trace was used to generate a group of 5 synthesized traces. Consequently, we have 3 groups of 5 synthesized traces; each group corresponding to a trace in the ensemble. We plot the categorical correlations for pairs of synthesized traces such that the synthesized traces constituting a pair belong to different groups. In the discussion that follows we only present the two categorical correlation plots that represent the strongest and weakest correlations.

In the following sections we discuss insights drawn by inspecting the REGs of the traces in each test case ensemble. We also detail the results of the Kruskall-Wallis test and categorical correlation plots for all test case ensembles.

7.2.1 Mean-change Ensemble

We constructed a piecewise independent stochastic process model for each trace in this ensemble. Each model thus constructed was used to generate 5 synthetic traces. Recall that our model of a trace depends on the REG of the trace. We present the REGs for traces in the Mean-change ensemble with seeds 10000001, 230523271, and 999999999 in Figures 7.1, 7.2, and 7.3 respectively. The REGS for all three traces in this ensemble are extremely similar to each other. Also, the REG paths are straight lines which makes these traces good candidates for the piecewise independent stochastic process model. Consequently, we expect models generated from the three traces in this ensemble to be similar to each other.

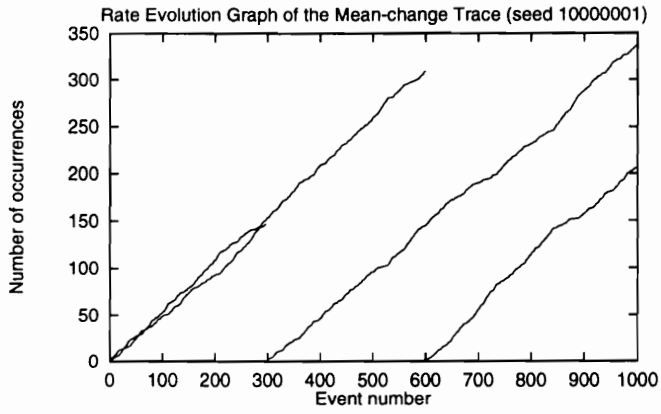


Figure 7.1: REG of the Mean-change Trace (seed 10000001)

Kruskall-Wallis Test The 15 synthetic traces generated from the three models for this ensemble were subjected to a 15 way Kruskall-Wallis test ($\alpha = 0.01$) as described in Section 6.2.2. We found that 60% of the trace segments passed the test.

Categorical Correlation Plots The Figures 7.4 and 7.5 display the categorical correlation plots (Section 6.2.4) for the synthesized traces that show the strongest and weakest correlation. The categorical correlation plots show that the 15 synthesized traces are highly correlated.

7.2.2 Periodic Ensemble

We constructed a piecewise independent stochastic process model for each trace in this ensemble. Each model thus constructed was used to generate 5 synthetic traces. We present the REGs for traces in the Periodic ensemble with seeds 10000001, 230523271, and 999999999 in Figures 7.6, 7.7, and 7.8 respectively. The REGS for all three traces in this ensemble differ slightly from each other. However, the REG paths are straight lines which makes these traces good candidates for the piecewise independent stochastic process model.

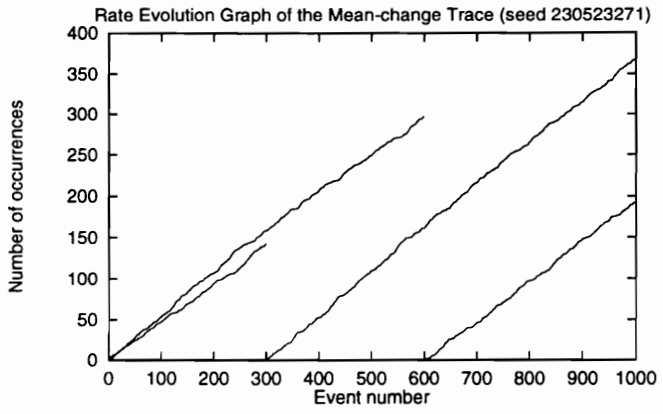


Figure 7.2: REG of the Mean-change Trace (seed 230523271)

Kruskall-Wallis Test The 15 synthetic traces generated from the three models for this ensemble were subjected to a 15 way Kruskall-Wallis test ($\alpha = 0.01$) as described in Section 6.2.2. We found that 50% of the trace segments passed the test.

Categorical Correlation Plots The Figures 7.9 and 7.10 display the categorical correlation plots (Section 6.2.4) for the synthesized traces that show the strongest and weakest correlation. The categorical correlation plots show that the 15 synthesized traces are highly correlated.

7.2.3 Time-trend Ensemble

We constructed a piecewise independent stochastic process model for each trace in this ensemble. Each model thus constructed was used to generate 5 synthetic traces. We present the REGs for traces in the Time-trend ensemble with seeds 10000001, 230523271, and 999999999 in Figures 7.11, 7.12, and 7.13 respectively. The REGS for all three traces in this ensemble are similar to each other. The REG paths, however, are smooth curves which our model approximates by segments of straight lines. This implies that estimation of model parameters may introduce error in the model (Section 6.5).

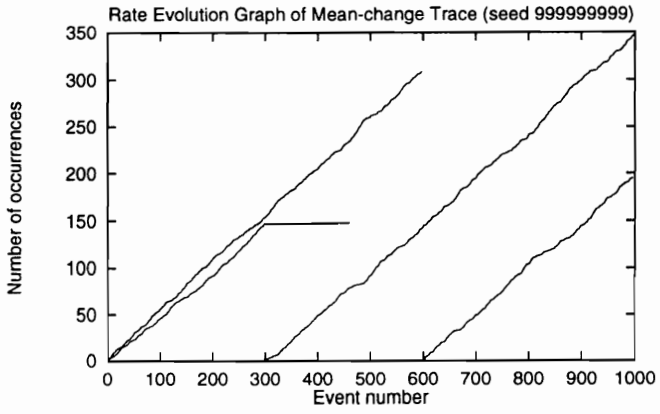


Figure 7.3: REG of the Mean-change Trace (seed 999999999)

Kruskall-Wallis Test The 15 synthetic traces generated from the three models for this ensemble were subjected to a 15 way Kruskall-Wallis test ($\alpha = 0.01$) as described in Section 6.2.2. We found that none of the trace segments passed the test.

Categorical Correlation Plots The Figures 7.14 and 7.15 display the categorical correlation plots (Section 6.2.4) for the synthesized traces that show the strongest and weakest correlation. The categorical correlation plots show that the 15 synthesized traces are correlated. This contradicts the result from the Kruskall-Wallis test.

7.3 Summary of Results

The validation results for the Mean-change ensemble and Periodic ensemble show that the piecewise independent stochastic process models constructed from any one member trace from these ensemble represents the other traces in the same ensemble adequately. The validation results for the Time-trend ensemble, however, contradict each other. The Kruskall-Wallis test for this ensemble shows poor results, while the categorical correlation plots contradicts that result. Inspite of the generally positive experimental results our experiments are limited by the fact that the test case ensembles represent simple stochastic systems with small variances. Consequently, it is desirable to augment our modeling

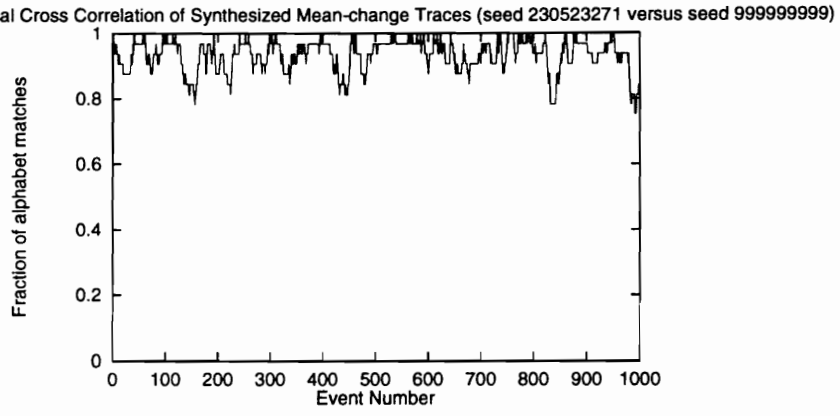


Figure 7.4: Categorical Correlation for Synthesized Mean-change Traces (seed 230523271 versus seed 999999999)

approach so that the model is built from an ensemble of data rather than a single trace.

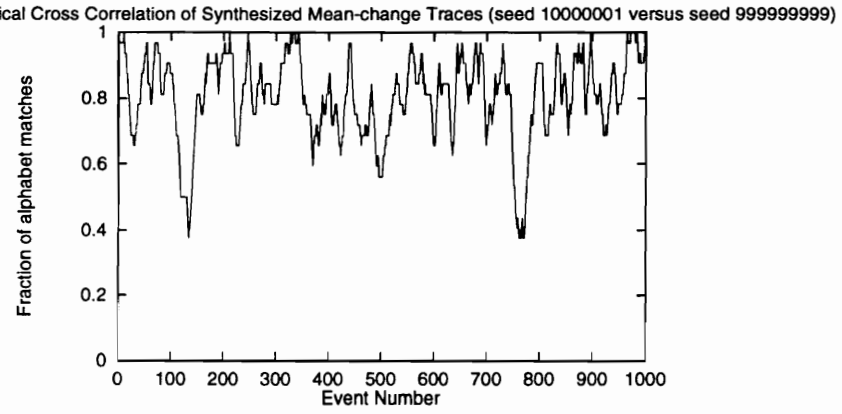


Figure 7.5: Categorical Correlation for Synthesized Mean-change Traces (seed 10000001 versus seed 999999999)

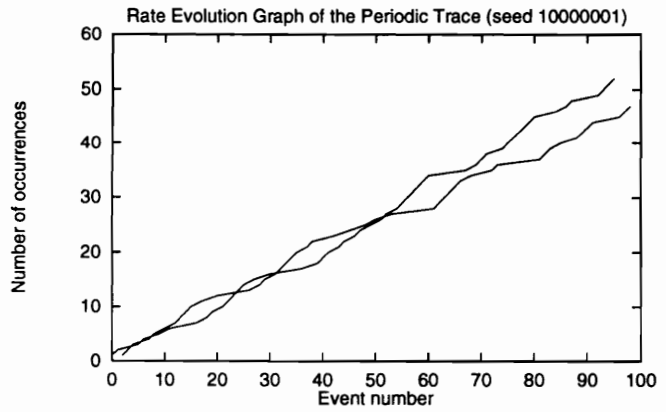


Figure 7.6: REG of the Periodic Trace (seed 10000001)

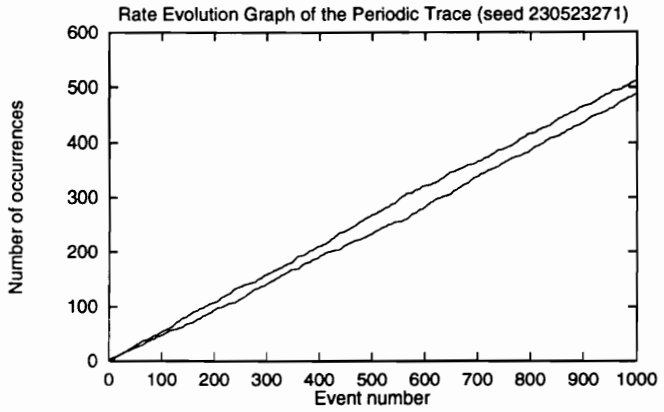


Figure 7.7: REG of the Periodic Trace (seed 230523271)

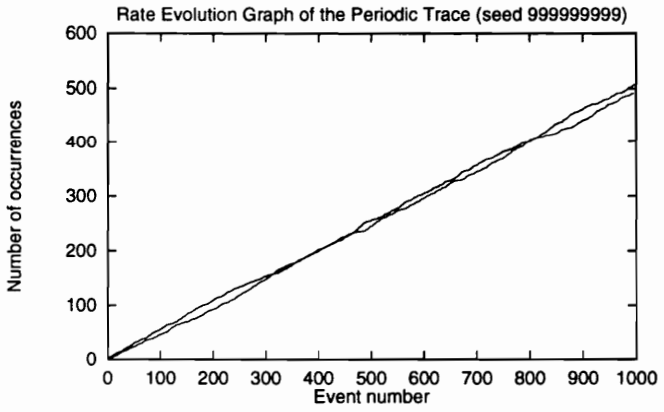


Figure 7.8: REG of the Periodic Trace (seed 999999999)

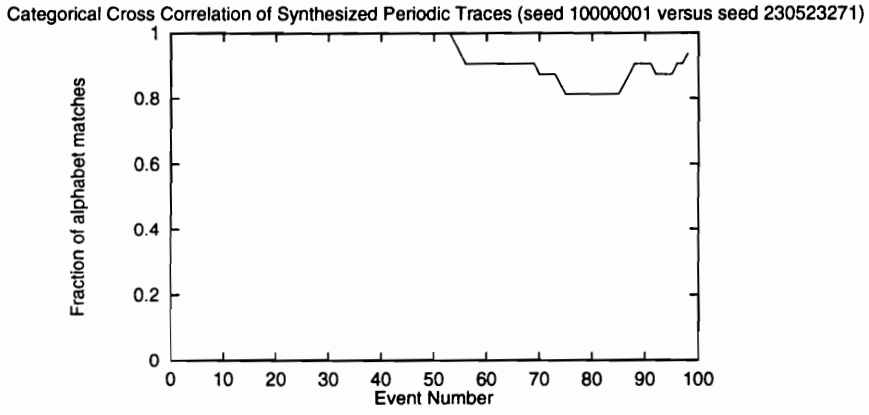


Figure 7.9: Categorical Correlation for Synthesized Periodic Traces (seed 10000001 versus seed 230523271)

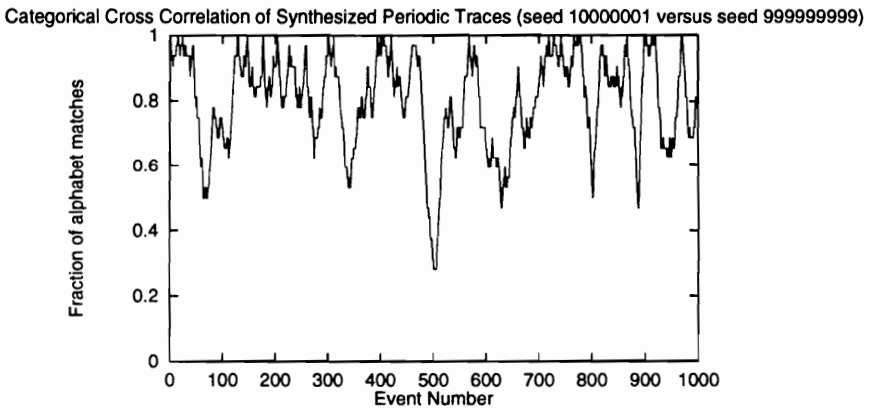


Figure 7.10: Categorical Correlation for Synthesized Periodic Traces (seed 10000001 versus seed 999999999)

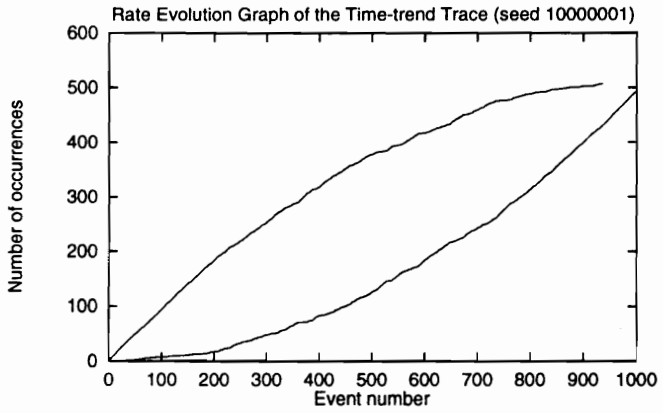


Figure 7.11: REG of the Time-trend Trace (seed 10000001)

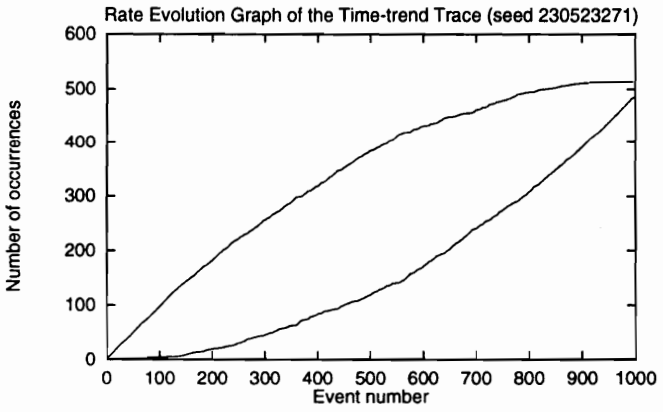


Figure 7.12: REG of the Time-trend Trace (seed 230523271)

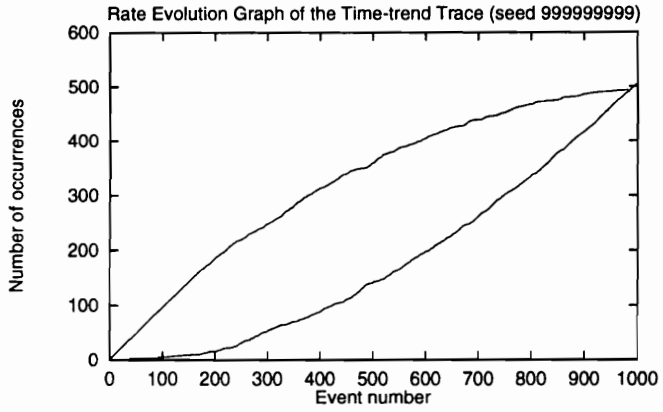


Figure 7.13: REG of the Time-trend Trace (seed 999999999)

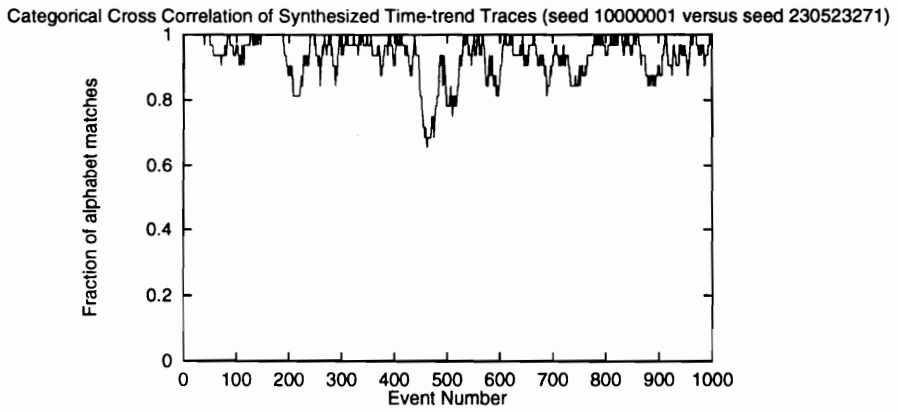


Figure 7.14: Categorical Correlation for Synthesized Time-trend Traces (seed 10000001 versus seed 230523271)

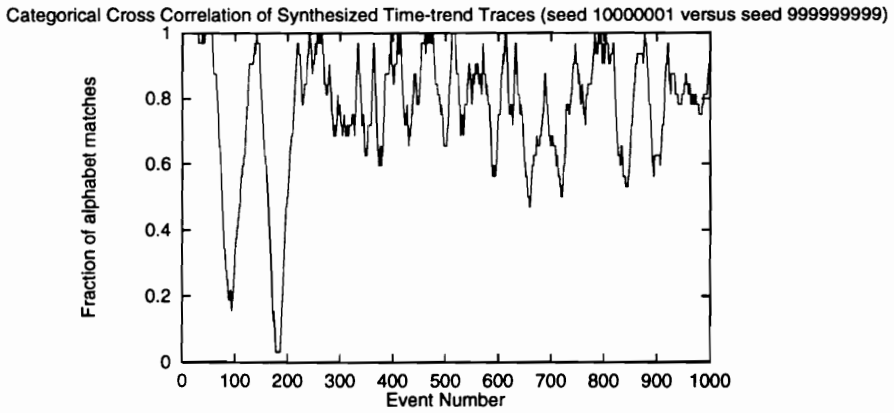


Figure 7.15: Categorical Correlation for Synthesized Time-trend Traces (seed 10000001 versus seed 999999999)

Chapter 8

CONCLUDING REMARKS

In this chapter we outline the salient features our solution approach. We also present a list of open problems related to this dissertation work and some possible solution ideas.

8.1 Problem Definition

Trace data representing real workloads often shows transient behavior. Transient trace data can be considered to be a sample realization of a stochastic process, $\{X_t\}$, such that random variables X_t follow different probability distributions. The primary objective of this dissertation research is to define a model for transient trace data and provide methods to build such a model for test case traces.

8.2 Solution Approach

In this dissertation we introduce the piecewise independent stochastic process as our solution for modeling traces of transient data. The piecewise independent stochastic process is expressed as a family of independent random variables. The domain of the index parameter of this stochastic process is divided into disjoint intervals such that the behavior of the stochastic process in an interval can be described by a probability mass function that is invariant to the index parameter of the stochastic process. We propose a method to estimate the parameters of the piecewise independent stochastic process model for sample traces. The parameter estimation method is based on the *Rate Evolution Graph* which is a unique visualization of trace data. The REG consists of paths, one for each unique state in the trace, that represent the evolution of the time dependent rate of occurrence of the

corresponding state. We use a collection of straight lines to fit each path in the REG. The slope of a fitted straight line segment is used as an estimate of the probability of occurrence of the corresponding state. Finally, the slopes of all the fitted straight lines for each REG path are used to construct an estimate of the time dependent joint probability mass function that describes the piecewise independent stochastic process model for the trace.

8.3 Summary of Results

The REG plots of most test case traces (Section 6.1.2) and corresponding validation results (Section 6.3) show that our model does quite well for traces with a small number of unique states and REG paths that can fitted with a few line segments. Notably accurate and parsimonious are the piecewise independent stochastic process models for the WWW Trace, Mean-change Trace, IBM CICS Trace, and the Periodic Trace. The PISP model does not track the autocorrelation present in some traces. This is evident by the validation results of the PISP model for the highly autocorrelated Star Wars trace.

8.4 Open Problems and Future Work

The following issues constitute open problem areas that are related to the research described in this dissertation.

- Our modeling approach builds a stochastic process model of trace data based on one sample realization. This approach is necessary in many instances because many traces contain observational data that cannot be replicated. For some simple systems (Chapter 7) a model built from one trace can be represent system behavior. In general, however, a model based on an ensemble of traces from the same system will be a better predictor behavior of that system. Building on our approach the REGs of individual traces in an ensemble can be used to build a model of the ensemble.

- Our model is appropriate for trace data which can be considered as a time series such that the index parameter of the time series is equally spaced and monotonically

increasing. Thus our model ignores the actual time stamps associated with the trace data. The analysis of trace data where the index parameter is not equally spaced is an open research area.

- A key assumption in the piecewise independent stochastic process model is that the joint probability mass function for the trace in one interval is independent of the joint probability mass function of the trace in any other interval. In reality this assumption is violated by many traces. The definition of the piecewise independent stochastic process model should be augmented such that the correlation structure of the trace factors into the model.

- We use a collection of straight lines to fit a REG path. Instead, as suggested in Section 3.2 polynomials can be fitted to REG paths. This approach may especially be appropriate when REG paths are smooth high order curves.

- State space explosion is a major problem in building models of trace data. We present one technique to reduce the number of unique states in the model. Alternative effective techniques to reduce state space size will have a significant impact on controlling the size of our model.

All the tools needed to construct a piecewise independent stochastic process model from traces are implemented in CHITRA, a performance analysis and visualization tool being built at Virginia Tech. CHITRA is available via WWW from http://www.cs.vt.edu/~chitra.

REFERENCES

[Abrams, 1986] Abrams, M. (1986). *Performance Analysis of Unconditionally Synchronizing Distributed Computer Programs Using The Geometric Concurrency Model.* PhD thesis, Computer Sci. Dept., Univ. of MD. TR-1696.

[Abrams et al., 1994] Abrams, M., Batongbacal, A., Ribler, R., and Vazirani, D. (1994). CHITRA94: A tool to dynamically characterize ensembles of traces for input data modeling and output analysis. Technical Report TR 94-21, Computer Sci. Dept., Virginia Tech, Blacksburg, VA 24061-0106. Available from World Wide Web location http://info.cs.vt.edu/chitra.

[Abrams et al., 1992] Abrams, M., Doraswamy, N., and Mathur, A. (1992). Chitra: Visual analysis of parallel and distributed programs in the time, event, and frequency domain. *IEEE Trans. on Parallel and Distributed Systems*, 3(6):672–685.

[Abrams et al., 1995] Abrams, M., Williams, S., Abdulla, G., Patel, S., Ribler, R., and Fox, E. (1995). Multimedia traffic analysis using chitra95. Technical Report TR 95-05, Computer Sci. Dept., Virginia Tech, Blacksburg, VA 24061-0106. Submitted of pub.

[Ahluwalia and Singhal, 1992] Ahluwalia, A. and Singhal, M. (1992). Performance analysis of the communication architecture of the connection machine. *IEEE Trans. on Parallel and Distributed Systems*, 3(6):728–738.

[Balci, 1994] Balci, O. (1994). Validation, verification, and testing techniques throughout the life cycle of a simulation study. *Annals of Operation Research*, 53:121–173.

[Beran, 1994] Beran, J. (1994). *Statistics for Long-Memory Processes.* Chapman Hall, New York.

[Berry and Jefferson, 1985] Berry, O. and Jefferson, D. (1985). Critical path analysis of distributed simulation. In *Soc. for Comp. Simulation*, San Diego, CA.

[Bhat, 1984] Bhat, U. N. (1984). *Elements of Applied Stochastic Processes.* John Wiley, New York, 2nd edition.

[Box and Jenkins, 1976] Box, G. E. P. and Jenkins, G. M. (1976). *TIME SERIES ANALYSIS forecasting and control, Revised Edition.* Holden-Day, Oakland, California.

[Crovella and Bestavros, 1995] Crovella, M. E. and Bestavros, A. (1995). Explaining world wide web traffic self-similarity. Technical Report TR-95-015, Computer Science Department, Boston University, 111 Cummington St, Boston, MA 02215.

[Denning, 1968] Denning, P. J. (1968). The working set model for program behavior. *Communications of the ACM*, 11(5):323–333.

[Denning, 1980] Denning, P. J. (1980). Working sets past and present. *IEEE Trans. on Software Engineering*, SE-6(1):64–84.

[D.Geist and B.Melamed, 1992] D.Geist and B.Melamed (1992). Testool: An environment for visual interactive modeling of autocorrelated traffic. In *Proc. of the 1992 International Conference on Communications*, pages 1285–1289, Chicago.

[Doraswamy, 1991] Doraswamy, N. (1991). Chitra: A visualization system to analyze the dynamics of parallel programs. Master's thesis, Computer Sci. Dept., Virginia Tech, Blacksburg, VA 24061-0106.

[Ferrari et al., 1983] Ferrari, D., Serazzi, G., and Zeigner, A. (1983). *Measurement and Tuning of Computer Systems.* Prentice-Hall, London.

[Garrett, 1993] Garrett, M. W. (1993). *Statistical Analysis of a Long Trace of Variable Bit Rate Video Traffic.* PhD thesis, Columbia Univ. Chapter IV.

[Garrett and Willinger, 1994] Garrett, M. W. and Willinger, W. (1994). Analysis, modeling and generation of self-similar vbr video traffic. In *Proc. SIGCOMM*, pages 269–279, London, UK. ACM.

[Goldberg and Hennessy, 1993] Goldberg, A. J. and Hennessy, J. L. (1993). Mtool: An integrated system for performance debugging shared memory applications. *IEEE Trans. on Parallel and Distributed Systems*, 4(1):28–40.

[Graham et al., 1982] Graham, S. L., Kessler, P. B., and McKusick, M. K. (1982). gprof: A call graph execution profiler. In *Proceedings of the SIGPLAN '82 Symposium on Compiler Construction*, pages 120–126. *SIGPLAN Notices*, 17(6).

[Heyman and Lakshman, 1996] Heyman, D. P. and Lakshman, T. V. (1996). Source models for vbr broadcast-video traffic. *IEEE Trans. on Networking*, 4(1):40–48.

[Heyman et al., 1992] Heyman, D. P., Tabatabai, A., and Laskshman, T. V. (1992). Statistical analysis and simulation study of video teleconference traffic in atm networks. *IEEE Trans. on Circuits and Systems for Video Technology*, 2(1):49–58.

[Hitson, 1990] Hitson, B. (1990). Knowledge-based monitoring and control of distributed systems. Technical Report TR 91-23, Computer Systems Laboratory, Stanford University, Stanford, CA.

[ICASE, 1993] ICASE (1993). *ICASE/NASA LaRC Workshop on Software Tools and Techniques for Performance and Reliability Estimation*, Langley, VA. NASA Langley Research Center.

REFERENCES

[Jain, 1991] Jain, R. (1991). *The Art of Computer Systems Performance Analysis.* John Wiley, New York.

[JPL, 1991] JPL (1991). *Time Warp Operating System Version 2.5.1, User Manual.* Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA.

[Law, 1977] Law, A. M. (1977). Confidence intervals in discrete event simulation: A comparison of replication and batch means. *Naval Res. Logist. Quart.*, 24(4):667–678.

[Law and Carson, 1978] Law, A. M. and Carson, J. S. (1978). A sequential procedure for determining the length of a steady-state simulation. *Operations Research*, 27(5):1011–1025.

[Law and Kelton, 1991] Law, A. M. and Kelton, W. D. (1991). *Simulation Modeling and Analysis.* McGraw-Hill, New York, 2nd edition.

[Leland et al., 1994] Leland, W. E., Taqqu, M. S., Willinger, W., and Wilson, D. V. (1994). On the self-similar nature of ethernet traffic (extended version). *IEEE Trans. on Networking*, 2(1):1–15.

[Martonosi et al., 1992] Martonosi, M., Gupta, A., and Anderson, T. (1992). MemSpy: Analyzing memory system bottlenecks in programs. In *Proc. SIGMETRICS*, pages 1–12, Newport, RI. ACM.

[McClave and Dietrich, 1985] McClave, J. T. and Dietrich, F. H. (1985). *Statistics.* Dellen Publishing Company, San Fransisco, CA, 3rd edition.

[Miller et al., 1990] Miller, B., Clark, M., Hollingsworth, J., Kierstead, S., Lim, S., and Torzewski, T. (1990). IPS-2: The second generation of a parallel program measurement system. *IEEE Trans. on Parallel and Distributed Systems*, 1(2):206–217.

[Molloy, 1982] Molloy, M. (1982). Performance analysis using stochastic petri nets. *IEEE Trans. on Computers*, C-31(9):913–917.

[Ott, 1988] Ott, L. (1988). *An Introduction to Statistical Methods and Data Analysis.* PWS-Kent, Boston, 3rd edition.

[Paxson and Floyd, 1994] Paxson, V. and Floyd, S. (1994). Wide-area traffic: The failure of poisson modeling. In *Proc. SIGCOMM*, pages 257–267, London, UK. ACM.

[Plateau and Atif, 1991] Plateau, B. and Atif, K. (1991). Stochastic automata network for modeling parallel systems. *IEEE Trans. on Software Engineering*, 1991(10):1093–1109.

[Reed et al., 1992] Reed, D. A., Aydt, R. A., Madhyastha, T. M., Noe, R. J., Shields, K. A., and Schwartz, B. W. (1992). The Pablo performance analysis environment. Dept. of Comp. Sci., Univ. of IL.

[Ribler et al., 1995] Ribler, R., Mathur, A., and Abrams, M. (1995). Visualizing and modeling categorical time series data. In *ICASE/LaRC Symposium on Visualizing Time-Varying Data*, pages 3–19, Williamsburg, VA.

[Rover and Waheed, 1993] Rover, D. T. and Waheed, A. (1993). Multiple-domain analysis methods. In *ACM/ONR Workshop on Parallel Debugging and Performance*, proceedings appeared in *ACM SIGPLAN Notices,* 28(12), Dec. 1993, pages 53–63, San Diego. ACM.

[Torgersen, 1991] Torgersen, E. (1991). *Comparison of Statisical Experiments.* Cambridge Univ. Press, Cambridge.

[Vazirani, 1996] Vazirani, D. (1996). Towards automated network traffic modeling for packetized video with Chitra. Master's thesis, Computer Sci. Dept., Virginia Tech, Blacksburg, VA 24061-0106.

[Wolfram, 1992] Wolfram, S. (1992). *Mathematica, A System for Doing Mathematics by Computer.* Addison Wesley, Reading, MA.

Appendix A

MODELS OF TEST CASE TRACES

In this appendix we display information about the piecewise independent stochastic process model for the test case traces introduced in Section 6.1. A table showing the number of partitioning events per state is presented for each test case trace. For the WWW Trace, the Mean-change trace, the Time-trend trace, and the Periodic trace we also present the joint probability distributions in tabular form. All other traces in Section 6.1 have a large number of unique states. Their joint probability distributions are therefore not included in this document.

GE Message Trace: The GE message Trace has 31 unique states. In Table A.8 we display the number of partitioning events (also the number of intervals) for each state in the model of this trace.

Dining Philosophers Trace: The Dining Philosophers Trace has 597 unique states. In Table A.9 we display the number of partitioning events (also the number of intervals) for each state in the model of this trace. The actual probability mass functiore.

UoS Ethernet Trace: The UoS Trace has 11 unique states. In Table A.10 we display the number of partitioning events (also the number of intervals) for each state in the model of this trace.

IBM CICS Trace: The IBM CICS Trace has 24 unique states. In Table A.11 we display the number of partitioning events (also the number of intervals) for each state in the model of this trace.

GE Memory Trace: The GE memory Trace has 48 unique states. In Table A.12 we display the number of partitioning events (also the number of intervals) for each state in the model of this trace.

WWW Trace: The WWW Trace has 7 unique states. In Table A.15 we display the number of partitioning events (also the number of intervals) for each state in the model of this trace. The joint probabilty mass function for this trace is displayed in Table A.1.

Ingalls Trace: The Ingalls Trace after reduction of state space has 100 unique states. In Tables A.13, A.14 we display the number of partitioning events (also the number of intervals) for each state in the model of this trace.

ACM Trace: The ACM Trace has 18 unique states. In Table A.16 we display the number of partitioning events (also the number of intervals) for each state in the model of this trace.

Star Wars Trace: The Star Wars Trace after reduction of state space has 118 unique states. In Tables A.17, A.18, and A.19 we display the number of partitioning events (also the number of intervals) for each state in the model of this trace.

Mean-change Trace: The Mean-change Trace has 4 unique states. In Table A.20 we display the number of partitioning events (also the number of intervals) for each state in the model of this trace. The joint probabilty mass function for this trace is displayed in Tables A.3, A.4, A.5, and A.6.

Time-trend Trace: The Time-trend Trace has 2 unique states. In Table A.21 we display the number of partitioning events (also the number of intervals) for each state in the model of this trace. The joint probabilty mass function for this trace is displayed in Table A.7.

Periodic Trace: The Periodic Trace has 2 unique states. In Table A.22 we display the number of partitioning events (also the number of intervals) for each state in the model of

this trace. The joint probabilty mass function for this trace is displayed in Table A.2.

Table A.1: Joint Probability Mass Function for the WWW Trace

Interval (open ended)		States						
From	To	0	1	2	3	4	5	6
0	1	1.00	0.00	0.00	0.00	0.00	0.00	0.00
1	2	0.28	0.72	0.00	0.00	0.00	0.00	0.00
2	269	0.15	0.38	0.47	0.00	0.00	0.00	0.00
269	9114	0.14	0.37	0.46	0.00	0.03	0.00	0.00
9114	9133	0.00	0.43	0.54	0.00	0.04	0.00	0.00
9133	9134	0.00	0.92	0.00	0.00	0.08	0.00	0.00
9134	9139	0.00	0.00	0.00	0.00	1.00	0.00	0.00

Table A.2: Joint Probability Mass Function for the Periodic Trace

Interval (open ended)		States	
From	To	0	1
0	2	1.00	0.00
2	96	0.53	0.47
96	99	0.00	1.00

Table A.3: Joint Probability Mass Function for the Mean-change Trace

Interval (open ended)		States			
From	To	0	1	2	3
0	1	1.00	0.00	0.00	0.00
1	18	0.66	0.34	0.00	0.00
18	19	1.00	0.00	0.00	0.00
19	21	0.25	0.25	0.25	0.25
21	28	1.00	0.00	0.00	0.00
28	47	0.51	0.49	0.00	0.00
47	50	1.00	0.00	0.00	0.00
50	56	0.25	0.25	0.25	0.25
56	62	0.00	1.00	0.00	0.00
62	75	0.50	0.50	0.00	0.00
75	82	0.00	1.00	0.00	0.00
82	105	0.47	0.53	0.00	0.00
105	135	0.00	1.00	0.00	0.00
135	142	1.00	0.00	0.00	0.00
142	156	0.62	0.38	0.00	0.00
156	158	1.00	0.00	0.00	0.00
158	171	0.25	0.25	0.25	0.25
171	172	1.00	0.00	0.00	0.00
172	197	0.39	0.61	0.00	0.00
197	202	1.00	0.00	0.00	0.00
202	203	0.47	0.53	0.00	0.00
203	209	0.00	1.00	0.00	0.00
209	225	0.61	0.39	0.00	0.00
225	232	0.00	1.00	0.00	0.00
232	246	0.62	0.38	0.00	0.00
246	252	0.00	1.00	0.00	0.00
252	271	0.59	0.41	0.00	0.00
271	278	0.25	0.25	0.25	0.25
278	287	0.60	0.40	0.00	0.00
287	288	1.00	0.00	0.00	0.00
288	292	0.25	0.25	0.25	0.25

Table A.4: Joint Probability Mass Function for the Mean-change Trace, Contd.

Interval (open ended)		States			
From)	To	0	1	2	3
292	293	1.00	0.00	0.00	0.00
293	297	0.60	0.40	0.00	0.00
297	300	0.00	1.00	0.00	0.00
300	304	0.00	0.52	0.48	0.00
304	309	0.00	0.00	1.00	0.00
309	316	0.00	0.51	0.49	0.00
316	319	0.00	1.00	0.00	0.00
319	323	0.00	0.65	0.35	0.00
323	329	0.00	0.00	1.00	0.00
329	331	0.00	0.67	0.33	0.00
331	332	0.00	1.00	0.00	0.00
332	344	0.00	0.56	0.44	0.00
344	346	0.00	0.00	1.00	0.00
346	362	0.00	0.65	0.35	0.00
362	369	0.00	0.00	1.00	0.00
369	371	0.00	0.47	0.53	0.00
371	373	0.00	1.00	0.00	0.00
373	381	0.00	0.34	0.66	0.00
381	389	0.00	1.00	0.00	0.00
389	390	0.00	0.58	0.42	0.00
390	395	0.00	0.00	1.00	0.00
395	402	0.00	0.69	0.31	0.00
402	403	0.00	1.00	0.00	0.00
403	413	0.00	0.00	1.00	0.00
413	429	0.00	0.46	0.54	0.00
429	433	0.25	0.25	0.25	0.25
433	443	0.00	0.61	0.39	0.00
443	448	0.00	1.00	0.00	0.00
448	460	0.00	0.52	0.48	0.00
460	463	0.00	1.00	0.00	0.00
482	497	0.00	1.00	0.00	0.00
497	503	0.00	0.64	0.36	0.00

Table A.5: Joint Probability Mass Function for the Mean-change Trace, Contd.

Interval (open ended)		States			
From	To	0	1	2	3
512	531	0.00	0.73	0.27	0.00
531	532	0.00	0.00	1.00	0.00
532	538	0.25	0.25	0.25	0.25
538	563	0.00	0.53	0.47	0.00
563	571	0.00	0.00	1.00	0.00
571	582	0.25	0.25	0.25	0.25
582	584	0.00	0.00	1.00	0.00
584	594	0.00	0.52	0.48	0.00
594	600	0.00	1.00	0.00	0.00
600	604	0.00	0.00	0.00	1.00
604	639	0.00	0.00	0.52	0.48
639	648	0.00	0.00	0.00	1.00
648	661	0.00	0.00	0.35	0.65
661	668	0.00	0.00	1.00	0.00
668	671	0.00	0.00	0.36	0.64
671	674	0.00	0.00	0.00	1.00
674	695	0.00	0.00	0.51	0.49
695	703	0.25	0.25	0.25	0.25
703	711	0.00	0.00	0.00	1.00
711	713	0.00	0.00	0.30	0.70
713	718	0.00	0.00	1.00	0.00
718	729	0.00	0.00	0.32	0.68
729	744	0.00	0.00	0.00	1.00
744	750	0.25	0.25	0.25	0.25
750	766	0.00	0.00	0.00	1.00
766	776	0.00	0.00	0.48	0.52
776	778	0.00	0.00	1.00	0.00
778	780	0.00	0.00	0.51	0.49
780	787	0.00	0.00	0.00	1.00
787	788	0.00	0.00	0.43	0.57
788	795	0.00	0.00	1.00	0.00
795	823	0.00	0.00	0.37	0.63
823	842	0.00	0.00	1.00	0.00
842	843	0.25	0.25	0.25	0.25

Table A.6: Joint Probability Mass Function for the Mean-change Trace, Contd.

Interval (open ended)		States			
From	To	0	1	2	3
843	849	0.00	0.00	0.00	1.00
849	860	0.00	0.00	0.71	0.29
860	867	0.00	0.00	0.00	1.00
867	869	0.00	0.00	0.66	0.34
869	872	0.00	0.00	1.00	0.00
872	875	0.00	0.00	0.78	0.22
875	889	0.00	0.00	0.00	1.00
889	897	0.00	0.00	0.77	0.23
897	907	0.00	0.00	1.00	0.00
907	908	0.00	0.00	0.49	0.51
908	918	0.00	0.00	0.00	1.00
918	928	0.00	0.00	0.46	0.54
928	931	0.00	0.00	0.00	1.00
931	964	0.00	0.00	0.51	0.49
964	968	0.00	0.00	0.00	1.00
968	998	0.00	0.00	0.48	0.52
998	999	0.00	0.00	1.00	0.00

Table A.7: Joint Probability Mass Function for the Time-trend Trace

Interval (open ended)		States	
From	To	0	1
0	28	1.00	0.00
28	56	0.91	0.09
56	92	0.00	1.00
92	187	0.91	0.09
187	195	0.00	1.00
195	197	0.50	0.50
197	234	1.00	0.00
234	319	0.69	0.31
319	341	0.00	1.00
341	349	0.66	0.34
349	361	1.00	0.00
361	464	0.61	0.39
464	503	0.00	1.00
503	504	0.50	0.50
504	517	1.00	0.00
517	572	0.43	0.57
572	589	1.00	0.00
589	620	0.36	0.64
620	623	1.00	0.00
623	655	0.40	0.60
655	675	1.00	0.00
675	737	0.42	0.58
737	747	0.00	1.00
747	762	0.50	0.50
762	766	1.00	0.00
766	792	0.32	0.68
792	801	0.00	1.00
801	810	0.50	0.50
810	874	0.00	1.00
874	875	0.50	0.50
875	895	1.00	0.00
895	938	0.10	0.90
938	999	0.00	1.00

Table A.8: Partitioning Events for the GE Message Trace

State(s)	Num of Partitioning Events
0-4	2
5	4
6	2
7	6
8-14	2
15	4
16-30	2

Table A.9: Partitioning Events for the Dining Philosophers Trace

State(s)	Num of Partitioning Events
0-10	2
11	3
12-53	2
54	3
55-596	2

Table A.10: Partitioning Events for the UoS Ethernet Trace

State(s)	Num of Partitioning Events
0	10
1	32
2	77
3	40
4	2
5	54
6	34
7,8	2
9	6
10	22

Table A.11: Partitioning Events for the IBM CICS Trace

State(s)	Num of Partitioning Events
0	2
1	82
2	72
3	59
4	392
5	16
6	68
7	52
8	18
9	38
10	2
11	100
12	32
13	39
14	40
15	3
16	2
17	6
18-23	2

Table A.12: Partitioning Events for the GE memory Trace

State(s)	Num of Partitioning Events
0	4
1-20	2
21	4
22-30	2
31,32	4
33-36	2
37,38	4
39	3
40	5
41	3
42-47	2

Table A.13: Partitioning Events for the Ingalls Trace

State(s)	Num of Partitioning Events
0	25
1	23
2	11
3	3
4-9	2
10	14
11	3
12	6
13	19
14	10
15	2
16	4
17	16
18	26
19	14
20	17
21	9
22	19
23	29
24	26
25	29
26	15
27	28
28,29	25
30	6
31	9
32	23
33	35
34	75
35	72
36	77
37	26
38	10
39	7
40	5
41	14

Table A.14: Partitioning Events for the Ingalls Trace, Contd.

State(s)	Num of Partitioning Events
42	9
43	10
44,45	4
46	20
47	2
48	22
49	17
50	11
51	7
52	9
53	11
54	7
55-57	9
58,59	14
60	20
61	12
62	6
63	2
64	4
65,66	6
67	7
68	11
69	7
70	2
71,72	14
73	6
74,75	3
76	6
77	8
78	5
79,80	6
81-99	2

Table A.15: Partitioning Events for the WWW Trace

State(s)	Num of Partitioning Events
0-6	2

Table A.16: Partitioning Events for the ACM Trace

State(s)	Num of Partitioning Events
0,2	2
3	4
4-7	2
8	6
9	4
10-17	2

Table A.17: Partitioning Events for the Star Wars Trace

State(s)	Num of Partitioning Events
0	25
1	3
2	32
3	33
4,5	49
6	56
7	58
8	38
9	20
10	7
11	3
12	87
13	9
14	40
15	25
16	39
17	45
18	71
19	95
20	84
21	124
22	153
23	156
24	183
25	236
26	246
27	244
28	247
29	351
30	241
31	326
32	294
33	331
34	250
35	184
36	238
37	233

Table A.18: Partitioning Events for the Star Wars Trace, Contd.

State(s)	Num of Partitioning Events
38	235
39	236
40	177
41	179
42	191
43	186
44	122
45	170
46	128
47	70
48	68
49	39
50	174
51	254
52	276
53	166
54	126
55	98
56	40
57	25
58	14
59	10
60	8
61	7
62	17
63	23
64	22
65	19
66	10
67	13

Table A.19: Partitioning Events for the Star Wars Trace, Contd.

State(s)	Num of Partitioning Events
68	15
69	9
70	7
71	9
72	5
73	7
74	6
75	4
76	9
77	5
78	6
79	4
80,81	2
82	32
83	236
84	240
85	34
86,115	2
116	4
117	2

Table A.20: Partitioning Events for the Mean-change Trace

State(s)	Num of Partitioning Events
0	22
1	36
2	48
3	20

Table A.21: Partitioning Events for the Time-trend Trace

State(s)	Num of Partitioning Events
0	14
1	20

Table A.22: Partitioning Events for the Periodic Trace

State(s)	Num of Partitioning Events
0	2
1	2

Appendix B

PRIMER FOR BUILDING A MODEL WITH CHITRA

As mentioned in Section 2.3.2 CHITRA is a collection of small programs, following the UNIX model of a library of programs with simple functions that read standard input and write standard output. Data is passed between programs in a Canonical Format (CF) file. A CF file represents data as space separated columns of data augmented with header lines that contain a pound sign in column one. The first step in analyzing trace data using CHITRA is to convert the trace data from its native form into the *Chitra Specification Language* (CSL) form. CSL files contain a trace as a time-stamped sequence of states in textual form. The state can be a vector of data values whose corresponding data types are declared at top of a CSL file in a header statement. For a complete description of the CSL syntax see [Abrams et al., 1994]. The trace as a CSL file is read by the CHITRA assistant utility `a.o.csl` and converted into a CHITRA *ensemble*. The CHITRA extractor utility `e.all` renders an ensemble into a file in CF format. Once the trace is available as a file in CF format it can be input to any appropriate CHITRA utility. In the discussion that follows we will assume that all inputs are available as CF files and that all outputs are in CF format also. The rest of this appendix is divided into three sections that discuss the task of building the piecewise independent stochastic process model of a trace, the reduction of model size by reducing the number of states in the trace, and the reduction of model size by reducing the number of partitioning events. Each of the above tasks is accomplished by executing several CHITRA utilities as a UNIX command pipeline. All the individual Chitra utilities are described at the end of this appendix as `man` pages.

Construction of the Model

To initiate the construction of the piecewise independent stochastic process model we must have the trace available as a file in CF format. The Figure B.1 shows the sequence of utilities that must be executed serially to build the model from the trace. Briefly, the `g.reg` utility takes the trace as input and outputs the REG. The sequence of utilities `g.regSlope`, `a.x.columns`, and `g.regSlopePart` take the REG as input and use *slope-based* partitioning (Section 4.1) to render the set of partitioning events for each path in the REG. The `g.reg.fitPath` utility uses linear least squares regression to fit straight line segments to each path in the REG. The partitioning events that mark the beginning and end of each straight line segment are available from the output of g.regSlopePart. The g.reg.fitPath utility outputs the slope of each straight line segment that it fits. The `g.reg.Pm` utility takes the output of g.reg.fitPath as input and outputs the model as it is stored internally (as a file in CF format). Let us denote the entire sequence of utilities invoked to build the model as the *PISP pipeline* (piecewise independent stochastic process pipeline). Now we have two choices. We can use the utility `v.reg.pmf` to output the

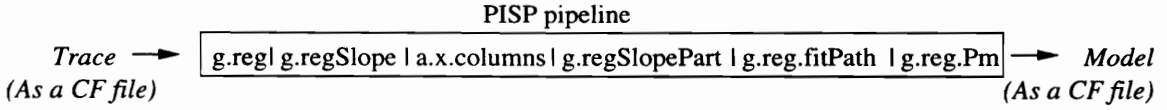


Figure B.1: Sequence of Chitra utilities that build a Piecewise Independent Stochastic Process Model

model as a joint probability mass function in tabular form (not CF). We can also generate a simulated synthetic trace from the model by invoking the utility `s.reg.gen`. Figure B.2 displays these two options.

Reducing the Number of States

To build our model with a reduced number of states we first invoke the `s.reg.filter` utility to remove a chosen (user specified) number of states. The input and output to

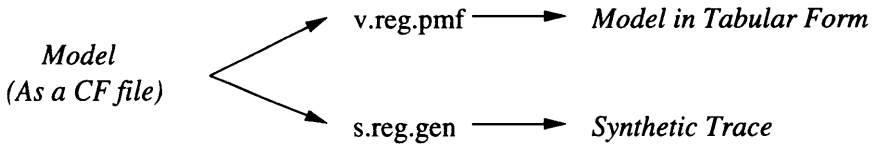


Figure B.2: Outputs from the Piecewise Independent Stochastic Process Model

(and from) s.reg.filter is a trace in CF format. The output from s.reg.filter is used as the input to the PISP pipeline. Consequently, we now have a model of the trace with reduced number of states. To generate a synthetic trace from our model we first use the model as input to the s.reg.gen utility. The trace that is output from s.reg.gen is then input to the `s.reg.noise_gen` utility which re-introduces the states, that were removed earlier, by using s.reg.filter, as random noise. Figure B.3 shows the sequence of utilities that must be invoked to reduce the number of states in our model of a trace.

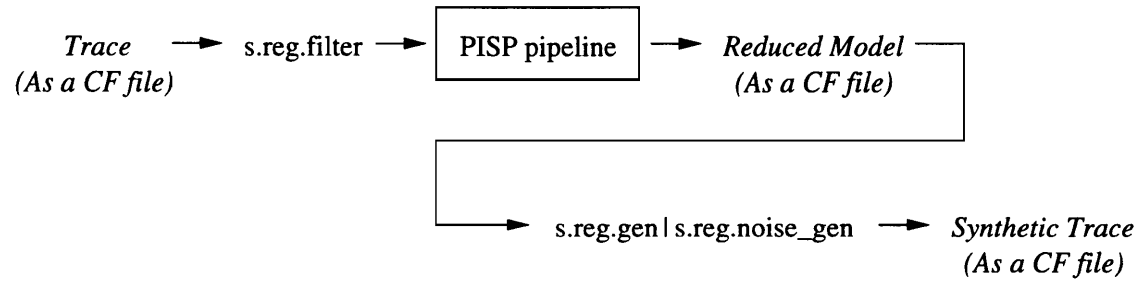


Figure B.3: Sequence of Chitra utilities that synthesize a trace with reduced number of states

Reducing the Number of Partitioning Events

Reducing the number of partitioning events is an iterative task (Section 5.1). Figure B.4 illustrates the pipeline of utilities that outlines this task. The first step in this task is to invoke the PISP pipeline on the trace and obtain the initial model. Next, we use the model as an input to the `g.regCompact` utility which generates a compact REG from the model. The output from g.regCompact then serves as the input to the `g.regRegressPart` utility

which uses the *incremental regression* partitioning (Section 5.1.2) to generate partitioning events from the compact REG. We feed the output from g.regRegressPart back to the PISP pipeline to obtain a new model (with less partitioning events than the original). In theory this process can be repeated until there is only a single partition left for each state. In practice, however, we must evaluate the reduced model for accuracy before initiating a new iteration of this pipeline.

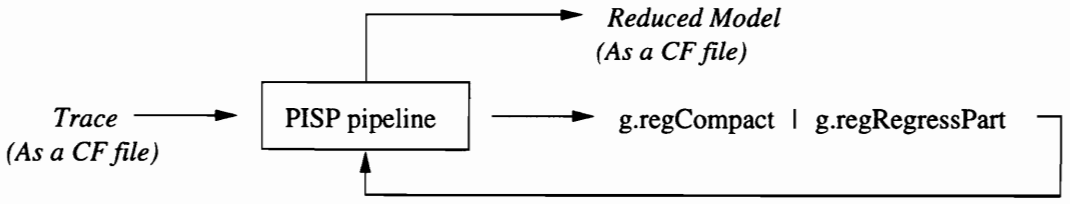


Figure B.4: Sequence of Chitra utilities that build a Piecewise Independent Stochastic Process Model with reduced number of partitioning events

NAME `g.reg`: Generate points for a rate evolution graph

SYNOPSIS `g.reg`

DESCRIPTION The g.reg command reads from standard input a CF file containing one object with 1-tuples of model states. (This implies that if the input CF file is created by `e.all`, the -suppressStateMap option must be given.) Let M denote the number of unique model states in the input CF file. It writes to standard output a CF file containing M objects, one per model state, containing 2-tuples listing (eventNum, numOccurrences). Plotting the tuples from one object yields one path in the rate evolution graph. Finally, the model state number to which object number i ($i = 0, 1, \ldots, M-1$) corresponds is written as a comment at the start of each object in the output CF file.

EXIT CODES

- 0 : Command terminated normally.
- 1 : Abnormal exit.

NAME `g.regSlope` - output slopes of REG paths

SYNOPSIS `g.regSlope`

DESCRIPTION g.regSlope is a command used in the pipeline to generate REG based models. This command reads the output from g.reg and produces a CF file of slopes as output. g.regSlope reads from standard input a CF file containing N objects. Each object contains 2-tuples that are (x,y) coordinates defining a path in 2 dimensional Cartesian space. g.regSlope writes to standard output a CF file containing N objects. Each object contains 2-tuples $(x, m(x, y))$.

EXIT CODES

- 0 : Command terminated normally.
- 1 : Abnormal exit.

NAME `a.x.columns` – assist by switching output order of CF tuples

SYNOPSIS `a.x.columns`

DESCRIPTION a.x.columns is a command used in the pipeline to generate REG based piecewise independent stochastic process models. This command reads the output from `g.regSlope` and produces a CF file of switched slopes as output. a.x.columns reads from standard input a CF file containing N objects. Each object contains 2-tuples such as (x, y). a.x.columns outputs a CF file of N objects with tuple order switched to (y, x).

EXIT CODES

- 0 : Command terminated normally.
- 1 : Abnormal exit.

NAME `g.regSlopePart` - Partitions objects in the output from `g.reg` into segments

SYNOPSIS `g.regSlopePart [-runlength <float>] [-threshold <float>] -megfile <string>`

DESCRIPTION g.regSlopePart is a command used in the pipeline to generate REG based piecewise independent stochastic process models. This command reads the output from `g.regSlope` (piped through `a.x.columns`) from standard input. It also reads the output from `g.reg` which is available through the `megfile` option.

It partitions each object in the output from g.reg into segments. Within each segment the slope of the points in the segment remains the same (within a `threshold`). g.regSlopePart outputs a CF file which has the same total number of tuples as the file output from g.reg. However, the CF file output from g.regSlopePart has many more objects because each object in the CF file output from g.reg has been partitioned into several (segments) objects.

The option `runlength` (default value 3) puts a lower bound on the number of points within a segment. The option `threshold` (default value 0.1) serves as cutoff value that determines the amount of difference in slope between consecutive points that mark the end of one segment and the beginning of the next.

The option `megfile` provides a way to read in the output from g.reg. Unlike the other two options this parameter is essential. The command will NOT work without it.

EXIT CODES

- 0 : Command terminated normally.
- 1 : Abnormal exit.

NAME `g.reg.fitPath` - fit a straight line to a REG path using linear least-squares data fitting.

SYNOPSIS `g.reg.fitPath [ -tolerance <float> ]`

DESCRIPTION `g.reg.fitPath` reads from standard input a CF file containing N objects. Each object contains 2-tuples that are (x, y) coordinates defining a path in 2 dimensional Cartesian space. `g.reg.fitPath` fits a straight line to each object in the input CF file. g.reg.fitPath writes to standard output a CF file containing N objects, one per input CF file object. Each object contains the same points as the input CF file object. However, the output CF object will contain attributes defining the straight line fitting the curve, along with information on the quality of fit. The attributes for object i are named: `slope.i`, `yIntersept.i`, and `rank.i`. The fitted line for object i is given by the equation

```
Y = (slope.i)*X + (yIntercept.i)
```

The value `rank.i` reports the number of uniquely determined parameters. If this number is less than the number of parameters being determined (two in our case) the least squares problem is degenerate. A degenerate problem also indicates that the model being fit is inappropriate. The option `tolerance` is usually set to the smallest positive real number that can be represented on the target machine. By default this is set to $1.0E-6$.

EXIT CODES

- 0 : Command terminated normally.
- 1 : Degenerate solution check input model.
- 2 : Abnormal exit.

NAME `g.reg.Pm` - outputs each object with its probability

SYNOPSIS `g.reg.Pm -<zero|megslope>`

DESCRIPTION g.reg.Pm is a command used in the pipeline to generate REG based piecewise independent stochastic process models. g.reg.Pm reads from standard input a CF file output from `g.reg.fitPath`.

The input CF file contains N objects and the following attributes for each object i: `slope.i, yIntersept.i, rank.i`. Each object i in the input CF file is a sequence of points (x, y) which have been fitted with a line with slope `slope.i` and y-axis intercept `yIntercept.i`.

g.reg.Pm writes CF file to the standard output. This CF file contains tuples such as (b, s) one for each object i in the input CF file. Here b is the x coordinate of the first tuple in object i, and s is the slope for object i (slope.i).

If the input file contains fitted paths such that each model state is represented by many straight line segments then the options `zero|megslope` play a significant role. Straight line segments fitting the curve (for a model state) in a REG are often separated by areas for which we have no data. The `zero` option assigns a 0 slope to such areas. The `megslope` option assigns a slope of the line XY for such areas, where X is the last point in the previous fitted segment and Y is the first point in the next fitted segment. NOTE: If areas with no data points are present they are added as segments to the g.reg.Pm output with slopes calculated as described in the above paragraph.

EXIT CODES

- 0 : Command terminated normally.
- 1 : Abnormal exit.

NAME `s.reg.gen` – create a synthesized trace from the REG based piecewise independent stochastic process model

SYNOPSIS `s.reg.gen -s<1-99> -t<1|2> -xpt<left|right|max|avg>`

DESCRIPTION `s.reg.gen` takes as input a CF file generated by the `g.reg.Pm` utility. It outputs a synthesized trace. The options for this utility are described below:

- `s` : Seed index in the range $< 1-99 >$ for the random number generator. The random number generator uses an array of seeds. A seed index is an index into this seed array.
- `t` : Number of tuples to be output. If $t = 1$ is then the output is a sequence of states, else the output is a sequence of states with corresponding event numbers.
- `xpt` : Transition point slope. A transition point marks the end of the current partition and beginning of the next one. The following options determine how the slope of a transition point is computed:
  - `right` : the slope of the transition point is the fitted slope of the next partition.
  - `left` : the slope of the transition point is the fitted slope of the current partition.
  - `avg` : the slope of the transition point is the average of the fitted slope of the current partition and the fitted slope of the next partition.
  - `max` : the slope of the transition point is the maximum of the fitted slope of the current partition and the fitted slope of the next partition.

EXIT CODES

- 0 : Command terminated normally.
- 1 : Abnormal exit.

NAME `v.reg.pmf` - outputs the joint probability mass function in tabular format.

SYNOPSIS `v.reg.pmf -xpt<left|right|max|avg> [-normalize] [-numcols<integer>]`

DESCRIPTION This program generates the joint probability mass function from the output of `g.reg.Pm`. The joint probability mass function is output in a tabular (human-readable) form. The options for this utility are described below:

- `xpt` : Transition point slope. A transition point marks the end of the current partition and beginning of the next one. The following options determine how the slope of a transition point is computed:
  - `right` : the slope of the transition point is the fitted slope of the next partition.
  - `left` : the slope of the transition point is the fitted slope of the current partition.
  - `avg` : the slope of the transition point is the average of the fitted slope of the current partition and the fitted slope of the next partition.
  - `max` : the slope of the transition point is the maximum of the fitted slope of the current partition and the fitted slope of the next partition.
- `normalize` : If present this option checks and forces the normalization of the probability mass functions such that the probabilities in a row add up to 1. This option is absent by default.
- `numcols` : This option specifies the number of columns to be printed per row. The default value is 5.

NAME `s.reg.filter` - takes the trace (in CF format) as input and outputs a trace with user specified number of states removed.

SYNOPSIS `s.reg.filter -l<integer> [-r<1-99>] -f<string>`

DESCRIPTION `s.reg.filter` reads the trace CF file from standard input. Let M be the maximum state in the trace. `s.reg.filter` converts the state space in the trace to a new state space given by the set 0.(M/l),..,(N-1)(M/l) where N is such that (N-1)(M/l) ¡ M. The option `l` determines the cardinality of the new state space. Next, every state in the trace is mapped to one of the two states in the new state space which form the smallest interval containing that state. The choice of the succeeding/preceding state is made based on a random variate chosen from a $U(0,1)$ distribution chosen with a probability = 0.5. The random seed option `r`, which is not a required parameter, is used to generate the $U(0,1)$ random variate. The option `f` is used to specify the *filename* of the file which contains information about the percentage of states in the trace that were actually filtered out.

EXIT CODES

- 0 : Command terminated normally.
- 1 : Abnormal exit.

NAME `s.reg.noise_gen` – add Normal or Uniform noise to a trace

SYNOPSIS `s.reg.noise_gen -s<1-99> -l<integer> -<n|u>`

DESCRIPTION The utility s.reg.noise_gen takes as input a CF file generated by the `s.reg.gen` utility. s.reg.noise_gen adds a normal (or uniform) noise to the trace synthesized from `s.reg.gen`. s.reg.noise_gen is used to generate a synthetic trace from a piecewise independent stochastic process model whose state space has been reduced by using the `s.reg.filter` utility. s.reg.noise_gen takes the following options:

- `s` : Seed index in the range $< 1-99 >$ for the random number generator. The random number generator uses an array of seeds. A seed index is an index into this seed array.
- `l` : The number of states filtered by the utility `s.reg.filter`. This option corresponds to the option by the same name in `s.reg.filter`.
- `n` : Generate noise made up of IID Normal random variates.
- `u` : Generate noise made up of IID Uniform random variates.

EXIT CODES

- 0 : Command terminated normally.
- 1 : Abnormal exit.

NAME g.regCompact - Generate and output a compact REG from the REG and model (output from `g.reg.Pm`) of a trace.

SYNOPSIS `g.regCompact -megfile <string>`

DESCRIPTION This utility constructs a compact REG (in CF format) that consists of ⟨ *event number, number of occurrences* ⟩ tuples, such that only those *event numbers* that appear in the model (output from `g.reg.Pm`) are chosen. The corresponding *number of occurrences* are determined by a look up to the original REG which is made available through the option `megfile`.

EXIT CODES

- 0 : Command terminated normally.
- 1 : Abnormal exit.

NAME `g.regRegressPart` - Partitions objects in the output from g.reg into segments by using incremental regression.

SYNOPSIS `g.regRegressPart [-z-or-tteststatistic <float>] [-runlength <integer>] -megfile <string>`

DESCRIPTION g.regRegressPart is a command used in the pipeline to generate piecewise independent stochastic process models of trace data. This command reads the output from `g.reg` through standard input. The output from `g.reg` is also read through the `megfile` option. The `megfile` option must be provided. The utility will not execute without it.

It partitions each object in the output from `g.reg` into segments. The algorithm keeps points that fit the same regression line within one segment. Every new point slope is tested for membership in the current segment on the basis of a prediction interval derived from the current segment. The option `-z-or-t-statistic`, 1.96 by default, determines the width of the prediction interval. If the new point is within the prediction interval the current segment continues, otherwise the new point starts a new segment. `g.regRegressPart` outputs a CF file which has the same total number of tuples as the `g.reg` output. However, the CF file output from `g.regRegressPart` has many more objects because each object in the CF file output from `g.reg` has been partitioned into several (segments) objects.

The option `runlength` (default value 3) puts a lower bound on the number of points within a segment.

The option megfile provides a way to read in the output from g.reg. Unlike the other two options this parameter is essential. The command will NOT work without it.

EXIT CODES

- 0 : Command terminated normally.
- 1 : Abnormal exit.

Appendix C

TRANSFORMING PACKET SIZE TRACES

Packet size traces such as the Ingalls trace (Section 6.1) have a large number of unique states. Direct modeling of such traces leads to a state space explosion. In this appendix we describe a technique to transform packet size traces that reduces state space.

Consider a packet size trace, $\{X_t\}$, such that the maximum packet size in the trace is a n bit integer. We divide each packet size, X_t, into k and l bit integers A_t and B_t respectively such that

$$n = k + l \qquad \text{and,}$$

$$X_t = A_t \textit{ bitwise-or } B_t.$$

Let us assume that A_t is the k most significant bits of X_t and B_t is the l least significant bits of X_t.

Denote as l^* the largest value of l $(1 \leq l \leq n)$ such that $\{B_t\}$ is a stationary time series. We can use one of the many stationary stochastic process models to model $\{B_t\}$. The problem is therefore reduced to modeling the transient time series $\{A_t\}$ where each A_t is the $(n - l^*)$ most significant bits of the corresponding X_t. Finally, a hierarchical model of $\{X_t\}$ can be constructed from the stationary model of $\{B_t\}$ and transient model of $\{A_t\}$. The procedure of building such hierarchical models is beyond the scope of this document. It is discussed in detail in [Heyman et al., 1992] and [Vazirani, 1996].

In this paper we concern ourselves with building transient models of the transient trace $\{A_t\}$. The Star Wars Trace, Ingalls Trace, and ACM Trace (Section 6.1) are the transformed transient traces that are the 9, 9, and 5 most significant bits of their original counterparts, respectively.

Appendix D

TRANSFORMING A STATE VECTOR

Consider a trace as the time series $\{X_t\}$ with state space **S** such that each member of **S** is an ordered k tuple. Let n denote the cardinality of **S**. Then, **S** can be described as the set

$$\{(x_0^0, x_0^1, \ldots, x_0^{k-1}), (x_1^0, x_1^1, \ldots, x_1^{k-1}), \ldots, (x_{n-1}^0, x_{n-1}^1, \ldots, x_{n-1}^{k-1})\}.$$

We map each member of **S** to a unique non-negative integer, consequently deriving a transformed state space **S**$'$. Finally, $\{X_t\}$ is modified to $\{X_t'\}$ such that every X_t, in **S**, maps to the corresponding $\{X_t'\}$, in **S**$'$.

We applied this transform to the Dining Philosophers trace (Section 6.1) thus reducing its state space from a set of 5 tuple vectors to a set of non-negative integers.

VITA

Anup Mathur was born in Kanpur, India. He received a High School diploma from Colvin Taluqdar's College in Lucknow, India and a Bachelor's degree from St. Stephens College in Delhi, India.

After the completion of his undergraduate studies, Anup joined the graduate program in Computer Science at the University of Georgia in Athens, Georgia. After successful completion of his Master's degree, Anup joined the doctoral program in Computer Science at Virginia Polytechnic Institute and State University located in scenic Blacksburg, Virginia. During his tenure as a doctoral candidate, Anup was also employed as a software engineer at INCODE, a software development company in Blacksburg, Virginia.

After receiving the Ph.D degree, Anup will work as a senior technical member at A&T systems in Silver Spring, Maryland.

Anup Mathur