CHITRA94: A Tool to Dynamically Characterize Ensembles of Traces for Input Data Modeling and Output Analysis*

Marc Abrams   Alan Batongbacal   Randy Ribler
Devendra Vazirani

June 22, 1994

Computer Science Department TR 94-21
Systems Research Center SRC-94-009

Virginia Polytechnic Institute and State University
Department of Computer Science
Blacksburg, VA 24061-0106
(703) 231-8457   abrams@vt.edu

Abstract

CHITRA94 is the third generation of a comprehensive system to visualize, transform, statistically analyze, and model ensembles of trace data and validate the resultant models. The tool is useful for deriving input data models from trace data as well as for analysis of trace data output by a simulation. The tool uses a stochastic (possibly non-Markovian) process as its fundamental modeling component. The tool is unique in several respects. First, it focuses on the dynamic characterization of systems. Consequently it includes tests for homogeneity of ensembles, stationarity of individual traces, and the ability to partition ensembles into transient and stationary pieces. Second, CHITRA94 is a *scalable* performance tool: its methods are designed to work with an arbitrary number of traces in an ensemble so that a user can examine the variability of system behaviors across different traces (representing different observation periods, different system configurations, etc.) To analyze multiple traces, the user can either (1) map the traces to a model that characterizes the dynamic behavior or (2) use a novel visualization, the *mass evolution graph*, that shows the probability mass distribution evolution of one or more traces. A mass evolution graph is easily constructed from trace data, and the derivatives of the paths it contains yield an estimate of probability mass as a function of time. Third, it analyzes categorical, not just numerical, time series data. Categorical data arises naturally in many computer and communication system modeling problems. Fourth, it includes a library of transforms that reduce the state space of the stochastic process generated. Fifth, CHITRA94 is implemented as a user extensible collection of small programs, organized as a library, which allows the user to write new library modules that use existing modules to automate analysis procedures. Instead, CHITRA94 may be invoked from command line, under a GUI, or integrated with another tool (e.g., a simulation model development or CASE tool). The use of CHITRA94 is illustrated on a variety of trace data, and the extensibility is illustrated on the problem of partitioning an ensemble of 60 traces of compressed entertainment video into mutually exclusive, exhaustive, and homogeneous subsets from which a hierarchical workload model is derived.

CR categories: I.6.5 [Simulation and Modeling]: Model Development, 1.6.6 [Simulation and Modeling]: Simulation Output Analysis], C.4 [Performance of Systems]: Modeling techniques

Keywords: Input data modeling, workload modeling, simulation output analysis, visualization, traces

---

[*] This work was supported in part by National Science Foundation grants NCF-9211342 and CDA-9121999. Funding for this work was also provided by the NSF to SUCCEED (Cooperative Agreement No. EID-9109853). SUCCEED is a coalition of eight schools and colleges working to enhance engineering education for the twenty-first century.

1 Introduction

Chitra[1] is an experimental lab for investigating the dynamic behavior of trace data [1, 2]. This paper describes Chitra94, the third generation of a tool that couples visualization and statistical analysis of trace data. Chitra94 is comprehensive in its approach to analyzing trace data:

1. Chitra94 does not analyze just a single trace file, but *multiple* traces, termed an *ensemble*, simultaneously. This is essential because one trace provides no information on how representative the observation is of the range of possible behaviors that the observed system can manifest. Chitra94 provides methods to partition an ensemble into mutually exclusive and exhaustive subsets, termed *sub-ensembles*, that each represent the "same" behavior, or in statistical terms, are homogeneous.

2. Chitra94 also addresses the *scale problem* in trace analysis: how to design a performance analysis tool that can simultaneously analyze an arbitrary number of traces. In contrast, existing trace visualization systems, while they are a breakthrough compared to looking at textual listings of trace data, are *not* scalable. One can perhaps open windows on a computer monitor to display two, three, or four traces and "eyeball" them for similarities, but how does one visualize an ensemble of a hundred traces? Chitra94 has two solutions.

   - The first is a novel method of visualizing trace data, called the *mass evolution graph*, that plots the time evolution of each state in a trace as a separate path in a way that shows the probability mass distribution evolution of the trace. The mass evolution distinguishes regions of stationary and non-stationary time intervals, and the way states are plotted distinguishes periodic from randomly occurring states. Data from an arbitrary number of traces can be superimposed to identify traces that are "outliers" or the existence of more than one "mode" of behavior in the observed system. In addition, the graph can be used with categorical data that often arises in workload modeling, not just numerical data.

   - The second solution is to build models. The user may visualize representative traces (e.g., one trace from each homogeneous sub-ensemble), and specify how Chitra94 is to map the ensemble data to a set of *model states*. Chitra94 will then estimate a stochastic process underlying each set of stationary sub-traces in each sub-ensemble. With this approach, the model, not just visualization of raw traces, characterizes dynamic behavior.

---

[1] One of Chitra's meanings is a Sanskrit word for beautiful or pleasing pictures and drawings.

3. State space explosion, a potential Achilles' heel of stochastic process models, is controlled in CHITRA94 by providing a variety of transforms that simplify, aggregate, and filter an entire ensemble simultaneously to reduce the state space size of the resultant model.

4. Building models necessitates validation. CHITRA94 contains modules to generate synthetic traces, to either use in simulations or compare to the original trace data as part of validation. Some simple validation tests are provided.

Based on experience gained from the earlier two generations of Chitra, the architecture of CHITRA94 consists of a set of small programs that communicate through a simple, standard file format. This architecture permits development of a library of performance analysis modules, and makes CHITRA94 user extensible, so that a performance analyst can incorporate new analysis procedures and models. This paper presents the modules currently in CHITRA94, which are continually being refined and augmented.

CHITRA94 has three uses. First, it can be used to derive a non-executable, artificial workload model (using Ferrari, Serazzi and Zeigner's terminology [5, p. 53]) from an ensemble of trace data. The model, based on stochastic processes, is typically used to drive a simulation of a system, or to solve an analytic system model. For example, CHITRA94 can derive a workload model of the connections established and released between hosts on a particular network, and the traffic characteristics over each connection. Second, CHITRA94 can analyze an ensemble of traces representing the output from multiple simulation replications to explore the dynamic behavior of the system being simulated. For example, the design of congestion control and routing algorithms for communication networks is often assisted by visualization of the dynamics arising in the system during simulation; see Zhang, Shenker, and Clark [18] for an example. A third use, illustrated by Abrams, Doraswamy, and Mathur [1], is to identify and diagnose performance and correctness problems in computer and communication protocol software from the model that CHITRA94 generates from traces.

1.1 Issues in Model Building

Chitra is organized around the analyst's need to address five issues: (1) Is the ensemble homogeneous, or must it be partitioned into a set of mutually exclusive, exhaustive, and homogeneous subsets (termed *sub-ensembles*)? (2) Is a given trace stationary, or must it be partitioned into transient and stationary sub-traces? (3) What type of model is appropriate? (4) How can state space explosion be avoided? (5) How can the model be validated? CHITRA94's current approach to these issues is explained in §3.

1.2 Problem Statement

Several definitions are required to make the problem statement precise. First, a system is described by a set Σ of *system states*. A system state is a vector of at least one component that describes the system over some interval of time. For example, the system state in a communication network might be an ordered pair *(host-name, length of packet last injected)*. The choice of state is governed by the performance study objective.

Informally, a trace consists of a sequence of ordered pairs representing an entrance time and a system state. Let Z denote the set of non-negative integers. Formally, a *trace of length* L is a sequence of ordered pairs (E_0, S_0), (E_1, S_1), $\ldots$, (E_{L-1}, S_{L-1}), where $\langle \forall i : 0 \leq i < L :: E_i \in Z \wedge \langle \exists s : s \in \Sigma :: S_i = s \rangle \rangle$, and $\langle \forall i : 0 < i < L :: E_{i-1} < E_i \rangle$. Each pair of successive states (S_k, S_{k+1}) (for $0 \leq k < L-1$) is a *transition*. An *event* causes a transition; the *event number* of S_i is i. An *ensemble* is a set of traces with equal length system state vectors, each representing a different observation of a system. An ensemble *member* is one trace in the ensemble; each ensemble member is identified by a unique, small non-negative integer.

Problem statement: Chitra94 provides tools to help solve the following problem: Given an ensemble, find one or more empirical models that parsimoniously fit the ensemble with sufficient closeness.

1.3 Numeric Versus Categorical Time Series Data

The statistical problem that is equivalent to the problem statement is to analyze and model time-series data. However, computer and communication system traces are unlike traditional time series data in one important way: the value of a system state component may not be numerical, but categorical. *Numerical* data has as its domain a set whose members are totally ordered, such as integers or reals. *Categorical* data is the complement; a partial order may exist among the domain values, but a total ordering does not. Computer and communication systems may require system states composed of a mixture of categorical and numerical components (e.g., the ordered pair (host name, packet length) from §1.2). The packet length is numerical, but the host name (e.g., *daphne.cs.vt.edu*, *alanlb.bevc.blacksburg.va.usa*) is categorical: *daphne* is not "less than" or "greater than" *alanlb.* Certain statistical literature exists to analyze categorical data, such as the CHAID method used in Chitra94 (see §3.5), but very little exists on categorical time series.

Given the abundance of techniques for analyzing numerical data, one is tempted to somehow map the categories to numbers. For example, one might map host names to their numerical network addresses. However this is dangerous because network addresses are assigned by organizations, and the relative magnitude of the addresses as number is arbitrary. Yet one might unknowingly use a numerical method on the data and come to a false conclusion. To illustrate, consider a simulation of a communication network that uses a synchronous adaptive shortest path routing algorithm. The algorithm has been shown theoretically to generate oscillations [3]. The simulation generates a finite sequence of *routes* used in the network. Routes are not numeric quantities because there no ordering relationship exists; thus routes are categorical. Suppose we wish to examine to the sequence to see if there exists any oscillations in routes. To do this, we define a function f to map the set of routes in the sequence to small non-negative integers. The function essentially maps categorical data to numeric values. First, one would like to visualize trace data. The most common way is as a Gantt chart: plotting state (in this case the route) on the y-axis and time, or event number, on the x-axis, as shown in Fig. 1(a). The function used as f in Fig. 1 maps the first route appearing in the trace to $y = 0$, the second to $y = 1$, and so on. The graph appears to be rather unpredictable, apparently without oscillations. One way to verify this is to examine the autocorrelation function (Fig. 1(b)). There is essentially no correlation of the function after it is shifted by at least 20 events. Thus one concludes that there is no oscillation present. Now consider Fig. 2. This shows the same trace file, but with a different definition of f. The graphs clearly show oscillatory behavior! The point is that if one visualizes categorical data, the mapping f chosen (e.g., Fig. 1 versus Fig. 2) might obscure an insight into the dynamic behavior (e.g., oscillations).

The ideal way to avoid such an ambiguity is to visualize the time evolution of categorical data *without* defining a mapping f. CHITRA94 provides such views (§3.2.2). In addition, CHITRA94 also flags trace data as numerical, categorical, or mixed, so that the analyst can be warned if a numerical analysis method (e.g., visualization using a Gantt chart or autocorrelation) is applied to categorical data.

1.4 Related Work

Little of the workload modeling literature (e.g., [16, 5, 9]) is concerned with representing the dynamic or time-dependent behavior of workloads. In recent years visualization tools have provided a new way to explore dynamic behavior in systems. General purpose commercial scientific visualization tools, such as PV-WAVE and AVS, can be used. In addition tools specific to computer and communication systems have been designed. The closest to our work are Traceview [11] and Pablo [15]. Pablo contains a library of Motif widgets to visualize and analyze data; furthermore a graphical interface is used to graphically build a data flow graph. These characteristics motivated us to design CHITRA94 as a set of small commands that are composed into an analysis procedure using a shell script.

These tools have two limitations. First, they work on *one* data set (trace) at a time, rather than an ensemble. Second, they are limited to traces with numerical system state vector components. CHITRA94's attempt to handle both ensembles and categorical data as first class problems is unique. However, one system of note is IPS-2 [12], which was designed for performance evaluation of parallel and distributed systems. IPS-2 constructs an activity graph from a single trace (similar to CHITRA94's construction of a state transition matrix in §3.5.1). In addition, IPS-2 contains an algorithm to partition a trace into phases, providing one solution for issue (2) in §1.1.

For simulation output analysis, simulation animation has also provided a new way to gain insight into dynamic behavior. In contrast to visualization, animation can work quite well for certain categorical data (e.g., animating the establishment of a connection in a network topology graph), if the system can be laid out naturally in two or three dimensions. A variety of commercial simulation tools include animation capabilities (e.g., Wolverine Software's Proof Animation^{TM}, Pritsker Corp.'s SLAMSYSTEM, and CACI's SIMGRAPHICS).

Certain functions have not been incorporated into CHITRA94, such as fitting distributions to data as part of workload modeling. This is because CHITRA94 is designed to be used in conjunction with other tools of the analyst's choice. It is straightforward to export CHITRA94's standard file format (described in §2.2) to other tools, such as UniFit II [17] for fitting distributions.

The remainder of the paper is organized as follows. §2 gives an overview of CHITRA94's commands, and describes the systems that influenced its architecture. §3 is organized around the major tasks that a performance analyst must perform, and describes how CHITRA94 supports each of them.

2 Overview of 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 [13]. 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.

2.1 CHITRA94's Approach

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 CHITRA94's 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 heaer 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 [15], 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 (see §3.8 for an illustration). 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.

2.2 CHITRA94 Commands

Fig. 3 illustrates CHITRA94's 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 (see §3.6).

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.

3 Analysis Tasks

This section describes the tasks involved in analyzing and modeling ensembles from the performance analyst's perspective.

3.1 Opening Ensembles of Traces

Analysis of an ensemble instance with Chitra94 starts by *opening* an ensemble. A user does this by typing the *a.o.Y* (assist by opening format Y trace files) command, where Y is the name of some trace file format. (Chitra94 allows a library of parsers that convert various trace file formats to CF format.)

When a user opens an ensemble, Chitra94 loads an *instance* of the trace files in the ensemble. Each ensemble instance $\mathcal{E}$ will have multiple *versions*, denoted $\mathcal{E}_0, \mathcal{E}_1, \ldots, \mathcal{E}_v, \ldots$. Version $\mathcal{E}_0$ is created when the ensemble is first opened; successive versions are created each time an ensemble is transformed (§3.6). A transform modifies the state space of the ensemble; for example it might aggregate states into a composite state by some rule. Version $\mathcal{E}_0$ is identical to the original disk copy; other versions modify the loaded instance, not the original disk copy. A user can open multiple instances of the same ensemble, permitting each instance to be modified independently. This is useful to allow an analyst to explore the effects of different transform sequences.

Let $\Sigma(\mathcal{E}_v)$ denote the set of all system states that appear in all traces in ensemble version $\mathcal{E}_v$. Σ is a function of v because applying transforms may change the state space. Chitra94 views each system state S_i in a trace file as an experiment outcome. For each ensemble instance, Chitra94 defines a random variable, $X_{\mathcal{E}_v}$, mapping all outcomes – the elements of $\Sigma(\mathcal{E}_v)$ – to non-negative integers, called *model states*. (An example of function X is function f in §1.3, illustrated in Figs. 1(a) and 2(a).) Let $\mathcal{M}(\mathcal{E}_v)$ denote the range of $X_{\mathcal{E}_v}$. Letting the domain of t be the interval bounded by the first and last timestamp in a particular trace file in $\mathcal{E}_v$, then $X_{\mathcal{E}_v}(t)$ represents the trace file as the realization of a stochastic process. $X_{\mathcal{E}_v}$ is subscripted by $\mathcal{E}_v$ because Chitra94 modifies the mapping whenever a new ensemble version is created. Subscripts are dropped when the ensemble and version are clear from context. When an ensemble is originally opened, Chitra94 generates an *initial* mapping $X_{\mathcal{E}_0}$, according to information provided on the *a.o.Y* command.

Chitra94 internally represents a trace file (E_0, S_0), (E_1, S_1), $\ldots$, (E_{L-1}, S_{L-1}) as an occupancy time sequence $E_1 - E_0, E_2 - E_1, \ldots, E_{L-1} - E_{L-2}$ and a model state sequence $M_0, M_1, \ldots, M_{L-2}$, where $M_i = X_{\mathcal{E}_0}(S_i)$. Note that the last system state, S_{L-1}, is dropped because its occupancy time cannot be computed.

Extracting data from an ensemble: The *e.** commands in Fig. 3 extract some or all of the data in the current ensemble version. The key ones are:

Extract all (e.all): For each ensemble member, extract (model state, time) pairs corresponding to all (S_i, E_i) pairs in the original trace. (Typically used for viewers or modelers, where one wants to see or model all trace data.)

Extract a random sample (e.rSample): The result differs from the previous case in that a random sample of the (S_i, E_i) pairs are used. (Typically used to reduce response time of analysis commands when a random sample yields results close to the full data.)

Extract a constant time increment sample (e.cSample): For each ensemble member, extract a sequence of model states representing the state of the system every Δt time units, where Δt is specified as a command argument. (Necessary for computing the FFT and autocorrelation views in the time domain.)

3.2 Getting a "Feel" for the Raw Trace Data

Before attempting to model, summarize with statistics, or statistically test ensemble data, it is useful to view the raw data through some form of visualization. Visualization commands that use *gnuplot* can direct their output either to a window on a monitor or to a file in various printer languages, including monochrome or color encapsulated postscript files.

3.2.1 Numerical Data Views

The three numerical data views in Chitra94 are conventional: *v.gantt*, view a Gantt chart, which plots model state versus time or event number (e.g., Figs. 1(a) and 2(a)); *v.fftPer*, view an FFT-based periodogram, illustrated in [1, Fig. 5]; and *v.autoCor*, view an autocorrelation graph (e.g., in Figs. 1(b) and 2(b)).

3.2.2 Categorical Data Views

Chitra94 provides three novel views suitable for categorical (as well as numerical) data: *v.meg* (view a mass evolution graph), *v.catPer,* (view a categorical periodogram), and *v.cCatPer* (view a cumulative categorical periodogram). These visualizations are based on two principles. First, the visualization method should *not* artificially map the categories to a spatial ordering. Doing so might distort a relationship that exists, as illustrated in Figs. 1 and 2. Second, the visualization should map model states with properities of interest (e.g., periodicity or known data with well known probability distributions) to simple graphical features, such as a lines. This of course is an old idea, having been used in probability plots.

Mass evolution graph (v.meg): The mass evolution graph, like the Gantt chart, contains time or event number as the abscissa. The ordinate represents the number of *occurrences* so far in the trace of the model state to which a point on the abscissa corresponds. For example, if $S_{191} = 69$, and model state 69 has occurred 14 times in ensemble member S_0, S_1, ..., S_{191}, then the point (191,14) is plotted. In general, let

$g : \Sigma \rightarrow Z$ map state $S_i = s$ (where $s \in \Sigma$) to the number of times that s occurs in $S_0, \ldots, S_i$. Then state sequence $S_0, \ldots, S_{L-1}$ is mapped to point set $\mathcal{P} = \{(0, g(S_0)), \ldots, (L-1, g(S_{L-1}))\}$.

Let n be the number of unique model states in the ensemble member; thus $n = ||\mathcal{M}(\mathcal{E})||$. A set of n paths is drawn through these points, where as follows. Consider any model state $s \in \mathcal{M}(\mathcal{E})$. One path connects all points $(k, y) \in \mathcal{P}$ satisfying $y = s$. A different color can optionally be used to plot each path and its points.

What is the physical interpretation of the n paths, and why are they useful to visualize? Let $\forall i$, $i \in \mathcal{M}(\mathcal{E})$, $p_i(t)$ denote the probability of the system being in model state i at time t. *The derivative of the path corresponding to model state i in a mass evolution graph is an estimate of the time dependent mass function* $p_i(t)$. Thus if one draws a line vertical line $x = t$ (if the x-axis represents time), then the points of intersection of this line with the n paths yields the *probability mass function at time t*. In addition, stationary behavior is easy to identify: by definition, the probability mass function is independent of time in a stationary period, and thus its integral must be a straight line in the mass evolution graph. Therefore *time intervals in the mass evolution graph in which all paths are straight lines represents stationary behavior.*

Figs. 4 to 7 illustrates mass evolution graphs corresponding to, respectively, (a) the cyclic model state sequence 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3; (b) a model state sequence containing states 0, 1 in a uniform distribution; (c) the discrete distribution $p_0 = 1/2, p_1 = 1/4, p_2 = p_3 = 1/8$; and (d) a model state sequence in which states 0 and 1 are periodic and states 2 through 7 are aperiodic: 0, 2, 1, 3, 0, 4, 1, 5, 0, 6, 1, 2, 0, 2, 1, 7, 0, 7, 1, 4, 0, 5, 1, 6, 0, 3, 1, 8, 0, 8, 1, 2. The fact that probability mass is the derivative of the paths in a mass evolution graph is illustrated by Figs. 4 to 7. For example, in Figs. 4 and 5 all four states have slope 1/4, which is the mass in the uniform distribution. In Fig. 6 states 0, 1, 2, and 3 have slopes 1/2, 1/4, 1/8, and 1/8, corresponding to $p_0 = 1/2, p_1 = 1/4, p_2 = p_3 = 1/8$.

The mass evolution graph also distinguishes periodic and aperiodic behavior: parallel paths (e.g., Fig. 4) represent cyclic states. A periodic state with constant period appears as a straight, diagonal line. The slope equals the frequency. For example, the four lines in Fig. 4 have slope 1/4, and each state in the sequence has period 4.

Finally, because the line $x = j$ (where j is an event number) yields the probability mass distribution at event j, the "closer" the path is to the line $y = x$ (respectively, $y = 0$), the more frequently (respectively, infrequently) the state occurs. For example, Fig. 7 visually distinguishes periodic states with larger mass from aperiodic states with smaller mass.

Figs. 8 and 9 illustrates the use of the mass evolution graph on two traces collected from real systems. Fig 8 shows a trace of transactions flow through a mainframe in a production system. Each model state, and hence each path in Fig 8, represents a particular resource requested or freed or an action performed by the transaction. The trace is from the CICS auxiliary trace facility on an IBM 3090. The particular meaning of each state is immaterial to the example; what is important is that the trace shows two periods of stationary behavior with a transition at about event 14,000. The two phases are evident because each path consists of either one or two line segments that share a common point at about event 14,000. Thus it is appropriate to partition the trace and derive a stationary model for each partition (e.g., the semi-Markov model of §3.5.1). Fig. 9 illustrates a trace of a program solving the Dijkstra's dining philosophers problem with five philosophers on a shared memory multiprocessor (see [1]) for details. This trace illustrates two stationary phases (intervals [500,4400] and [4400,5100]), transient periods before and after these intervals, and periodic behavior (e.g., the paths of the nine states with the greatest probability mass never cross each other).

A mass evolution graph may also be used to plot multiple members of an ensemble by merely superimposing the plots of the individual members. A different color is optionally used for each ensemble member. If all members have the same distribution, the "bands" of lines contributed by all members will lie in identical graph regions. A dissimilar trace will result in paths in an otherwise empty graph region. In this way "outlier" traces may be identified.

Periodogram #1 (v.catPer): Consider a trace $S_0, S_1, \ldots, S_{L-1}$. Let $S_i = s$, for some $0 \leq i < L$, where $s \in \Sigma$. Recall that E_i is the entry time of state S_i, and that $g(i)$ is the number of times that s occurs in $S_0, \ldots, S_i$. If $g(i) > 1$ then let S_j (where $j < i$) be the last occurrence of s prior to S_i (formally, S_j is the "last" occurrence if $s = S_j$ and $\forall k, j < k < i, s \neq S_k$). Then

$$\tau(i) = \begin{cases} 0 & \text{if } g(i) = 1 \\ E_i - E_j & \text{if } g(i) > 1 \end{cases}$$

A *categorical periodogram* (command *v.catPer*) plots $\tau(i)$. Fig. 10 illustrates *v.catPer* (view a categorical periodogram). The x-axis represents event numbers: one point for each trace event. The y-axis represents period, in the time units of the trace data. Specifically, the y-axis coordinate for x-coordinate i equals the duration of time from the entrance time E_i of event i to the entrance time of the *last* occurrence of state S_i in the trace file. If a state occurs periodically, say every 10000 time units, then the contribution of the state to the graph is a horizontal line with y-coordinate 10000. Therefore periodic states correspond to horizontal lines, which are present (e.g., at $y = 8$) in the transaction trace in Fig. 10(a). In contrast, a trace with no periodic behavior generates a distribution of points throughout the plane in a region bounded above by $y = x$. This is illustrated by Fig. 10(b), which shows the sequence of frame lengths generated by a variable bit encoding of the movie *Star Wars*. (This trace is considered further in §3.8 to create a traffic model for a video source in a communication network.) The bound arises because no points can lie above the line $y = x$ because each frame is generated at a constant rate (normalized to one frame per unit time in the trace file). Thus for event i on the x-axis, the last occurrence of the state S_i cannot occur more than i time units earlier.

Periodogram #2 (v.cCatPer): One would like to know how the density of points varies along any horizontal line through a categorical periodogram, as one moves from the bottom to the top. This is a *cumulative* categorical periodogram (command *v.cCumPer*). For example, this follows a geometric distribution in Fig. 10(b), indicating that if a frame length is repeated, it is more likely to be repeated sooner than later.

Let T denote the set of periods generated by a categorical periodogram; formally $T = \{\tau(i) | 0 \leq i < L\}$. Power, denoted $P(t)$, for $t \in T$, is defined as the number of states in a trace that have t as their period; formally $P(t) = ||\{\tau(i) | \tau(i) = t \wedge 0 \leq i < L\}||$. A cumulative categorical periodogram plots $P(t)$. The resulting graph conveys the distribution of the periods of all states in an ensemble member.

3.3 Testing Ensembles for Homogeneity

Informally, an ensemble is *homogeneous* if its members all display "similar" behavior. Formally, an ensemble is homogeneous if a hypothesis of homogeneity is true. Examples of hypotheses are: the occupancy times of a particular state are drawn from the same distribution, the same set of transitions are present in each member, a run (sequence of state transitions [7]) arises with about the same frequence in each member.

A user tests an ensemble for homogeneity. If the test fails, then the user partitions the ensemble members such that members in each partition are homogeneous.

In statistics, one must identify a universal population, and then calculate statistics of a sample of the population. In CHITRA94, the universal population is the set of all possible execution sequences of a system, and the sample is the execution sequence in a single trace file. A set of trace files are homogeneous with respect to some system attribute if the samples of those attributes contained in the traces are drawn from the same population or from identical distributions. CHITRA94 provides three methods; the first is a formal statistical test. Abrams, Lee, Cadiz, and Ganugapati [2] illustrate the use of all three methods to collect evidence as to whether an ensemble representing a database on a parallel computer is homogeneous.

Kruskal-Wallis rank sum test (t.kw): One homogeneity test in CHITRA94 is a classical statistical test of the hypothesis that sets of samples are drawn from the same distribution: the Kruskall-Wallis (KW) rank sum test [14, pp. 422–426], which is an analysis of ranks test. This nonparametric test was selected because it does not require any assumption about the distribution underlying the data. (Recall that distribution fitting is left to a tool outside of CHITRA94.) One of two quantities may be tested: (a) the model states arising in each trace, or (b) the occupancy time samples of a selected state arising in each state. (These two essentially test the first two of four definitions of "representativeness of a model" given by Ferrari, Serazzi and Zeigner [5, pp. 45-46].) To use the test, the user selects a significance level for the χ^2 test that underlies the KW method and, for (b), a single state that occurs in all traces. The null hypothesis tested is that the samples in all traces are drawn from identical distributions.

Comparison of adjacency matrices (v.am): Recall that CHITRA94 can map a trace to a semi-Markov process. The process contains a state transition diagram, giving the single step probability of transition from one model state to another. Therefore one hypothesis of homogeneity is that the set of transitions that arise in all traces in the ensemble are approximately the same. For example, if 9 of 10 traces have at least one transition from state 3 to 5, but the other does not, then only 9 of the 10 may be homogeneous.

The adjacency matrix for an ensemble $\mathcal{E}$ is formed as follows. Recall that n denotes the number of model states in the current version of an ensemble. Let $m_0, m_1, \ldots, m_{n-1}$ denote a sorted list of the elements of $\mathcal{M}(\mathcal{E})$. The adjacency matrix is an $n \times n$ matrix. Element i, j is equal to the percentage of the traces that contain at least one transition from state m_i to m_j. Thus, if element i, j is 0, no traces have a transition from model state m_i to m_j; while an entry of 100% indicates that all traces contain at least one instance of the transition. Nearly all non-zero entries in the adjacency matrix of a homogeneous ensemble should be close to 100%. The adjacency matrix is displayed by the *v.am* (view adjacency matrix) command, illustrated in Fig. 11.

Comparison of runs arising in files (x.a.run.ens): §3.6 describes several transforms which a user can perform on an ensemble. In one of these (*x.a.run.ens*), a user identifies a *run*, and CHITRA94 will identify the number and location of all occurrences of the run in all ensemble members. This is useful in traces containing periodic states. Therefore a test for homogeneity is that a run arises with roughly the same frequency in all ensemble members.

3.4 Testing Ensembles for Stationarity

At present, CHITRA94's models are homogeneous stochastic processes, which means that each trace in an ensemble must be stationary. To test stationarity, CHITRA94 provides a test: *t.ensStny*. This command partitions the time axis of each trace in the ensemble into N pieces, such that each piece has about the same time duration or number of events. The N pieces are then tested for homogeneity with the KW test (*t.kw*), as described in §3.3 (either on model states or state occupancy time).

When the test fails, a user must identify whether a trace can be partitioned into stationary components. To assist, one uses *v.meg* (§3.2.2) to identifies event numbers or times at which paths suddenly change their mean slope, as was illustrated in Figs. 8 and 9. Given a list of times or event numbers, the *x.sPartEns* (transform by stationary partitioning of an ensemble) command partitions a trace into an ensemble of sub-traces.

CHITRA94 also provides a command to superimpose the mean of a process (*v.smooth*), because a shift in the mean indicates non-stationarity. Several smoothing functions are provided, based on a uniformly and an exponentially weighted moving average with an optional window over which data is averaged. Traceview [11] also plots a uniformly weighted smoothing curve, although for a different purpose.

3.5 Modeling Ensembles

CHITRA94 at present contains the three models described below.

3.5.1 Model by a semi-Markov Chain the Ensemble: m.sm.ens

The normal representation of a stochastic process $X_{\mathcal{E}_v}(t)$ in CHITRA94 is as a time invariant, continuous time semi-Markov chain [4, pp. 290–294]. The state space of the chain is the set of model states, $\mathcal{M}(\mathcal{E}_v)$, that occur in ensemble version $\mathcal{E}_v$. The definition of a semi-Markov chain requires enumeration of the occupancy time distribution for each state in $\mathcal{M}(\mathcal{E}_v)$ and enumeration of a transition rate matrix. For each model state, CHITRA94 represent as a histogram the cumulative distribution function of occupancy time samples. The synthesizers (§3.7) generate a [0,1] random variable, and inverse map it using the cumulative distribution function to obtain a random variate [5, pp. 79-80]. CHITRA94 displays in model view statistics of occupancy time and a one step probability transition matrix, P. The transition rate matrix can be calculated from the occupancy time statistics and P (Fig. 12). The semi-Markov view displays P rather than the rate matrix because we feel that a user gets more intuition about the sub-ensemble behavior from the former than from the later. Matrix P contains one row and one column for each element of $\mathcal{M}$, similar to the adjacency matrix (§3.3). Element i, j of P is the percentage of transitions in $\mathcal{E}_v$ matching (m_i, m_j).

3.5.2 Model by CHAID the Ensemble: m.chai.ens

Often the semi-Markov model can be improved. The semi-Markov model defines the probability distribution of the next model state conditioned only on the current model state. Although the semi-Markov model is fast to compute – the time complexity is proportional to the sum of the lengths of current trace versions used to build the model – there is no inherent reason why a system cannot violate this assumption.

Therefore CHITRA94 incorporates an alternative (but more computationally costly) model: CHAID-based. The CHAID-based model identifies which past or future state best predicts each occurrence of one or more particular states of interest. The answer is an integer, called a *position*. The position -1 means that the previous state is the best predictor of the positions examined. Position -1 is a Markov model. A position of +4 means that the state of interest is best predicted by the state that arises four states later.

The prediction is obtained using CHAID (Chi-squared automatic interaction detection) [10]. CHAID originated as a technique to analyze a data set that contains vectors in which one component represents the categories of a dependent variable and the remainder represent the categories of a set of independent variables. The objective is to identify which dependent variable has the most significant interaction with an independent variable. The method analyzes sets of contingency tables formed from the data and uses the χ^2 values of the tables as a measure of significance.

CHITRA94 uses CHAID by extracting (via *e.some*) samples of W successive states in each ensemble member as the vectors in the data set. W is called a *window size*. The CHAID-based model identifies which position in the window best predicts the states that appear in the dependent variable position. The output of the *m.chai.ens* command is an i-step transition matrix, where i is the best predictor position, along with the p-values of each position in the window to justify why the selected position was chosen. Details on the CHAID-based model and an illustration of its use appear in [2].

3.5.3 Model by Univariate the Ensemble: m.uni.ens

This is a special case of CHAID, used when one wants to find the best predictor for a single state, rather than multiple states.

3.6 Avoiding State Space Explosion

To reduce the number of states required to model the ensemble, a user applies transforms that aggregate model states. A transform is invoked from *one* view, but is applied to all members of the ensemble, even if they are not being viewed. In addition, the result of the transform appears in *all* views of any member of the ensemble, to keep all views consistent.

A transform maps each trace in an ensemble to a simpler trace, either containing fewer model states or representing a shorter duration. Recall that an ensemble is treated as a set of realizations of a stochastic process $X(t)$. Transforms either reduce the range of X or reduce the interval over which t is defined. CHITRA94 uses the classical method of controlling state space explosion: a hierarchical model, in which all but the bottom level contain aggregate model states.

CHITRA94 has three categories of transforms: aggregation, which reduces the range of X by replacing model states by aggregate model states and hence defining a hierarchical model; clipping, which reduces the interval over which t is defined by extracting a portion of a trace; and projection, which reduces the range of X by applying user defined system state transformation rules. Transforms are defined and illustrated in [1, 2]. Projections currently in CHITRA94 are listed in Table 1. One projection, *x.p.sr*, to scale the range of each state vector component, is used in §3.8.

3.7 Validating CHITRA94's Models

CHITRA94 provides several tools to help validation. Essential to validation is CHITRA94's commands to generate an ensemble of one or more synthetic traces (using different random number seeds) from each type of model that Chitra provides: *s.sm.ens* for semi-Markov, *s.chai.ens* for CHAID-based, and *s.uni.ens* for univariate.

CHITRA94 supports at present three validation methods. In the first, one combines the original ensemble with an ensemble of synthetic traces, and then applies the normal homogeneity tests (§3.3). The hypothesis of homogeneity then becomes a hypothesis of model validity. Second, because many traces from computer and communication systems contain multiple runs of states, tests for the number, type, and distribution of runs can be used (*t.ensRnd* command). Finally, one can compare the autocorrelation functions of the original and synthetic traces (i.e., by the *v.autoCor* command in Figs. 1(b) and 2(b)) for ensembles with only numerical system state vector components.

3.8 Automating the Analysis Procedure

This section illustrates the process of writing both a command and a shell script to automate a particular algorithm to partition an ensemble into homogeneous sub-ensembles using adjacency matrices. We consider an analysis procedure formerly performed manually by one of the authors in constructing an input data model for a simulation study of a communication protocol for multimedia networks. The problem is to derive an input data model from traces of the sequence of frame sizes generated by a variable bit rate encoding of randomly chosen samples of entertainment video. Each frame will be sent in a network packet whenever someone watches a video that is delivered over the network. Frames are transmitted at a constant rate, so the model needs only to generate a frame length sequence.

The specific data we use in our illustration here is 60 segments of the movie *Star Wars* available on the Internet from Garrett and Vetterli [6]. The data uses a discrete cosine transform and entropy encoding of monochrome frames for display in a window of size 480 lines by 504 pixels.

First observe that none of CHITRA94's models can be used directly. There are two reasons. First, entertainment movies display time dependent behavior: a dialogue scene with a white backdrop resembling teleconferencing traffic characteristics (e.g., [8]) may be followed by a high motion rate scene with complex detail. Second, the frame lengths are positive integers bounded above by 78,459 bytes. Thus a direct representation as a Markov process in which model state x represents a frame of length x bytes requires an excessively large state space (naively storing the transition matrix requires about six trillion bytes of storage). Unless the state space is very small, the raw traces themselves will require less storages space and thus obviate the need for a model.

To cope with state space size, we use the standard solution of a hierarchical model. The frame length, viewed as a five digit decimal number, can be expressed as $10000x + y$, where $0 \leq x \leq 7$ and $0 \leq y < 10000$. Therefore our problem reduces to finding a model for the random variables X and Y, from which x and y are drawn, respectively.[2]

A CHITRA94 projection, *x.p.sr* (transform by projection to scale range – see Table 1), is used twice to produce two ensembles that represent X and Y separately. *x.p.sr* is first used to divide the original frame lengths by 10000 and then truncate them to integers to obtain samples of X. The second use of *x.p.sr* computes the frame length modulo 10000 to obtain samples of Y. The stationarity test (*t.ensStny*) reveals that traces of Y are stationary; a tool external to CHITRA94 (UniFit II) shows that they follow a uniform distribution. The ensemble containing traces of X will be partitioned into homogeneous sub-ensembles, and one Markov chain of up to eight states (because $0 \leq X \leq 7$) will be generated for each sub-ensemble.

One interpretation of the hierarchical model as saying that the upper digit of the frame length is a time-dependent quantity, while the remaining digits follow a uniform distribution because for a sufficiently large number of frames, one expects the set of numbers representing the encoded lengths to be random in length.

The user extension requires two parts. The first is a new command for CHITRA94 named *g.am.hPS* (generate by adjacency matrices a homogeneous partition state), that reads the adjacency matrix (in the form of a CF file) and identifies a *partition state* (defined below). The second is a shell script, named *g.am.hPart* (generate by adjacency matrices a homogeneous partition) (Fig. 13), which is the "glue" that allows CHITRA94's commands, along with *g.am.hPS*, to be used as building blocks to construct the overall algorithm. Also, a synthesizer is added that executes the existing *s.sm.ens* command to generate synthetic traces from the hierarchical model; this is used to drive simulations and for the purpose of validation. Finally, the commands and manual entries are added to the library of CHITRA94 extensions used by the analyst's organization.

---

[2] Also considered were models with more levels, but the two level model described is both simple and accurate.

g.am.hPS: This program (a few dozen lines of C++ code in our implementation) maps an adjacency matrix to a *partition state*. Observe in Fig. 11 that certain states (5,6,7) occur in only a few of the ensemble members (recall from §3.3 that the table values represent the percentage of ensemble members containing the listed transition). We choose one such state, and call that state the *partition state*. The partition state is chosen as follows: First, add the probabilities in each row. Choose the state whose row has minimum sum. If there is a tie, then choose the state whose row contains the most zeroes. If there is still a tie (say between states i and j), look at the columns for states i and j and choose the state with the most zeroes in its column. If there is still a tie, choose one state arbitrarily. For example, state 7 is chosen in Fig. 11 as the partition state. Finally, the algorithm will declare that there is no partition state if the state it computes appears in *all* traces, which is equivalent to the existence of a 1.0 frequency in any value in the row for the chosen state.

g.am.hPart: Given that we can find a partition state, the algorithm for ensemble partitioning works as follows (Fig. 13). We will use two data structures, TBPL (to be partitioned list: holds sub-ensembles that are believed to need partitioning) and DONE (holds sub-ensembles that can be partitioned no further). The shell script makes use of CHITRA94's support for two data types: set and CF file. TPBL and DONE are sets, manipulated with the *a.mod.set* (assist by modifying set) command. AMFILE is a CF file. Initially, TBPL contains one sub-ensemble equal to the original ensemble. First (step 1 in Fig. 13), some sub-ensemble on TBPL is removed (call it X). Then (step 2) an adjacency matrix is computed and stored in AMFILE. Then (step 3) a partition state (SPLITSTATE) is calculated for X using the program from the preceding paragraph. If no state can be found (step 4), X is inserted on the DONE list. Otherwise (step 5) we find which ensemble members contain SPLITSTATE, denoted as M. If (step 6) $X = M$ then all members contain the partition state, and the sub-ensemble cannot be partitioned further; thus X is inserted on DONE. Otherwise both M and $X - M$ are inserted into TBPL as two different sub-ensembles. Finally, (step 7), the algorithm terminates when there are no more sub-ensembles in TBPL. The list of sub-ensembles in DONE is the final partitioning. Thus the process generates a tree of partitions (Fig 14), with the leaves containing no partition state.

The final model has seven sub-ensembles, each no larger than eight states. A naive representation of each sub-ensemble model using 8 × 8 bytes results in only 448 bytes of storage, compared to the approximately 1M bytes required to store the original ensemble – a thousand-fold reduction.

4 Conclusions

CHITRA94 is the result of several years of building tools to explore and model the dynamic behavior of ensembles of trace data. It is our hope that other performance analysts will find CHITRA94 useful, and possibly contribute new library modules similar to the ones outlined in §3.8 to establish a library available on the Internet. The CHITRA94 source code is copyrighted but free to at least educational and non-profit organizations.

A number of challenging problems remain. Among these are:

- Development of a new model of transient behavior by fitting a curve to each path in a mass evolution graph and then taking the derivative of the curves to obtain the time dependent probability mass function of each model state.
- Development of automated algorithms to partition a non-stationary trace into components that are labeled stationary or transient.
- Analyzing ensembles of simulation output that represents *different* parameter values (right now we analyze multiple replicas of the *same* parameter values). Analysis should characterize the functional dependence of the dynamic behavior on the parameter values. A similar problem exists for input data modeling.

References

[1] M. Abrams, N. Doraswamy, and A. Mathur. 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, Nov. 1992.

[2] M. Abrams, T. Lee, H. Cadiz, and K. Ganugapati. Beyond software performance visualization. *submitted for publication*, Feb. 1994. also appeared as TR 94-07, Computer Science Dept., VPI&SU.

[3] D. P. Bertsekas. Dynamic behavior of shortest path routing algorithms for communication networks. *IEEE Trans. on Automata Control*, AC-27:60–74, 1982.

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

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

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

[7] J. D. Gibbons. *Nonparametric Statistical Inference*, chapter 3, pages 50–66. McGraw-Hill, New York, 1971.

[8] D. P. Heyman, A. Tabatabai, and T. V. Laskshman. 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, Mar. 1992.

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

[10] G. V. Kass. An exploratory technique for investigating large quantities of categorical data. *Applied Statistics*, 29(2):119–127, 1980.

[11] A. D. Malony, D. H. Hammerslag, and D. J. Jablonowski. Traceview: A trace visualization tool. *IEEE Software*, 8(5):19–28, Sept. 1991.

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

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

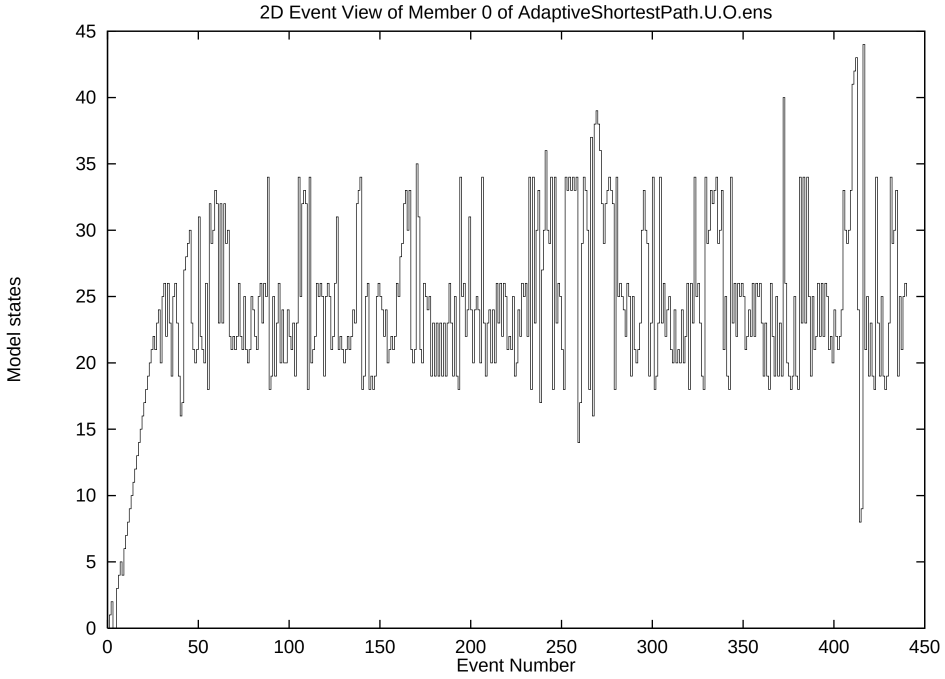
[14] L. Ott. *An Introduction to Statistical Methods and Data Analysis.* PWS-Kent, Boston, 3rd edition, 1988.

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

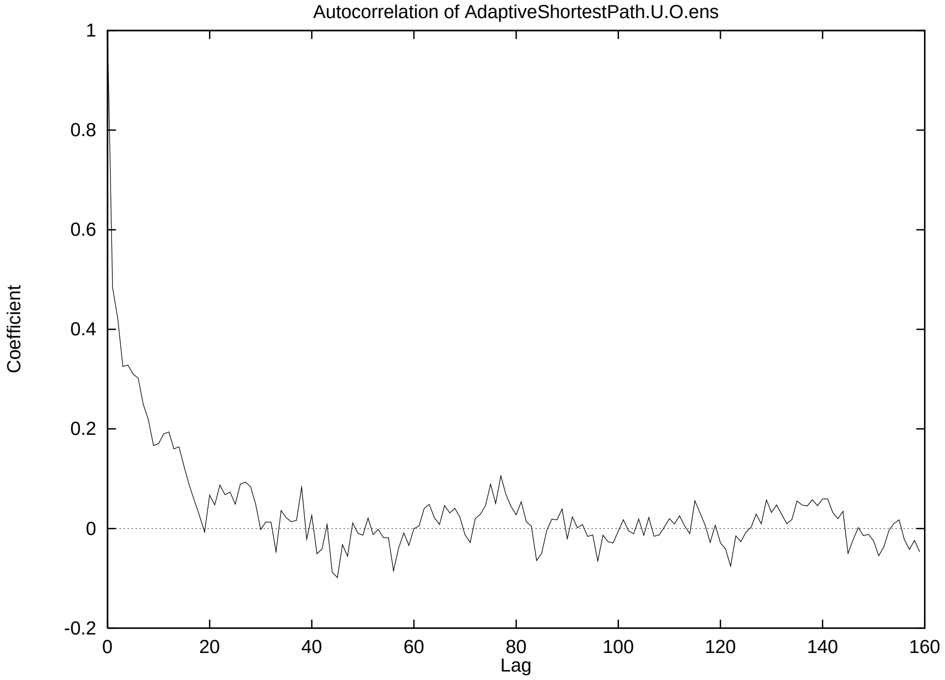
[16] G. Serazzi, editor. *Workload Characterization of Computer Systems and Computer Networks.* North-Holland, Amsterdam, 1986.

[17] S. G. Vincent and A. M. Law. UniFit II: Total support for simulation input modeling. In *Proc. Winter Simulation Conference*, pages 371–376, Arlington, VA, Dec. 1992.

[18] L. Zhang, S. Shenker, and D. D. Clark. Observations on the dynamics of a congestion control algorithm: the effects of two-way traffic. In *Proc. SIGCOMM*, pages 133–147, Zurich, Sept. 1991. ACM.

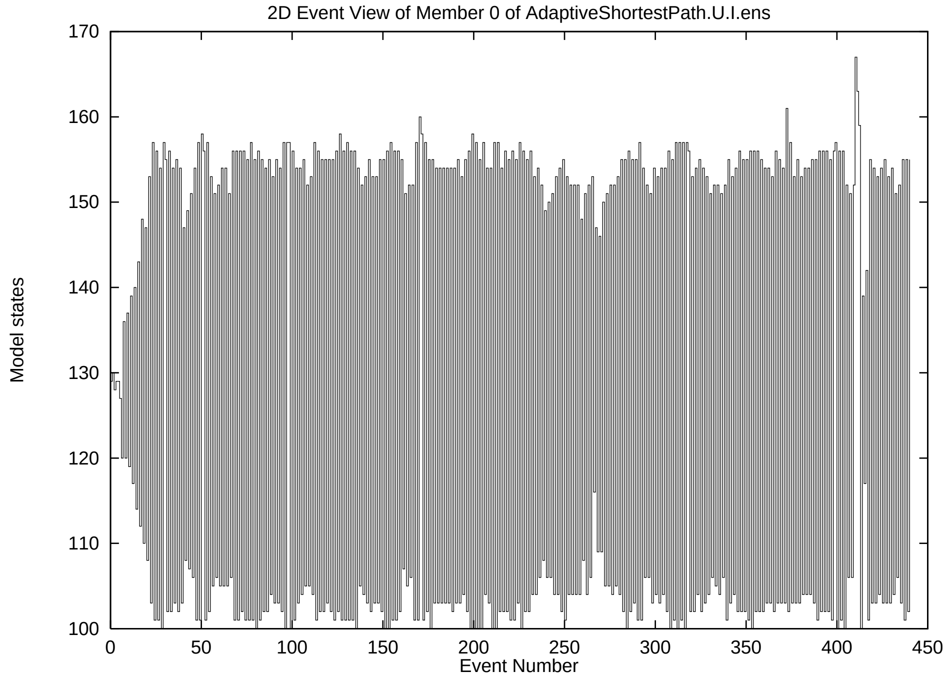


(a) Gantt chart (*v.gantt* command)

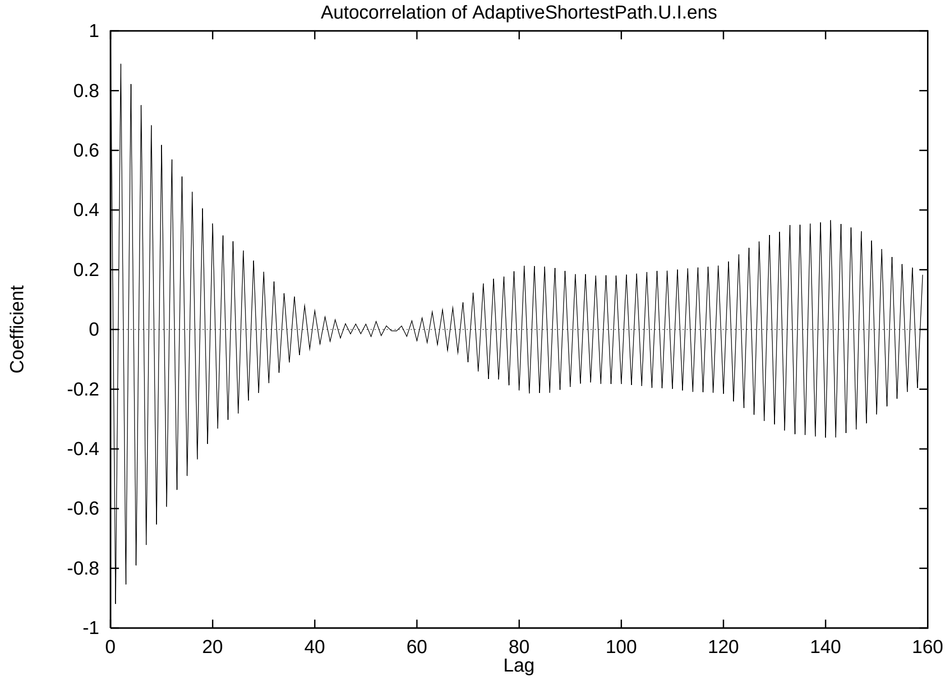


(b) Autocorrelation (*v.autoCor* command)

Figure 1: Simulation trace of routes chosen by adaptive shortest path routing algorithm using one mapping of routes to y-axis coordinates. Oscillatory behavior is obscured.



(a) Gantt chart (*v.gantt* command)



(b) Autocorrelation (*v.autoCor* command)

Figure 2: Same trace as Fig. 1, using a different mapping of routes to y-axis coordinates. Oscillatory behavior is now evident.

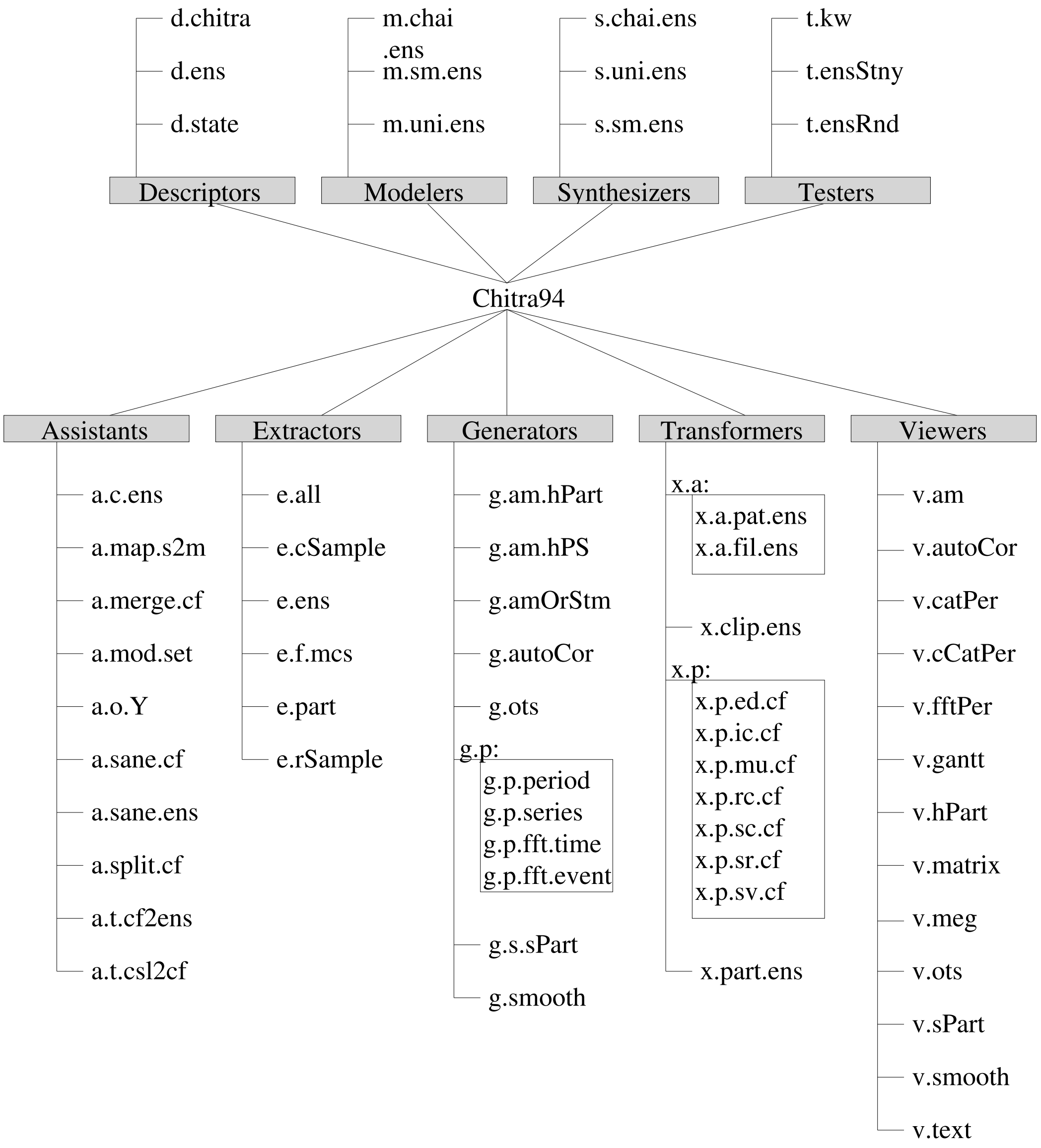


Figure 3: CHITRA94 commands.

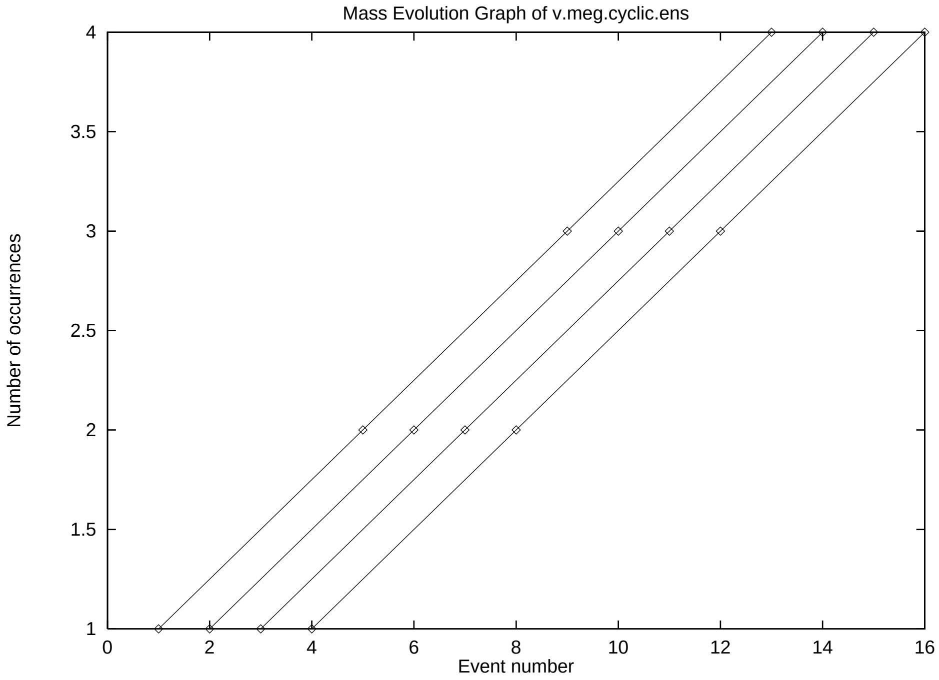


Figure 4: Mass evolution graph of trace with cyclic state sequence (*v.meg* command).

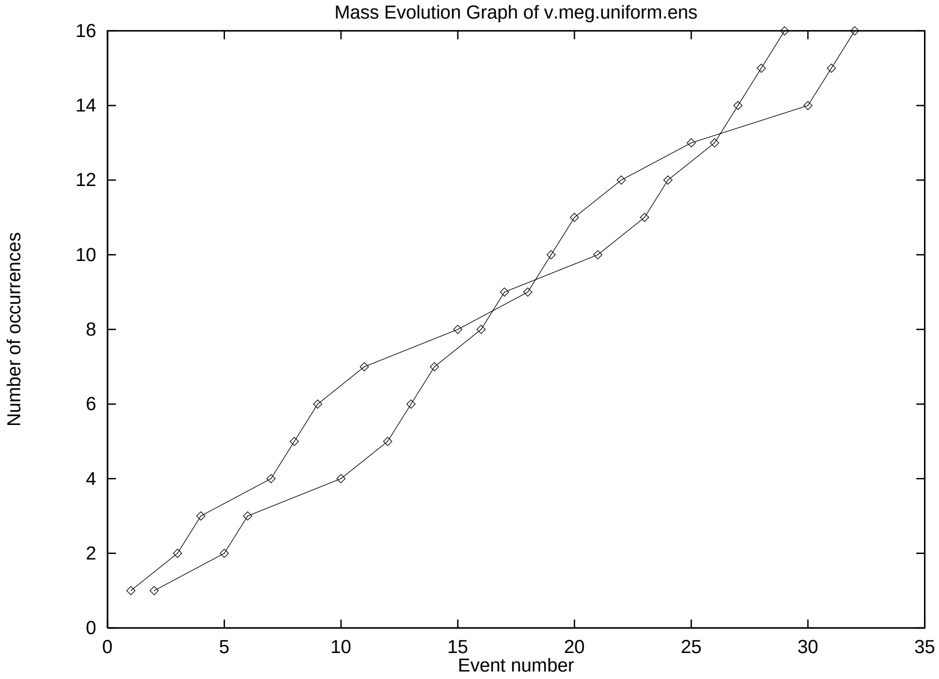


Figure 5: Mass evolution graph of trace with uniformly model state distribution (*v.meg* command).

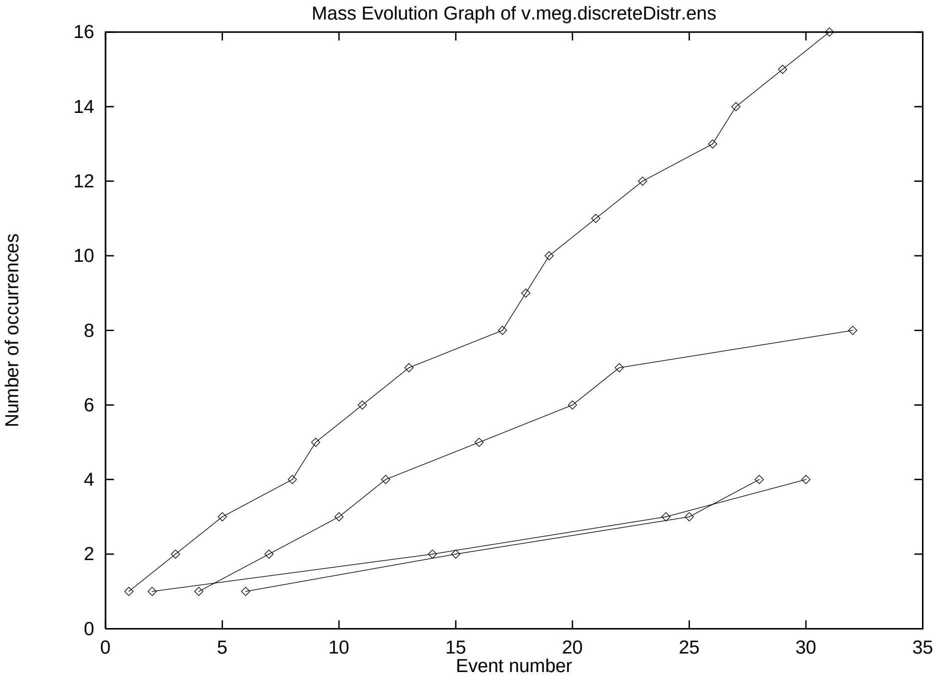


Figure 6: Mass evolution graph of trace with distribution $p_0 = 1/2, p_1 = 1/4, p_2 = p_3 = 1/8$ (*v.meg* command).

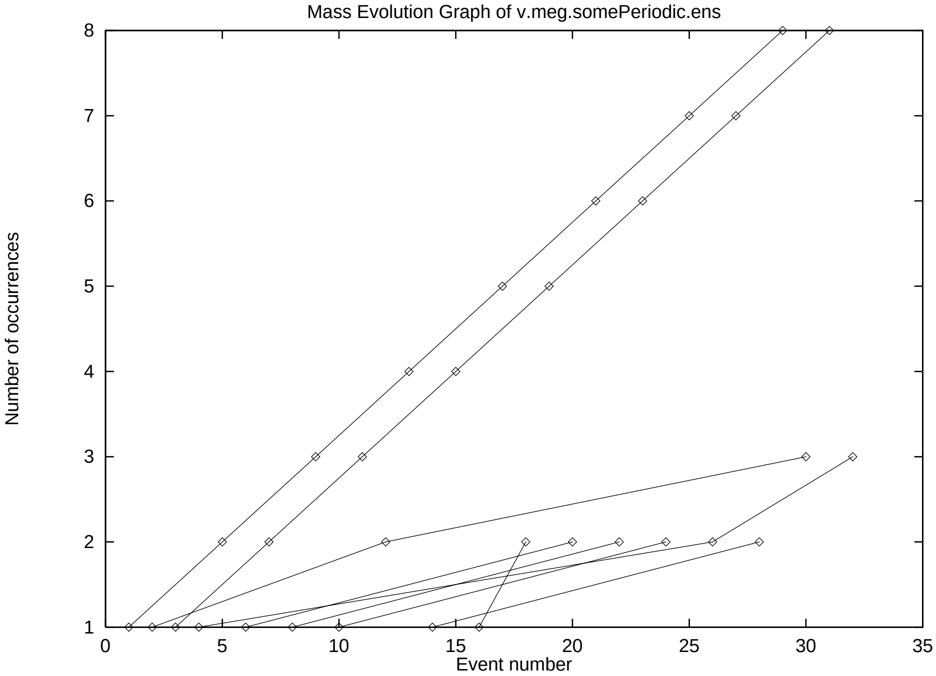


Figure 7: Mass evolution graph of trace with two periodic and two aperiodic states (*v.meg* command).

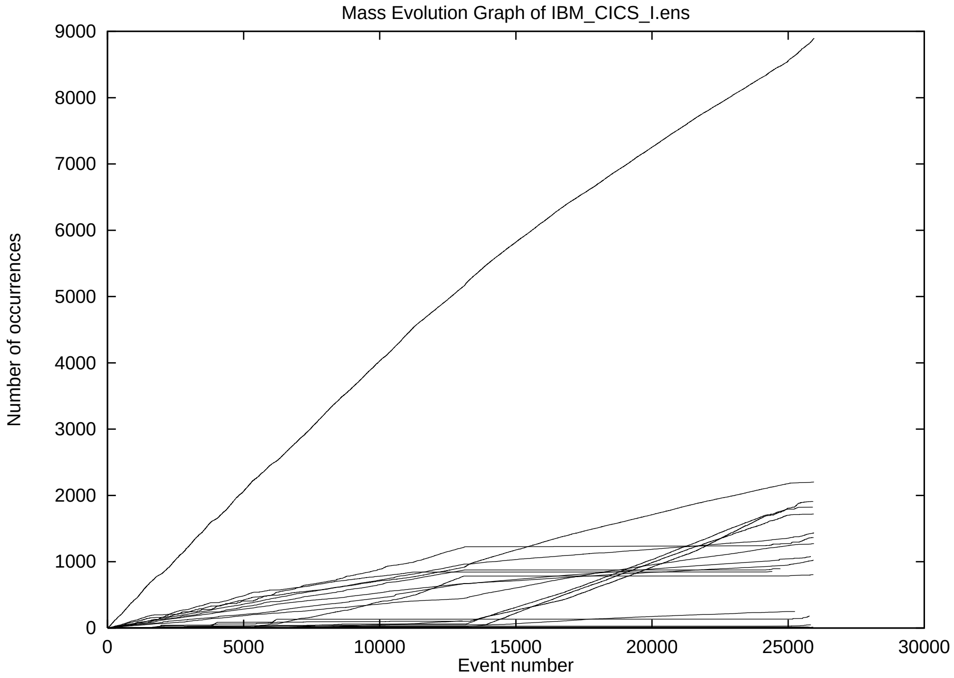


Figure 8: Mass evolution graph of trace of IBM CICS transactions; shows two stationary phase with transition at about event 14,000 (*v.meg* command).

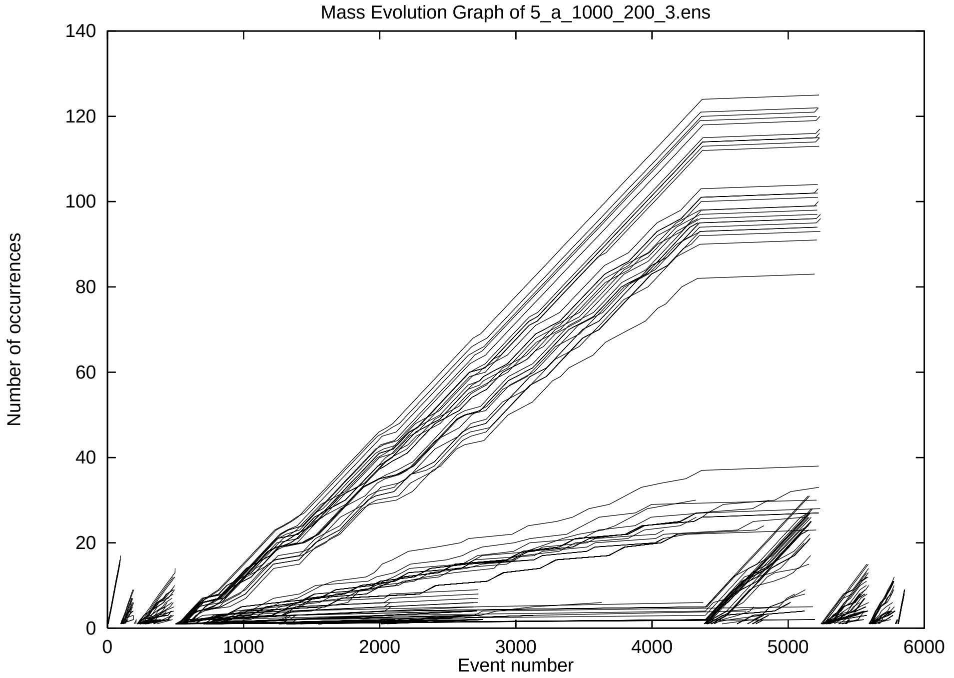
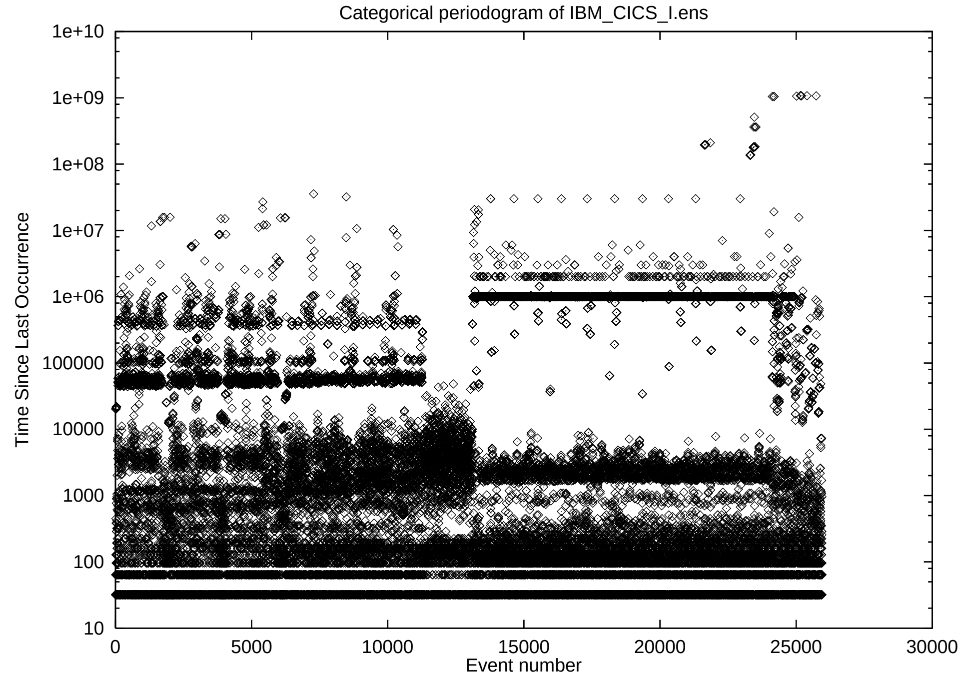
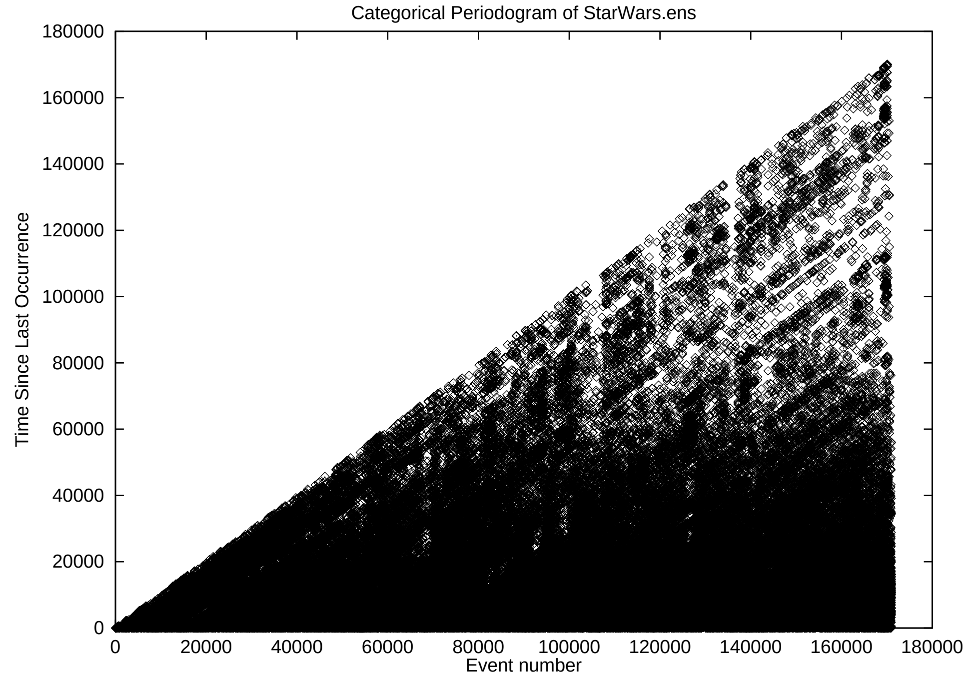


Figure 9: Mass evolution graph of five process dining philosophers program on a shared memory multiprocessor (*v.meg* command).

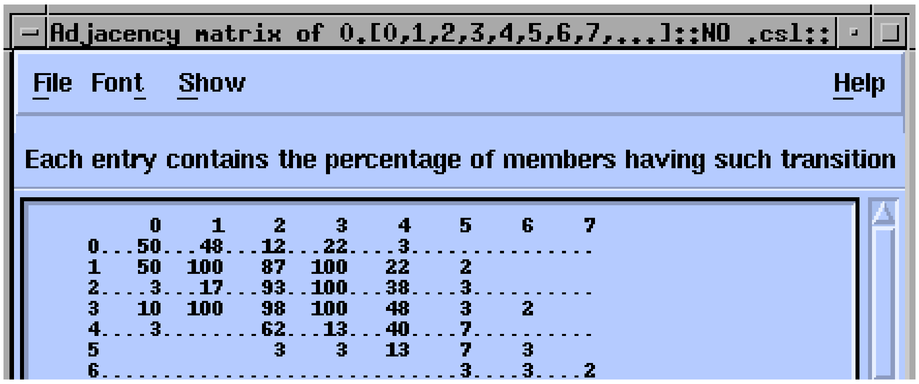


(a) Trace of IBM CICS transactions; horizontal lines indicate presence of periodic states.



(b) Packet lengths from the entire 120 minute movie *Star Wars*, illustrating randomness.

Figure 10: Periodograms for categorical data (*v.catPer* command).



Each entry contains the percentage of members having such transition

```
        0     1     2     3     4     5     6     7
  0...50...48...12...22....3................
  1   50   100   87   100   22     2
  2....3...17...93..100...38....3..........
  3   10   100   98   100   48     3     2
  4....3.........62...13...40....7..........
  5               3     3    13     7     3
  6.............................3....3....2
```