The Development of a CHAID-based Model for Chitra93

by

Horacio T. Cadiz

Project submitted to the faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

APPROVED:

[illegible]

Dr. Marc Abrams, Chairman

[illegible]

Dr. Gerald McLaughlin

Osman Balci

Dr. Osman Balci

February, 1994

Blacksburg, Virginia

c.2

LD
5655
V851
1994
C335
c.2

The Development of a CHAID-based Model for CHITRA93

by

Horacio T. Cadiz

Committee Chairman: Dr. Marc Abrams

Computer Science

(ABSTRACT)

The complexity of the behavior of parallel and distributed programs is the major reason for the difficulties in the analysis and diagnosis of their performance. Complex systems such as these have frequently been studied using models as abstractions of such systems. By capturing only the details of the system which are considered essential, a model is a replica of the complex system which is simpler and easier to understand than the real system. CHITRA92, the second generation of the performance analysis tool CHITRA, builds a continuous time semi-Markov chain to model program behavior. However, this model is limited to representing relationships between states which are only immediate predecessors or successors of each other. This project introduces and implements a new empirical model of the behavior of software programs which is able to represent dependencies between non-sequential program states. The implementation combines deterministic and probabilistic modeling and is based on the Chi Automatic Interaction Detection (CHAID) statistical technique designed for investigating categorical data. The empirical model, constructed by analyzing an ensemble of program execution sequences, is stochastic and non-Markovian in the form of an N-step Transition Matrix. The algorithm is integrated as one of the modeling subsystems of CHITRA93, the third generation of CHITRA.

ACKNOWLEDGEMENTS

The author thanks his wife Joanne and daughter Maria Frances for their love, inspiration, support, and sacrifice; his family and friends for their prayers and support; all the members of the CHITRA group for their help and guidance; and Dr. Gary Perlman for allowing him to use the chi-square functions from "|STAT."

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1
Introduction

Parallel and distributed programs exhibit behavior which is complex and difficult to analyze. The complexity arises from the fact that, in general, different runs of the same parallel or distributed program may execute statements in different orders and consume different amounts of resources [14]. This *nondeterministic* behavior is caused by the competition for system resources between multiple and concurrent threads of execution in such programs. It is a major impediment to making generalizations and, ultimately, gaining knowledge about the characteristics of parallel and distributed programs.

Analysts mostly rely on performance data collected from the execution of programs as the source of the details about the behavior of programs. However, especially for parallel and distributed programs, the details which explain the observed behavior of the system are frequently obscured by the sheer volume and complexity of the collected data. *Visualization*, a technique which transforms symbolic data into geometric shapes [24], has been incorporated into many modern performance analysis tools with the hope that better insights into the data could be obtained.

CHITRA[1] is a tool that uses traditional visualization techniques for performance analysis. However, it is unique among visualization tools because it integrates construction of an empirical model fitting the data with visualization. An empirical model is important because it provides a replica of the system under study which could be manipulated and examined. Frequently, the model developed from the data points towards other useful ways of manipulating the data. It may show that some previous steps taken to develop the model were unnecessary or erroneous or that perhaps some important details had been excluded. This leads to further data manipulation and model development. Questions about the system are often framed in terms of the model and, if unanswered by the model, lead to further study of the data for developing a better model. Thus, CHITRA's answer to the complexity of the behavior of a system is the incremental development of a simple, accurate, and coherent model to explain the complex behavior.

CHITRA has integrated its visualization and modeling modules to make it convenient for an analyst to iterate through the steps of data visualization, data manipulation, and model development. CHITRA is built on the premise that fast, simple, convenient, and accurate model construction is crucial for performance analysis [2]. CHITRA, therefore, offers a methodology for building an empirical model of the behavior of a system under analysis.

[1] *Chitra* is a Sanskrit word for a beautiful or pleasing picture.

1.1 Overview of CHITRA

CHITRA91[2], CHITRA92[14], and CHITRA93 are the three generations of the performance analysis tool CHITRA. They all describe the behavior of a system by enumerating the sequence of *states* entered by the system during execution. Each *system state* is an n-tuple representing n attributes of the system. A tuple, for example, may represent the name of a program variable and another tuple the value of that program variable. The choice of attributes to include in a system state is dependent on the objectives of the study. The sequence of system states is called the *system state sequence* (SSS). Let $S = \langle s_1, s_2, \ldots, s_E \rangle$ be the SSS of a system under study where E is the length of the SSS. Let t_i be the time when the system entered state s_i, $1 \leq i \leq E$. The sequence of ordered pairs $\langle (t_1, s_1), (t_2, s_2), \ldots, (t_E, s_E) \rangle$ is called the *timed* SSS (TSSS). The ordered pair of successive states from S, (s_n, s_{n+1}) where $1 \leq n < E$, defines a *transition.* The state s_n is called the *predecessor* of s_{n+1} while s_{n+1} is the *successor* of s_n. A transition is said to to be *out* of a predecessor state and *into* the successor state.

CHITRA92 has three major functional modules to support the construction of a model based on a SSS.

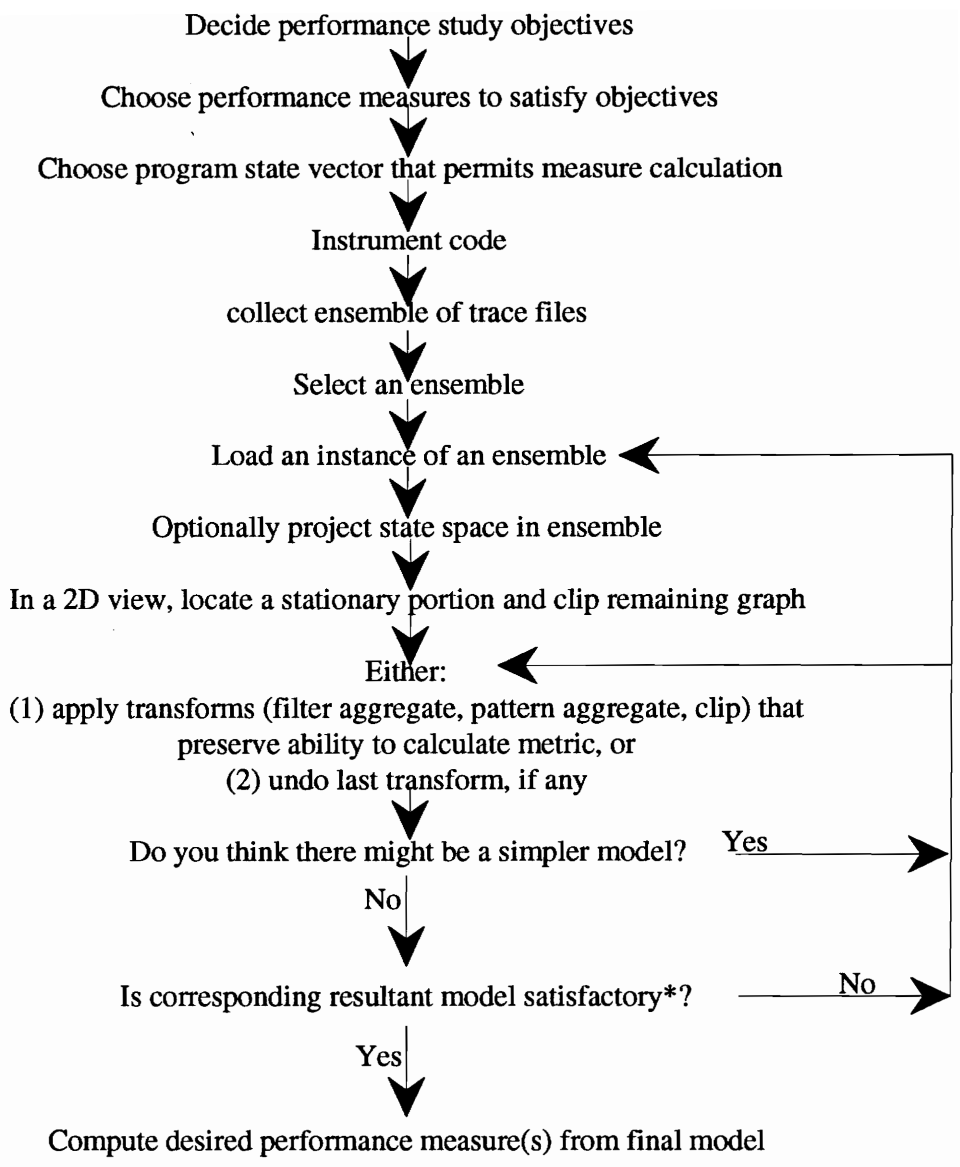
1. The first is a *visualization* module which maps the SSS into one or more display images. This provides plots of states versus time (by using the corresponding TSSS),

of states versus event number, and periodograms representing the Fourier transforms of the previous two plots.

2. The second is a *transform* module containing four transform functions which either map sequences of one or more states into one or more *aggregate* states or delete a sequence of states. *Projection*, *pattern aggregation*, and *filter aggregation* are the three mapping functions while *clipping* is the function for deleting states. The application of any of these functions results in the creation of a new SSS.

3. The third is a *modeling* module which produces an empirical model fitting a SSS. The model (discussed in the next section) is a homogeneous, continuous-time, semi-Markov stochastic process constructed to fit the SSS. Occupancy time statistics, a set of histograms, and a transition matrix are calculated on the basis of the states in the model.

These modules are integrated together to allow the user to iterate through the process of model building as summarized in Figure 1.1, which is adapted from [14]. This integration is the source of one of the main strengths of CHITRA – the convenient exploration of different alternatives for building a model.


Decide performance study objectives
Choose performance measures to satisfy objectives
Choose program state vector that permits measure calculation
Instrument code
collect ensemble of trace files
Select an ensemble
Load an instance of an ensemble
Optionally project state space in ensemble
In a 2D view, locate a stationary portion and clip remaining graph
Either:
(1) apply transforms (filter aggregate, pattern aggregate, clip) that preserve ability to calculate metric, or
(2) undo last transform, if any
Do you think there might be a simpler model?
Yes
No
Is corresponding resultant model satisfactory*?
No
Yes
Compute desired performance measure(s) from final model

"Satisfactory" means (1) measure computed from model is sufficiently close to measure computed from ensemble, and (2) model state spae is sufficiently small and simple to permit insight.

Figure 1.1: Performance analysis using CHITRA

1.2 The Empirical Model of Program Behavior in CHITRA

CHITRA91 and CHITRA92 model program behavior as a homogeneous, continuous time semi-Markov (CTSM) process. A program state is mapped to a stochastic process model state through *aggregation*. Through this process, history is embedded in the state.

Example 1 *Consider a program solving Djikstra's "Dining Philosophers" problem [1]. N philosophers sitting around a table spend their lives alternating between eating and thinking. On the table is a bowl of food and N forks arranged in a circle such that each philosopher has one fork to the left and another on the right. When a philosopher is hungry and wishes to eat, the philosopher must use* both *left and right forks in his position. Therefore, no neighboring philosophers may eat simultaneously. When a philosopher finishes eating, both forks are placed back on the table and the philosopher begins thinking. The program is an algorithm for philosopher*$_i$, $0 \leq i \leq N-1$.

We model the behavior of philosopher$_i$ *cycling through the following states: Think Short (TS), Acquire Left Fork (A1), Acquire Right Fork (A2), Eat Short (ES), Release Left Fork (R1), Release Right Fork (R2), Think Long (TL), Acquire Left Fork (A1), Acquire Right Fork (A2), Eat Long (EL), Release Left Fork (R1), Release Right Fork (R2). ES (respectively, TS) represents eating (thinking) for a short period of* **x** *time units. EL (TL) represents a long eating (thinking) period of length* **10x** *time units. Each philosopher thus* *perfectly cycles between short and long eating (thinking) times. If the philosopher is currently in state ES (TS), then the next eat (think) state will be EL (TL). Furthermore, state ES is always followed by state TL. Similarly, state EL is followed by state TS. We have* S*, the sequence of program states as* $S = \langle TS,\ A1,\ A2,\ ES,\ R1,\ R2,\ TL,\ A1,\ A2,\ EL,\ R1,\ R2,\ TS,\ A1,\ A2,\ ES,\ R1,\ R2,\ TL,\ A1,\ A2,\ EL,\ R1,\ R2,\ \ldots,\ R2\rangle$ *with alternating subsequences of* $\langle TS,\ A1,\ A2,\ ES,\ R1,\ R2\rangle$ *and* $\langle TL,\ A1,\ A2,\ EL,\ R1,\ R2\rangle$*.*

Figure 1.2 shows a CTSM model chain of S in Example 1. State A denotes the sequence $\langle A1, A2\rangle$ and is an example of an aggregate state. Because this sequence is deterministic (i.e., $A1$ is *always* followed by $A2$), there is no loss of information when the sequence is aggregated into one new state, A[2]. The same holds true for state R which is the aggregation of the sequence $\langle R1, R2\rangle$. Aggregating sequences of states into a single model state allows an analyst to generate a simpler model of the sequence of program states under study.

In general, CHITRA's *model state* is an aggregate state. Two important properties of the CHITRA CTSM model result from aggregation.

1. Some history is encoded in the model state. We note that the history that state $A2$ is preceded by state $A1$ is embedded in the model state A.

[2]In general, the loss of information resulting from an aggregation, is proportional to the probability of the transition from the initial state of the aggregated sequence to final state of the aggregated sequence. In our example, if $A2$ were to follow $A1$ for only 90% of the time, the aggregation of $A2$ and $A1$ would have resulted in a loss of information about the other 10% of the transitions from $A1$.

2. State transitions are based only on the current program state. The transition from state A to ES in Figure 1.2, for example, considers only state A. There is no possibility of considering the other states that have preceded A to determine that ES or EL is its successor.

The following definition, based on [15], formally defines the *Markov-dependence* property of the CTSM model. Given a set of indices T and a set of states $\Sigma = \{s_t \mid t \in T\}$, Σ has the *Markov-dependence* property if for any $t_0 < \ldots < t_m < t_1, \ldots, t_n \in T$, the conditional probability distribution $F(s_{t_1}, \ldots, s_{t_n} \mid s_{t_0}, \ldots, s_{t_m}) = F(s_{t_1}, \ldots, s_{t_n} \mid s_{t_m})$. That is, for any state s_t at some $t \in T$, the future state s_v for $v > t$ is independent of the past state s_r for $r < v$.

Any program execution can be viewed as a finite state machine (FSM). The enumeration of the values of all the memory locations and registers of the computer it is running on is the FSM's state space Σ. A program state is merely an abstraction for a computer's state

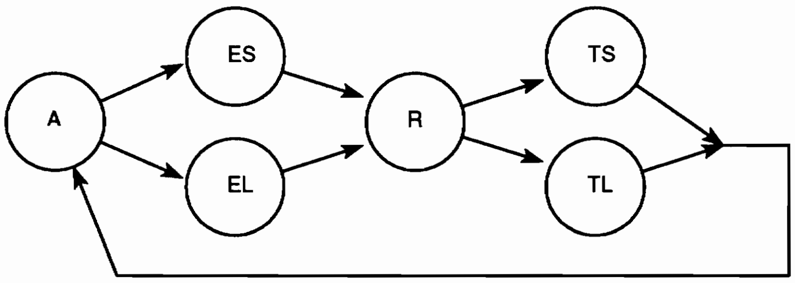


Figure 1.2: The model for S

(i.e., a combination of a computer's memory locations and registers). We note that the FSM's transition function is defined by the sequence of states realized by the program. Because a program is a FSM, its next state (i.e., the computer's next state) is based solely on its current state. This finite state machine can be used as the semi-Markov chain of the CTSM model with each unique program state represented by a model state. Thus, in theory, software (i.e., the set of program states in the software) is Markov-dependent and its behavior can be modeled as a homogeneous, continuous time semi-Markov chain [2]. We note that the one-to-one correspondence between the set of program states and the set of the model states implies that the transition functions of the model and the finite state machine are equivalent as well. Because of these relationships, we describe this CTSM model as an *ideal* model.

1.3 Limitations of the Chitra CTSM Model

Consider the state space Σ of the ideal model. Each of its elements is a component with *every* variable in the program as a component. This means that in a non-trivial program, not only would the size of Σ be prohibitively large but it would also contain references to variables which might not be of any interest to the analyst for the objective of the model to be built. In practice, therefore, the modeler will most likely choose another definition of the state space to reduce its size. Usually, only the "essential" or "interesting" variables will be included as components of the state vectors. This new state space, denoted as Σ', is then used as input to CHITRA.

In Σ, any given state is reached through a combination of some probabilistic and deterministic transitions. The deterministic transitions represent the execution of deterministic program statements (e.g., assignment statements, function calls). On the other hand, the probabilistic transitions represent the execution of random events (e.g., page faults, asynchrony). When the reduced state space Σ' is used, the one-to-one correspondence between the transition functions of the CTSM model and the FSM is lost. The states in Σ' may *not* obey the Markov-dependence property. Thus, in practice, a simplified and reduced state space is traded for the loss of the Markov property.

Consider S in Section 1.2 and its CTSM model in Figure 1.2. The best information that can be gathered from this model is that there is a 50% probability of transitions from state A to either ES or EL. The model is unable to capture the deterministic relationship between states TS and ES which we had described in Example 1. The better model would be as in Figure 1.3 which shows that the occurrence of state ES is *determined* by the occurrence of state TS.

In fact, because we have deterministic sequences in $\langle TS, A, ES\rangle$ and $\langle TL, A, EL\rangle$ (i.e., there are no other sequences which begin with TS (TL) and end with ES (EL) in the SSS), we can further aggregate the two sequences into two different model states $T1$ and

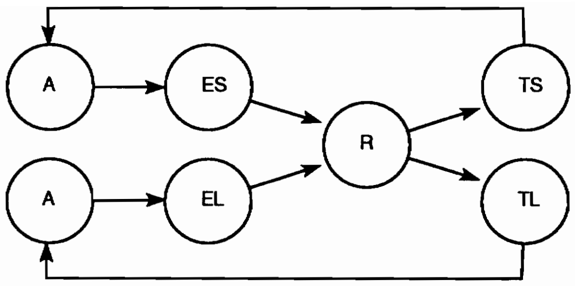


Figure 1.3: The deterministic relationship between states TS (TL) and ES (EL)

$T2$ respectively. The simpler model is shown in Figure 1.4.

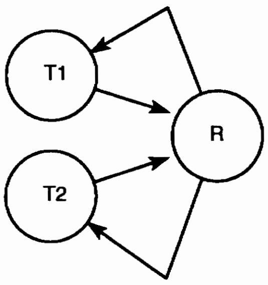


Figure 1.4: A simpler model of Figure 1.3 with the aggregate states $T1$ and $T2$

Determining the existence of a relationship between program states is important in the performance analysis of software. If a state under analysis were to represent say a "packet storm" in a network, memory thrashing by an operating system, or any other problem state, the objective of an analyst would be to determine the causes of this state with the ultimate goal of preventing or minimizing their occurrence. This, however, is dependent on discovering what previous states could have triggered the occurrence of state d. Uncovering associations between the other states and state under analysis is therefore crucial.

In general, given any SSS $S = \langle s_1, s_2, \ldots, s_E \rangle$, the CTSM model for S can not directly represent the relationship between s_n and s_{n+k} $(1 \leq n < E)$ when $k \neq 1$. In other words, the transition (s_n, s_{n+1}) is defined in the semi-Markov model but not (s_n, s_i) where $i < n$ or $i > n + 1$. This exposes the two main shortcomings of the CTSM model:

1. only one state is examined for possible dependencies with a particular state;
2. only the transitions from the past state can be represented in the model.

However, associations between program states are not always limited to those states which are immediate predecessors or successors of each other. A model which is able to represent the relationships between such states would be a powerful aid to software performance analysis.

1.4 Problem Definition

This project proposes a stochastic, non-Markovian model designed to uncover dependencies between model states such as those found in Figure 1.3. That is, for any given SSS S and state s_n, the new model is able to consider *and* represent the transition (s_n, s_{n+k}) for some $k \neq 1$. The new model developed by this project is unencumbered of the two main shortcomings of the current CTSM model because it is capable of representing dependencies between non-sequential model states. The new model is called a *CHAID-based* model because it is based on a statistical technique (described more fully in the succeeding sections) called Chi Automatic Interaction Detection (CHAID)[18]. The CHAID-based model is an empirical model developed from an *ensemble* or collection of one or more SSS's, each of which is obtained from an execution of a program. The model consists of model states and state transition probabilities in the form of a transition matrix.

Chapter 2

Foundations of the CHAID-based Model

This chapter introduces and defines several terms necessary for developing the CHAID-based model. We present the definitions necessary for applying the CHAID-based statistical technique to an SSS and give the algorithm used in CHITRA93.

2.1 Preliminary Definitions

The SSS from a program is not directly analyzed in developing a CHAID-based model. Rather, we use subsequences obtained from a SSS as the basic unit of analysis. We first describe how such subsequences can be obtained from a SSS.

Consider the SSS $S = \langle s_1, s_2, \ldots, s_i, s_{i+1}, \ldots, s_{i+n}, \ldots, s_E \rangle$, where $1 \leq i \leq i+n \leq E$. The sequence $S' = \langle s_i, s_{i+1}, \ldots, s_{i+n} \rangle$ is a *subsequence* of S. The *length* of S', $n+1$, is the number of states in the subsequence. $Sub(S, n+1)$ is the set of subsequences of S of length $n+1$. Let W be any integer satisfying $2 \leq W \leq E$. An *analysis set* of S is *some* subset of $Sub(S, W)$. W is called the *window size* of the analysis set.

A *tallied analysis set* can be obtained from a base analysis set by augmenting each sequence in the analysis set with an integer to denote the frequency of occurrence of the sequence in the SSS. Let π denote a sequence in an analysis set; and $k > 0$ denote the frequency of occurrence of π in S. Then, the tallied analysis set is a set of ordered pairs (k, π) for all π in the analysis set and their frequencies $k > 0$. Henceforth, we use the term analysis set to refer to the tallied analysis set.

The next two examples show how an analysis set may be obtained from a SSS. Example 2 uses an analysis set based on *all* the subsequences of length W from a SSS. Example 3, on the other hand, uses an analysis set based on a *subset* of the set of subsequences of length W from a SSS.

Example 2 *Consider a SSS* $S = \langle g,g,f,a,g,f,a,e,d,c,d,e,a,g,f,a,e,d,c,d,e,a,g,f,a,g,a,c,d \rangle$, *which has* $E = 29$ *states, 6 of which are unique. If the window size is* $W = 4$, *a possible analysis set is* $\Pi = \{$ (1, $\langle g,g,f,a \rangle$), (2, $\langle g,f,a,g \rangle$), (1, $\langle f,a,g,f \rangle$), (3, $\langle a,g,f,a \rangle$), (2, $\langle g,f,a,e \rangle$), (2, $\langle f,a,e,d \rangle$), (2, $\langle a,e,d,c \rangle$), (2, $\langle e,d,c,d \rangle$, (2, $\langle d,c,d,e \rangle$), (2, $\langle c,d,e,a \rangle$), (2, $\langle d,e,a,g \rangle$), (2, $\langle e,a,g,f \rangle$), (1, $\langle f,a,g,a \rangle$), (1, $\langle a,g,a,c \rangle$), (1, $\langle g,a,c,d \rangle$)$\}$.

Example 3 *Consider a SSS* $S = \langle g,g,f,a,g,f,a,e,d,c,d,e,a,g,f,a,e,d,c,d,e,a,g,f,a,g,a,c,d \rangle$. *We have the same window size* $W = 4$ *as in Example 2. However, we are only interested in the subsequences whose final state is either* **d** *or* **a**[1]. *The analysis set is* $\Pi' = \{$ (3, $\langle a,g,f,a \rangle$),

[1]We denote the states in bold whenever there is a possibility that they could be mistaken for letters. This often happens when the states are not surrounded by other mathematical symbols which would give their context. In such cases, the reader should note that a and **a** refer to the same state.

(2, $\langle f,a,e,d \rangle$), (2, $\langle e,d,c,d \rangle$, (2, $\langle c,d,e,a \rangle$), (1, $\langle f,a,g,a \rangle$), (1, $\langle a,g,a,c \rangle$), (1, $\langle g,a,c,d \rangle$)}.

Our choice of an analysis set is thus governed by two criteria. The first is the window size W and the second is the use of either a proper or the improper subset of $Sub(S, W)$. Deciding which subset of $Sub(S, W)$ to use for generating the analysis set is based on the type of analysis a user wants to perform. This is fully discussed in Section 2.7.

Because an ordering of the states is implicitly defined by our subsequences, *positions* may be assigned to the states in any sequence. Consider the subsequence $S' = \langle s_i, s_{i+1}, \ldots, s_{i+n} \rangle$ of length $n+1$, for some integers i and n. The *absolute position* of a state s_{i+j} in S' is $j+1$, $0 \leq j \leq n$. Furthermore, because the subsequences in an analysis set are all of equal length, all the subsequences have the same number of positions. We may consider the positions as properties of the analysis set and speak of analysis sets as having positions.

Example 4 *Consider the subsequence $\langle f, a, e, d \rangle$ in Π of Example 2. The states* **f**, **a**, **e**, *and* **d** *are in absolute positions 1, 2, 3, and 4 respectively. These same four positions are defined for all the subsequences in* Π. *We can say that* Π *has positions 1, 2, 3, and 4.*

Recall that the limitation of the CTSM model arises from the fact that the transitions to any given state are based only on the state's immediate predecessor. As we had previously mentioned, the CHAID-based model is free from such a constraint. Let $S = \langle s_1, s_2, \ldots, s_E \rangle$ be a SSS of length E; and F and H, be two integers such that $W - H - F \geq 1$. Then the ordered pair of states from S, (s_n, s_{n+j}), $1 \leq n < E - F$ and $1 \leq j \leq F$, defines a *forward transition* for state s_n; and (s_{n-j}, s_n), $E - H \leq n \leq E$ and $1 \leq j \leq H$, defines a *reverse transition* for state s_n. We say that the state s_{n+j} (respectively, s_{n-j}) is *j steps forward from* (*steps backward to*) s_n. F is called the *future window size* and H is called the *past window size.*

F and H, respectively, determine the number of future and past states to be considered in the analysis. For convenience, we follow the convention of assigning positive integers for the positions of the future states and negative integers for the positions of the past states. We refer to these position numbers as the *relative positions* to contrast them with the absolute positions which are all positive integers. The *relative position* of a state s in the absolute position k in some sequence S' of length W is defined to be $(k - W) + F$.

Example 5 *Consider* $(2, \langle f, a, e, d\rangle)$ *in* Π *of Example 2. If* $F = 2$ *(thereby setting* $H = 1$*), the states* $\mathbf{f}, \mathbf{a}, \mathbf{e}$, *and* $\mathbf{d}$ *have relative positions -1, 0, 1, and 2 respectively. If* $F = 3$ *(or* $H = 0$*), the relative positions are 0, 1, 2, and 3.*

The following two definitions hold true for either relative or absolute positions. A position I is a *predecessor* of position N if and only if $I < N$. We say that I *precedes* N. A position I is a *successor* of position N if and only if $I > N$. We say that I *succeeds* N. Henceforth, unless otherwise specified, the term *position* is used to refer to *relative position.*

We now relate some basic concepts in statistics with the items we have defined. Given a SSS S, and a window size W, the *population* is an analysis set Π from S. The units of study, the *individuals*, are the subsequences in Π. The *variables*, properties of individuals that may take two or more values but not at the same time for the same individual [22], are the positions. The *dependent variable* or *dependent position* is position 0. A *predictor variable* or *predictor position* is any position other than position 0. The domain of a position I, denoted Σ_I, is the set of states that appear in position I for all subsequences of the analysis set Π; formally, $\Sigma_I = \{s_I \mid \langle s_i, s_{i+1}, \ldots, s_I, \ldots, s_{i+W-1}\rangle \in \Pi \ \wedge \ i \leq I \leq i+W-1\}$.

Example 6 *Consider* Π *in Example 2. If we have* $F = 2$*, the variables are the positions -1, 0, 1, and 2. The set of states in position 1 is* $\Sigma_1 = \{a, c, d, e, f, g\}$.

2.2 The Categories of Positions

The categories define a mutually exclusive and exhaustive partitioning of the states that occur in the position. The CHAID-based analysis (Section A.1) will:

1. search for similar categories of an independent position and group them together; and
2. search for associations between the (possibly grouped) categories of any given independent position and the categories of the dependent position.

Our use of the term "category" is equivalent to its use in statistics. The notion of a category of a position is formalized below.

Let $\mathcal{P}_I$ denote the set of all possible partitions of Σ_I, for any position I, such that $\{\rho_1^I, \rho_2^I, \ldots, \rho_m^I\} \in \mathcal{P}_I$ iff

$$\bigcup_{1 \leq i \leq m} \rho_i^I = \Sigma_I \ \wedge \ m \geq 2.$$

Let $\mathcal{C}_I$ denote a set of categories of position I; formally, $\mathcal{C}_I \in \mathcal{P}_I$. Let M_I denote a function that maps states in Σ_I to the chosen set of categories. Formally, $M_I : \Sigma_I \rightarrow \mathcal{C}_I$ as $M_I(s) = \rho_j^I$ where $s \in \rho_j^I$. Then, ρ_j^I is said to be the *category* of state s.

We adopt the following convention for naming each category in $\mathcal{C}_I$.

1. If $\mathcal{C}_I$ is the set of singletons, we denote each category by the singleton. Category $\{f\}$ is denoted as f. This conforms to the notion that a state and its category are indistinguishable when the state and only that state is mapped to the category.

2. If $\mathcal{C}_I$ is composed of two sets and one of the set is a singleton, we denote the singleton as in item(1) and the other set as the complement of the singleton. Thus, if the singleton is $\{a\}$, the categories are denoted as a and $\overline{a}$.

3. If $\mathcal{C}_I$ is composed of more than two sets and only one of the sets is not a singleton, we denote all the singletons as in item (1) and denote the remaining set as the complement of the union of all the singletons.

4. If $\mathcal{C}_I$ is none of the above, we refer to a category as a composite of its elements. We denote category $\{c, d, e\}$ as c_d_e.

Example 7 *Consider* $\Sigma_1 = \{a,\ c,\ d,\ e,\ f,\ g\}$ *in Example 6. Five examples of a partition of* Σ_1 *are the sets* $T = \{\{a\},\{c\},\{d\},\{e\},\{f\},\{g\}\}$, $T' = \{\{f\},\ \{a,\ c,\ d,\ e,\ g\}\}$, $T'' = \{\{a\},\{c\},\{d\},\{e\},\{f,g\}\}$, $T''' = \{\{f,c\},\ \{a,\ d,\ e,\ g\}\ \}$, *and* $K = \{\{a,\ c,\ d,\ e,\ f,\ g\}\}$. *Either* T, T', T'', *or* T''' *can be used as* $\mathcal{C}_1$. *On the other hand, although* K *is a partition of* Σ_I, $K \notin \mathcal{P}_1$ *because* $\| K \| < 2$. *Hence,* K *can not be used as* $\mathcal{C}_1$.

Using T *as* $\mathcal{C}_1$, *we have* $\rho_1^I = \{a\}$, $\rho_2^I = \{c\}$, $\rho_3^I = \{d\}$, $\rho_4^I = \{e\}$, $\rho_5^I = \{f\}$, *and* $\rho_6^I = \{g\}$. *Following item (1) of the naming convention, the categories are* **a**, **c**, **d**, **e**, **f**, *and* **g***m, respectively. We have* $M_1(a) = a$, $M_1(c) = c$, $M_1(d) = d$, $M_1(e) = e$, $M_1(f) = f$, *and* $M_1(g) = g$.

Using T' *as* $\mathcal{C}_1$, *we have* $\rho_1^I = \{f\}$ *and* $\rho_2^I = \{a,\ c,\ d,\ e,\ g\}$. *Following item (2) of the naming convention, our categories are* $\mathbf{f}$ *and* $\overline{\mathbf{f}}$. *We have* $M_1(a) = \overline{f}$, $M_1(c) = \overline{f}$, $M_1(d) = \overline{f}$, $M_1(e) = \overline{f}$, $M_1(f) = f$, *and* $M_1(g) = \overline{f}$.

Using T'' *as* $\mathcal{C}_1$, *we have* $\rho_1^I = \{a\}$, $\rho_2^I = \{c\}$, $\rho_3^I = \{d\}$, $\rho_4^I = \{e\}$, *and* $\rho_5^I = \{f,g\}$. *Following item (3) of the naming convention, the categories are* **a**, **c**, **d**, **e**, $\overline{\mathbf{a} \vee \mathbf{c} \vee \mathbf{d} \vee \mathbf{e}}$. *We have* $M_1(a) = a$, $M_1(c) = c$, $M_1(d) = d$, $M_1(e) = e$, $M_1(f) = \overline{\mathbf{a} \vee \mathbf{c} \vee \mathbf{d} \vee \mathbf{e}}$, *and* $M_1(g) = \overline{\mathbf{a} \vee \mathbf{c} \vee \mathbf{d} \vee \mathbf{e}}$.

Using T''' *as* $\mathcal{C}_1$, *we have* $\rho_1^I = \{f,c\}$ *and* $\rho_2^I = \{a,\ d,\ e,\ g\}$. *Following item (4) of the naming convention, our categories are* **f_c** *and* **a_d_e_g**. *We have* $M_1(a) = a\_d\_e\_g$, $M_1(c) = f\_c$, $M_1(d) = a\_d\_e\_g$, $M_1(e) = a\_d\_e\_g$, $M_1(f) = f\_c$, *and* $M_1(g) = a\_d\_e\_g$.

For the dependent position, we will mostly use partitions composed of two sets (e.g., T' in Example 7) and partitions composed of singletons (e.g., T in Example 7). For the independent positions, as we shall see in Section 2.5), there is no favored pattern for the partition. Unless otherwise noted, we use the partition composed of singletons as the set of categories for any position.

2.3 Classification Function

The mapping function M_I, together with C_I, were defined to partition the position I. However, it appears that the categories of a position could also be used for partitioning *subsequences* in Π. One group, for example, could be for subsequences which have the state a in position -1, another for the subsequences with the state b in position -1, and so on.

Example 8 *Consider* Π *in Example 2 with* $F = 2$. *If the categories of position 1 were based on the presence (or absence) of state* **a** *or state* **d** *in position 1, we have a partition of* Π *into three sets:* $A = \{\ (2, \langle g,f,a,g\rangle), (2, \langle g,f,a,e\rangle), (2, \langle d,e,a,g\rangle), (1, \langle a,g,a,c\rangle)\}$ *is the set of subsequences with state* **a** *in position 1;* $D = \{\ (2, \langle a,e,d,c\rangle), (2, \langle d,c,d,e\rangle)\}$ *is the set of subsequences with state* **d** *in position 1; and* $\overline{(A \vee D)} = \{\ (1, \langle g,g,f,a\rangle), (1, \langle f,a,g,f\rangle), (3, \langle a,g,f,a\rangle), (2, \langle f,a,e,d\rangle), (2, \langle e,d,c,d\rangle, (2, \langle c,d,e,a\rangle), (2, \langle e,a,g,f\rangle), (1, \langle f,a,g,a\rangle), (1, \langle g,a,c,d\rangle)\}$ *is the set of subsequences with states which are neither* **a** *nor* **d** *in position 1.*

Example 8 shows how the categories of a position can be used as categories of the subsequences in an analysis set. The subsequences can be assigned to the categories of any position I (e.g. position 1, for this example). When M_I assigns the states of a position to $\mathcal{C}_I$, it also has the effect of mapping the subsequences into these categories. Thus, although our mapping function M_I was not defined to operate on subsequences, it could be used to categorize subsequences.

Our primary interest, however, is to uncover the relationships between the states of the independent positions and the states of the dependent position. To examine whether a state in an independent position affects a state in the dependent position, we need to isolate subsequences from the analysis set based solely on the states in the independent and dependent position. We will define a function which classifies subsequences in an analysis set based only on the states which occur in the dependent and a particular independent position. We can think of this function as analogous to the function M_I we had already defined.

We first present the function informally. The function takes a subsequence π from Π and examines two states in π: the state which appears in the dependent position and the state in the independent position. It categorizes π based on the categories of the dependent position and the categories of the independent position. The function, therefore, maps every subsequence in Π to an ordered pair composed of the categories of the dependent and independent position.

We now formally define the function as follows. Denote an analysis set as Π; a chosen independent position as I; the dependent position as D; an element of Π to be classified, as (k, π); the state in position D of π, as s_D; the state in position I of π, as s_I; the mapping function for I as M_I; and the mapping function for D as M_D. Then, the classification function $C : \Pi \to \mathcal{C}_I \times \mathcal{C}_D$ is

$$C((k, \pi)) = (\rho_i^I, \rho_d^D) \tag{2.1}$$

where $M_D(s_D) = \rho_d^D$ $(1 \leq d \leq \parallel \mathcal{C}_D \parallel)$ and $M_I(s_I) = \rho_i^I$ $(1 \leq i \leq \parallel \mathcal{C}_I \parallel)$. We say that C *classifies* π as (ρ_i^I, ρ_d^D) or π is *classified by* C as (ρ_i^I, ρ_d^D). The function partitions Π into disjoint populations by using the categories of the dependent position together with the categories of a given independent position.

Example 9 *Consider* Π *in Example 2 with* $F = 1$. *The positions are -2, -1, 0, and 1. Let position* -1 *be the predictor position* I. *Thus,* $\mathcal{C}_I$ ={g,f,a,e,d,c}. *Let position* $D = 0$ *have two categories,* $\mathcal{C}_D = \{\mathbf{a}, \bar{\mathbf{a}}\}$. *Consider the ordered pair* $(1, \langle g, g, f, a \rangle)$. *It has* **g** *in position -1 which belongs to category* **g**. *It has* **f** *in position 0 which belongs to* $\bar{\mathbf{a}}$. *Thus, we have* $C((1, \langle g, g, f, a \rangle) = (g, \bar{\mathbf{a}})$. *Consider the ordered pair* $(2,, \langle g, f, a, g \rangle)$. *It has* **f** *in position -1 which belongs to category* **f**. *It has* **a** *in position 0 which belongs to category* **a**. *Thus, we have* $C((1, \langle g, f, a, g \rangle) = (f, a)$.

2.4 Testing for Dependencies

The introduction stated that our aim is to uncover dependencies between states in a SSS. In this section, we formally define *independence* and present a method for testing whether two variables are independent.

2.4.1 The Null Hypothesis

Let I denote an independent position with categories $\mathcal{C}_I$, $\| \mathcal{C}_I \| = m$; D denote the dependent position with $\mathcal{C}_D$, $\| \mathcal{C}_D \| = n$; Π denote an analysis set; and p_{qr} denote the number of ordered pairs $o \in \Pi$ such that $C(o) = (\rho_q^I, \rho_r^D)$, $2 \leq q \leq m$ and $2 \leq r \leq n$. Then, the *hypothesis of homogeneity* [4] states that

$$p_{1r} = p_{2r} = \cdots = p_{mr} \qquad \text{for all } r = 1, \ldots, n.$$

Two positions I and D are considered to be *statistically independent* if and only if the hypothesis of homogeneity holds for the two positions.

Statistical analysis is used to determine the independence between the states in positions I and D of an analysis set Π. The null hypothesis, H_0, is that the occurrence of a particular state in position D is independent of the state in position I. The null hypothesis may be tested by using a *contingency table.* We first describe its construction and then proceed to use it for hypothesis testing.

Given an analysis set $\Pi = \{ (k, \pi) \mid \pi \in Sub'(S, W)\}$ with N subsequences, an independent position I with $\mathcal{C}_I$ ($\| \mathcal{C}_I \| = m$), a dependent position D with $\mathcal{C}_D$ ($\| \mathcal{C}_D \| = n$), we construct a table with m rows and n columns for the categories of positions $\mathcal{C}_I$ and $\mathcal{C}_D$, respectively. Using the Classification Function (Equation 2.1), the *observed frequency*, O_{ij}, for row i and column j of the table is given by

$$O_{ij} = \sum_{l=1}^{N} x_l \tag{2.2}$$

where $x_l = k$ iff $C((k, \pi)) = (\rho_i^I, \rho_j^D)$, otherwise $x_l = 0$.

For each row i, the observed frequency is summed to give the *row total.* The same is done for each column j to obtain the *column total.* The sum of the column or row totals is the *sample size*, N. This table is the contingency table for positions I and D.

Example 10 *Consider* Π *in Example 2 with* $F = 2$. *Let* $I = -1$ *be the predictor position and* 0 *be the dependent position in* Π. $\mathcal{C}_I = \{\{a\}, \{c\}, \{d\}, \{e\}, \{f\}, \{g\}\}$. *The number of categories of* I *is* $m = 6$. *In this example, we are only interested in the presence (absence) of state* $\mathbf{a}$. *Hence, we only consider the* $n = 2$ *categories* $\mathbf{a}$ *and* $\bar{\mathbf{a}}$ *for position* 0.

We use the categories of position -1 and position 0 to classify an individual in the population Π *using the Classification Function (Equation 2.1). Representing the 2* classes *of position 0 as the columns and the 6 classes of position -1 as the rows, we obtain the two-way contingency table in Table 2.1. The observed frequencies,* O_{ij}, *from* Π *are gathered for* $i = 1 \ldots 6$, $j = 1, 2$. *Consider row* $\mathbf{f}$ *for instance.* $O_{fa} = 4$ *because the following ordered* *pairs have* **f** *in position -1 and state* **a** *in position 0 and the sum of their numbers is 4:* (1, $\langle f,a,g,f\rangle$), (2, $\langle f,a,e,d\rangle$), *and* (1, $\langle f,a,g,a\rangle$). $O_{f\overline{a}}$ *is 0 because there are no subsequences with state* **f** *in position -1 other than those with state* **a** *in position 0. The numbers in parentheses are discussed later.*

Table 2.1: The contingency table for Example 10

Position −1	Position 0				Row Total
	a		$\overline{a}$		
g	1	(1.62)	5	(4.38)	6
f	4	(1.08)	0	(2.92)	4
a	0	(1.62)	6	(4.38)	6
e	2	(1.08)	2	(2.92)	4
d	0	(1.08)	4	(2.92)	4
c	0	(0.54)	2	(1.46)	2
Column Total	7		19		26

The previous example used an analysis set Π composed of all the subsequences of the SSS with window size 4. In the next example, we use a smaller analysis set.

Example 11 *Consider* Π' *in Example 3 with* $F = 2$. *Let* $I = -1$ *be the predictor position and 0 be the dependent position in* Π. *We choose this* Π *because we are only interested in both states* **a** *and* **d**. *We therefore consider only the subsequences of* Π *which have either state* **a** *or* **d** *in the dependent position. Our categories are* **a** *and* **d** *for position 0. The number of categories of* I *is* $m = 3$ *with* $C_I = \{\{g\}, \{a\}, \{d\}\}$.

We use position -1 as I and position 0 as D in the Classification Function. Representing the 2 classes *of position 0 as the columns and the 3 classes of position -1 as the rows, we* *obtain the two-way contingency table in Table 2.2.*

Table 2.2: The contingency table for states **a** and **d** of Example 11

Position −1	Position 0				Row Total
	a		*d*		
g	4	(2.33)	0	(1.67)	4
a	2	(2.33)	2	(1.67)	4
d	1	(2.33)	3	(1.67)	4
Column Total	7		5		12

Given a contingency table with m rows and n columns, the column total K_j for column j, $1 \leq j \leq n$, and the row total R_i for row i, $1 \leq i \leq m$, the *expected frequency* F_{ij} for a cell in row i and column j is given by

$$F_{ij} = \frac{K_j}{N} R_i, \quad \text{where N is the sample size.} \tag{2.3}$$

The expected frequencies are the figures enclosed in parentheses in Table 2.1 for Example 10 and Table 2.2 for Example 11. They are "expected" if and only if H_0 is true. Thus, a difference between the observed and expected frequencies is an *indicator* that H_0 is false.

2.4.2 The Chi-square Test

The chi-square statistic χ^2 is calculated to determine the likelihood of the difference between the expected and observed values. We use the symbol χ_0^2 to represent a particular value of the statistic. Given a contingency table with m rows and n columns, the calculated

χ_0^2 statistic for the table is

$$\chi_0^2 = \sum_{i=1}^{n}\sum_{j=1}^{m} \frac{(O_{ij} - F_{ij})^2}{F_{ij}}. \tag{2.4}$$

The χ^2 test is a simple test for uncovering evidence of associations [5] between variables. It is a measure of how the observed frequencies differ from the expected values of the frequencies. It is dependent on the magnitude of the difference between the observed and expected frequencies *and* on the number of items which contributed to the statistic.

Being a cumulative test (i.e., each cell in the table contributes to its value), the magnitude of χ^2 is directly proportional to the number of cells in the table with all things being equal. This is accounted for by the *degrees of freedom* denoted as v. The value of degrees of freedom measures the "number of ways in which a set of data can vary"[12]. For an $m \times n$ contingency table, $v = (m-1)(n-1)$ because the frequency in any cell in the table may be calculated from the frequencies of the other $m-1$ cells in the same row and the row total. The same is true for the column total and $n-1$ cells in the same column. We have $\chi_0^2 = 18.04$ and $v = 5$ for Table 2.1 for Example 10. Table 2.2 for Example 11 has $\chi_0^2 = 4.80$ and $v = 2$. The reader is directed to [4], [5], and [12] for more comprehensive discussions on the χ^2 test.

The χ^2 test for H_0 is considered *significant* if and only if the χ_0^2 statistic does not exceed a predetermined critical value. Otherwise, the test is considered *not significant.* A state in position D is deemed to be statistically independent or independent of the states in position

I if and only if the χ^2 test for H_0 is not significant.

The null hypothesis is that there is no interaction between the way the states are distributed between the positions -1 and 0. In other words, except for the differences in distribution which could be attributed to sampling error, there is no difference between the observed frequencies in the contingency table and the expected frequencies based on the null hypothesis.

2.4.3 Significance Test

We have already seen how the χ_0^2 statistic is calculated. From the formula, we can see that the value of the χ^2 statistic is inversely proportional to the likely truth of the null hypothesis. We have not yet, however, established the magnitude of the test statistic at which the null hypothesis should be rejected. Establishing this critical level is the domain of significance testing.

The primary objective is to balance the risk of rejecting the null hypothesis when it is in fact true (Type I error) with the risk of accepting the null hypothesis when in fact it is false (Type II error). The probability of a Type I error is denoted as α while the probability of a Type II error is denoted as β. The problem is that decreasing the probability of one type of error increases the probability of the other. The usual solution to this problem is to set α to a certain small value with the hope that β is also acceptably small [4]. For exploratory purposes, a high α level would be acceptable because it presents relationships which might be worth examining but not really significant. However, when rigor and correctness are the primary aims of the test, a high α would be unacceptable. Suffice it to say that it is the analyst's responsibility to set the value of α.

Let α be a certain percentage, say 5%. By establishing this 5% level, we have decided to consider a test statistic χ_0^2 which is so distant from 0 that, if the null hypothesis were true, a larger statistic would be computed less than 5% of the time. We consider, in other words, a less than 5% chance for the null hypothesis to be true as being so improbable that the null hypothesis should not be believed.

Two equivalent methods for testing the validity of the null hypothesis can be used. The first method calculates the probability that H_0 is true when $\chi^2 = \chi_0^2$. This probability value, called the *test statistic p-value* or *p-value* and denoted as p_0, is then compared with α. As had been previously discussed, $p_0 < \alpha$ would cause us to reject the null hypothesis. We call α as the *critical p-value* and denote it as p_{crit}. The other method is to calculate the *critical test statistic*, χ_{crit}^2, which is the χ^2 value for p_{crit}. For the same reasons as in the above, $\chi_0^2 > \chi_{crit}^2$ would be cause to reject the null hypothesis.

Our choice of method is largely influenced by the type of information we wish to obtain. Although both methods allow us to test the validity of the null hypothesis, the first method gives more information to the analyst. It shows the magnitude of the χ_0^2 value in terms of probability of H_0 being true. It presents the α which would lead an analyst to consider H_0 to be true. Without the p-values, the second method only allows us to say whether the null hypothesis is valid or not.

Consider Example 10. With $\alpha = p_{crit} = 0.05$ and $v = 5$, the critical test statistic χ^2_{crit} is 11.1, based on the χ^2 table in [4]. Interpolating from the same table, $\chi^2_0 = 18.04$ has $p_0 = 0.003$. Because $p_0 < p_{crit}$, by using the first method of testing, the null hypothesis is rejected. Equivalently, by using the second method of testing, we reject the null hypothesis because $\chi^2_{crit} < \chi^2_0$. The rejection of the null hypothesis means that the observed frequencies in the contingency table are so different from the calculated frequencies that it is too improbable for the hypothesis of homogeneity to be true for Π. We therefore conclude that there is a dependence between the state a in position 0 and the states in position -1.

By comparing the observed frequencies with the expected values of the frequencies in Table 2.1, we infer that H_0 is false largely due to state f in position -1. The sum of the differences between its observed frequencies and expected frequencies (4-1.08 and 0-2.92) is the largest contributor to the computed χ^2. Furthermore, we can be reasonably certain that when state f occurs in position -1, state a would also occur in position 0.

So far, we have seen how the significance test allows us to reject or accept the null hypothesis. Significance testing can also be used as a *strength-of-relationship* function [22]. Recall that a significant χ^2_0 means that the dependence between the dependent and predictor variables is strong enough to be believed. Given any two *significant* test statistics χ_0^2 and $\chi_0^{2'}$ from the respective tests of two hypotheses H_0 and H_0', the relationship between the two statistics is a measure of the relative strengths of the relationships between their respective dependent and predictor variables. That is, the magnitude of χ_0^2 is directly related to the degree that the the null hypothesis is rejected. In turn, this is directly related to the strength of the relationship between the dependent and predictor variables being tested. This function of significance testing will be used in the next sections.

2.5 Merges

Up to this point, we have used standard statistical techniques for testing relationships between the independent variables and the dependent variables. Sometimes, the χ^2 test concludes that there is no relationship between the dependent and independent position. However, a different choice of categories for the independent position could yield a significant result. In particular, categories which are very similar to each other will almost always yield a "no relationship" result because one cannot discriminate the effects of one category from the other. In some cases, the results of the test are due to the fact that no relationship actually exists. However, when the data analysis is exploratory as is true during experimentation, the "no relationship" result might have been caused by the unfortunate choice of categories for the variable.

Example 12 *Consider Table 2.3, which has $v = 9$. At $\alpha = 0.01$, the critical value is $\chi^2_{crit} = 21.66$. Because $\chi^2_0 = 19.06 < \chi^2_{crit}$, the null hypothesis can not be rejected. The conclusion is that the positions 0 and -1 are independent.*

Suppose, however, without knowing anything about Table 2.3, the analyst had chosen to consider **g** *and* **f** *as one category. This possibility is shown in Table 2.4. Table 2.4 has $\chi^2_0 = 17.11$ and $v = 6$. The critical value for this table, at $\alpha = 0.01$, is $\chi^2_{crit} = 16.81 < \chi^2_0$. Because the test statistic exceeds the the critical value of the test, the null hypothesis is rejected. The conclusion is that the positions 0 and -1 are not independent.*

We see that two different choices of categories for position -1 yield different results.

Table 2.3: Table with no composite categories, yielding $\chi^2_0 = 19.06 < \chi^2_{crit} = 21.66$

Position −1	Position 0				Row Total
	a	*b*	*c*	*d*	
g	7	39	107	17	170
f	13	34	112	17	176
a	11	30	93	26	160
e	13	34	80	36	163
Column Total	44	137	392	96	26

Table 2.4: Table for the same data as in Table 2.3 but with a composite of category g and f, yielding $\chi^2_1 = 17.11 > \chi^2_{crit} = 16.81$

Position −1	Position 0				Row Total
	a	*b*	*c*	*d*	
g_f	20	73	219	34	346
a	11	30	93	26	160
e	13	34	80	36	163
Column Total	44	137	392	96	26

In experiments, when an analyst relies more on intuition than on actual knowledge, the choice of categories could be quite arbitrary. At the data collection stage, an analyst could choose to collect together the occurrences of g and f considering them to be just one state g_f. The result would be Table 2.4. The analyst could have just easily chosen not to group f and g together and get the result as in Table 2.3. In the latter case, the analyst might run experiments and collect more data, perhaps explore new ways of determining how the different categories are related. However, the computational complexity of exhaustively testing all possible permutations of l categories of an independent position is presented by Kass in [18] to be $O(2^l)$. This is extremely prohibitive and would very likely curtail any data exploration.

We present a process called a *merge* which is crucial to building our CHAID-based model. It is a method to augment the statistical tests discussed in the previous sections. We characterize this as a heuristic technique because we can not use the test results as *exact* statistics. The merge algorithm uses knowledge gained from the results of previous tests to determine which categories to merge. As we shall see, our algorithm specifically modifies the testing procedure based on what is known about the sample. This merge algorithm reduces the computational complexity to only $O(l^3)$ through the judicious choice of categories to combine and an effective rule for terminating the exploration of other combinations of the categories. Integrated with the data analysis, this procedure allows the analyst to automatically explore the relationship between the categories of the variables using the *same* data. For systems used in experimentation, such a merge procedure appear would be advantageous.

The term *similar* is used throughout this section to describe states or categories. However, we postpone our definition of *similarity* until the latter part of the section. For now, we just assume the existence of a function by which two states can be ruled to be similar to or different from each other. We caution the reader that this use of the merge is a departure from the original use in the CHAID technique. We will return to this point later on after we discuss the process itself.

The iterative merge process operates on the set of categories (e.g., the states $\{g, f, a, e, d, c\}$) of a predictor position. Each iteration combines the least different categories to form a new compound category composed of two categories which are homogeneous with respect to how each affects the dependent position. The result of a merge is a new *set* of categories composed of the new category and the rest of the unmerged categories. The new set of categories is then used for the next iteration. The merge process is stopped when no similar categories can be found.

Rather than exhaustively testing *all* possible combinations of the categories, we only merge the categories which result in an increase in the association between the independent position and the dependent position. Because the most similar categories are combined in each iteration, the categories in the new set are those which are most different from each other. Therefore, at the end of a merge process, the presence of any association between the dependent and predictor variables is exaggerated. In other words, our merge process is a systematic way for finding the partition of the categories of the independent position which increases the chance of finding an association between the the dependent and independent positions. The merge process is really an automatic test for uncovering interactions between the categories of the predictor position based on how they affect the dependent position. The final result of a merge is *maximized configuration* for a particular independent position. As we shall see, the χ^2 test for significance testing is incorporated into the merge process.

The independent positions are ranked based on the p-values of their maximized configuration. The position with the smallest p-value is considered to be the *optimal* position. This is our heuristic for deciding which of the independent positions is the best predictor.

Given any two distinct but similar states s and t at the same position N, their replacement with a new state s' is called a *state merge.* The new state s' is called a *compound state.* A *compound merge* or a *merge* is the replacement with s' of any two distinct states s and t, either of which could be a compound or a non-compound state, in the same position N. The result of a merge is a *merged analysis set.*

Example 13 *Given* Π *in Example 2 with* $F = 2$. *The state merger of* **a** *and* **d** *in position* -1 *into a_d results in a new* $\Pi'' = \{$ (1, $\langle g,g,f,a \rangle$), (2, $\langle g,f,a,g \rangle$), (1, $\langle f,a,g,f \rangle$), (3,

$\langle a\_d,g,f,a\rangle$), (2, $\langle g,f,a,e\rangle$), (2, $\langle f,a,e,d\rangle$), (2, $\langle a\_d,e,d,c\rangle$), (2, $\langle e,d,c,d\rangle$, (2, $\langle a\_d,c,d,e\rangle$), (2, $\langle c,d,e,a\rangle$), (2, $\langle a\_d,e,a,g\rangle$), (2, $\langle e,a,g,f\rangle$), (1, $\langle f,a,g,a\rangle$), (1, $\langle a\_d,g,a,c\rangle$), (1, $\langle g,a,c,d\rangle$)}.

The merged analysis set Π'' *has the same number of positions as the original analysis set* Π. *However, as a result of merging a and d in position* -1, *only five categories remain for this position. The categories for the other positions remain as in* Π.

The use of automatic interaction detection (AID) and the χ^2 (Chi) test gives us the CHAID *part* of the name of our model. As originally described by Kass [18], CHAID is a technique which "partitions the data into mutually exclusive, exhaustive, subsets that best describe the dependent variable" [18]. This is achieved through the following steps

1. finding the most significant partition of each predictor;

2. comparing the predictors with each other based on each predictor's most significant partition and choosing the most significant partition among all the partitions as the optimal position and

3. partitioning the data using the merged categories of the optimal position.

In our algorithm, although we follow the first two steps listed above, we do not use the merged categories for partitioning the data. Rather, we use the merged categories for deciding which position is to be used for partitioning the data. The original *unmerged* categories are used for partitioning the data. The main reason for presenting the model's results based on the original categories is our desire to present information to an analyst in terms of what the analyst is familiar with. The "problem" states that an analyst would be interested in studying are the original unmerged states as found in the SSS rather than the synthetic compound states that our model-building algorithm develops which, although based on the SSS, do not exist in the SSS.

The merge process prevents the interpretation of the χ^2 as a test statistic because it specifically modifies the χ^2 values and the degrees of freedom based on what is known about the sample. We thus use the χ^2 values as the basis of our decision rule for determining which position is chosen as *the* predictor of the dependent position. Because of these two differences, our model is CHAID-based rather than CHAID.

Types of Merges

The first type, called a *free merge*, is a method which attempts to merge each category of the predictor variable with any other category. *All* possible pair-wise merges are tried to find the two states which are most similar to each other.

Example 14 *Consider a two-component SSS composed of the percentage of refused connections in a network server and the number of concurrent transactions the server is processing. We list the the categories of the predictor variable, the percentage of refused connections as* $0, 10, 20, \ldots, 100$. *Because the list is sorted in ascending order, we know that adjacent val-*

ues, say 0 *and* 10*, are more similar to each other than values not adjacent to each other, say* 0 *and* 100*. In fact, attempting to merge states* 0 *and* 100 *together makes very little sense because these two values are the complete opposites of each other. In this example, a relationship between the states are implied in their ordering.*

In general, the free merge would be the applicable type for the analysis of SSS's if the way the states are grouped together does not reflect the idea of a scale. When the grouping of the categories of the predictor variable implies the idea of a scale or magnitude (i.e. the categories are listed in some order), then adjacency is indicative of similarity. That is, two states which follow each other are more similar to each other than two states which do not follow each other. Typically, it does not make sense to merge categories which are not adjacent to each other in the list. This second type, called a *monotonic merge*, confines the merges to categories which are adjacent to each other.

Some SSS's can be analyzed using either type. In these cases, it is the analyst's aim and understanding of the data which would be the basis for preferring one type over the other.

Example 15 *Consider a simplification of the sliding-window protocol used for network communications. Two communicating nodes negotiate the number of packets, n. The receiver uses n to determine how many packets are to be sent before it considers unacknowledged data as lost and starts retransmitting. The sender, on the other hand, uses n to determine how many packets are to be received before acknowledging what had been sent.*

Note that one danger of a small window size is "packet starvation" where the ratio of data packets over control packets (e.g. requests for packet, acknowledgment packet) is small resulting in slow data transfers. On the other hand, a large window size could cause the sender to overrun the capacity of the receiver resulting in discarded packets and retransmission.

Consider a two-component SSS composed of the percentage the speed of data transfer in a network and the window size used, say 1 to 25. At first glance, it makes sense to monotonically merge the window size categories if they are already sorted (either in ascending or descending order). After all, a window size of 1 is more closely related to a size of 2 than it is to a size of 25. Therefore, a monotonic merge would show how the increase in window size relates to the speed of data transfer.

Using only a monotonic merge, however, might be shortsighted. Having too small a window size and too large a window size have the same net effects – slower network data transfer rates. Allowing the categories to freely merge might lead to the discovery of the optimal window size.

It is not the monotonicity of the categories of the predictor variable alone which determines when a monotonic merge should be used. The presence or absence of a monotonic relationship between predictor and dependent variables play an equally important part in deciding which type of merge should be performed. The previous example shows the possibility of having a monotonic predictor variable which does not have a monotonic relationship with the dependent variable. That is, at some point in the predictor versus dependent variable graph, a knee is reached such that an increase (decrease) in the magnitude of the predictor variable causes a decrease (increase) in the magnitude of the dependent variable. Unfortunately, the presence of such a knee is not so obvious in many cases. The analyst's knowledge about the data is ultimately the basis of preferring one type of merge over another.

Merge Algorithm

Given the analysis set Π with the m categories $I_1, \ldots, I_m$ of position I, let Π_{ij} be the merged analysis set created by the merger of categories I_i and I_j for $i = 1, \ldots, m$ and $j = 1, \ldots, m$, $j \neq i$. Let χ^2_{ij} be the χ^2_0 statistic for testing the null hypothesis on I and D of Π_{ij}. The categories I_i and I_j are said to be *not significantly different* if and only if χ^2_{ij} is not significantly large.

Example 16 *Consider* Π *in Example 2 with* $F = 2$. *There are* $\begin{pmatrix} 6 \\ 2 \end{pmatrix} = 15$ *possible* free *merges of the six categories of position* -1 *in* Π. *We show three of the possible merges, their associated contingency tables, and the* χ^2 *statistic when the dependent position is categorized into the states* a *and* $\overline{a}$.

Position -1	Position 0				Row Total
	a		$\bar{a}$		
g	1	(3)	5	(3)	6
f	4	(2)	0	(2)	4
Column Total	5		5		10

$\chi^2_{12} = 6.66$

Position -1	Position 0				Row Total
	a		$\bar{a}$		
a	0	(0)	6	(6)	6
d	0	(0)	4	(4)	4
Column Total	0		10		10

$\chi^2_{35} = 0$

For $\alpha = 0.05$ and a degree of freedom $v = 1$, χ^2_{crit} is 3.84. Because all the fifteen tables have the same number of rows and columns (hence the same v), they all have the same critical values. Of the three pairs, the merges show that only a and d are not significantly different from each other. The other two mergers have their χ^2_0 statistic beyond the critical value.

Note the column total of 0 for column a of the middle table which results in expected values of 0 for the cells in column a. Equation 2.4 is not defined for such a table. We therefore define the χ^2 value to be 0 for such cases.

The above example showed the use of free merge. The following shows the use of a monotonic merge on the same analysis set.

Position -1	Position 0				Row Total
	a		$\bar{a}$		
f	4	(2)	0	(2)	4
d	0	(2)	4	(2)	4
Column Total	4		4		8

$\chi^2_{25} = 8.0$

Example 17 *Consider* Π *in Example 2 with* $F = 2$. *The* $6 - 1 = 5$ *possible* monotonic *merges of the six categories of position* -1 *in* Π *are* g_f, f_a, a_e, e_d, *and* d_c. *We show three of the possible merges, their associated contingency tables, and the* χ^2_{12} *statistic.*

Position -1	Position 0		Row Total	
	a	$\overline{a}$		
g	1 (3)	5 (3)	6	$\chi^2_{12} = 6.66$
f	4 (2)	0 (2)	4	
Column Total	5	5	10	

Position -1	Position 0		Row Total	
	a	$\overline{a}$		
a	0 (1.2)	6 (4.8)	6	$\chi^2_{34} = 3.75$
e	2 (0.8)	2 (3.2)	4	
Column Total	2	8	10	

Position -1	Position 0		Row Total	
	a	$\overline{a}$		
d	0 (0)	4 (4)	4	$\chi^2_{56} = 0$
c	0 (0)	2 (2)	2	
Column Total	0	6	6	

As in Example 16, χ^2_{crit} *is 3.84. Of the three possible merges shown above, the merger of states d and c is the most insignificant. Note that the final table in this example is a similar case to the middle table in Example 16. Similarly, we define the* χ^2 *value for this table to be* 0.

The categories I_n and I_l are said to be *the least significantly different* categories if and only if χ^2_{nl} is not significant *and* $\chi^2_{nl} < \chi^2_{ij}$ for $1 \leq n \leq m$, $1 \leq l \leq m$, $i \neq n$, $i = 1, \ldots, m$, $j \neq l$, and $j = 1, \ldots, m$. Informally, two categories are not significantly different if the relationship between their position and the dependent variable is not significantly affected by their merger. The least significantly different categories are those whose merger has the least significant effect on the relationship between their position and the dependent variable. The possibility of finding a significant relationship between the predictor position and the dependent variable is increased when the least significantly different categories of the predictor position are merged together. A merger tests for the possibility that an interaction between the two merged variables exists which produces an effect on the dependent position.

Example 18 *Consider* Π *in Example 2 with* $F = 2$. *All the fifteen tables from the fifteen possible free merges of the categories of position* -1 *have the same degrees of freedom. Hence the* χ_0^2 *statistic is an indicator of the significance level of the association between the variables position* -1 *and* 0. *The table with the smallest value belongs to the categories which are the least different from each other.*

Three possible merges, **a** *with* **d**, **a** *with* **c**, *and* **d** *with* **c** *have the lowest* $\chi_0^2 = 0$. *Thus we may choose any of the three, say the merger of* **a** *and* **d**. *As a result of this merger, we now just have five categories* $\{g, f, a\_d, e, c\}$ *for position* -1. *The new contingency table is shown below.*

Note that the p-value, $p_0 = 0.003$, of the contingency table in Example 10 is greater than that of the contingency table in Example 18. The result of this merger is the exaggeration

Position -1	Position 0 a		$\overline{a}$		Row Total
g	1	(1.6)	5	(4.4)	6
f	4	(1.08)	0	(2.92)	4
a_d	0	(2.69)	10	(7.31)	10
e	2	(1.08)	2	(2.92)	4
c	0	(0.54)	2	(1.46)	2
Column Total	7		19		26

$\chi_0^2 = 16.68, v = 4, p_0 = 0.0022$

of the relationship between the predictor position and the dependent position.

The next iteration subjects the modified contingency table to the merge process. This process is repeated until the point when no insignificantly different pair of categories can be found. At this point, the p-values for each iteration are compared. The iteration with the smallest value has the grouping of the predictor position's categories which maximizes the association between the dependent position and the predictor position. With this grouping, the position is said to be *optimally merged.*

Example 19 *Consider the contingency table in Example 18. We show the sequence of contingency tables after the merger of the least significantly different categories of position* -1.

Position -1	Position 0 a		$\overline{a}$		Row Total
g	1	(1.6)	5	(4.4)	6
f	4	(1.08)	0	(2.92)	4
a_d_c	0	(3.23)	12	(8.77)	12
e	2	(1.08)	2	(2.92)	4
Column Total	7		19		26

$\chi_0^2 = 16.68, v = 3, p_0 = 0.00082$

Position -1	Position 0				Row Total
	a		$\overline{a}$		
g_e	3	(2.14)	7	(7.86)	10
f	4	(1)	0	(3)	4
a_d_c	0	(3.23)	12	(8.77)	12
Column Total	7		19		26

$\chi_0^2 = 15.32, v = 2, p_0 = 0.00047$

The next three possible merges, **g_e** *with* **f** *($\chi_0^2 = 5.6$),* **g_e** *with* **a_d_c** *($\chi_0^2 = 4.16$), and* **f** *with a_d_c ($\chi_0^2 = 16.0$) all exceed $\chi_{crit}^2 = 3.84$. Hence, no further merges are attempted and we stop iterating through the merge process.*

The p-values for all the tables (including the initial unmerged table in Example 10 and the table for Example 18) may now be compared. The smallest p-value is 0.00047*, the p-value of the final table in this example. We therefore use the grouping of the categories found in this table. Finally, we have the merged analysis set for position* -1*, as* $\Pi_2'' = \{$ (1, $\langle g\_e,g,f,a\rangle$), (2, $\langle g\_f,f,a,g\rangle$), (1, $\langle f,a,g,f\rangle$), (3, $\langle a\_d\_c,g,f,a\rangle$), (2, $\langle g\_e,f,a,e\rangle$), (2, $\langle f,a,e,d\rangle$), (2, $\langle a\_d\_c,e,d,c\rangle$), (2, $\langle a\_e,d,c,d\rangle$, (2, $\langle a\_d\_c,c,d,e\rangle$), (2, $\langle a\_d\_c,d,e,a\rangle$), (2, $\langle a\_d\_c,e,a,g\rangle$), (2, $\langle g\_e,a,g,f\rangle$), (1, $\langle f,a,g,a\rangle$), (1, $\langle a\_d\_c,g,a,c\rangle$), (1, $\langle g\_e,a,c,d\rangle$)$\}$.

The monotonic merge of the states of the same predictor variable is shown in the next example.

Example 20 *Again, consider Π in Example 2 with $F = 2$ and the independent position* -1*. We show the sequence of contingency tables after the monotonic merger of the least significantly different categories.*

Position -1	Position 0 a		$\overline{a}$		Row Total
g	1	(1.6)	5	(4.4)	6
f	4	(1.08)	0	(2.92)	4
a	0	(1.6)	6	(4.4)	6
e	2	(1.08)	2	(2.92)	4
d_c	0	(1.6)	6	(4.4)	6
Column Total	7		19		26

$\chi_0^2 = 16.68, v = 3, p_0 = 0.00022$

Position -1	Position 0 a		$\overline{a}$		Row Total
g	1	(1.6)	5	(4.4)	6
f	4	(1.08)	0	(2.92)	4
a_e	2	(2.69)	8	(7.31)	10
d_c	0	(1.6)	6	(4.4)	6
Column Total	7		19		26

$\chi_0^2 = 13.63, p_0 = 0.00345$

The possible merger of **g** *and* **f** *has* $\chi_0^2 = 6.66$ *while that of* **f** *and* **a_e_d_c** *has* $\chi_0^2 = 11.66$. *Both are beyond the* χ_{crit}^2 *of 3.84. This last table is the final iteration through the monotonic merge process. Of the p-values of all the tables (including the one in Example 10, the smallest is the one for the last table in this example. We therefore accept the grouping in this last table as the best configuration for the states of position* -1.

From the examples for the free and monotonic merge, we can conclude that the use of different merge types could lead to two different groupings of the categories of the predictor

Position -1	Position 0 a		$\overline{a}$		Row Total
g	1	(1.6)	5	(4.4)	6
f	4	(1.08)	0	(2.92)	4
$a\_e\_d\_c$	2	(4.3)	14	(11.7)	16
Column Total	7		19		26

$\chi_0^2 = 12.87, p_0 = 0.00165$

position. The p-values for the optimally merged configuration of the predictor variable can not be guaranteed to be the same for both the free ($p_0 = 0.00047$) and monotonic merge ($p_0 = 0.00165$).

We point out two important considerations resulting from the merge process.

1. The merge process is iterative and takes advantage of the sample population's characteristic in order to find the succeeding merges. Once we create new groupings based on previous groupings, we are prevented from further use of the χ^2 test as an *exact* probability test. Thus, after the first merge, the χ^2 statistics of the succeeding contingency tables must be considered as *approximate* tests. Our χ^2 tests are decision rules rather than exact tests of the null hypothesis at the α level. Kass [18] reports successful uses of these approximations in simulation and in practice.

2. The use of $(m - 1)$ multiple χ^2 testing violates the α level for the experiment. We will show in the next section how the the α level may be adjusted to account for this.

2.6 The Optimally Merged Analysis Set

All our previous examples have shown the merge process for position -1. By subjecting the other two predictor positions (1 and 2) of the analysis set Π in Example 2 to the same merge process that has been described for position -1, the optimal grouping of states for each position will eventually be obtained. At that point, our merged analysis set would be composed of positions which are optimally merged. We formalize this notion with the following definition.

Let Π be an analysis set with window size W and the positions $\{P_k, P_{k+1}, \ldots, P_{k+W-1}\}$, $0 \leq k+W-1$, and Π'' be a merged analysis set of Π. Π'' is an *optimally merged analysis set* if and only if every position P_j, is optimally merged, $k \leq j \leq k+W-1$ and $j \neq 0$. That is, when all the *predictor* positions of Π'' are optimally merged, Π'' is optimally merged.

Example 21 *Consider Example 2 with its contingency table Example 10. We have already shown in Example 19 the optimally (free) merged position* -1. *We list here the merged categories for the predictor positions* 1 *and* 2 *obtained from the application of the merge process for the two positions. For position* 1, *the three categories are* **g**, **f_a_d** *and* **e_c**. *For position* 2,*the two categories are* **g_a_e_c** *and* **f_d**.

Π'' = { (1, ⟨*g_e*,*g*, *f_a_d*, *g_a_e_c*⟩), (4, ⟨*g_e*, *f*, *f_a_d*, *g_a_e_c*⟩), (1, ⟨*f*, *a*, *g*, *f_d*⟩), (4, ⟨*a_d_c*, *g*, *f_a_d*, *g_a_e_c*⟩), (2, ⟨*f*,*a*, *e_c*, *f_d*⟩), (4, ⟨*a_d_c*, *e*, *f_a_d*, *g_a_e_c*⟩), (2, ⟨*g_e*, *d*, *e_c*, *f_d*⟩), (2, ⟨*g_e*, *a*, *g*, *f_d*⟩), (1, ⟨*f*,*a*,*g*,*g_a_e_c*⟩), (1, ⟨*g_e*, *a*, *e_c*, *f_d*⟩)}

Recall that our objective is to determine which predictor position has the strongest association with the state a of the dependent position. To achieve this, each position was subjected to the merge process so that each position's association with state a was maximized. After each predictor position had been optimally merged, we were able to get the optimally merged analysis sequence for state a as seen in Example 21. To determine which predictor position has the strongest relationship with state a, we again use the χ^2 test. Our null hypothesis remains the same, H_0 : the predictor position is independent of state a in the dependent position. However, we now use the optimally merged analysis set as our population which in turn affects how the significance level is measured.

2.6.1 Bonferroni Adjustment

The type I probability error rate α is set on the assumption that there is only one population from which we collect one sample and perform one test for the validity of the null hypothesis. Repeated testing of the same sample, as had been done in the previous subsections, violates the one population, one sample, and one test assumption [9]. We increase the probability of finding a relationship (i.e. rejection of the null hypothesis) simply by increasing the number of times we search for such a relationship – by chance and not because such a relationship actually exists. This either *increases* the possibility of finding spurious relationships or, when such relationships actually exist, *magnifies* their strengths. As a result, the α level initially set is no longer reflective of the desired error rate.

To counter this increased chance of being in error, α must be divided by an adjustment factor called the *Bonferroni adjustment factor*, to maintain the desired error level [9]. The result would be a new type I error rate α' which is stricter (i.e., smaller) than the initial α.

Thus we have,

$$\alpha' = \frac{\alpha}{B(m)} \tag{2.5}$$

where α is the target type I error level, m is the number of states in the predictor position, $B(m)$ is the Bonferroni adjustment factor, and α' the new type I error level.

In our step-wise procedure, every merger reduces the number of possible states that can be merged in the next step. At the r^{th} step, there are $m - r + 1$ states from which 2 states are to merged together to form $m - r$ states for the next step. We denote this number as $N(m - r + 1, m - r)$. For a free merge, we have the combination

$$N(m - r + 1, m - r) = \frac{(m - r + 1)!}{(m - r - 1)!2!}. \tag{2.6}$$

We have

$$N(m - r + 1, m - r) = m - r \tag{2.7}$$

for a monotonic merge. This gives the Bonferroni adjustment factor derived in [9]

$$B(m) = 1 + \sum_{r=1}^{m-2} N(m - r + 1, m - r) \tag{2.8}$$

where r is the possible number of times a merger may be performed (a minimum of two groups must be left after every merger), $N(m - r + 1, m - r)$ is the number of ways of grouping two categories of the $m - r + 1$ categories into $m - r$ categories (dependent on the type of merge).

Example 22 *Consider Example 10. The Bonferroni adjusted critical p-value for* $\alpha = 0.05$ *is*

$$\alpha' = \frac{0.05}{1 + \sum_{r=1}^{4}(6-r)} = 0.0033$$

for a monotonic merge. For a free merge we have

$$\alpha' = \frac{0.05}{1 + \sum_{r=1}^{4}\frac{(7-r)!}{(5-r)!2!}} = 0.0014$$

2.6.2 The Optimal Position

Let Π'' be the optimally merged analysis set with window size W and the relative positions $\{P_k, P_{k+1}, \ldots, P_{k+W-1}\}$, $0 \leq k + W - 1$. Let P_0 as the dependent position. Let p_i be the p-value of χ_i^2, the χ_0^2 test statistic for testing the null hypothesis on P_i and P_0 and p_{least} the smallest p-value from among all p_i where $i = k, \ldots, k + W - 1$, $i \neq 0$. The *candidate set* is defined to be $P_{can} = \{P_i \mid p_i = p_{least}\}$. The *optimal position* P_{opt} is:

1. P, if and only if $P_{can} = \{P\}$;

2. P, if and only if $P \in P_{can}$ and $|P| < |P_i|$ for all $P_i \in P_{can}$, $P_i \neq P$;

3. $-P$, if and only if $P_{can} = \{P, -P\}$ and $P > -P$.

In other words, the optimal position is chosen to be the position with the smallest p-value which is closest to the dependent position. If there are two such closest positions (e.g., P and $-P$), the predecessor position of the dependent position is chosen. We default to the predecessor position to maintain some form of similarity with the output of the CTSM model. This would allow an analyst to cross-validate the results of the CHAID-based model with CTSM model.

Example 23 *Consider the optimally merged analysis set Π'' in Example 21. The contingency tables for the best grouping of the states for each predictor position using a free merge are Tables 2.5, 2.6, and 2.7.*

Table 2.5: The best table for position 2 of Example 23

Position 2	Position 0				Row Total	
	a		$\overline{a}$			
f_d	6	(2.15)	2	(5.85)	8	$p_2 = 0.00023$
$g\_a\_e\_c$	1	(4.85)	17	(13.15)	18	
Column Total	7		19		26	

Table 2.6: The best table for position 1 of Example 23

Position 1	Position 0				Row Total	
	a		$\overline{a}$			
f_a_d	0	(4.04)	15	(10.96)	15	
g	4	(2.96)	0	(8.04)	4	$p_1 = 0.00017$
e_c	3	(1.88)	4	(5.12)	7	
Column Total	7		19		26	

Table 2.7: The best table for position -1 of Example 23

Position -1	Position 0				Row Total	
	a		$\overline{a}$			
g_e	3	(2.14)	7	(7.86)	10	
f	4	(1)	0	(3)	4	$p_{-1} = 0.00047$
a_d_c	0	(3.23)	12	(8.77)	12	
Column Total	7		19		26	

Ranking the p-values from the tables, we get $p_1 < p_2 < p_{-1}$. We conclude that the optimal position for state a of the dependent position is position 1.

Each table in Example 23 was tested independently of each other. We had chosen the optimal position from among all the positions we had tested. Once again, we had violated the one population, one sample, one test assumption of significance testing. Again, we have to adjust the significance level for testing position 1 by using the Bonferroni adjustment to account for this violation. To achieve the desired significance level of α for testing the optimal position, we have (adapted from [9]) an adjusted critical value

$$\alpha' = \frac{\alpha}{(N_v)B(j)} \tag{2.9}$$

where N_v is the number of predictor positions tested and $B(m)$ is the Bonferroni adjustment factor for the chosen predictor position. For position 1 of Example 23, we have

$$\alpha' = \frac{0.00143}{3} = 0.00048$$

and because $p_1 < \alpha'$, p_1 is still significant.

As a final note, the Bonferroni adjustment factor accounts for the number of tests we had performed but does *not* adjust for the merges of the categories which take advantage of sampling error.

2.6.3 Classifiers

As previously mentioned, the categories of a predictor position partition the analysis set into mutually exclusive and exhaustive subsets which describe the analyzed state(s) in the dependent position. The optimally merged analysis set was created to be the basis for comparing the degrees of association of each predictor position with the dependent position. Among all the predictor positions, the categories of the most significant position is the *best* description of the analyzed state(s) in the dependent position.

One result of the merges is the creation of compound states from the original states found in the independent position of the SSS. For example, the state *f_d* in Example 23 does not in fact exist in the SSS in Example 2 from which the merges are based. We assume that an analyst's familiarity with the states as they appear in the original SSS, make the original states better frames of reference than the merged states would. In view of this, the output of the model is based on the original unmerged states found in the SSS. Once the optimal position has been chosen, the optimally merged analysis set becomes useless. It is discarded and the analysis set reverts to the original unmerged set. In our succeeding examples and definitions, we revert to using Π in Example 2 as our analysis set rather than using Π'' in Example 21.

A *classification rule*, based on the definition in [11], is a partition of a population X into n disjoint subsets $X_1, \ldots, X_n$, $X = \bigcup_n X_n$ such that for every $x \in X_j$, the predicted class is j. We already have the states of the ordered pair of the state in the optimal position and the state in the dependent position as our classifier. The classification rule for the CTSM states is the Classification Function (Equation 2.1), with position 1 as the predictor position.

Example 24 *Consider* Π *in Example 2. From Example 23, we know that the optimal position is position* 1. *Using the Classification Function with position* 1 *as the predictor position, we have the classification of the first four subsequences of* Π: $C(\langle g,g,f,a\rangle) = (f,g), C(\langle g,f,a,g\rangle) = (a,f), C(\langle f,a,g,f\rangle) = (g,a), C(\langle a,g,f,a\rangle) = (f,g)$

The ordered pairs (f,g), (a,f), *and* (g,a) *are obtained by discarding the states from the* other *independent positions.*

Our CHAID-based algorithm assures us that this ordered pair composed of a state in the optimal position and the analysis state (i.e. state a) in the dependent position is the best general classifier.

2.7 Types of Analysis

The examples we have shown were based on Example 10 where the the absence (presence) of a single state (i.e., state a) determined the population. The result was a contingency table with only two columns a and $\overline{a}$. How would an analysis proceed when we have a set of states to be analyzed rather than just a single state?

Two methods present themselves. We first describe the method called the *independent analysis*. Denote the set of states to be analyzed as Δ; recall that the window size is W and that the dependent position is D. Then, for every $\delta \in \Delta$, the the statistical technique described in this chapter, beginning from Section 2.4 to Section 2.5, is performed with:

1. $\mathcal{C}_D = \{\delta, \overline{\delta}\}$;

2. $\Sigma_\delta = \mathcal{C}_D$;

3. $\Pi = Sub(S, W)$.

The method is called an independent analysis because for every δ, the analysis proceeds without considering the other elements of Δ. Because the optimal position is obtained during each analysis, there could possibly be $\| \Delta \|$ unique optimal positions, one for each state.

The second method is called the *joint analysis* which we now describe. Denote the set of states to be analyzed as Δ; the window size as W; and the dependent position as D. Then, *one* analysis is performed with:

1. $\mathcal{C}_D = \{\{\delta\} \mid \delta \in \Delta\}$;

2. $\Sigma_\delta = \Delta$;

3. $\Pi = \{\langle s_1, s_2, \ldots, s_W\rangle \in Sub(S, W) \mid s_D \in \Delta, 1 \leq D \leq W\}$.

This method analyzes the set Δ as a group rather than analyzing each individual states in the set. The population is determined based on *all* the states in Δ. Example 3 shows how such a population is determined with $\Delta = \{\ a,\ d\ \}$. Example 11 presents the contingency table from such a population. The test results are therefore obtained for the group rather than the individual states in the group.

2.8 Summary

In section 1.3, we had mentioned that the limitation of the CTSM model stems from the fact that the transitions to a state are based only on the state's immediate predecessor. Hence a state's association is confined only to that predecessor. We illustrated in this section that our proposed CHAID-based model has a "window of analysis" of arbitrary size compared to CTSM model which only considers two states at a time: a state and its successor. By specifying a window size W and specifying the set of states A to be analyzed, the CHAID-based algorithm extends the search for any state associated with the members of A to include any number of predecessor or successor states of each member. Through the use of the χ^2 for significance testing, the most associated state encompassed within W could be determined. Furthermore, the analyst is allowed to explore the results of using different window sizes for the best model results.

This chapter illustrated how the CHAID statistical technique was adapted for analyzing states in a SSS. Section 2.1 began the chapter with a discussion of how the sample population, the analysis set, is obtained from the SSS. It then proceeded to define the positions in the analysis set as our variables. Section 2.3 defined how subsequences of the analysis set are classified based on the categories of the dependent variable. This was followed by the definition of statistical independence in Section 2.4. The method for testing statistical independence through the use of contingency tables and the χ^2 test was also presented in this section. Section 2.5 discussed the use of automatic interaction detection as a means of maximizing the effects of the predictor variable on the dependent variable. The criterion for selecting one of the many independent positions as the optimal position for predicting the dependent position was then presented in Section 2.6. Finally, the two methods for analyzing a group of states in the dependent position were discussed in Section 2.7.

Chapter 3

The CHAID-based Model

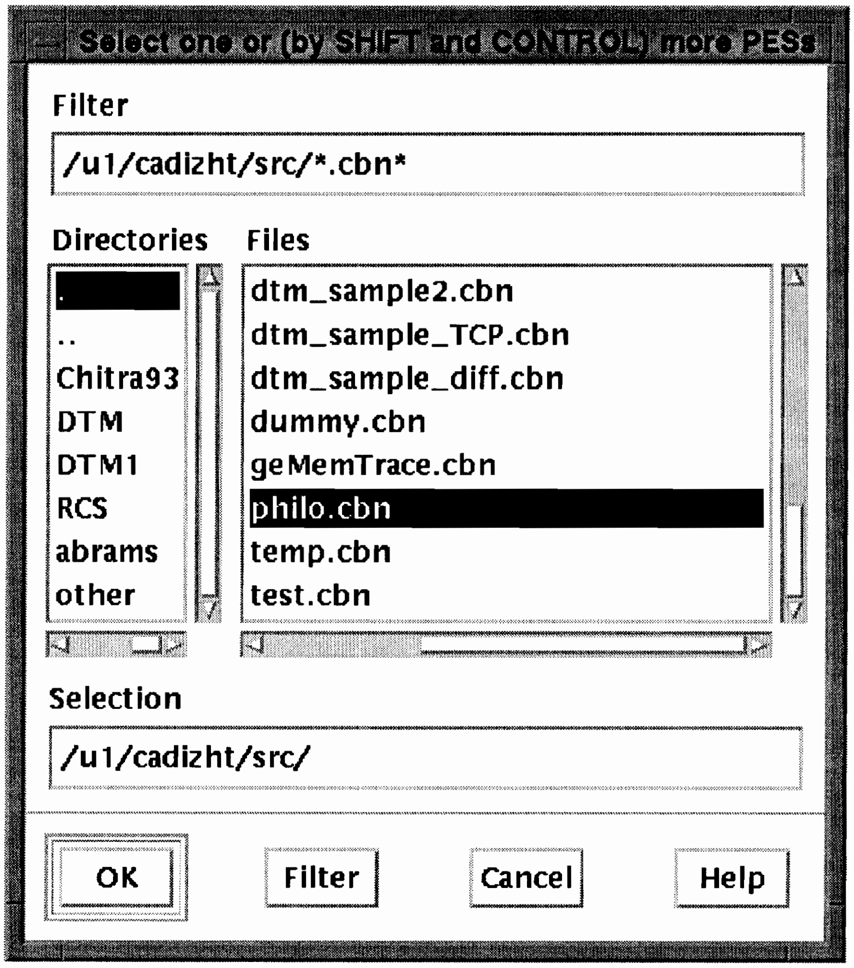
The previous chapter discussed the theoretical basis of the CHAID-based model. In this chapter, we present an overview of the CHAID-based modeling method as it is integrated into CHITRA93.

3.1 Generating a CHAID-based Model

We used a short SSS in Example 2 to make it easy to follow the discussions in our examples. For our tour of the CHAID-based model, we use a SSS generated from an actual parallel program developed by the CHITRA group as a solution to Djikstra's "Dining Philosophers" problem which we had introduced in Section 1.2. The SSS is as discussed in Example 1.

3.1.1 Specifying the Parameters

Figure 3.1 shows the opening screen where the user selects the SSS to be analyzed. The selected file *philo.cbn* is the binary format representation of the SSS generated by the above program. Recall that CHITRA uses a SSS as the representation of the data generated from a program. The CHAID-based model strips the time stamps from the TSSS to generate the SSS it uses.



/u1/cadizht/src/*.cbn*

Directories	Files
.	dtm_sample2.cbn
..	dtm_sample_TCP.cbn
Chitra93	dtm_sample_diff.cbn
DTM	dummy.cbn
DTM1	geMemTrace.cbn
RCS	philo.cbn
abrams	temp.cbn
other	test.cbn

/u1/cadizht/src/

Figure 3.1: Selecting a SSS to be modeled

When the user selects the *OK* button from the window in Figure 3.1, the display for the CHAID-based model view is created (Figure 3.2). The user is presented with a dialog box from where the parameters for generating the model may be selected or input. The dialog box shown in the window labeled "Model Parameters" in Figure 3.2, when first presented to the user, contains the default settings of the different parameters. The grey color shows which of the choices are currently active. The following subsections explain each element of the dialog box.

Merge Method

The user may select the type of merge to be used, *Free* or *Monotonic*, by clicking on the appropriate button. Section 2.5 explains the difference between these two types of merge. In this sample analysis, we choose a *Free* merge because the Dining Philosophers states are not monotonic.

Type of Analysis

There are two different ways of analyzing the selected states as discussed in Section 2.7. An *Independent analysis* analyzes the effect of the categories of the independent positions based solely on each state without regard to any other state which may have been selected. *Joint analysis* considers all the selected states as a group and analysis is performed on the group rather than on the individuals in the group. We choose to perform a joint analysis on the two states we had selected so we click (not shown) on the *Joint* button of the parameter box.

Dependent position

Recall that the size of the "window of analysis", W, was defined in terms of the number of steps in the future or the past (Section 2.1). Given a state to be analyzed, its predecessor and/or successor states may be analyzed for the associations. Choosing *Analyze past* restricts the search for the states in the previous positions. This sets the value for H, the number of predecessor states to analyze, to whatever is chosen by the user. The number of successor states, F, is then automatically set to 0. This results in $W = H$. Conversely, *Analyze future* restricts the analysis to only the successor states. This sets the value for F to the number chosen by the user while automatically setting $H = 0$. This results in $W = F$. *Analyze both* allows the analyst to search for associations with both the successor and predecessor states.

Clicking on any of the three buttons pops up a dialog box which allows the user to enter the desired window size. For *Analyze future* and *Analyze past*, the user is allowed to enter only one value, F or H respectively. *Analyze both* allows the user to enter the values for F and H. Figure 3.3 shows the dialog box that pops up when the user clicks on *Analyze both* of the *Dependent position* section. The user may then enter the number of positions to include in the analysis.

The computational cost of the CHAID-based analysis is directly proportional to the window size (Appendix A). The choice of a window size must be governed by the time the user wants to allow for analysis together with the estimate of the distance of the optimal position for the state(s) to be analyzed. In Figure 1.2, we showed the probabilistic transition from the model state $A2$ to states ES and EL. Because we want to examine the relationship between states TL and TS with the states ES and EL, we want to ensure that the subsequences we examine include $\langle TS, A1, A2, ES\rangle$ and $\langle TL, A1, A2, EL\rangle$, which are both of length 4. We thus need to define our window size to be greater than or equal to 4. We choose *Analyze future* and enter 4 as the number of positions to analyze as shown in Figure 3.4.

Significance level

The α level for significance testing is entered here. It is set to 5% by default. This means that 1 of 20 times we perform a test of the null hypothesis, we may reject the null hypothesis when it is in fact true. This is the Type I error rate which is fully discussed in Section 2.4.3. Note, however, that this value will be adjusted to account for multiple testing (Section 2.6.1).

List of states to analyze

The user chooses the states to be analyzed through the *List of states to analyze* part of the "Model Parameters" window. The tool determines what states in the SSS actually appear in the dependent position of the SSS. The *All* button selects all the states in the dependent position for analysis. Typically, however, the user would choose to analyze only a subset of these states perhaps composed of some "problem" or other "interesting" states. In our case, we are interested in discovering what states are associated with a long/short think time. We click on *Specify* which pops up the dialog box shown in Figure 3.5.

For space saving reasons and ease of data manipulation, CHITRA assigns a unique number to each state in the SSS. The selection box shows the state names together with their corresponding state numbers. We refer to either as *states* although technically the numbers are CHITRA's representation of the states. Our SSS has eight states appearing in the dependent position. We select the states TS and TL because we suspect that these states determine the transitions from state $A2$. We click on *OK* after our selection.

3.1.2 Interpreting the Model View

There is no specific order on how the user should define the parameters. At any given point, the user may return to the different parameters and modify the selections. Clicking on *Apply* of Figure 3.1, generates the CHAID-based model using the parameters that had been set. The result is a three-paned window as shown in Figure 3.6. In this figure, we have the *N-step Transition Matrix* as the top pane, the table for the p-values in the middle pane, and a replica of the Markov-chain state occupancy time statistics based on a CTSM model.

1. ***N*-step Transition Matrix**

   We call the matrix shown in the top window pane the *N-step Transition Matrix* to distinguish it from the *Transition Matrix* generated by the CTSM model in CHITRA92. The column labels are the state numbers[1] which correspond to the state names as shown in the bottom window pane. These are the states which were chosen in the *List of states to analyze.* The rows are the state numbers for *all* the states which appear in the dependent position of the SSS.

   - The numbers in the brackets below the column numbers are the optimal positions. In our example, these numbers are all the same because we performed a *joint analysis.* They could possibly be different for an *independent analysis.* It would be more useful to think of the optimal position as the distance of the states in this position from the states in the dependent position. For example, the states in Position 1 may be considered to occur 1 step *later* than the states in dependent position, the states in Position -1 are 1 step *earlier* than the dependent position states, and so on.

[1]We use the state numbers rather than the state names to conserve screen real estate. The matrix would be difficult to read with state names of varying lengths. CHITRA actually allows the state names to vary up to a certain MAXLENGTH although all the state names in our examples are only two characters long.

- Interpreting the meaning of the matrix elements is dependent on the direction of the optimal position n. If $n < 1$, the matrix element m_{ij} in row i and column j is the percentage of transitions in the SSS *to* state j from state i some n *previous* steps. If $n > 1$, the element m_{ij} is the percentage of transitions *from* state j *to* state i some n *future* steps. In Figure 3.6, the N-step Transition Matrix shows that whenever the philosopher is in state TS, it will *always* (100% of the time) be in ES after 3 states. Similarly, the matrix shows that whenever the philosopher is in state TL, it will *always* be in EL after 3 states.

2. **P-value Table**

As an aid to understanding how the different independent positions compare to each other, another table is provided the user and is shown in the middle pane of Figure 3.6. The column labels are also the state numbers of the selected states. The row labels are the positions. For an *independent analysis*, the element m_{ij} in row i and column j is the p-value of the optimal i versus dependent position contingency table using the state j and $\bar{j}$ as the categories of the dependent position (Examples 2 and 10). The rule for selecting the optimal position is found in Section 2.6.2. For a *joint analysis*, the element m_{ij} is the p-value of the optimal i versus dependent position contingency table using *all* the states in the column as the categories of the dependent position

(Examples 3 and 11). This implies that all the elements across any row are all equal. The critical p-values shown are the significance levels α for the optimal contingency tables after the Bonferroni adjustments as discussed in Section 2.6.1. Figure 3.6 shows that the independent position 3 was the optimal position.

3. **Occupancy Time Statistics**

   The bottom window pane contains occupancy time statistics based on the CTSM model. Briefly, it contains the frequency of occurrence, the minimum duration, maximum duration, the sample mean of the duration, the sample variance for the duration, and the standard deviation of the duration of each state in the SSS. These statistics and their calculations are fully explained in [14].

3.1.3 The Embedded Chain of Model States

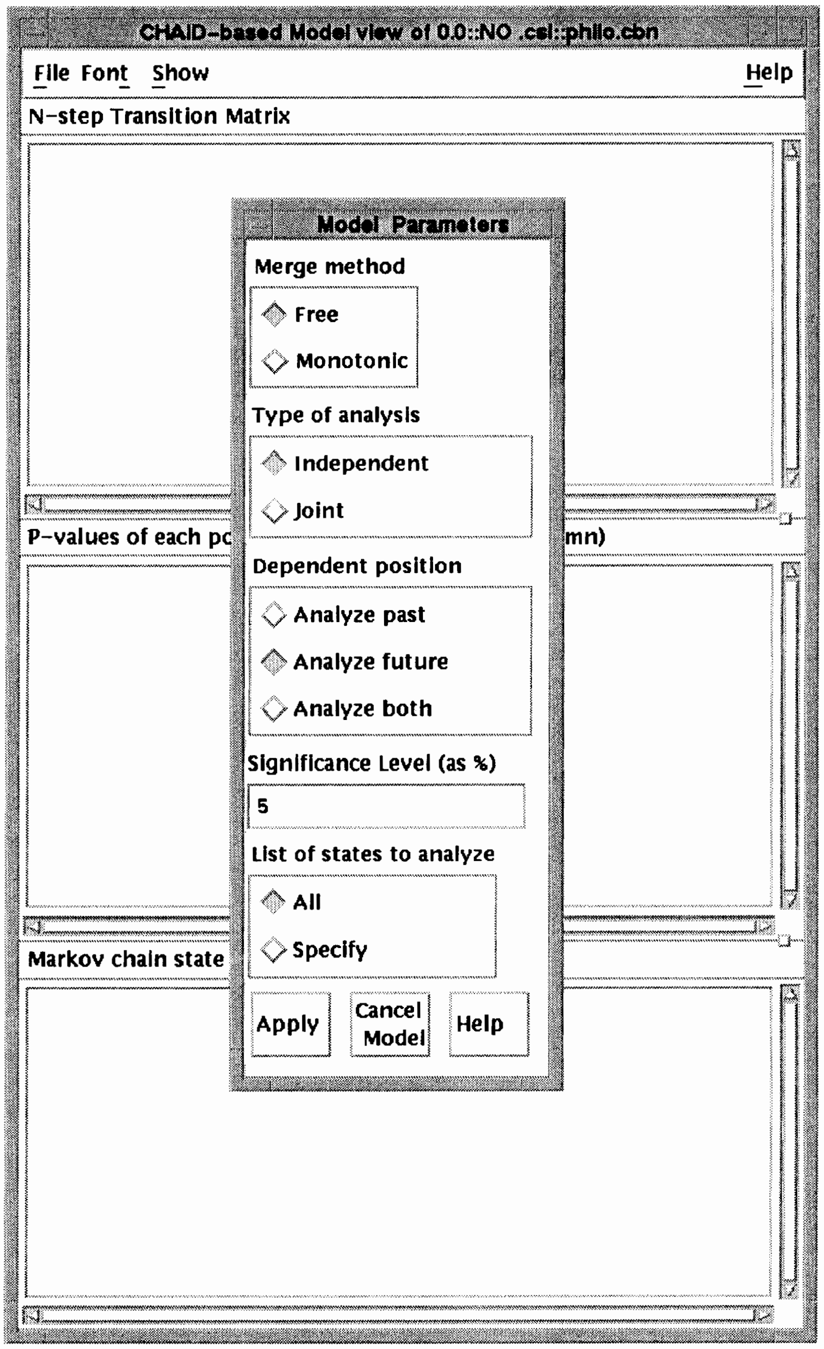
The generation of the embedded model chain for the CHAID-based model is relatively straight forward. The states in the chain are the states i and j for every element $m_{ij} \neq 0$ of the N-step Transition Matrix. If the optimal position n is greater than 0, then the direction of the graph is from state i to state j. If $n < 0$, then the direction of the graph is from state j to state i. Note that $n = 0$ can never occur because position 0 is the dependent position. Only the independent positions are analyzed for relationships with the dependent position. The embedded model chain for the N-step Transition matrix in Figure 3.6 is shown in

Figure 3.7. The graph is labeled with the optimal position. From the N-step Transition Matrix, we can see that ES is determined by state TS. Similarly, state EL is determined by state TL.

By replacing the probabilistic transition from state A in Figure 1.2 with the deterministic transitions in in Figure 3.7, we can generate Figure 1.3. In fact, because the transitions from state TL to ES and state TS to EL are completely deterministic we can aggregate the sequence of states beginning with TL (TS) and ending with ES (EL) into state $T1$ ($T2$) without any loss of information. That is, we can replace all the subsequences of length 4 which begin with state TL (TS) and end with state ES (EL) in the SSS with the new model state $T1$ ($T2$). The resulting model is shown in Figure 1.4.

3.2 Summary

Using the SSS from the Dining Philosophers program presented in Section 1.2 as the sample input for analysis, this chapter shows how a CHAID-based model is generated in CHITRA93. The reader is guided through the specification of the various parameters for developing a CHAID-based model. We then generated the model view – showing how an embedded chain model of the analyzed states could be obtained from N-step Transition Matrix.



CHAID-based Model view of 0.0::NO .csl::philo.cbn

File Font Show Help

N-step Transition Matrix

P-values of each pc

mn)

Markov chain state

Model Parameters

Merge method

[✓ Free] Monotonic

Type of analysis

[✓ Independent] Joint

Dependent position

Analyze past [✓ Analyze future] Analyze both

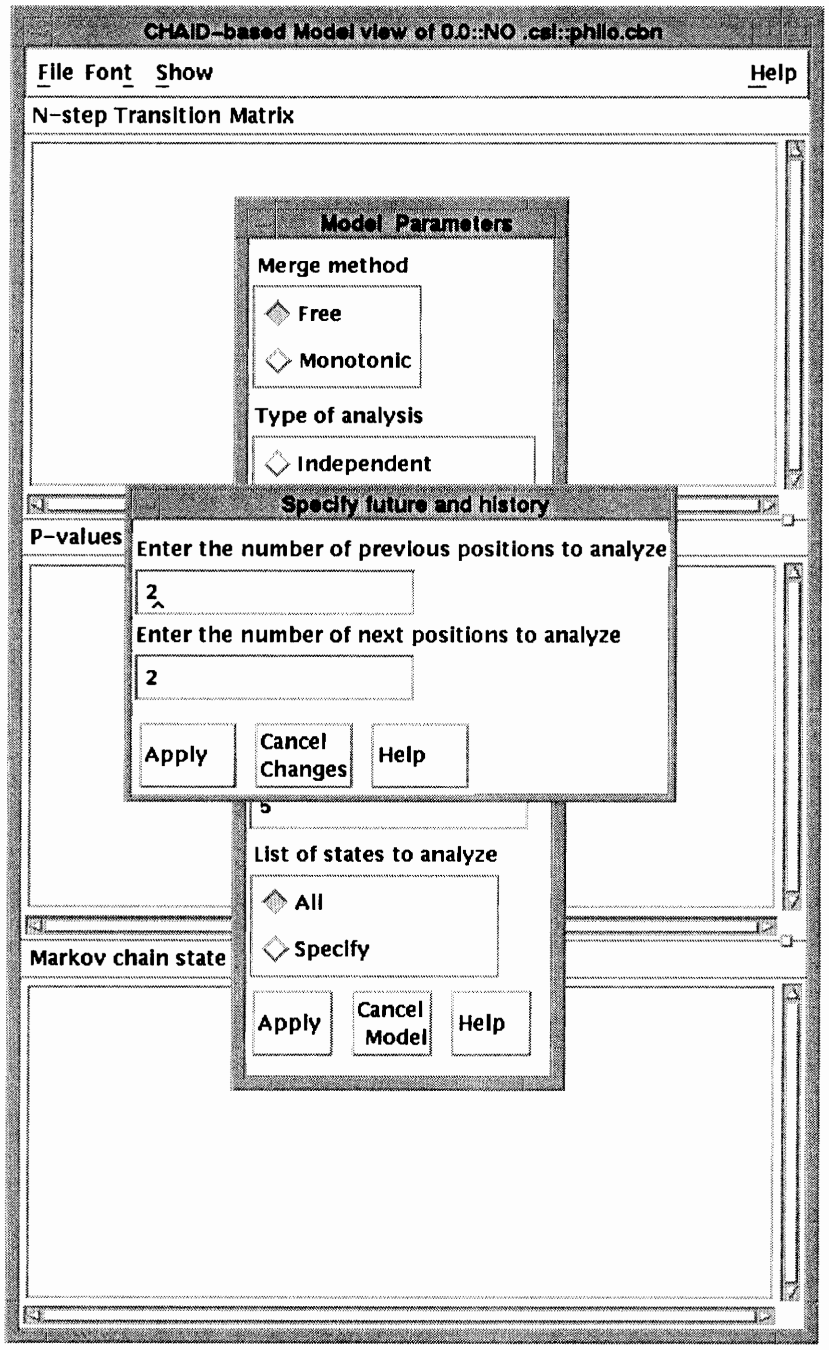
Significance Level (as %)

5

List of states to analyze

[✓ All] Specify

Figure 3.2: The parameter dialog box.



Merge method: [✓ Free] Monotonic

Type of analysis: Independent

List of states to analyze: [✓ All] Specify

Enter the number of previous positions to analyze: 2

Enter the number of next positions to analyze: 2

Figure 3.3: Dialog box for *Analyze both*

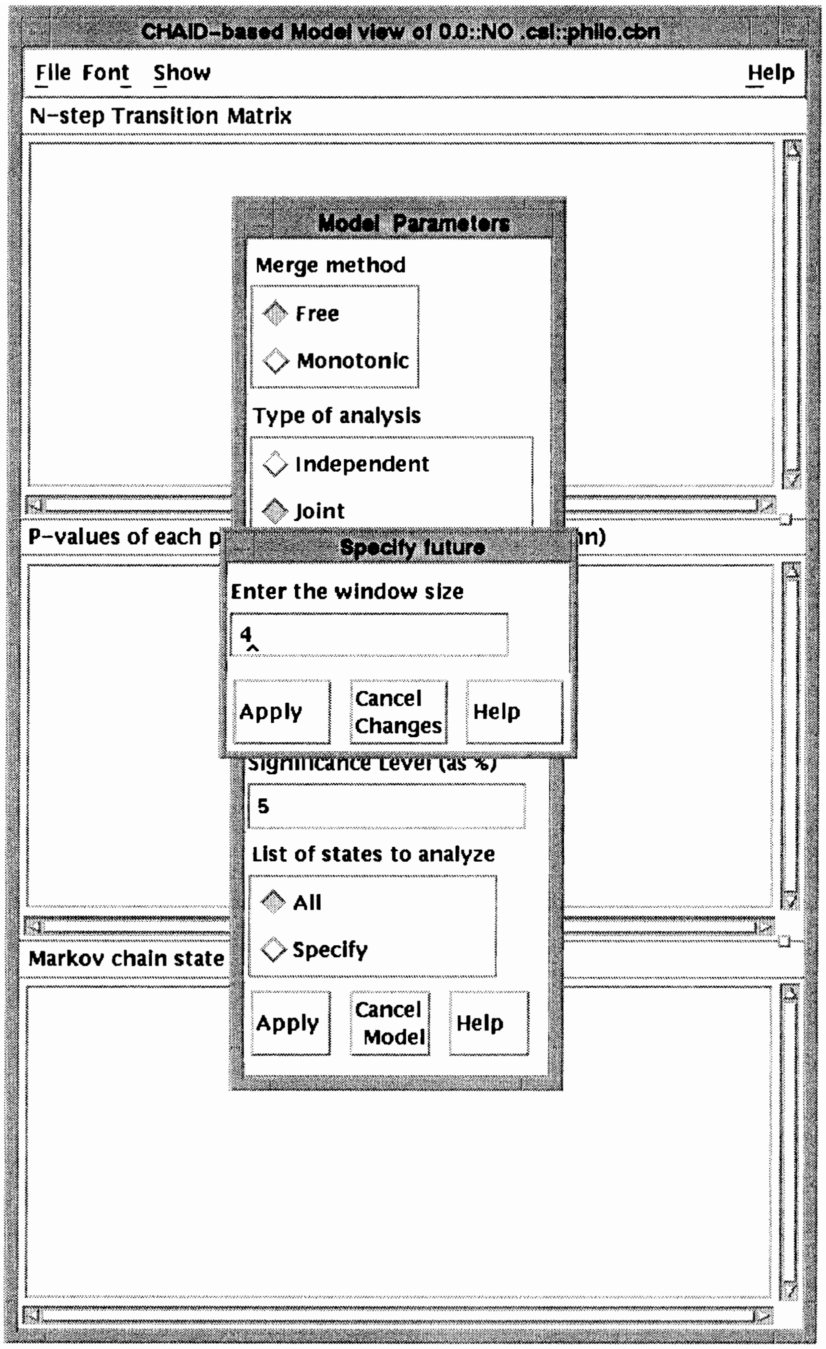


Figure 3.4: Dialog box for *Analyze future*

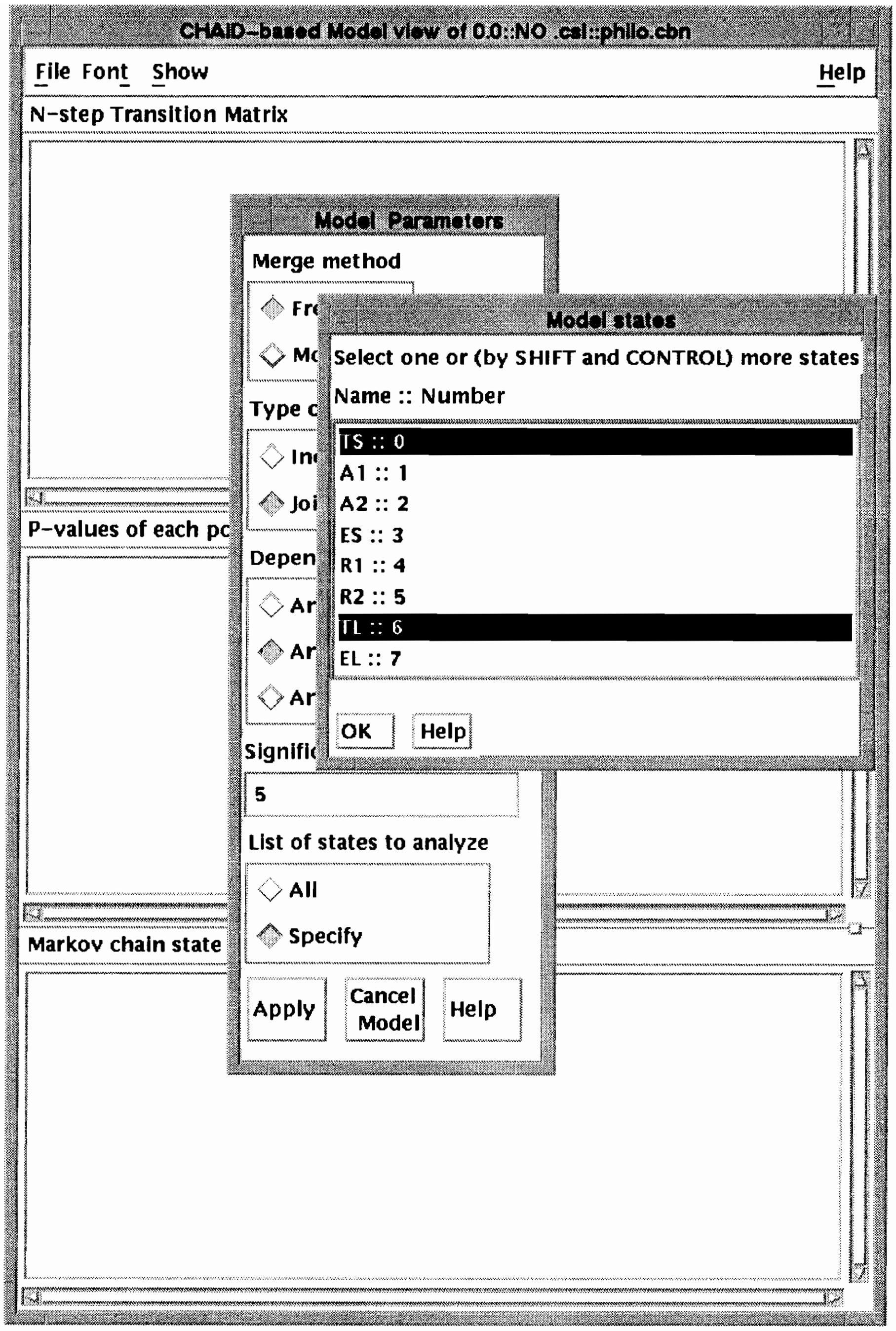
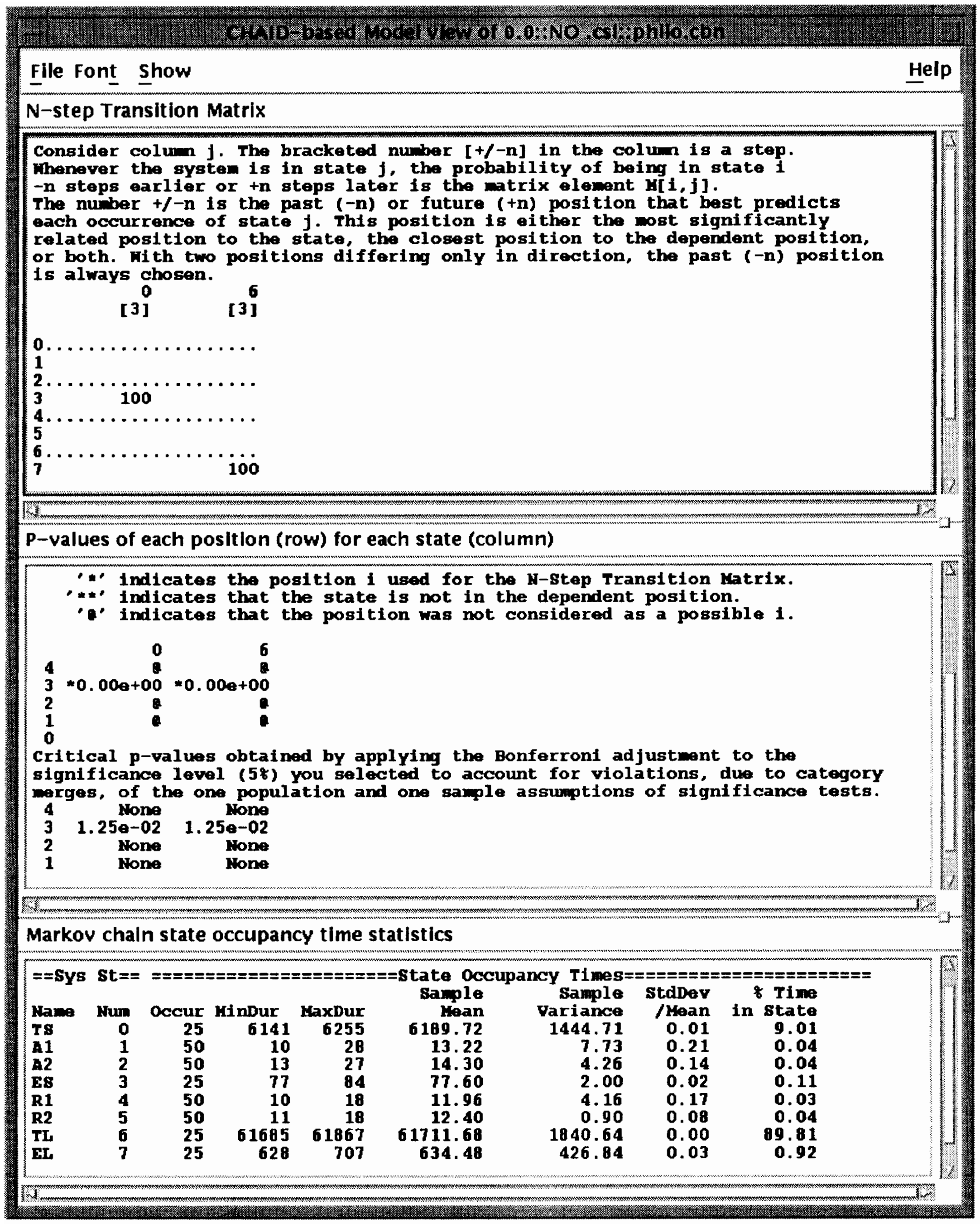


Figure 3.5: Selecting the states for analysis.


CHAID-based Model View of 0.0::NO .csl::philo.cbn
File Font Show
Help
N-step Transition Matrix
P-values of each position (row) for each state (column)
Markov chain state occupancy time statistics

```
Consider column j. The bracketed number [+/-n] in the column is a step.
Whenever the system is in state j, the probability of being in state i
-n steps earlier or +n steps later is the matrix element M[i,j].
The number +/-n is the past (-n) or future (+n) position that best predicts
each occurrence of state j. This position is either the most significantly
related position to the state, the closest position to the dependent position,
or both. With two positions differing only in direction, the past (-n) position
is always chosen.
          0          6
         [3]        [3]

0....................
1
2....................
3        100
4....................
5
6....................
7                   100
```

```
    '*' indicates the position i used for the N-Step Transition Matrix.
   '**' indicates that the state is not in the dependent position.
    '@' indicates that the position was not considered as a possible i.

           0          6
 4         @          @
 3 *0.00e+00 *0.00e+00
 2         @          @
 1         @          @
 0
Critical p-values obtained by applying the Bonferroni adjustment to the
significance level (5%) you selected to account for violations, due to category
merges, of the one population and one sample assumptions of significance tests.
 4      None       None
 3  1.25e-02   1.25e-02
 2      None       None
 1      None       None
```

```
==Sys St== =======================State Occupancy Times=======================
                                          Sample          Sample   StdDev    % Time
Name   Num   Occur MinDur   MaxDur          Mean        Variance    /Mean  in State
TS       0      25   6141     6255       6189.72         1444.71     0.01      9.01
A1       1      50     10       28         13.22            7.73     0.21      0.04
A2       2      50     13       27         14.30            4.26     0.14      0.04
ES       3      25     77       84         77.60            2.00     0.02      0.11
R1       4      50     10       18         11.96            4.16     0.17      0.03
R2       5      50     11       18         12.40            0.90     0.08      0.04
TL       6      25  61685    61867      61711.68         1840.64     0.00     89.81
EL       7      25    628      707        634.48          426.84     0.03      0.92
```

Figure 3.6: The model view of *philo.cbn* for states TS and TL.

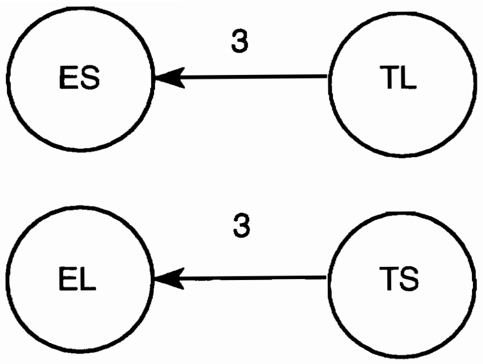


Figure 3.7: The embedded chain from the N-step Transition Matrix.

Chapter 4

CHAID-based Modeling of Ensembles of SSS's

4.1 Motivation

Many software performance analysis tools base their analysis on a *single* execution of a program. Because reliable performance measures on a system can not be obtained from a single execution, analysts typically observe many executions of a program and use a tool to generate performance measures from all executions. The individual results are analyzed in the hope that a general performance model could be made to fit the individual runs. Collating the different individual results is mostly done manually and is quite time-consuming. Frequently, data from some SSS's may skew the model being developed. In such cases, either the data are considered outliers and discarded or the model itself is discarded as being invalid and a new better fitting model attempted to be developed. In any case, the result is a costly and time-consuming procedure especially when analyzing parallel programs.

Program's are executed with a set of data items referred to as the program's *parameter set* [14]. For the Dining Philosophers problem, examples are the number of philosophers or the eat/think policy (e.g., fixed or random eating/thinking time). Multiple SSS's are generated through different runs of a program which, in general, may use different parameter sets for each run. Because parallel programs exhibit an inherently non-deterministic behavior, multiple executions of the same parallel program using exactly the same parameter set may lead to different SSS's. This adds complexity to the task of fitting one model for multiple SSS's.

One of the on-going research activities of the CHITRA research group is the generation of a single model from a collection of one or more SSS's. This collection, called an *ensemble*, is composed of SSS's obtained from multiple executions of a program. Membership in an ensemble may be based on certain characteristics of the parameter sets used in generating the SSS. For example, if a program were run a multiple number of times with the same parameter set, an analyst could group together the different SSS's from all executions into an ensemble. CHITRA's task would be to generate an empirical model fitting all the SSS members of the ensemble. The model generated would be dependent on the parameters of the ensemble (i.e. the parameters which were used by the program which generated the SSS).

4.2 Definitions

We develop a single CHAID-based model from an ensemble of SSS's by proceeding as if only one SSS were being analyzed. An analysis sequence is generated for *each* ensemble member. Because a single CHAID-based model is to be developed, the window size and future windows size would be the same for all analysis sequences. Each analysis sequence would thus have the same positions.

Let $e_1, e_2, \ldots, e_q$ be the q SSS's comprising an ensemble. Define Π_i as the analysis set from ensemble member e_i. Let Σ_I^i be the set of states in position I of the analysis set for e_i. The set of states (i.e., the categories) in position I for the ensemble is defined to be $\Sigma_I^{all} = \bigcup_{i=1}^{q} \Sigma_I^i$. The null hypothesis can then be tested as in Section 2.4 with the modification that the set of states used for each position I is Σ_I^{all} rather than Σ_I^i.

We will create one table for the ensemble based on the tables from the ensemble members. Let $T^1, T^2, \ldots, T^q$ be the contingency tables for each of the q SSS's. Let T^{all} be the single contingency table for the ensemble. Then, the element in row i and column j of T_{all} is defined as

$$T_{ij}^{all} = \sum_{r=1}^{q} T_{ij}^r$$

where T_{ij}^r is the element in row i, column j of the table T^r if it exists and 0 otherwise. This definition for T^{all} is a direct consequence of the definition for Σ_I^{all}.

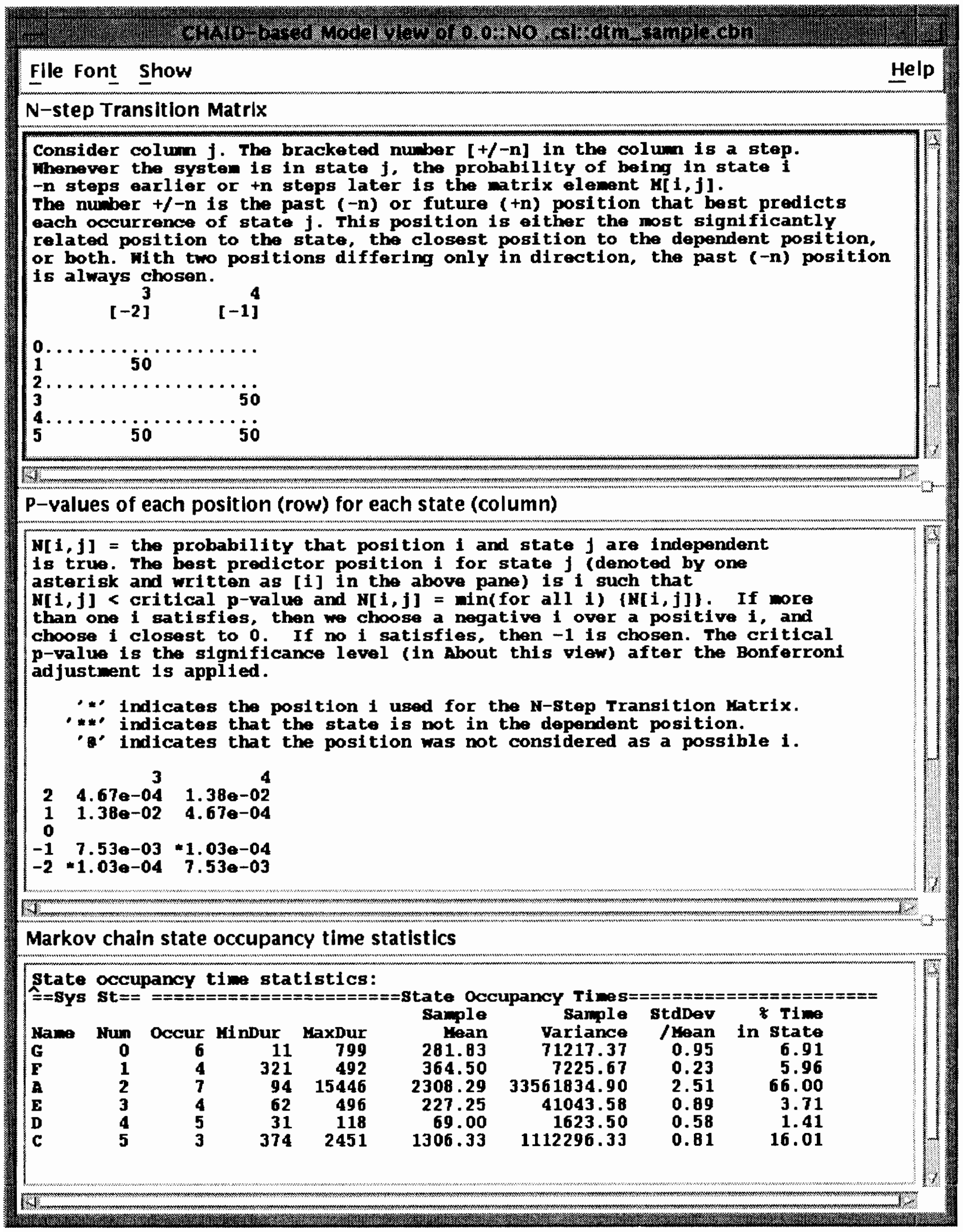
Example 25 *Consider S, the SSS in Example 2 as the SSS from one of the two SSS members of an ensemble. Let S' be the SSS from the other SSS member and $S' = S$. We construct* two *analysis sets, one for each of the SSS's. Because the SSS's are equal, our analysis sets would also be equal.*

The result would be the doubling of the frequencies in the contingency table Table 2.1 of Example 10. Table 4.1 shows this contingency table for the ensemble.

Table 4.1: The contingency table for Example 25

Position -1	Position 0				Row Total
	a		$\overline{a}$		
g	2	(3.24)	10	(8.76)	12
f	8	(2.16)	0	(5.84)	8
a	0	(3.24)	12	(8.76)	12
e	4	(2.16)	4	(5.84)	8
d	0	(2.16)	8	(5.84)	8
c	0	(1.18)	4	(2.92)	4
Column Total	14		38		52

The algorithm we have described generates the model from the contingency tables. Once the contingency tables have been set up, the rest of the analysis could proceed as had been described for a single SSS. Compare the model view of the single SSS in Example 2 shown in Figure 4.1 with the model view in Figure 4.2 of the ensemble as defined in this example. The N-step Transition Matrix are equal. The p-value table, however, shows an increase in the strength of the relationship (i.e., a decrease in the p-values) between the predictor positions and the states.



N-step Transition Matrix

```
Consider column j. The bracketed number [+/-n] in the column is a step.
Whenever the system is in state j, the probability of being in state i
-n steps earlier or +n steps later is the matrix element M[i,j].
The number +/-n is the past (-n) or future (+n) position that best predicts
each occurrence of state j. This position is either the most significantly
related position to the state, the closest position to the dependent position,
or both. With two positions differing only in direction, the past (-n) position
is always chosen.
           3          4
         [-2]       [-1]

0....................
1         50
2....................
3                    50
4....................
5         50         50
```

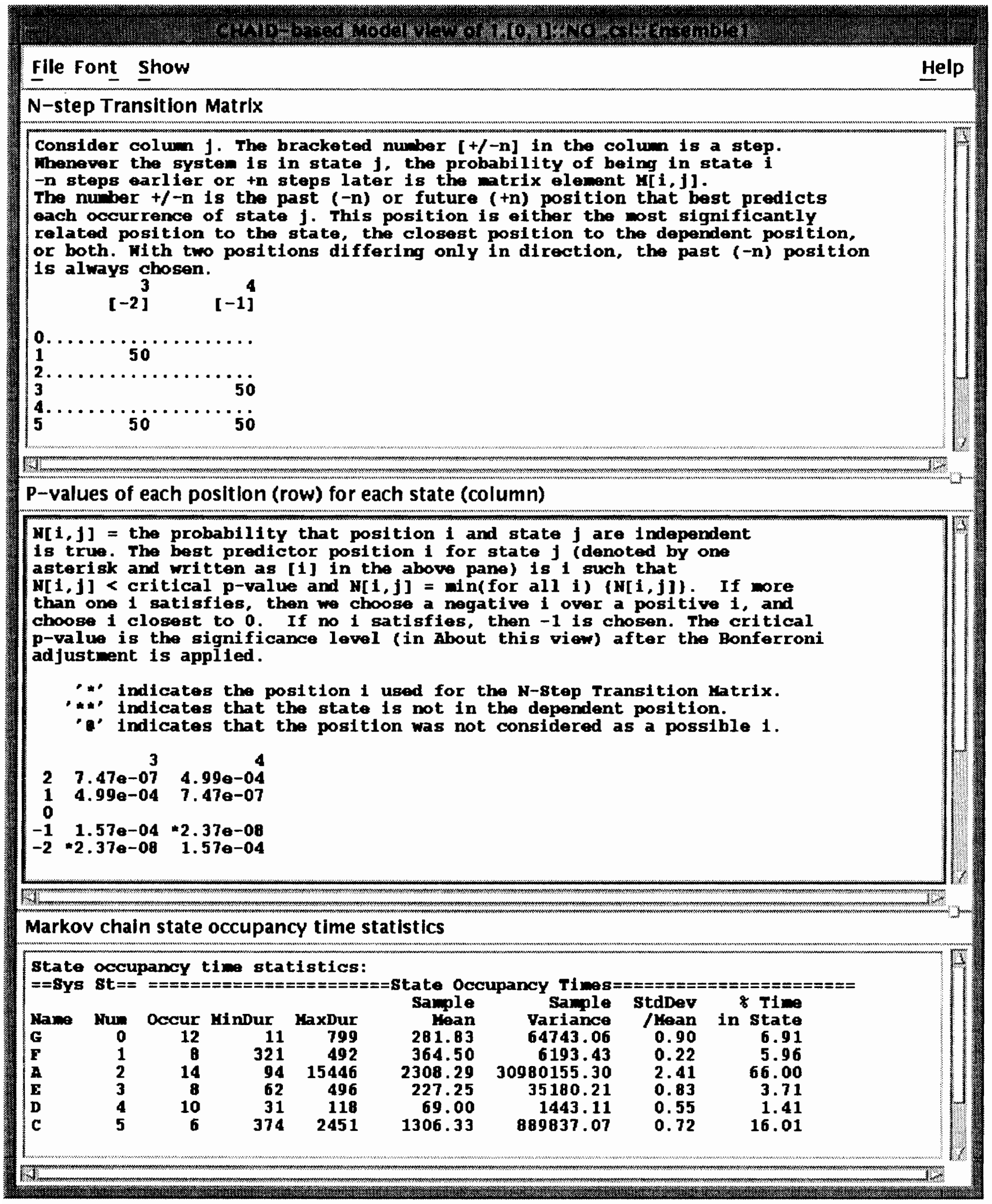
P-values of each position (row) for each state (column)

```
M[i,j] = the probability that position i and state j are independent
is true. The best predictor position i for state j (denoted by one
asterisk and written as [i] in the above pane) is i such that
M[i,j] < critical p-value and M[i,j] = min(for all i) {M[i,j]}.  If more
than one i satisfies, then we choose a negative i over a positive i, and
choose i closest to 0.  If no i satisfies, then -1 is chosen. The critical
p-value is the significance level (in About this view) after the Bonferroni
adjustment is applied.

    '*' indicates the position i used for the N-Step Transition Matrix.
   '**' indicates that the state is not in the dependent position.
    '@' indicates that the position was not considered as a possible i.

          3         4
 2  4.67e-04  1.38e-02
 1  1.38e-02  4.67e-04
 0
-1  7.53e-03 *1.03e-04
-2 *1.03e-04  7.53e-03
```

Markov chain state occupancy time statistics

```
State occupancy time statistics:
==Sys St== =======================State Occupancy Times=======================
                                         Sample          Sample  StdDev     % Time
Name   Num  Occur MinDur  MaxDur          Mean        Variance   /Mean   in State
G        0      6     11     799        281.83        71217.37    0.95       6.91
F        1      4    321     492        364.50         7225.67    0.23       5.96
A        2      7     94   15446       2308.29     33561834.90    2.51      66.00
E        3      4     62     496        227.25        41043.58    0.89       3.71
D        4      5     31     118         69.00         1623.50    0.58       1.41
C        5      3    374    2451       1306.33      1112296.33    0.81      16.01
```

Figure 4.1: CHAID-based model view for the SSS in Example 2.



N-step Transition Matrix

```
Consider column j. The bracketed number [+/-n] in the column is a step.
Whenever the system is in state j, the probability of being in state i
-n steps earlier or +n steps later is the matrix element M[i,j].
The number +/-n is the past (-n) or future (+n) position that best predicts
each occurrence of state j. This position is either the most significantly
related position to the state, the closest position to the dependent position,
or both. With two positions differing only in direction, the past (-n) position
is always chosen.
           3         4
       [-2]      [-1]

0....................
1        50
2....................
3                  50
4....................
5        50        50
```

P-values of each position (row) for each state (column)

```
N[i,j] = the probability that position i and state j are independent
is true. The best predictor position i for state j (denoted by one
asterisk and written as [i] in the above pane) is i such that
N[i,j] < critical p-value and N[i,j] = min(for all i) {N[i,j]}.  If more
than one i satisfies, then we choose a negative i over a positive i, and
choose i closest to 0.  If no i satisfies, then -1 is chosen. The critical
p-value is the significance level (in About this view) after the Bonferroni
adjustment is applied.

    '*' indicates the position i used for the N-Step Transition Matrix.
   '**' indicates that the state is not in the dependent position.
    '#' indicates that the position was not considered as a possible i.

             3         4
 2  7.47e-07  4.99e-04
 1  4.99e-04  7.47e-07
 0
-1  1.57e-04 *2.37e-08
-2 *2.37e-08  1.57e-04
```

Markov chain state occupancy time statistics

```
State occupancy time statistics:
==Sys St== ======================State Occupancy Times======================
                                    Sample        Sample  StdDev    % Time
Name  Num  Occur MinDur  MaxDur       Mean      Variance   /Mean  in State
G       0     12     11     799     281.83      64743.06    0.90      6.91
F       1      8    321     492     364.50       6193.43    0.22      5.96
A       2     14     94   15446    2308.29   30980155.30    2.41     66.00
E       3      8     62     496     227.25      35180.21    0.83      3.71
D       4     10     31     118      69.00       1443.11    0.55      1.41
C       5      6    374    2451    1306.33     889837.07    0.72     16.01
```

Figure 4.2: CHAID-based model view for ensemble in Example 25.

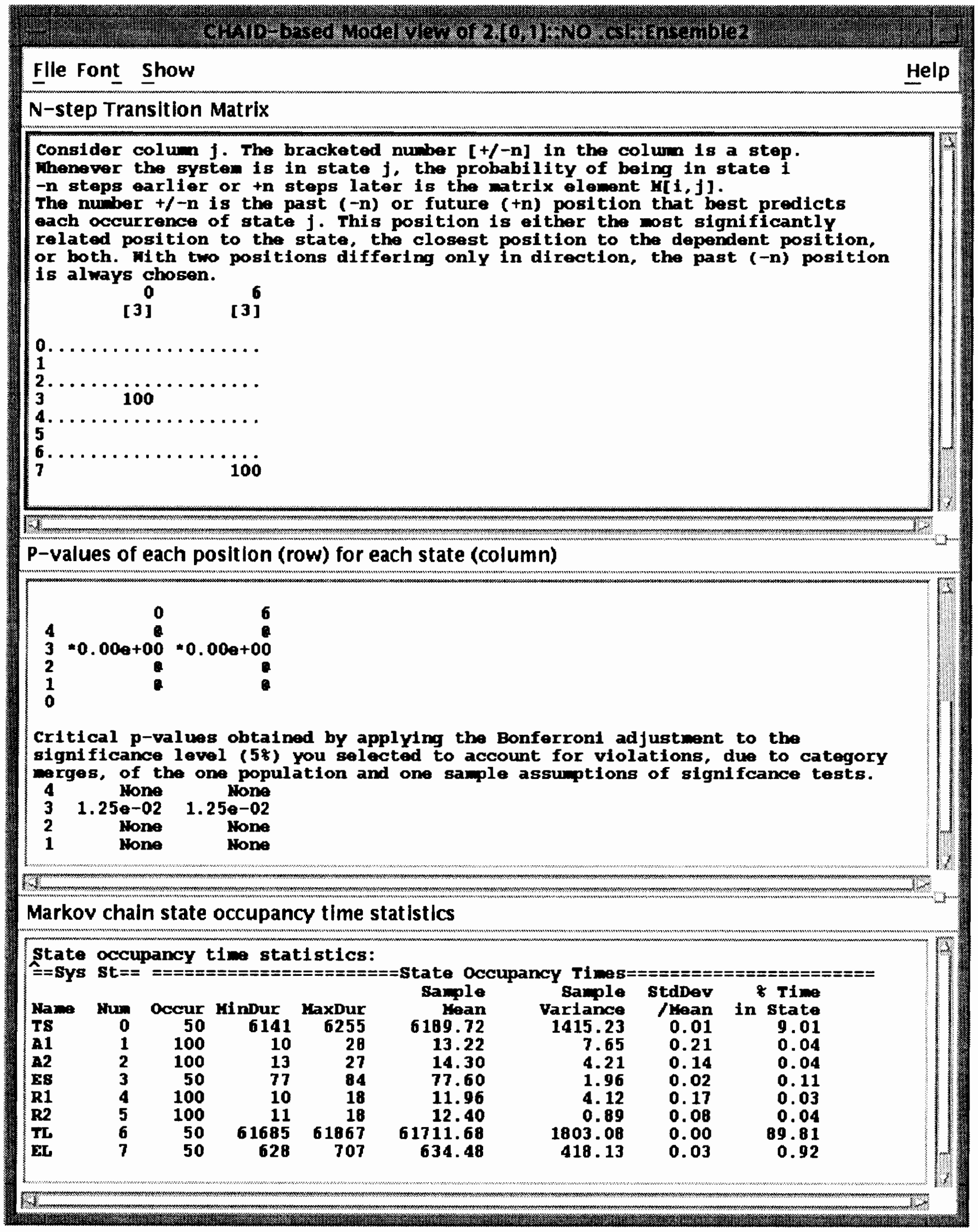
We apply the same analysis with an ensemble composed of two copies of the SSS *philo.cbn* used in Chapter 3. However, because the p-value for position 3 in Figure 3.6 is already at the lowest possible level (i.e., 0.00), the strength of the relationship between the predictor positions and the states can no longer increase. This is shown in the CHAID-based model view of the ensemble shown in Figure 4.3.

4.3 Summary

This chapter shows how a CHAID-based model for an ensemble of more than one SSS can be generated. The procedure for developing a model for an ensemble is as follows:

- A SSS is generated for each ensemble member.
- For each SSS, a contingency table for each independent position is created as described in Chapter 2.
- The set of categories for each position is the union of the categories in that position for all the SSS's.
- An overall contingency table is created for each independent position in the ensemble. The rows of the overall contingency table for an independent position are for the union of the categories described above.
- The merge algorithm is performed on the overall contingency table.

- The model is developed from these overall contingency tables.



N-step Transition Matrix

```
Consider column j. The bracketed number [+/-n] in the column is a step.
Whenever the system is in state j, the probability of being in state i
-n steps earlier or +n steps later is the matrix element M[i,j].
The number +/-n is the past (-n) or future (+n) position that best predicts
each occurrence of state j. This position is either the most significantly
related position to the state, the closest position to the dependent position,
or both. With two positions differing only in direction, the past (-n) position
is always chosen.
          0         6
        [3]       [3]

0....................
1
2....................
3       100
4....................
5
6....................
7                 100
```

P-values of each position (row) for each state (column)

```
          0         6
 4        @         @
 3 *0.00e+00 *0.00e+00
 2        @         @
 1        @         @
 0

Critical p-values obtained by applying the Bonferroni adjustment to the
significance level (5%) you selected to account for violations, due to category
merges, of the one population and one sample assumptions of signifcance tests.
 4      None      None
 3  1.25e-02  1.25e-02
 2      None      None
 1      None      None
```

Markov chain state occupancy time statistics

```
State occupancy time statistics:
==Sys St== ======================State Occupancy Times======================
                                     Sample        Sample   StdDev    % Time
Name   Num   Occur  MinDur  MaxDur     Mean      Variance    /Mean  in State
TS       0      50    6141    6255  6189.72       1415.23     0.01      9.01
A1       1     100      10      28    13.22          7.65     0.21      0.04
A2       2     100      13      27    14.30          4.21     0.14      0.04
ES       3      50      77      84    77.60          1.96     0.02      0.11
R1       4     100      10      18    11.96          4.12     0.17      0.03
R2       5     100      11      18    12.40          0.89     0.08      0.04
TL       6      50   61685   61867 61711.68       1803.08     0.00     89.81
EL       7      50     628     707   634.48        418.13     0.03      0.92
```

Figure 4.3: CHAID-based model view for ensemble of two copies of *philo.cbn* to be compared with Figure 3.6.

Chapter 5

Related Work

Our search for an alternative to CHITRA's CTSM model is motivated by our twin goals of discovering *and* representing dependencies between any arbitrary model states. We note that this motivation is not confined to the field of performance analysis but is present in many other areas where researchers are interested in identifying patterns in experimental data obtained in an investigation. In medicine, for example, a dependency between a patient's blood pressure and the patient's risk of morbidity (e.g., high-risk, low-risk, no-risk) have been discovered. In meteorology, the levels of some airborne pollutants have been associated with the magnitude of health risks to the the general populace [11] and are therefore used to determine the warning levels (e.g, none, first-stage) raised by public health officials.

We cite the above two examples only to show that the techniques used in developing our new model did not develop from a vacuum but have already been successfully utilized in other disparate fields. Our contribution is in adapting these techniques to create a new model for performance analysis of software. The previous sections attempted to show how to develop a CHAID-based model for CHITRA. We now describe the techniques from which our CHAID-based model is founded.

5.1 Automatic Interaction Detection

Automatic Interaction Detection (AID) is a technique which had been developed as a descriptive method for large data sets [17]. Kass traces AID's roots to a paper by Belson in 1953. He attributes the current AID method to that which had been described by Morgan and Sonquist in 1963.

The data sets AID operates on must be composed of a single dependent variable and at *least* one categorized (i.e., it must have discrete values) predictor variable. A predictor variable, also known as an *independent* variable, is a variable which is considered to be affecting the realization of the dependent variable.

AID operates by attempting to successively split the data set into 2 subgroups. The sole criterion for splitting is to find the two subgroups which have the maximum "between subgroup sum of squares" [17]. In statistical experimental design, this sum of squares is an estimate of the combined effects of treatment levels (i.e. the categories of the predictor variable) and chance on the dependent variable [19]. By performing a split only when the effect of chance is below a certain threshold level, we can be assured that the predictor variable defines how a split is performed. The algorithm continues until each of the subgroups formed in the previous stage can no longer be split.

A simplistic split, for example, would be to separate data elements into those which the predictor variable has an effect on from those on which the predictor has no effect. The set of categories of the predictor variable which effect the dependent variable would define one subgroup. The other subgroup would be defined by the set's complement. This ensures that the between subgroup sum of squares is maximum.

Each subgroup is composed of elements which are homogeneous (i.e., similar to each other) with respect to how the predictor variable affects them. Conversely, the between group differences are maximized. Through the splitting process, AID maximizes the effect of the categories of the predictor variable on the dependent variable.

In most data sets, more than one predictor variable would be defined. Because the effect of one predictor variable on the dependent variable may differ from the effect of another, each predictor variable defines its own split of the data set. What is important to remember is the fact that AID ensures that each final split is maximized for the predictor variable which defined the split.

As described by Morgan and Sonquist, AID has two basic uses. The first is as a tool for preliminary analysis of data. The aim is to gather enough information about the data set so that the analyst can determine what statistical technique would be most appropriate for further analysis. The second use is for forming homogeneous groups (i.e., groups with elements which are homogeneous). Our interest is in the latter use. By maximizing the effects of the predictor variables on the dependent variable, AID uncovers the presence of a relationship between the dependent variable and predictor variable. In particular, for our PES, the presence of a relationship between the categories (i.e. the states) of the dependent and predictor variable (i.e. the predictor and dependent positions) can be established.

5.2 Chi Automatic Interaction Detection

Two main shortcomings had been cited against AID. It had been criticized as not accounting for the "variability inherent in data" [10] and for the bias in its splitting process. Because the possibility of finding the maximum sum of squares as being that of one predictor variable is directly proportional to its number of categories, AID was biased towards predictors with more categories [18]. To remedy both limitations, Kass added a significance testing step during the partitioning process. This allowed for the creation of multi-way splits as opposed to binary splits. It also had the effect of minimizing but not, as we shall see in the next subsection, completely nullifying the bias.

Kass further criticized AID for using only the standard t-test for testing whether the two newly formed groups are significantly different from each other. While an insignificant t-test result would indeed indicate that the split is insignificant, Kass claims that the converse is not necessarily true. A significant t-test result does not imply that the split is significant.

Since the nature of AID is to maximize the between-group t-statistic, the probability that a "significant" split is actually due to chance is considered by Kass to be unacceptably high.

Kass introduced the two-way contingency table as the basis of analysis for finding the significant splits of the data. Using the chi-square test for the hypothesis of homogeneity (or, conversely, the hypothesis of independence) of a contingency table, Kass analyzed the effect of a predictor variable on the dependent variable. The predictor variable for which the dependent variable showed the greatest dependence on, the one with the greatest χ_0^2 statistic for the contingency table, was chosen as the variable to split the data set. Its categories (classes) were chosen as the splits.

The chi-square test is inadequate for an exhaustive study of a data set because it is unable to extract certain information about the data set. It does not, for example, give an indication of the direction or the nature of an association between the variables. Rather, it is a simple test for uncovering evidence of associations [5]. Its simplicity in performing this role makes the chi-square test ideal for partitioning the data.

Together with the chi-square test, Kass introduced a new type of predictor which he called the floating predictor. This type was meant for situations where categories fit an ordinal scale save for one exceptional category whose position in the scale is unknown. Missing or unknown values fit into this new category. With this new category type and the use of the chi-square as the test of significance, Kass had developed a new technique called

Chi Automatic Interaction Detection (CHAID).

5.3 Knowledge Seeker

CHAID is not without its flaws. Biggs, De Ville, and Suen showed that the significance levels used in the CHAID significance testing were still unduly related to the number of categories of a predictor variable. This time, however, the favored predictor variables were those with small numbers of categories. In addition, as the number of categories of a floating category type became large, the significance levels became too low – by up to a factor of 10. This had the effect of discriminating against the newly introduced floating category predictor variables [9].

Biggs, De Ville, and Suen refined CHAID by including a method for adjusting the significance level by considering the type and number of categories of the predictor variable and allowing for the fact that the maximum sum of squares was used (i.e. the "best" split was used) to partition the data set. The result was an algorithm which removed the bias against a particular category type and allowed for the ranking of each variable in order of importance at each split.

This algorithm is incorporated in a software called Knowledge Seeker [13]. It is available for MS-DOS based PC's and was used by the author to do the preliminary work on this project.

5.4 Classification and Regression Trees

Recall the two steps in AID. The first is to find the the predictor variable with which the dependent variable has the greatest dependence. The second is to use the categories of the chosen predictor variable as the classifiers for the dependent variable. The result of the second step is a partition of the data set.

The techniques embodied in AID and CHAID are essentially means to classify a dependent variable based on its relationship with the predictor variables. The classes are the categories of the predictor variable. Measurements (i.e. t-test for AID, and chi-square for CHAID) are made on the strength of dependence between the categories of the predictor variable and the dependent variable. Based on these measurements, a dependent variable is assigned to a class.

The roots of Classification and Regression Trees (CART) can be traced to Breiman and Friedman who, independently of each other, began to use tree methods in classification in 1973. After the two began their collaboration, they were joined by Stone whom they credit for the methodological development of tree classification. They were then joined by Olshen who applied the methods in the field of medicine and who contributed to CART's theoretical development [11].

In CART, a binary tree classifier is constructed by a recursive split of a node into two descendant sub-nodes. The root is the entire data set. The sub-nodes are disjoint subsets of the parent node. The terminal nodes (i.e. subsets which are not split), form a partition of the data set.

In Figure 5.1, which is adapted from page 21 of [11], we have the data set X split into two disjoint subsets X_1 and X_2. Both are further split into two disjoint subsets. The former into X_3 and X_4 and the latter into X_5 and X_6, and so on.

The splits of a node are formed based on the measurements done on the elements of the node. For example, the first split could be $X_1 = \{x \in X\colon x \text{ is dependent on } Y\}$ and $X_2 = \{x \in X\colon x \text{ is not dependent on } Y\}$. X_3 of Split 2 could be $X_3 = \{x \in X_1\colon x \text{ is not dependent on } Z\}$ That is, $X_3 = \{x \in X\colon x \text{ is dependent on } Y \text{ and not dependent on } Z\}$. Thus from the definition of the first split, for any element x in X, it can be determined whether x goes to X_1 or X_2 and so on until x reaches a terminal node where it is finally classified.

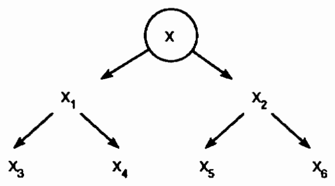


Figure 5.1: A classifier tree with 4 classes.

The entire construction of a tree, as quoted from [11], is dependent on the following:

1. The selection of the splits

2. The decision when to declare a node terminal or to continue splitting it

3. The assignment of each terminal node to a class

For CHAID, the χ_0^2 statistic is the basis for the selection of splits. A node is declared terminal when no further significant split is possible (i.e., the groups formed by a further split are still homogeneous). The classes are formed automatically by the categories of the predictor variable used to split the data.

Chapter 6

Conclusions and Future Work

The goal of this project was to design and implement an alternative to CHITRA92's CTSM model that shows what previous or succeeding states influence the occurrence of any particular state. The CHAID-based model presented in this report has been successfully implemented and is now an integral part of the latest generation of CHITRA93.

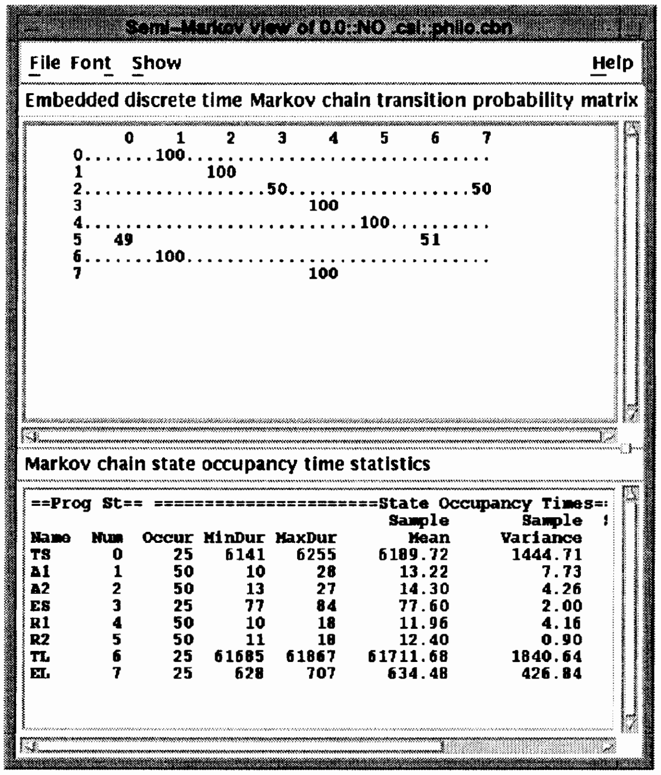
We summarize the potential impact of the CHAID-based model in performance analysis studies.

6.1 Enhanced Model Accuracy

We had mentioned that program states are reached through some probabilistic and deterministic transitions. In large traces, the deterministic relationships between states are obscured by the volume of data to be examined. Furthermore, when the probabilistic transitions occur between two, otherwise completely related states, the relationship between the states are occluded. That is, related states are frequently separated by seemingly random sequences of other states which hide the relation.

The CHAID-based model is able to “filter out” the randomness between related states with its ability to represent and examine relationships between states which are not immediate predecessors or successors of each other.

Consider the Transition Matrix in Figure 6.1 showing the generated CTSM model for the dining philosophers program we had discussed in Example 1. We show the corresponding transition diagram in Figure 6.2



Embedded discrete time Markov chain transition probability matrix

```
        0    1    2    3    4    5    6    7
0.......100.............................
1            100
2..................50..................50
3                      100
4...........................100..........
5   49                            51
6.......100.............................
7                      100
```

Markov chain state occupancy time statistics

```
==Prog St== =====================State Occupancy Times==
                                        Sample        Sample !
Name   Num  Occur MinDur MaxDur           Mean      Variance
TS       0     25   6141   6255        6189.72       1444.71
A1       1     50     10     28          13.22          7.73
A2       2     50     13     27          14.30          4.26
ES       3     25     77     84          77.60          2.00
R1       4     50     10     18          11.96          4.16
R2       5     50     11     18          12.40          0.90
TL       6     25  61685  61867       61711.68       1840.64
EL       7     25    628    707         634.48        426.84
```

Figure 6.1: The CTSM model view of *philo.cbn*

It only shows the fact that the philosopher moves to state $A2$ to either ES or EL

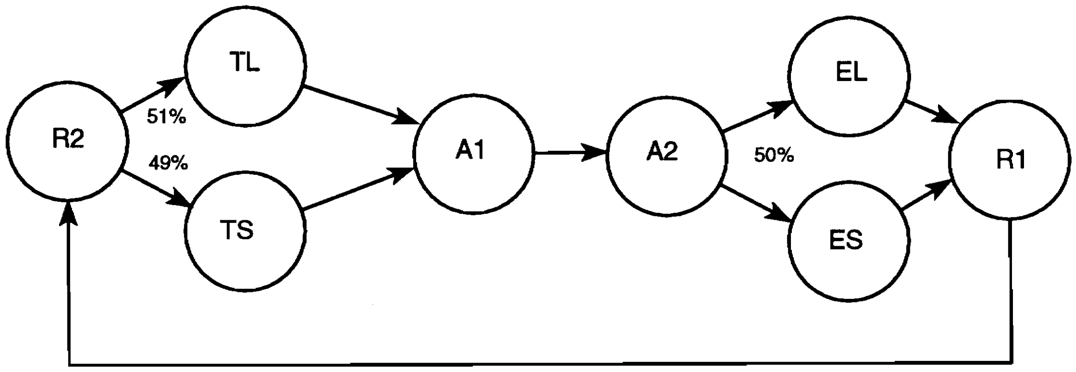


Figure 6.2: The corresponding transition diagram of Figure 6.1

with equal probability. Thus, we have a probabilistic transition out of state $A2$. The N-step Transition Matrix in Figure 3.6, on the other hand, shows that ES and EL are determined by TL and TS, respectively. Our CHAID-based model has allowed us to replace a probabilistic branch in the semi-Markov model by a deterministic branch it has uncovered. The resulting model is non-Markovian but *more accurate*. Figure 6.1 allows the sequence $\langle TL, A1, A2, ES\rangle$ which is illegal. The model combining the results of the CTSM model and the CHAID-based model is shown in Figure 6.3.

In addition, because the deterministic transition occurs in step $N > 1$, replacement of the probabilistic branch results allows us to aggregate the sequence $\langle TS, A1, A2, ES\rangle$ into a single state $T1$. Similarly for the sequence $\langle TL, A1, A2, EL\rangle$ into a single state $T2$. This results not only in a reduction of the SSS, but a reduction without loss of information. Transition graphs like the one for this dining philosopher program have been observed in

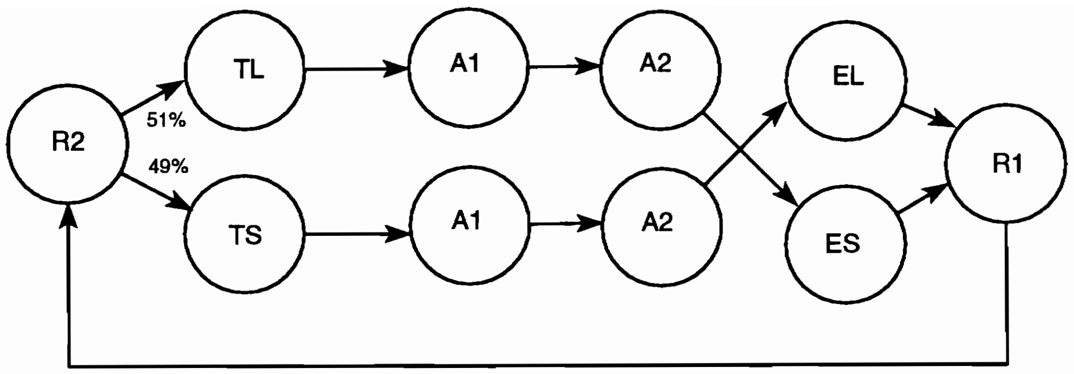


Figure 6.3: The model of the program behavior of *philo.cbn* combining the results of the CTSM model and the CHAID-based model.

other software traces [3].

Model Generation for Trace-Driven Simulation Computer architects extensively rely on actual input data which had been generated from the operation of the real system as input to their simulation model. The major advantage of using an actual trace, of course, is the quality assurance that the model is valid with respect to observed traces. When no adequate methods for generating the input data are available, there is no other choice but to use actual traces.

The primary disadvantage of using actual traces is the overwhelming volume of such data. Using the unmodified trace data may be extremely time consuming to process and space consuming to store. In simulating cache references for example, various techniques such as *address stripping* have been developed for reducing the volume of the trace [25].

As usual, the objective is to reduce the length of the trace while maintaining its ability to represent the real input.

Developing an empirical model of large trace data is a novel technique for capturing the characteristics of the trace data in a more concise form. Could model states

6.2 Future Work and Enhancements to the CHAID-based model

Three future work on CHITRA related to the CHAID-based model are presented. The first and most important is the validation of the CHAID-based model for performance evaluation. The second is the use of the CHAID-based model to support future enhancements to CHITRA. The third is the use of other statistical techniques which could be used as alternatives to the CHAID-based technique we had presented.

The Validation of the CHAID-based Model Extensive validation of the model is necessary to determine the applicability and robustness of the CHAID-based model for performance analysis. Many techniques for model validation are described in [7]. Although the CHITRA group has already used the model in analyzing Dining Philosopher programs and parallel database applications with promising results, the formal validation of the CHAID-based model is still a primary concern.

The CHAID-based Model as a Similarity Measure The ability to group together SSS's collected from different runs of the same program is a powerful heuristic for predicting program behavior [3]. We had discussed in Chapter 4 how SSS's already grouped together into ensembles could be analyzed. A related case is when the ensemble members are SSS's with similar but not equal parameter sets[1]. Within an ensemble, some SSS's could be more similar to each other than to the rest of the members of the ensemble. How could an ensemble be partitioned into different sub-ensembles which are more homogeneous than the original ensemble?

Our CHAID-based model uses the predictor positions to describe the analyzed states of a SSS. Our examples in Chapter 4 had shown that an ensemble of two identical SSS's results in an increase in the strength of the p-values (compare the middle panes of Figures 3.6 and 4.3). A separate analysis of the two SSS's results in two exactly equal model views. Can modeling a subset of all the states in a SSS be a good characteristic function for the SSS or should all the states be modeled to provide a better characteristic function? Could the p-value tables of the models of the ensemble members be accurate indicators of similarity?

Another possibility is the use of the merged categories of an independent position as classifiers for a SSS. These mergers, as had been previously mentioned, are partitions of the

[1]The definition of what is similar is largely dependent on the objectives of the analysis. For example, SSS's with fixed eat/think time, although the time units are different for each SSS, could be deemed to be similar. Those with a random eat/think time policy could be in a different group.

dependent position. In developing our CHAID-based model, we had discarded the resulting categories of the independent positions from the merges because our sole interest was in obtaining the optimal independent position. These categories however, could be used as classifiers of *each* independent position as discussed in Section 5.4. In turn, the classifiers for each independent position could be grouped together and taken as a classifier of the SSS.

Consider two SSS's A and B from two different SSS's, each with m independent positions. Let A_i, $1 \leq i \leq m$ be the classifier for the independent position i of SSS A. Let B_i, $1 \leq i \leq m$ be the classifier for the independent position i of SSS B. We can construct a similarity function for the SSS's (hence, a similarity function for their corresponding SSS's) by comparing A_i with B_i for $1 \leq i \leq m$. Two SSS's could be considered to be *equal* if and only if $A_i = B_i$ for $1 \leq i \leq m$. A and B could be called *not equal* if and only if $A_i \neq B_i$ for $1 \leq i \leq m$. Similarity could then be measured in terms of the percentage of classifiers which are equal to each other.

Developing an Alternative Technique for Merges Our search for associations between the independent positions and the dependent position used two-dimensional tables tested with the chi-square statistics. In our procedure, we had *sequentially* tested each independent position against the dependent position for significant associations. We then selected the independent position with the smallest p-value for the chi-square statistic as the optimal position for the categories of the dependent position. Implicit in this decision is the belief that the table with the smallest p-value is *significantly* different from the other tables with bigger p-values. In other words, we were concluding that the independent position with the smallest p-value had an effect on the dependent position which was significantly different from the effect the other independent positions with bigger p-values had on the dependent position. We had not, however, specifically tested whether the subsequences which fall into the respective categories of the different independent positions do in fact differ appreciably in their response to the dependent position.

Statistical techniques referred to as *log-linear contingency table analysis*, *logit analysis*, or more frequently as *log-linear models.* could be used to explicitly compare each predictor position. These methods for analyzing the different independent variables such as our predictor positions is presented in [16]. Furthermore, a hierarchy of log-linear models which could be used and tested sequentially for determining which independent variable predicts the dependent variable could be incorporated in CHITRA.

REFERENCES

[1] M. Abrams and A. K. Agrawala. Performance study of distributed resource sharing algorithms. *IEEE Dist. Processing Technical Committee Newsletter*, 7(3):18–26, November 1985.

[2] 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, November 1992.

[3] M. Abrams, T. Lee, H. Cadiz, and K. Ganugapati. Chitra93: Beyond software visualization. This paper describing Chitra93 is yet to be published., 1994.

[4] A. A. Afifi and S. P. Azen. *Statistical Analysis: A Computer Oriented Approach.* Academic Press, Inc., 1972.

[5] A. Agresti. *Categorical Data Analysis.* John Wiley and Sons, 1990.

[6] M. S. Aldenderfer and R. K. Blashfield. *Cluster Analysis.* SAGE Publications, 1984.

[7] O. Balci. Guidelines for successful simulation studies. Course Notes, March 1987.

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

[9] D. Biggs, B. de Ville, and E. Suen. A method of choosing multiway partitions for classification and decision trees. *J. of App. Stat.*, 18(1):49–62, January 1991.

[10] Y. M. Bishop, S. E. Fienberg, and P. W. Holland. *Discrete Multivariate Analysis.* MIT Press, 1975.

[11] L. Brieman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees.* Wadsworth, 1984.

[12] E. Caulcott. *Significance Tests.* Routledge and Kegan Paul, 1973.

[13] FirstMark Technologies Ltd. *Knowledge Seeker User's Guide*, Version 2.1 edition.

[14] K. Ganugapati. The design and implementation of Chitra92, a system to empirically model concurrent software performance. Master's thesis, Computer Sci. Dept., Virginia Tech, Blacksburg, VA 24060, April 1993.

[15] P. G. Harrison and N. M. Patel. *Performance Modelling of Communication Networks and Computer Architectures.* Addison-Wesley, 1992.

[16] D. E. Hinkle, J. T. Austin, and G. W. McLaughlin. Using log-linear models in higher education research. In B. Yancey, editor, *New Directions for Institutional Research*, volume 58, pages 23–41. Jossey-Bass, 1988.

[17] G. V. Kass. Significance testing in automatic interaction detection (A.I.D.). *Applied Statistics*, 24(2):178–189, 1975.

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

[19] R. E. Kirk. *Experimental Design: Procedures for the Behavioral Sciences.* Brooks/Cole, 2nd edition, 1982.

[20] T. Lehr et al. Visualizing performance debugging. *Computer*, pages 38–51, October 1989.

[21] M. Maekawa, A. E. Oldehoeft, and R. R. Oldehoeft. *Operating Systems Advanced Concepts.* The Benjamin/Cummings Publishing Company, 1987.

[22] L. Mohr. *Understanding Significance Testing.* SAGE University Paper series on Quantitative Applications in the Social Sciences no. 07-073. SAGE Publications, 1990.

[23] A. Papoulis. *Probability, Random Variables, and Stochastic Processes.* McGraw-Hill, Inc., 3rd edition, 1991.

[24] M. Simmons and R. Koskela, editors. *Performance Instrumentation and Visualization.* ACM Press, New York, 1990. Based on the Workshop on Parallel Computer Systems: Instrumentation and Visualization, May 1989.

[25] H. S. Stone. *High-Performance Computer Architecture.* Addison-Wesley, 2nd edition, 1990.

[26] D. D. Wolff and M. L. Parsons. *Pattern Recognition Approach to Data Interpretation.* Plenu Press, 1983.

Appendix A

The Computational Complexity of the Algorithm

This appendix analyzes the time and space complexity of the CHAID-based algorithm of CHITRA93. The algorithm has its foundations on the discussions in Section 2. We present the pseudo-code in a Pascal-like language.

A.1 Pseudo Code

```
PROGRAM main ($\Delta$)
      BEGIN
      Create $Sub(S, W)$
      $\Pi = Sub(S, W)$
[1]   FOR each independent position $I$ DO BEGIN
[2]         IF *independent analysis* THEN BEGIN
                  smallest = MAXREAL
                  $\mathcal{C}_D = \emptyset$
[3]               FOR each state $\delta \in \Delta$ DO BEGIN
                        $\Sigma_D = \{\delta, \overline{\delta}\}$
                        $\mathcal{C}_D = \{\delta, \overline{\delta}\}$
                        $p\_value$ = Merge($I, \mathcal{C}_D$)
                        IF smallest > $p\_value$ THEN
                           smallest = $p\_value$
                  END
                  $p_I$ = smallest
            END
[4]         ELSE BEGIN
                  $\mathcal{C}_D = \{\{\delta\} \mid \delta \in \Delta\}$
                  $\Pi = \{\langle s_1, s_2, \ldots, s_W \rangle \in Sub(S, W) \mid s_D \in \Delta, 1 \leq D \leq W\}$.
                  $p_I$ = Merge ($I, \mathcal{C}_D$)
```

```
             END
         END
         least p_value = MAXREAL
[5]      FOR each independent position $I$ DO BEGIN
             IF least p_value $> p_I$ THEN BEGIN
                 least p_value $= p_I$
                 optimal position $= I$
             END
         END
END PROGRAM.

MERGE($I, \mathcal{C}_D$) BEGIN:
[6] IF *free merge* THEN BEGIN
        $B = \mathcal{C}_I$
        $\Pi'' = \emptyset$
[7]     REPEAT
            $E = B$, the working set
[8]         WHILE $\| E \| > 2$ DO BEGIN
                    Get one element, $m$, $m \in E$
                    $E = E - \{m\}$
                    $T = \emptyset$
[9]                 FOR each $e \in E$ DO BEGIN
                        $t =$ p_value of the similarity test of $e$ and $m$
                        IF $t$ indicates that $e$ and $m$ are similar THEN $T = T + \{t\}$
                    END
                    IF $T = \emptyset$ THEN $\Pi'' = \Pi'' + \{m\}$
                    ELSE BEGIN
                        Select the most significant p_value $t \in T$, the p_value for
                        testing the similarity between state $e$ and $m$.
                        Create a new category $m \oplus e$.
                        $\Pi'' = \Pi'' + \{m \oplus e\}$
                        $E = E - \{e\}$
                    END
            END
            IF $\| E \| = 1$ THEN $\Pi'' = \Pi'' + \{e\}$
            $B = \Pi''$
        UNTIL $\| B \| \leq 2$ or when no new category is created
        $\mathcal{C}_I = \Pi''$
        Build the contingency table for position $I$ versus $D$.
        RETURN (the p_value of the above table)
[10]END
```

```
[11]ELSE BEGIN
        $B = C_I$
[12]    REPEAT
            $E = B$
            $\Pi'' = \emptyset$
[13]        WHILE $\| E \| > 1$ DO BEGIN
                    Get the smallest element $m$ from $E$
                    $E = E - \{m\}$
                    Get the smallest element $n$ from $E$
                    $E = E - \{n\}$
                    $t$ = p_value of the similarity test of $m$ and $n$
                    IF $t$ indicates that $m$ and $n$ are similar THEN $T = T + \{t\}$
                    IF $E$ has more than 1 element THEN BEGIN
                        Get the smallest element $j$ from $E$
                        $t$ = p_value of the similarity test of $n$ and $j$
                        IF $t$ indicates that $n$ and $j$ are similar THEN $T = T + \{t\}$
                    END
                    IF $T = \emptyset$ THEN $\Pi'' = \Pi'' + \{m\} + \{n\}$
                    ELSE BEGIN
                        Select the most significant p_value $t \in T$,
                        the p_value for testing the similarity between state $e$ and $n$.
                        IF $e = j$ THEN BEGIN
                            $\Pi'' = \Pi'' + \{m\} + \{n \oplus e\}$
                            Assign $n \oplus e$ the value of $n$
                            $E = E - \{j\}$
                        END
                        ELSE BEGIN
                            $\Pi'' = \Pi'' + \{m \oplus n\}$
                            assign $m \oplus n$ the value of $n$
                        END
                    END
            END
            $B = \Pi''$
        UNTIL $\| B \| \leq 2$ or when no new category is created
[14]END
END MERGE.
```

A.2 Computational Time Complexity

Two components of the algorithm are the major contributors to the complexity of the algorithm. The first is the Merge() function which is called by the main program (on steps[3] and [4]) to find the best configuration of a given independent position. The second component is the section which determines how the states of the dependent position are examined (i.e., type of analysis). After calculating the complexity of the two components, we will then use the results of the analysis to calculate the time complexity of the entire program.

A.2.1 Merge

The time complexity of Merge() is dependent on the type of merge to be performed. We discuss the two types separately.

1. **Free merge (Steps [6]–[10])**

   The complexity of this section is just the complexity of the REPEAT-UNTIL loop of step [7] which is dependent on the WHILE loop of step [8]. The WHILE loop at step [8], merely attempts to find the most similar pairs of categories in set E. There are at most $\begin{pmatrix} l \\ 2 \end{pmatrix} = \frac{l(l-1)}{2}$ possible pairs, where $\| E \| = l$, $2 \leq l \leq M_I$. This is the cost of step [8]. However, step [8] is contained within the REPEAT-UNTIL loop of step [7] which only terminates when no similar pairs of categories are found within step

[9] or when the elements of M_I have been merged together such that there are only two categories left. The best case occurs when, after one pass through step [8], no similar pairs are found. This results in a cost of $\frac{||M_I||(||M_I||-1)}{2}$, giving the best case time complexity of step [8] as

$$O(|| M_I ||^2). \tag{A.1}$$

The worst case is when a merge is performed during every pass through step[8] which costs $\sum_{l=2}^{||M_I||} \frac{l(l-1)}{2}$, giving the worst case complexity of step[8] as

$$O(|| M_I ||^3). \tag{A.2}$$

2. **Monotonic merge (Steps [11]–[14])**

Similar to the free merge, the complexity of this section is the complexity of the REPEAT-UNTIL loop of step [12] which is dependent on the WHILE loop of step [14]. The WHILE loop at step [13] attempts to find the most similar pairs of categories in set E. However, in contrast to step [8], it only considers adjacent pairs for comparisons. The cost of step [13] is therefore just $l-1$, the number of possible pairs, where $|| E ||= l$, $2 \leq l \leq M_I$. The REPEAT-UNTIL loop of step [12] has the same conditions as loop step [7]. The best case occurs when, after one pass of step [13], no similar pairs are found. The cost is $|| M_I || -1$, which gives the best case time complexity of step [12] as

$$O(\| M_I \|) \tag{A.3}$$

The worst case is when every pass of step [13] finds a pair to be merged. The cost is $\sum_{l=2}^{\|M_I\|} l - 1$, giving the worst case time complexity of step [12] to be

$$O(\| M_I \|^2) \tag{A.4}$$

A.2.2 Type of analysis

The type of analysis determines how the states of the dependent position are analyzed. The independent analysis, step [2], has an inner FOR-DO loop in step [3] which is executed $\| \Sigma_d \|$ times. In contrast, the joint analysis in step [4], does not contain a loop because it uses the entire Σ_d for the merge. For a joint analysis, therefore, the function Merge() is called only once.

A.2.3 Overall Time Complexity

We show the calculation of the time complexity for three cases.

Worst Case Time Complexity

The worst case time complexity results from a free merge and an independent analysis of the states in the dependent position. Using Equation A.2 for the complexity of Merge(), we have

$$\| \Sigma_d \| \sum_{I=1}^{\|V\|} \Sigma_I O(\| M_I \|^3) \tag{A.5}$$

where V is the set of independent positions. We make the simplifying assumptions that the set of categories used for the independent positions are the set of single states (Section 2.2) that occur in these positions (i.e., $\Sigma_I = M_I$) and that every $Sigma_I$, $1 \leq I \leq \| V \|$, has n elements. We get the worst case time complexity of

$$O(\| \Sigma_d \| \, \| V \| \, n^4). \tag{A.6}$$

A.2.4 Best Case Time Complexity

The best case time complexity is for a monotonic merge and a joint analysis. Using Equation A.3 for the time complexity of Merge(), we have

$$\sum_{I=1}^{\|V\|} \Sigma_I O(\| M_I \|). \tag{A.7}$$

With the same simplifying assumptions used in Equation A.6, we get the best case time complexity of

$$O(\| V \| \, n^2). \tag{A.8}$$

A.2.5 Typical Case Time Complexity

For the SSS's the CHITRA group has analyzed, the best models have been generated using free merge of the categories of the independent positions and joint analysis of the states in the dependent position [3]. Considering this as the typical case, we have

$$\sum_{I=1}^{||V||} \Sigma_I O(|| M_I ||^3) \tag{A.9}$$

With the same simplifying assumptions used in Equation A.6, we get the typical case time complexity of

$$\Theta(|| V || n^4) \tag{A.10}$$

Except for Equation A.8, the time complexity of the CHAID-based algorithm looks very prohibitive indeed. Fortunately, however, all the equations are linear with respect to the number of independent positions, $|| V ||$. This means that the running time of the algorithm will not tyrannically grow when the window size is increased. Furthermore, the number of unique states occurring in certain positions do not increase without bound when the size of a SSS input file is large. For problems like the Dining Philosophers, for example, the number of states are bound by the problem and do not increase when the SSS size increases.

A.3 Computational Space Complexity

The data structures for the contingency tables are the major contributors to the space complexity of the program. At any given time there is a contingency table for each independent position versus the dependent position. This gives us $\sum_{i=1}^{W-1} || \Sigma_i || \; || \Sigma_D ||$. During Merge(), a contingency table is created for the categories about to be merged which is just

$2(\| \Sigma_D \|)$. Thus the space complexity is

$$O(\| \Sigma_D \| \| \Sigma_I \|). \tag{A.11}$$

This gives us a linear space complexity for the input. We had traded space complexity for time complexity. Many of the data elements are computed every time they are needed rather than stored and retrieved.

Appendix B

Verification of the Implementing Algorithm

This chapter describes how the algorithm implementing the CHAID-based model in CHITRA93 was verified.

B.1 The Test SSS

One test SSS was taken from the SSS of the TCP/IP communication protocol program analyzed in [2]. The first 29 events were isolated from that SSS and used as the test SSS for generating a CHAID-based model. The only modification was the replacement of the state names in the SSS with the letters $a - g$. The resulting SSS from this SSS was what we had used in our examples in Chapter 2 – the SSS in Example 2.

This test SSS was chosen for two reasons. First, it comes from an actual program which had already been studied using CHITRA92. Second, it is a short SSS with a small number of unique states. This brevity allows the use of a traditional software verification technique called *desk checking* [7].

B.2 Functional Testing using the SSS

The accuracy of the model's input-output transformation was verified using an adaptation of the Functional Testing technique. As explained in [7], this technique relies on the decomposition of the model into submodels where the inputs and outputs for each submodel are identified. We tested the following three critical steps for developing the CHAID-based model using the test SSS described in Section B.1.

B.2.1 Building the Contingency Tables

Given a window size $W = 4$ and $F = 2$, the three initial contingency tables for the independent positions $-2, -1, 1$ versus position 0 were generated. Table 2.1, for example, was both manually calculated and checked against the table generated by the system. We verified that:

1. the categories for each position were indeed in the respective positions in the analysis sequence,
2. the observed frequencies were indeed accounted for in each subsequence, and
3. the expected frequencies were indeed calculated from the respective row and column totals.

B.2.2 Merges

The fundamental assumption in each step of the merge algorithm is that the two categories to be merged are the pairs with the least significant interaction (i.e., the smallest p-value) between them. The module implementing the merge algorithm was instrumented to verify that this assumption remained inviolate during every merge.

For each independent position, the statistics (i.e., χ^2 and p-values) of each candidate merge were generated and the statistics of three randomly chosen merges checked manually. Then, the p-values of all the candidate merges were manually compared to each other for verifying that the two categories with the smallest p-value were the ones being merged together to form a new category. Finally, the resulting contingency table were then checked against the initial tables generated in Section B.2.1. The column and row totals for the two tables were verified to match each other and the frequencies for the newly formed category were ensured to be the sum of the merged categories.

All the statistical functions (e.g., χ^2 functions) were obtained, without any modification, from the collection of statistical programs called "|STAT" developed by Gary L. Perlman. These functions were adapted by Perlman from the collected algorithms of the ACM. Although some of the p-values and the χ^2 values calculated by these functions were randomly tested against the χ^2 Distribution Table found in Appendix II of [4], no exhaustive tests were performed by the author to verify these functions.

All the calculations, contingency tables, and statistics presented as examples in Section 2 were taken from the results of the functional tests that had been performed as described in this chapter.

B.3 Summary

This chapter presents the steps we had followed to verify the correctness of the algorithm. The function which creates the contingency tables from the SSS was tested by comparing the contents of the generated contingency tables with the contingency tables generated manually. The functions which implemented the merge algorithm was verified by checking the χ^2 values from each merge. The statistical functions were taken from the collection of statistical functions |STAT written by Gary L. Perlman. They were accepted without any modifications and were not exhaustively tested.

VITA

Horacio Cadiz was born in Lipa City, Batangas in the Republic of the Philippines. He was graduated from Ateneo de Manila University in the Philippines with a B.S. Math in Computer Science degree in 1985. Sixteen years of education under the Jesuits (8 years in elementary school, 4 years in high school, and 4 more years as an undergraduate) seem to have left a mark on his impressionable mind. He joined the Jesuit Volunteers Philippines program as a lay volunteer and taught in a parish high school in Malipayon, Bukidnon in the island of Mindanao, in the southern Philippines. He is proud to have been "ruined for life" by the Jesuit Volunteers Program.

In 1986, after his year as a JVP volunteer, he joined the faculty of Xavier University, the Jesuit university in Cagayan de Oro City. He taught introductory computer science courses with the Xavier University Computer Center and developed software for the university. It was there in Xavier University, in 1988, where he met Joanne Ferraris.

Two years of graduate studies at VPI appear to have unbalanced Horacio. He returned home to the Philippines and married Joanne in June of 1991. That Fall semester, he returned to Tech for what he thought was his final year as a graduate student. Within a couple of months, Joanne joined him in Blacksburg. A year later, while Horacio was ostensibly working on his M.S. degree, Joanne gave birth to a baby girl. They named her Maria Frances after Horacio's Jesuit friend and Philosophy teacher Fr. Francis Reilly, S.J., who also officiated their wedding.

Failing to receive tenure as a graduate assistant at the Office of Institutional Research, Planning, and Analysis at VPI, Horacio will shortly be returning to Xavier University in the Philippines. There, he will join his wife and teach in the newly established computer science program of the university. He will be a dutiful husband and doting father to Maria Frances as Joanne writes her thesis for her M.S. in Computer Science from Ateneo de Manila University. As a teacher, he hopes to be as precise and knowledgeable, and have the same drive and energy as his adviser. Pity his students.

[illegible]