Learning Predictive Generalizations for Multiple Streams: An Incremental Algorithm

Matthew D. Schmill and Paul R. Cohen
Computer Science Dept., LGRC
University of Massachusetts
Box 34610
Amherst, MA 01003-4610
schmill@cs.umass.edu, cohen@cs.umass.edu

March 2, 1995

Abstract

We present an approach to learning complex dependencies among multiple streams of time-series data incrementally. Given a set of input streams that contain categorical values that change over time, we characterize recurring structure with a set of *dependency rules* that can be used to predict stream values in the future. These rules are general in the sense of ignoring noisy values in streams.

Keywords: dependency detection, time series, data reduction

This research is supported by ARPA/Rome Laboratory under contract F30602-93-C-0010 and by NTT Data Communications Systems Corporation.

1 Introduction

Many dynamic situations can be represented as *streams* or time series of tokens.[2] For example, the progress of a patient in intensive care might be characterized by readings from several machines at regular intervals. We want to learn rules to predict the state of a system—the token values in all streams at a particular time—from earlier states. The problem we address here is simpler in some respects and more challenging in others than the intensive care example. We learn only *lag 1* rules, which predict state at time $t+1$ from state at time t; and we assume the tokens in streams are from a relatively small alphabet, instead of real numbers. [1] On the other hand, we assume streams contain noise tokens, so we learn general rules that ignore noisy streams; and we learn incrementally, as data appear in streams. In this paper we present the Incremental Multi-stream Dependency Detection (IMSDD) algorithm and describe some of the factors that affect its performance. We compare it with MSDD, a batch algorithm that has been shown to solve multi-stream dependency detection problems quite successfully [4].

2 Definitions

Operating in the world of MSDD and IMSDD requires a few definitions and representations to keep the concepts clear. We will adopt the following terminology for the remainder of this paper.

- A *stream* is a source of categorical data values that change over time. We denote a stream by its values over time, for example *aabac*.
- A *token* t is a datum in a stream. For example, a is the first token in the stream *aabac*. The set of all possible tokens is our *alphabet*, which is of size k.
- A *multitoken* is a vector of all streams' values at a particular point in time. We represent a multitoken as a list of tokens $(t_1 t_2 \ldots t_n)$ and denote the length of a multitoken by n. Suppose we have streams *adbc* and *caac*. The second multitoken is (da).
- A *rule* is a pair of multitokens, one of which predicts the other. We denote a rule by *precursor* $\rightarrow$ *successor*. For the purposes of this paper, the successor of a rule will appear in the timestep directly after the precursor. We believe that IMSDD can be modified to generate rules of arbitrary scope (i.e., lag > 1).
- A *word* is a rule presented as a single entity. We represent a word with the form $\langle t_1 t_2 \ldots t_w \rangle$, and denote the length of a word by n_w.
- A *wildcard* is a token that indicates that a particular stream value is to be ignored. A wildcard is denoted $*$.

[1] Continuous valued time-series data is appropriate to many domains and is discussed in[1] [3][5].

2.1 The MSDD Approach to Dependency Detection

The MSDD algorithm, IMSDD's batch predecessor, approaches dependency detection cautiously. Starting with the completely general rule $(* * \ldots *) \rightarrow (* * \ldots *)$ as a root of a *generalization hierarchy*, MSDD iteratively expands that hierarchy by instantiating a single wildcard of a leaf rule at a time. Figure 1 shows an example MSDD generalization hierarchy. Each iteration of the search for predictive rules extends the frontier of the tree a single level until a predetermined size limit has been reached. At that point, MSDD concludes its training session.

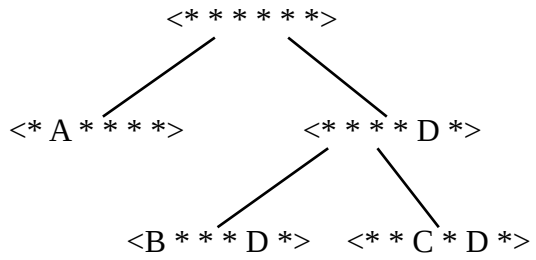

Figure 1: A simple MSDD generalization hierarchy.

MSDD accrues several benefits from this approach. Because it considers a large batch of data simultaneously, any rule that MSDD proposes can be evaluated and either rejected or accepted immediately. This allows MSDD to explore only fruitful paths in its search space, and results in a succinct, powerful model for learning rules. In the next section, we will see why these benefits are important and why an incremental approach lacks them.

2.2 Overview of Incremental Dependency Detection

The move towards an incremental algorithm serves dual purposes. An incremental system is not dependent on the quality of the training batch; this makes such an algorithm well suited for environments in which the rules change, as well as environments that present a difficult learning curve (i.e., the rules are difficult to discern). Further, an algorithm that learns progressively rather than in large, time-consuming chunks, is better suited for real-time applications. In essence, the incremental algorithm sacrifices the ability to examine and recount data in batches in order to gain adaptable, any-time behavior.

As a consequence, IMSDD is forced to abandon the generalization hierarchy exploited by MSDD. The loss of the training batch (and thus the ability to recount) makes it impossible to start with general rules and work towards more specific rules; doing so would result in a disastrous, context-less search in IMSDD's k^n generalization space.[2] IMSDD must take a data-driven, bottom-up approach to forming rules. As IMSDD receives input, it stores multitoken pairs as fully instantiated words. Tokens in streams that exhibit no contingent structure are *generalized* as a move towards representing the data's true structure. Such *noisy* streams in generalized rules are given wildcards for their values to indicate that they should be ignored.

[2]The alternative involves faking incremental behavior by maintaining a batch in memory.

The basic operation of IMSDD follows a *predict* $\rightarrow$ *verify* $\rightarrow$ *generalize* $\rightarrow$ *update* loop. Based on the current input (multitoken), IMSDD predicts what the next multitoken will contain, evaluates that prediction, uses the input to form new generalizations, and then updates its internal data structures.

At this point it is worth considering the worst-case complexities of the predict, verify, generalize, update loop. For a given word w of length n_w, there are 2^{n_w} possible generalizations. We can expect a worst case of $O(2^{n_w})$ for each of the phases except verify[3]. In i timesteps, a naive algorithm may approach $O(i2^{n_w})$ and generate the entire search space of $O((k+1)^{n_w})$.[4]

2.3 The IMSDD Memory Structure

The first step to combating the combinatorics of the dependency detection algorithm is defining a memory structure that facilitates efficient storage and retrieval of rules. With an $O((k+1)^{n_w})$ search space, a dependency detection algorithm needs a data structure that inherently limits focus to at most the 2^{n_w} generalizations that are actually relevant to any given input (and ideally, far fewer). IMSDD makes use of a structure that resembles a parse tree for the precursor of a rule, aptly named the *precursor tree*. Figure 2 shows an illustrative parsing of the precursor portion of $\langle abac \rangle$, (ab). Starting with the token a at the root, IMSDD parses the precursor by moving down the branch that corresponds to the current token, and moves on to the next token, which in the example is b. By the time IMSDD reaches a leaf in the structure, IMSDD has fully parsed the precursor multitoken of a rule. Information about what the precursor predicts is stored in the leaf as a *successor table*. Each row of a successor table represents a stream in the successor, and keeps a history of token frequencies in each of the successor streams. For example, in figure 2, row one of the successor table has recorded that an a in stream one has followed (ab) twelve times.

This structure narrows IMSDD's focus for a given precursor to 2^n rules by storing precursor information as paths in a tree and recording the successors as tables that require a simple $O(n)$ operation to query. It also provides a simple means for keeping successor counts current with respect to a particular precursor, which will become important in the prediction phase.

3 IMSDD in Detail

It is now possible to consider each step in the predict, verify, generalize, update cycle in greater detail. Further, we add and consider a *pruning* step to the cycle.

[3]Which is a simple $O(n)$ comparison.

[4]The complexity derivations are quite simple. The 2^{n_w} cases involve all possible wildcard combinations with a fully instantiated word, and $(k+1)^{n_w}$ is the full set of n_w length words with k possible tokens as well as the wildcard token.

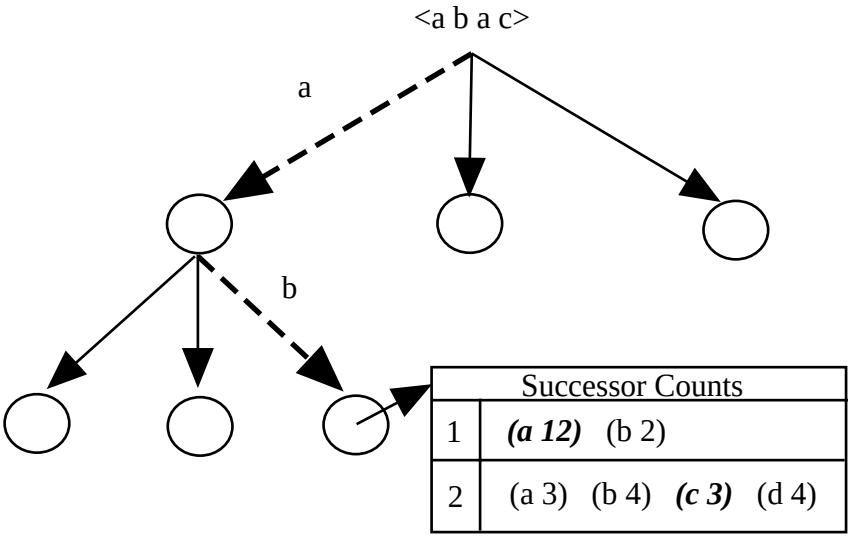

Figure 2: An IMSDD precursor tree and the parsing of $\langle abac \rangle$.

3.1 Predicting and Verifying

Predict and verify are perhaps the simplest phases of the IMSDD cycle. Given a multitoken, IMSDD's goal is to predict the next multitoken, based on the contents of the relevant successor tables in the precursor tree. Once a prediction has been made, its prediction is verified against the observed successor. Since the verification process is a trivial token-by-token comparison, we turn to a closer examination of the prediction scheme.

Suppose IMSDD observes the multitoken p (e.g. $p = (ab)$ as in figure 2). To predict its successor, IMSDD parses p through the parse tree to generate the set of all leaves whose path matches p.[5] We call this set $\mathcal{S}$, and observe that it contains all of the possible successor tables for p supported by data we have already seen. Each successor table in $\mathcal{S}$ *suggests* a unique successor s based on the frequency information it contains. This is accomplished by choosing the "best" token t from each of the rows in the successor table as rated by the function S:

$$S(t) = \alpha(N_{Hits}(t)) - (1 - \alpha)(N_{FP}(t))$$

where α is the *aggressiveness coefficient* of the algorithm, or the extent to which it values hits versus false positives, N_{Hits} is the number of times t has occurred in the successor table row, and N_{FP} is the number of times t did not occur in the successor table row. For each row j of the successor table, IMSDD suggests a token t_j which maximizes S. The result is the complete successor multitoken s. As an example, given the precursor (ab) in figure 2, IMSDD would suggest either (ab) or (ad), since a clearly dominates row one, while b and d have matching counts in row two.

It is now possible to think of $\mathcal{S}$ as a set of suggested successors to p. Next IMSDD will need to rate each complete successor in $\mathcal{S}$ to decide which is the best to apply. For this purpose, we define S_{total} on successor multitoken s :

[5] At first, it may seem that this set would contain a single leaf. This would be the case were it not for precursor generalizations, which are considered in section 3.2.2.

$$S_{total} = \alpha \sum_{i=0}^{|s|} (N_{Hits}(t_i)) - (1 - \alpha) \sum_{i=0}^{|s|} (N_{FP}(t_i))$$

All that remains is to choose the successor in $\mathcal{S}$ that maximizes S_{total}.

3.2 Making Generalizations

Intelligent control when forming generalizations (deciding that a stream should be ignored in a rule) is perhaps the single most important aspect of the IMSDD problem. Because the generalization space is exponential, the brunt of our effort in developing IMSDD was devoted to addressing this problem. Since precursors and successors to rules are treated differently in the counting scheme, it is appropriate to treat the two differently in the generalization scheme. We will look first at the simpler case of forming generalizations in the successor portion of a rule.

3.2.1 Successor Generalizations

The impetus for successor generalizion is successor data that provides little predictive power. IMSDD uses the layout of the successor table to decide when generalization is beneficial. Based solely on the distribution of token frequencies in the rows of a successor table, IMSDD can decide whether it is best to make a prediction or abstain from making one (i.e., predict a wildcard).

Upon startup, IMSDD assigns a constant S value, τ, to the wildcard token. This value is based on the aggressiveness of the algorithm, α, and the probability of correctly guessing a token by chance. Each time IMSDD attempts to predict a stream value, it considers a wildcard token along with all those tokens that were actually observed to occur. Should no token's S value be high enough (greater than τ) for IMSDD to believe it did not occur by chance, the algorithm will suggest a wildcard for that stream.

The result of this simple scheme is a substantial saving in time and space requirements. However, it relies on an important assumption. By storing only token frequencies in the rows of a table rather than complete multitokens, IMSDD assumes that streams within a successor are independent, or because they are dependent, the dependent stream values will always occur together, and thus either all of them will dominate the successor table, or none will. The benefits of this assumption are a succinct means of recording successor frequency, and a generalization mechanism that requires little extra time and space of the algorithm.

3.2.2 Precursor Generalizations

Generalizations in the precursor of a rule come at a somewhat higher cost. The reason is that successor tables are only valid for a single precursor, and so when IMSDD generalizes, it must have a unique successor table for the new, generalized precursor. The result is a procedure that generates precursor generalizations opportunistically, and creates a new path in the precursor tree to represent the generalized precursor. Figure 3 shows the path $(*b)$

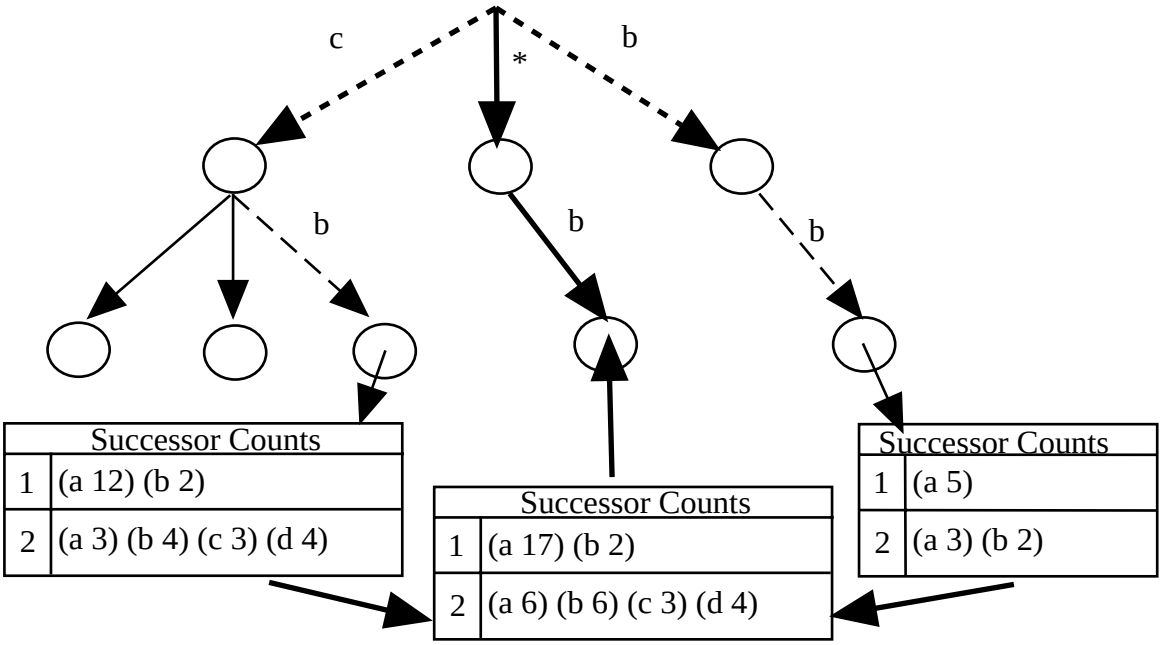

Figure 3: Forming a precursor generalization ($*b$) from (cb) and (bb).

in an illustrative precursor tree, along with a successor table that is the composition of two successor tables whose precursors match the new generalization.

The greatest challenge in generating precursor generalizations is deciding when to make a generalization, because there are $O(2^n)$ possibilities and no easy answers as there are with successor generalizations. To control the explosiveness of the generalization routine, IMSDD constrains precursor generalizations in the following ways :

- A precursor path must exist in the precursor tree which differs from the current precursor in exactly one stream (any token matches a wildcard).
- The successors to the nearly-matching precursors must also match or nearly match.

The first of these constraints is implemented in the search for matching precursors. A mechanism called the *wrong turn procedure* locates all matches to a precursor that are off by one token. The wrong turn procedure parses the new precursor into the precursor tree, and at each level of the parse, considers what would result if the token at that level was parsed *incorrectly*. When the algorithm is left to run its course, it will return precisely those leaves in the tree whose paths differ from the new precursor by exactly one stream value.

In addition, IMSDD offers another speedup option. Instead of considering all wrong turns at each level of the parse, IMSDD samples randomly some proportion of the possible wrong turns. Here, IMSDD operates under the assumption that a uniform distribution of stream values (which indicate that a generalization is going to be useful) will ensure that all important matches will be manifest at least once through random sampling.

Once the matching precursors have been found, IMSDD has a set of plausible generalizations and the actual precursors which support them. A generalization will be accepted

as good if all those precursors which match it suggest similar successors. The new generalization will then be recorded as a path in the precursor tree. At the end of the path, as in figure 3, a new successor table is formed, and the supporting precursors' successor tables are combined to fill in the token frequencies.

As the number of streams presented to IMSDD increases, it becomes increasingly unlikely that two successor tables will be similar, especially if some of the streams are noisy. This causes precursor generalizations to fail on the matching-successor constraint. Consequently, IMSDD could run for many timesteps before having the opportunity to make a single good generalization. IMSDD offers as a satisficing solution a mechanism that resembles *simulated annealing*. When IMSDD is filtering the possible generalizations, it will with some probability γ, which diminishes with time, *trivially accept* each one whether it meets the constraints or not. In effect, this property overcomes the bootstrapping problem by agitating the rule base early on until there are some good generalizations. These good generalizations will have higher counts in the successor that can enable noisy streams to be identified.

3.3 Updating

Counting successor frequencies is vital to correctly quantifying the predictive accuracy of a rule. When IMSDD observes a word, it parses the precursor portion in its precursor tree, finding all paths that the precursor matches. For example, the precursor (bb) would match $(*b)$ and (bb) in the tree of figure 3. In each path's successor table, the token count for each row is incremented according its corresponding stream value. Using the same example, if precursor (bb) was followed by successor (ad), the counts in the $(*b)$ successor table frequencies would change to $(a\ 18)$ in row one and $(d\ 5)$ in row two.

3.4 Pruning

The final component to the IMSDD algorithm is the pruning component. By pruning, IMSDD attempts to bound the search space to contain only those rules that have occurred recently or have proven useful. Rules are selected for pruning during the prediction phase. During IMSDD's selection process to find the best successor to predict, the pruning component selects a fixed number of the worst rated rules. These rules, unless used under different circumstances within a certain period of time, will be pruned.

4 Empirical Evaluation

Because of the potential complexity of dependency detection, IMSDD was under constant evaluation to determine the impact of design decisions on its performance. For the purposes of testing, a system for creating artificially structured datasets was developed. For a given number of streams, ASG, the artificial data generator, produces a set of general rules, and generates a series of multitokens, some containing noise, and some containing actual structure. We define the metrics *adjusted hit rate (ahr)* to be the number of correct token predictions divided by the number of tokens seeded in the dataset,[6] and *fp-rate (fpr)* to be

[6]Since IMSDD may correctly predict tokens that occur by chance, the *ahr* metric often exceeds 1.0.

the number of incorrectly predicted tokens divided by the total number of tokens.

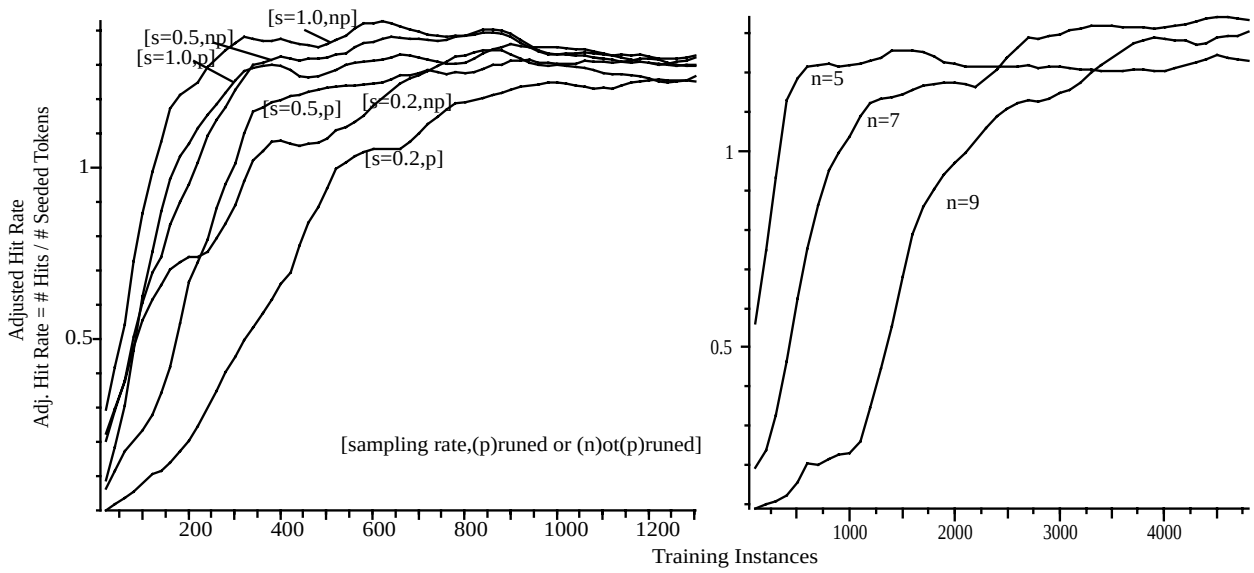

Figure 4: **(a)** Effects of sampling and pruning on learning curves. **(b)** Learning curves for $n = (5, 7, 9)$.

Figure 4 shows IMSDD's learning curves for *ahr* and *fpr* on IMSDD experiments with varying parameters. The curves were produced by recording IMSDD's performance on a fixed test batch every 100 timesteps during the learning process.

The first mechanisms we examined were the pruning strategy and the sampling policy for precursor generalization. We recorded the learning curves generated by IMSDD as we varied the sampling rate and turned pruning on and off. Figure 4a suggests that both pruning and sampling have small effects. The effects of pruning and sampling on false positive rate (not pictured) were somewhat surprising; both sampling and pruning led to slower increase rates and lower peaks in the curves[7], while the steady state of each curve was similar.

We next explored the effects of increasing n, the number of streams. Figure 4b shows the effect of increasing n from 5 to 9. First note that the overall slope of the learning curve is similar for all three values of n. Second, the onset of the learning curve and the point at which IMSDD can account for 100% of the structure differ are delayed as n increases. The first result suggests that the learning algorithm, when scaled up might exhibit the same facility for learning rules and accounting for structure. The second result implies that due to the larger stream size, there is some degree of difficulty learning good initial generalizations given the added dimensions in search space size.

While these results were encouraging, it remained to test IMSDD against its tried and true predecessor MSDD [4]. Using identical data, we ran the 5 stream dataset through MSDD, varying the size of its training batch, to see how IMSDD compared. Examination of the *ahr* curve showed that MSDD was quicker to account for 100% of the present structure, but that IMSDD was not far behind. The *fpr* curve suggested that MSDD's false positive

[7]Some of the fpr curves rose sharply and then dropped.

rate was somewhat lower at 0.23, and constant, while IMSDD's *fpr* curve appeared similar to its learning curve, scaled down to peak at 0.3.

5 Conclusions

We have presented an algorithm that incrementally learns rules which characterize contingent structure in multi-stream, time series data. The IMSDD algorithm uses a bottom-up, specific-to-general approach to dependency detection to refine stream data into a set of rules that correctly identify predictive relationships as well as noisy streams. Further, we introduced a precursor tree representation, sampling and pruning strategies, and a compact representation of successor frequencies to reduce the inherent exponential nature of the problem to a manageable level. Empirical results suggest that IMSDD is capable of efficiently managing its rule base to make accurate predictions of structure in time series data. Results of testing against MSDD, our standard for dependency detection algorithms, indicate that IMSDD does not learn as quickly as a batch algorithm might, but given adequate time, can exhibit equal performance.

Acknowledgments

This research is supported by ARPA/Rome Laboratory under contract F30602-93-C-0010. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes not withstanding any copright notation hereon. Additional support came from NTT Data Communications Systems Corporation.

References

[1] John Fox and J. Scott Long. *Modern Methods of Data Analysis.* Sage, 1990.

[2] Adele E. Howe and Paul R. Cohen. *Selecting Models from Data: AI and Statistics IV*, chapter Detecting and Explaining Dependencies in Execution Traces, pages 71–77. Springer-Verlag, 1994.

[3] Berndt Donald J. and James Clifford. Using dynamic time warping to find patterns in time series. In *Proceedings of the AAAI-94 Workshop on Knowledge Discovery in Databases*, pages 359–370, 1994.

[4] Tim Oates, Dawn E. Gregory, and Paul R. Cohen. Detecting complex dependencies in categorical data. In *Preliminary Papers of the Fifth International Workshop on Artificial Intelligence and Statistics.*, pages 417–423, 1995.

[5] John W. Tukey. *Exploratory Data Analysis.* Addison-Wesley, 1977.