ORNL/TM-13522

ornl

OAK RIDGE NATIONAL LABORATORY

LOCKHEED MARTIN

CLASSIFICATION OF TIME SERIES PATTERNS FROM COMPLEX DYNAMIC SYSTEMS

Jack C. Schryver
Nageswara S. V. Rao

MANAGED AND OPERATED BY
LOCKHEED MARTIN ENERGY RESEARCH CORPORATION
FOR THE UNITED STATES
DEPARTMENT OF ENERGY

ORNL-27 (3-96)

ORNL/TM-13522

Computer Science and Mathematics Division

CLASSIFICATION OF TIME SERIES PATTERNS FROM COMPLEX DYNAMIC SYSTEMS

Jack C. Schryver
Nageswara Rao
Computer Science and Mathematics Division

Date Published: July 1998

Research sponsored by the
Laboratory Directed Research and Development Program
of the Oak Ridge National Laboratory,
managed by Lockheed Martin Energy Research Corp.
for the U. S. Department of Energy
under Contract No. DE-AC05-96OR2264

[illegible]

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Abstract

Numerical time series data is becoming more prevalent in large-scale databases created by empirical research conducted particularly within the transportation research community. Locating and identifying temporal patterns of interest in numerical time series data is an increasingly important problem for which there exists few available techniques. Numerical time series pose many interesting problems in classification, segmentation, prediction, diagnosis and anomaly detection. This research focuses on the problem of classification of numerical time series data where prior knowledge of pattern form is available. Our approach is nonparametric and allows for a wide variety of abstract template families for recognition of temporal patterns under a unified and well-understood mathematical framework (automata theory) without having to specify functional forms. Finite state automata, which are normally used with nominal alphabets, were extended to recognize temporal patterns constructed from numerical alphabets to allow arbitrary stretching and compression, rescaling, alternative or repeated subpatterning, concatenation, and recursive patterning. This approach offers several advantages over ad hoc solutions or efforts to develop new special-purpose languages for temporal pattern recognition. It operates on raw time series and does not depend on an intermediate feature extraction step. It is particularly suitable when large training sets are not available, e.g., for supervised learning of multilayer perceptrons. The new algorithm builds upon previous work in dynamic programming algorithms for approximate string-automata matching by exploiting additional structure in numerical strings as compared with strings composed from nominal alphabets. The matching algorithm accepts numerical strings as input, and uses logical preconditions and memory for special variables to restrict search in a discretized range. It computes the substitution cost of the best-matching numerical string to the input recognized by the automaton. The automata are very compact and intuitive for even the casual user. The matching algorithm is implemented in an interactive graphical environment in which the user is given considerable freedom to explore the capabilities of the matching algorithm. The interactive interface allows the user to browse large numerical time series in a graphical window and select a template or recognizer from a template library. The user frames a subsequence for classification with the use of sliders, and simply presses a button with the mouse to apply the matching algorithm to the subsequence. The interface not only shows goodness-of-fit to the input string, but also displays the best-matching string by plotting it over the input data. The area of mismatch between input string and the template is highlighted in red, thus providing the user with valuable insight into the quality of fit. In order to demonstrate the power of automata for temporal pattern-matching, templates were developed for recognition of flat, piecewise linear, monotonic decreasing, monotonic increasing, oscillatory, concave, convex, and stopping sequences. It was demonstrated how more complex temporal patterns could be recognized using concatenation and recursion of simpler automata. The matching algorithm was validated on artificial input strings generated from seven different pattern families for which automata had been constructed. All costs were computed to fill a 7 x 7 confusion matrix. Costs along the diagonal were lower overall than off-diagonal costs. More general automata generated costs at least as low as specific automata contained by the general automata. Additional validation runs were completed to fill a 4 x 4 confusion matrix using templates and signals representing continuous downward, smooth, step and pump stopping strategies. The stopping signals were derived from actual field data. All stopping templates fit each of the stopping signals quite well, although in general there was no incremental benefit for templates designed for specific signals. Automata or templates were sometimes able to provide good fits to temporal patterns for which they had not been designed, suggesting that template designers need to carefully define restrictive preconditions to narrow the set of recognizable temporal patterns. The matching algorithm shows great promise as a potential tool for the transportation analyst interested in characterization of time series data.

1.0 INTRODUCTION

An increasing availability of high-performance computing and data storage media at decreasing cost is making possible the proliferation of large-scale numerical databases and data warehouses. Numeric warehousing enterprises on the order of hundreds of gigabytes to terabytes are a reality in many fields such as finance, retail sales, process systems monitoring, biomedical monitoring, surveillance and transportation. Large-scale databases are becoming more accessible to larger user communities through the internet, web-based applications and database connectivity. Consequently, most researchers now have access to a variety of massive datasets. This trend will probably only continue to grow over the next several years. Unfortunately, the availability of integrated tools to explore, analyze and understand the data warehoused in these archives is lagging far behind the ability to gain access to the same data. In particular, locating and identifying patterns of interest in numerical time series data is an increasingly important problem for which there are few available techniques. Temporal pattern recognition poses many interesting problems in classification, segmentation, prediction, diagnosis and anomaly detection. This research focuses on the problem of classification or characterization of numerical time series data.

Time series data is becoming commonplace in databases containing measurements from complex dynamic systems. We define a time series of length N as an ordered sequence $x_1x_2x_3 \ldots x_n$, where x_i is the scalar value of the variable 'x' at time t_i. Increments of time between adjacent values x_i and x_{i+1} are typically small and equal-intervaled.

Highway vehicles and their drivers are examples of complex dynamic systems (CDS) which are being used by transportation agencies for field testing to generate large-scale time series datasets. Tools for effective analysis of numerical time series in databases generated by highway vehicle systems are not yet available, or have not been adapted to the target problem domain. However, analysis tools from similar domains may be adapted to the problem of classification of numerical time series data.

1.1 Classification and Data Exploration of Time Series (CADETS)

Many large-scale data collection efforts in CDS have been motivated by an intent to test a small number of hypotheses. Statistical testing exploits a tiny percentage of the information available in a large-scale database. After planned statistical tests have been performed, it is often the case that no further analysis of the data is attempted. However, the database may contain additional patterns of significant interest or value to other researchers.

The development of a more comprehensive understanding of large-scale databases has a significant exploratory and opportunistic component, as contrasted with specific hypothesis testing. The potential information value of large-scale databases is so extensive that a research community simply cannot anticipate all relevant hypotheses and sources of regularity among the data.

What are the ways in which time series data can be manipulated to understand the behavior of systems which generate them? We sketch a general process model of time series data understanding for CDS, and refer to it as Classification and Data Exploration of Time Series (CADETS). The process highlights the search for interesting temporal patterns in databases.

The first requirement in exploiting a large data warehouse is to access relevant data. The analyst identifies time series segments of potential interest. A typical time series may extend over long time intervals, and the next task would be to preview a time series in order to spot outliers, prominent trends and obvious correlations. Strip charts can be stacked in vertical format, preserving synchronization of time scales of each chart in the column. Long-term time series data may contain several distinct episodes that are explainable by different processes or conditions. By segmenting these data, the analyst can effectively narrow the search. For example, the fast whole-sequence and subsequence matching techniques of Agrawal et al. (1993) and Faloutsos et al. (1994) provide a solid foundation for segmentation. The identification of regularities or patterns in a segment is called classification. If the data is contaminated with too much noise, it may be necessary to transform a time series representation into a more compact feature space as a preprocessing step for efficient classification. Feature extraction is a common aspect of pattern recognition techniques in many application domains. The output of classification may be formulated as a more abstract representation than raw time series, where the data are specified at the event level. The new event representation may be descriptively modeled or re-visualized in new ways. In the final step rules are extracted from abstract data descriptions which deepens understanding of relationships in the time series.

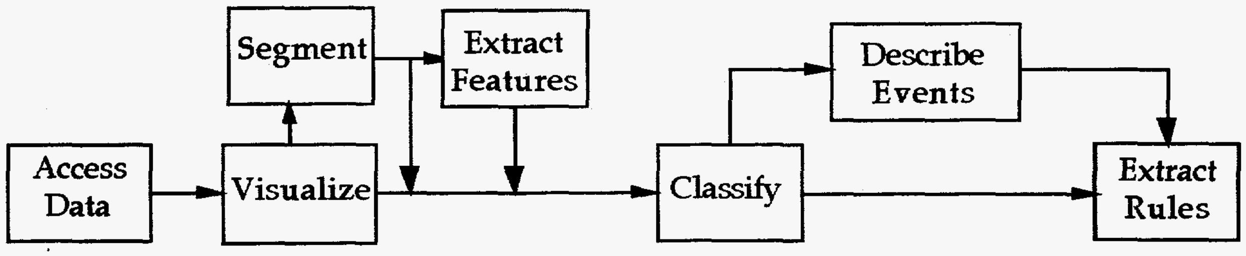


Figure 1. CADETS Process Model.

Some aspects of the CADETS process are supported by existing software packages, although few or none incorporate the entire process. Software tools are available for visualization of multiple time series. Raw time series data can also be imported into spreadsheet and statistical analysis programs for more detailed analysis. Segmentation and classification are significant operations that are not adequately addressed by classical time series analysis techniques. Many feature extraction methods have been developed, but it is not known which features will turn out to be useful for classification of temporal patterns in highway vehicle driving and other CDS.

1.2 Applications of Time Series Classification in Highway Vehicle Analysis

Large-scale multivariate time series data (e.g., speed, yaw rate, lane position, steering angle) can be generated during the performance of even relatively simple tasks such as driving to work. If an automobile is outfitted with sensors that sample vehicle/driver parameters at relatively high rates, e.g., 20 Hz., then a single time series contains 72,000 data points per variable after a one-hour test.

The transportation researcher may want to find specific temporal patterns in these long time series. A typical pre-analytic task is the segmentation of time series data with respect to basic driving tasks (lane-following, lane-changing, curve-following, overtaking and stopping at an intersection). Alternatively, suppose the analyst possesses data describing braking behavior on icy road surfaces just before collision or near-miss. The investigator wishes to know whether an undesirable outcome is preceded by a particular style of braking behavior.

A final example focuses on specific highway vehicle safety research needs. Suppose we have normative driving data with a large sample of drivers under varied roadway, environmental and traffic conditions. This data is compared with other data collected under controlled conditions where proposed safety countermeasures are taken. One set of countermeasures involves vision enhancement schemes (especially for elderly drivers) to mark salient roadway features. A simple vision enhancement countermeasure is a sequence of reflective surfaces placed along the centerline and shoulder of a curved roadway segment. An analysis tool capable of discriminating prescriptive (safe) and nonprescriptive (marginally unsafe) vehicle performance is desired to test the hypothesis that countermeasures are more likely to be associated with safe curve driving. We define safe performance as well-controlled steering and braking inputs within roadway geometry constraints. This definition allows for a wide variety of individual curve-driving strategies, each of which are equally acceptable. We avoid strong (and probably unnecessary) assumptions that penalize any deviation from the centerline of the lane through the curve. A successful classifier will accept different curve-following strategies, but reject marginally unsafe performances, even if there is no evidence of roadway departure.

Figure 3 shows a set of steering input functions created to represent instances of several different curve-following strategies. Smooth steering is an economical strategy that attempts to

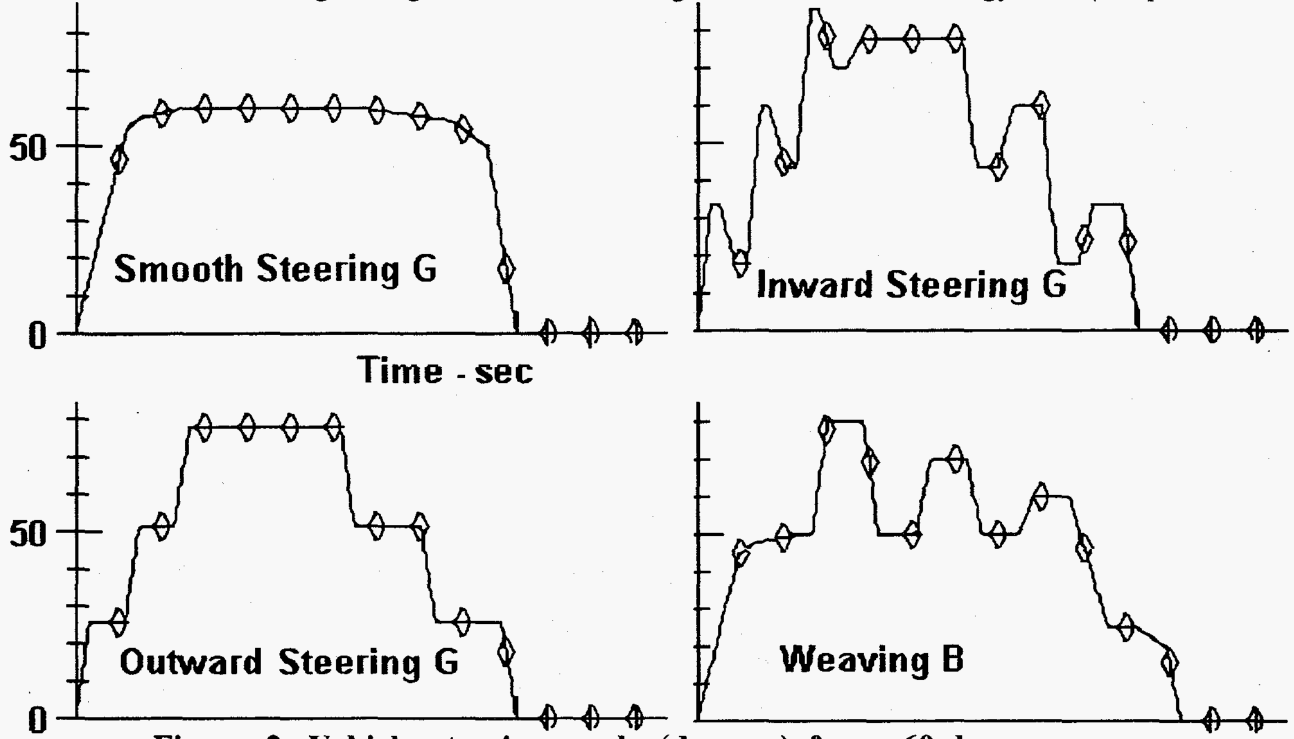


Figure 2. Vehicle steering angle (degrees) for a 60-degree curve.

minimize steering reversals. Inward steering aims for the inside of the curve with strong open loop responses, and relies on quick small steering angle adjustments to keep the vehicle within the lane boundary. The vehicle is kept to the outside in outward steering, correcting steering angle input only enough to avoid driving onto the shoulder. Everyday driving may lie somewhere between these pure strategies. The final plot illustrates a marginally unsafe steering pattern for curve-following; the weaving pattern is kind of a less-controllable deformation of smooth steering input. Each pattern was submitted as steering input to CarSim, a vehicle dynamics simulation program available through the University of Michigan Transportation Research Institute. The resulting trajectories in the horizontal plane were nearly identical for all steering input patterns, so each strategy, even weaving, avoided any roadway departures.

If one examines the lateral acceleration plots in Figure 3, it appears as if simple measures such as number of zero crossings or reversals cannot correctly distinguish the three acceptable steering strategies from the weaving pattern. On the other hand, notice that each plot has its own distinctive features, e.g., smoothness in the top-left plot, saw-tooth features in the top-right plot, gentle scalloped curves in the bottom-left plot, and triangular pointed shapes at bottom-right. Although differences among these curves are visually apparent, traditional classifiers and ad hoc methods would not likely perform well on the curve-following classification problem. The problem is further compounded if one wishes the classifier to be invariant across a range of radii of curvature, vehicle speeds, automobile handling characteristics and environmental conditions.

In each case described above, prior knowledge is available about the pattern we wish to observe. The patterns are abstract because an entire family of nonparametric curves fits the template equally well. For example, we might wish to define a template that is invariant with respect to transformations in scale, or expansion and compression of the fundamental signal. However, the availability of appropriate tools for this kind of temporal pattern recognition is quite limited.

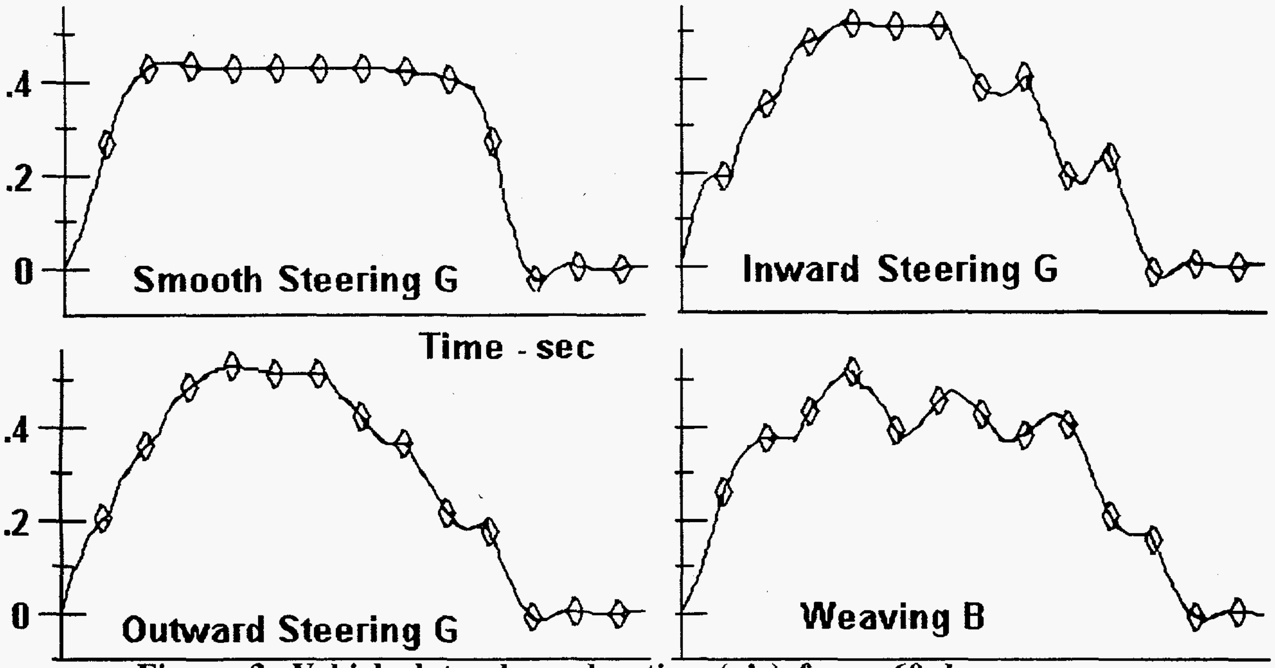


Figure 3. Vehicle lateral acceleration (g's) for a 60-degree curve.

The next section begins by briefly reviewing classical time series methods and concludes that they are not particularly relevant to the problem of classification. Following subsections introduce methods employed in similar problem domains, and which may be applicable to analysis of time series transportation data.

2.0 BACKGROUND

2.1 Classical Time Series Analysis

Time series methods have been successfully employed for several decades. The classical model-based approach is represented by autoregressive, moving average and integrative processes that are subsumed under the ARIMA model, and are characterized by Box, Jenkins, and Reinsel (1994). These highly parameterized models are chiefly used for prediction and control. A typical application of the ARIMA model is to forecast retail demand for a certain stock item, for the upcoming quarter, given a time history of sales over the past year. The periodogram is another classical method used to analyze sinusoidal or periodic frequencies of time series buried in noise. Transfer function models are a third set of classical methods derived from feedback control theory.

One problem with model-based approaches is that they tend to make restrictive assumptions including linearity, stationarity and (piecewise) homogeneity. These assumptions are known to be false for time series originating in many real systems. Stock market prices and highway vehicle time series are probably both nonlinear and nonstationary. One reason is that the process governing a single time series variable is often a complex function of environmental, system and human response functions. Bendat and Piersol (1995) acknowledge that little can be done to parameterize this kind of data, unless one has a detailed understanding of the specific nonlinear and nonstationary process which generates each time series variable.

It is clear that ARIMA models, transfer function models, and other classical approaches are the wrong class of methods for the problem of understanding trends in CDS. They are useful for prediction and control, but not segmentation and classification operations that are needed in order to understand the underlying processes in a time series. We find a more convenient formulation of the problem and data in applications of speech recognition, plant signal processing, syntactic pattern recognition and DNA sequence analysis. The following sections survey classification methods from these areas, and are evaluated for their potential usefulness for time series analysis of CDS.

2.2 Speech Recognition Methods for Time Series Classification

Any time series can be regarded as a signal. Understanding and extraction of knowledge from time series can be framed as a signal processing problem. Methods to search for temporal patterns in the signal are needed in order to provide meaning or explain the empirical process. In many cases reference patterns or templates are available or can be derived from prior knowledge. In speech recognition, a continuous signal is searched in order to identify phonemes, words and sentences. A reference pattern is used to represent the phoneme which when voiced may stretch or compress in time, or register loudly or softly. At each level of abstraction, the speech recognition problem is one of classification of a signal.

State-of-the-art speech recognition systems are relatively speaker-independent, with vocabularies of up to 40,000 words in a benign environment. Isolated word recognition can perform with error rates less than 5%, whereas continuous sentence recognition does not fare as well, with error rates exceeding 10% (Rudnicky, Hauptmann, and Lee 1994).

A digital signal is subjected to analysis which generates a sequence of feature vectors. The recognizer searches an event space to find the phoneme or word with the highest likelihood of generating the feature vectors. There must be a complete set of models for all events, and an algorithm for computing the probability or goodness of fit with the data for each model or reference pattern.

Originally speech recognition proceeded directly from the signal, but it was soon discovered that feature extraction minimized the large variability of the speech signal. Computing the envelope of the short-term spectrum eliminates source characteristics, such as whether the sound is voiced or fricated, and periodicity or pitch. The short-term spectrum may be estimated over a 20-msec frame, and updated every 10 msec. Frequency warping usually occurs after the features are extracted. The most popular features for speech recognition are the mel-frequency cepstral coefficients. The coefficients are obtained by performing a mel-scale warping and logarithmic transform of the spectrum, followed by an inverse Fourier transform. The first dozen cepstrum coefficients are retained to capture essential information in the spectral envelope. During the 1980s the primary technique used for speech recognition was dynamic time warping. Modern techniques generally utilize hidden Markov models, time-delay neural networks, or a combination.

2.2.1 Dynamic Time Warping (DTW). Although DTW is now little used for speech recognition (see Silverman and Morgan, 1990 for a recent review), there is renewed interest in the method within the data mining community (Berndt and Clifford 1996). DTW is based on a conceptually simple algorithm and is relatively easy to program. DTW can handle the raw signal as input, eliminating the tedious feature extraction step. This option may prove to be critical if discriminative feature sets for a particular set of patterns and signals are not well-known. If both the signal and reference pattern (template) are time series of length N and M respectively, an N X M grid of signal by template can be formed, where each gridpoint corresponds to an alignment between an element from the signal (S) and template (T). Figure 4 illustrates the warping path, W, which aligns pairs of signal and template elements such that the distance between them is minimized. If W lies directly on the main diagonal between S and T there is no timing offset. We can define the distance between elements of S and T as the absolute difference or the square of the difference between values.

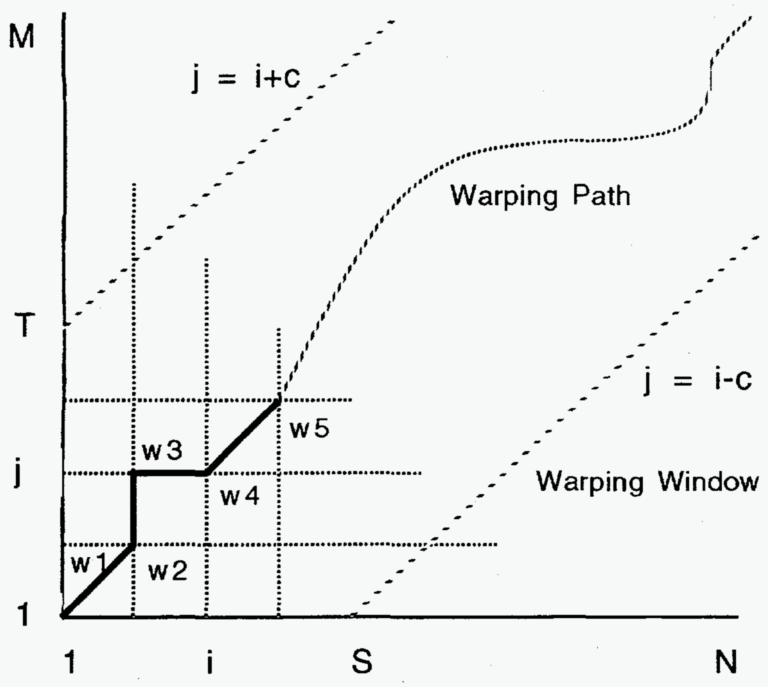


Figure 4. Warping Path from signal to template.

The DTW problem is formulated in terms of dynamic programming, i.e., the minimization over potential warping paths based on the cumulative distance for each path. Several decision variables are employed to restrict the search space and the permissible paths between two grid points. The restrictions are monotonicity (points in W are monotonically ordered with respect to time), continuity (only neighboring points are allowed), size of the warping window (c), slope constraint (on the warping path), and boundary conditions, such as endpoint anchoring. The recursive definition of cumulative distance, $\gamma(i,j)$, for each point is given by:

$$\gamma(i,j) = \delta(i,j) + \min\left[\gamma(i-1,j), \gamma(i-1,j-1), \gamma(i,j-1)\right]$$

The cumulative distance is just the sum of the simple distance for the current point and the minimum of the cumulative distances of neighboring points. The dynamic programming algorithm fills in a table of cumulative distances as computation fans out, so that previous calculations are not redone. When the algorithm halts, the optimal warping path is found by tracing backward in the cumulative distance table and following neighbors with the least cumulative distances. The goodness-of-fit of the optimal warping path is assessed by computing a normalized cumulative distance score in order to compensate for scale effects, and comparing it with a baseline value. Alternatively, one could classify the signal by selecting the reference pattern yielding the greatest normalized cumulative distance score.

2.2.2 Hidden Markov Models (HMM). A classical Markov chain consists of a set of states and a set of transitions among pairs of states. Each transition is associated with a probability (such that probabilities for all transitions from each state sum to unity), and each state possesses an output symbol. An output sequence is uniquely defined by a specific route through the set of states. HMMs acquired their name because a unique mapping between states and outputs cannot be established. Figure 5 shows a discrete probability density function associated with each of three states. The output symbol for each state transition during run time is randomly selected from the associated distribution. The stochastic model does not guarantee the same result each time the same sequence of state transitions occurs. HMMs are doubly stochastic because both transition and output probabilities come into play. The Viterbi search algorithm is used to find the sequence of states most likely to generate a given output sequence, without searching all possible sequences (Makhoul and Schwartz 1994).

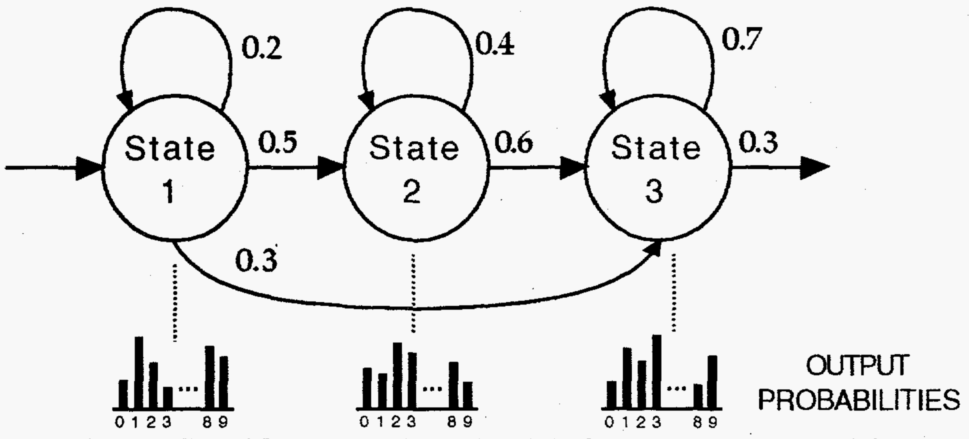


Figure 5. Hidden Markov Model for speech recognition.

Most phonetic HMMs are 3-state left-to-right models as in Figure 5 which represent the beginning, middle and end stages of an utterance. Transitions are restricted to either the current state or the state to the right of the current state.

2.2.3 Time-Delay Neural Networks. Bottou, et al. (1990) and Lang, Waibel and Hinton (1990) describe the use of time-delay neural networks (TDNN) to perform respectively digit and isolated word recognition. The TDNN architecture is just a backpropagation feedforward neural network with a single hidden layer which encodes sharp discriminative features for speech recognition. In the latter study the input was a spectrogram composed of short time frames (3 ms) and eight features. Both hidden and output layers contain replicated units which were constrained to ensure that copies of a given unit have the same weights. The network factors out the position of features in its input patterns by summing the activations of replicated output units connected to small receptive fields. The replication units impart a kind of translation invariance or tolerance for alignment error between signal and units. Another way of describing the architecture without using replication units is to connect hidden units to each input unit by several different links, where each link has a time delay of 0, 1, 2, ..., n. The same multiple time delay links are built into the hidden-output connections. The input spectrogram is scanned one frame at a time, and activation is iteratively clocked upwards through the network.

Although the TDNN is a very good speech recognizer, it requires feature extraction as a preprocessing step, and the availability of a large training set to identify the large set of weights.

2.3 Partially-Recurrent Neural Networks (PRNN)

A traditional feedforward ANN representation of time series input typically leads to fixed input length, liability to alignment error, and a large number of weights that must be estimated during training. Time-delay neural networks also require a large number of weights relative to the length of the input vector. Elman (1990) introduced a PRNN architecture that has been proven to be a powerful tool (Kremer 1995) for classification of time series data. An entire time subseries is not accepted as a single pattern in an Elman network. Rather, individual time samples are posed in sequence. Since the time samples are arranged in a stack of arbitrary length, the fixed input length and alignment problems are elegantly handled by the Elman PRNN. The length of the input series is free to vary. Training is tractable because networks are more compact due to smaller input vector size. A greater number of patterns are also available for training and testing because there is exactly one pattern associated with each time step.

The Elman network closely resembles a multilayer perceptron with the addition of a context layer. Each time sample is presented to an Elman ANN at the input layer. The input layer is fully interconnected with the hidden layer which encodes the emergent features of the input. The output layer encodes the classification learned by the network, and is connected to all units in the hidden layer. Elman networks store sequence information in a state vector or context layer. After each element in the series is processed, activations in the hidden layer are copied directly into the context layer. The context layer is a memory storage which preserves the current network activation state, allowing the network to learn sequence information. The context layer feeds back into the hidden layer during presentation of the next input.

Elman networks have been used for discovering syntactic and semantic feature for words, speech recognition, phonetic to acoustic mapping, and recognition of trends in time-ordered medical parameters. Since it is a supervised learning procedure, successful application depends on the availability of a large number of temporal patterns for training and testing.

2.4 Qualitative Modeling of Plant Signals

Cheung and Stephanopoulos (1990a,b) adopted an artificial intelligence approach to representation of process trends embedded in plant sensor signals. Although the motivation was to find a meaningful descriptive representation of system state for monitoring, diagnosis and control, the formal notion of qualitative trend may be very useful for classifying time series patterns in other systems.

Cheung and Stephanopoulos introduced a triangular representation of process trends. A triangular description is a contiguous series of triangular episodes. Episodes are bounded on both sides by zero crossings, i.e., extrema and inflection points. There are only seven triangular primitives, and only four are usually needed:

A. Concave Downward; Monotonic Increase - first derivative positive & second derivative negative
B. Concave Downward; Monotonic Decrease - first and second derivatives negative
C. Concave Upward; Monotonic Decrease - first derivative negative & second derivative positive
D. Concave Upward; Monotonic Increase - first and second derivatives positive
E. Linear Increase - first derivative positive and second derivative zero
F. Linear Decrease - first derivative negative and second derivative zero
G. Constant - first derivative zero

This representation is complete, correct and robust, and allows for explicit description of important information about a trend. A trend can be represented in purely qualitative terms, as for example, the string "ABDC" describes a particular trend. But the triangular representation also contains quantitative information which can be used to compute characteristic parameters such as gain, rise time, settling time, overshoot, decay ratio, period of oscillation and dead time. Figure 6 shows the landmark values a,b,ya, and yb as well as the rate values c and yc for the triangular episode described in (D). A high-level description of a trend such as "The reactor feed is improving from 290 K to approach 310 K during time interval t1 and t2" can easily be generated from the information in a triangular representation. A very good reconstruction of the original signal can be obtained from a triangular description.

A problem arises in working with a triangular description of a raw process signal. Noise and events are represented equally well, and it is difficult to distinguish important features. Underlying processes that produce the signal may occur at different scales. It is therefore desirable to analyze a signal over various scales by constructing a multiscale, hierarchical description. Since many processes are localized in time, the appropriate domain for analysis is time-frequency space. The wavelet is an efficient smoothing function localized in time and frequency domains that is capable of providing a multiscale representation with an appropriate choice of a scaling function. It is possible to locate inflection points (and therefore triangular episodes) of a smoothed signal with the extrema and zero crossings of a cubic spline wavelet and scaling function.

Bakshi and Stephanopoulos (1994a,b) discuss the idea of using a wavelet interval tree to represent the structure of scale. First, consider a multiscale decomposition of a signal where wavelet representations of increasing scale (or decreasing resolution) are lined up vertically.

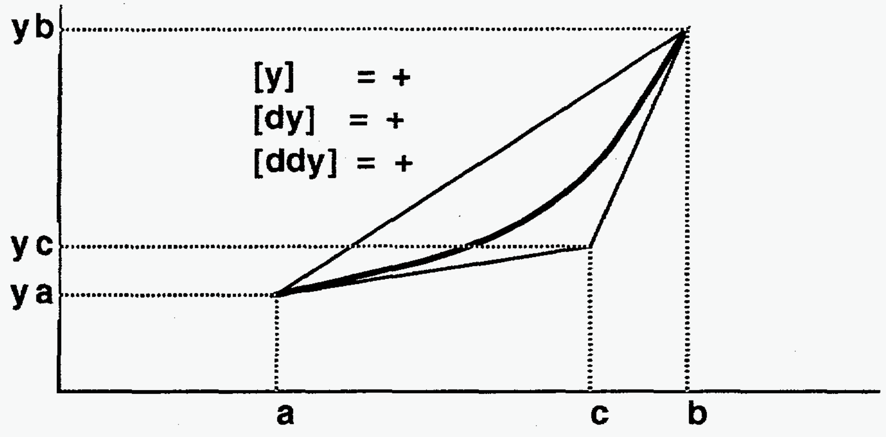


Figure 6. A triangular representation.

Connecting wavelet transform extrema across vertically aligned scales using inflection points located in the detail signal provides a hierarchical interval tree structure. By identifying the triangular representation at each interval, it is possible to concatenate a qualitative description of trend at each scale. At larger scales, only the most obvious features will be represented. The most stable episodes are defined by the number of scales (as scale is decreased) over which no new extrema appear between the extrema defining the episode. The identification of stable episodes is a powerful and intuitive method of extracting significant features.

The multiscale decomposition is not an ideal structure to handle noise, because it doesn't filter noise, but merely tends to settle it toward the finer levels in the multiscale representation. The method also generates an additional dimension of complexity to the analysis of the original data in the time domain.

2.5 Special-Purpose Languages for Numeric Shape Pattern Recognition

A different approach to approximate language-string matching is found in Agrawal, Psaila, Wimmers and Zait (1995). They present a special-purpose shape definition language for retrieving objects based on shapes contained in the time histories associated with these objects. The language performs blurry matching in which the user cares about the overall shape but not specific details. Some of the objects in the alphabet might include a slightly increasing transition, highly increasing transition, slightly decreasing transition, highly decreasing transition, transition from a zero value to a non-zero value, transition from a non-zero value to a zero value, final value nearly equal to initial value, and both initial and final values are zero. Complex shapes can be derived through recursive combination of elementary shapes and previously defined shapes. The shape definition language has a great deal of expressive power, although it does not appear to provide additional power over formal languages which have already been studied extensively. Unfortunately, the shape definition language has not yet been implemented on any known platform.

2.6 Syntactic Pattern Recognition and Sequence Analysis

The multiscale representation is a suitable description of the signal for matching with reference patterns. Syntactic pattern recognition techniques offer an elegant framework for matching signals to reference patterns. We begin by observing that a triangular description at a given scale in the wavelet interval tree is just a string. Therefore, any string processing algorithm is applicable. In the simplest case a reference pattern is represented by a unique string. Several fast approximate string matching algorithms are available. A match occurs when the string-edit distance between the signal- and reference-string is less than a threshold value (Stephen 1992).

A deeper and more powerful approach is possible by modeling reference patterns as instances of formal languages or automata. A grammar is defined by a set of nonterminal symbols, a set of terminal symbols, a set of productions over the union of the symbols sets, and a unique start symbol. The various grammars are nested and distinguished by their types of allowable productions. A formal grammar can be formulated as a set of recursive rewriting rules (or productions) used to generate patterns of strings, and consist of the following components:

- a set of terminal symbols, which are the characters of the alphabet that appear in the strings generated by the grammar.
- a set of nonterminal symbols, which are placeholders for patterns of terminal symbols that can be generated by the nonterminal symbols.
- a set of productions, which are rules for replacing (or rewriting) nonterminal symbols (on the left side of the production) in a string with other nonterminal or terminal symbols (on the right side of the production).
- a start symbol, which is a special nonterminal symbol that appears in the initial string generated by the grammar.

To generate a string of terminal symbols from a grammar, we begin with a string consisting of the start symbol. One of the productions with the start symbol on the left hand side is applied, replacing the start symbol with the right hand side of the production. The process of selecting nonterminal symbols in the string is repeated, and replacing them with the right hand side of some corresponding production, until all nonterminals have been replaced by terminal symbols.

The various classes of languages are defined by the types of productions that are allowed. A considerable amount of computational power can be achieved even with the regular grammars and the equivalent finite state automata. A regular language has two types of productions. The first type has a terminal and nonterminal on the right side of the production; the other type has only a terminal on the right side. A context-free grammar (CFG) may also have two nonterminals on the right side of the production.

Salient features of a reference pattern can be abstractly represented in a grammar or automaton. Now the problem is reformulated as approximate string-to-grammar matching. A suitable parsing algorithm (Young and Fu 1986) is employed in order to determine whether a signal-string is accepted by a particular reference grammar. Noise and uncertainty in the signal can be handled with an error-correcting parser. There exist efficient algorithms which search for a string in the grammar to minimize string-edit-distance to the signal-string for all strings in the grammar. The reference grammar associated with the smallest string-edit-distance is matched to the signal-string.

Formal languages and automata have a long history of use in pattern recognition where the target pattern is a symbol string based on a discrete finite alphabet. Recently, syntactic recognition techniques have been employed successfully in parsing the biological language of DNA. For example, Searls (1993) used a language containing the context-free language to parse genes and other high level features of DNA nucleotide sequences. The nucleotides form a small alphabet containing the bases ‘a’, ‘c’, ‘g’, and ‘t’.

2.7 Attribute Grammars and Medical Signals

Trahanias and Skordalakis (1990) and Koski, Juhola and Meriste (1995) developed pattern recognition methods for ECG signals based on an attributed grammar. Attributed grammars are convenient ways to analyze numerical strings because they combine syntactic and statistical pattern recognition. Nonregular structures can often be avoided in attribute grammars by representing complex constraints with attributes. Attributes also offer a convenient mechanism for performing numerical analysis of waveforms, which is not an easy task with pure formal grammars. The mathematical structure of attribute grammars share much with the triangular representations described in a previous section.

ECG signals are analyzed by converting time series to a stream of symbols from a very small alphabet or set of primitive temporal patterns. Trahanias and Skordalakis use three primitive patterns: the peak, straight line segments and parabola segments. Seven attributes are assigned to each peak: the left boundary coordinates, extremum coordinates, right boundary coordinates and the energy of the peak. The start and ending coordinates are assigned as attributes to the straight line segment. The first step is to extract the primitive patterns from the signal such that each waveform is transformed into a sequence of primitive symbols accompanied by their attribute values. Pattern templates were formulated, based on attribute grammars, for the description of ECG waveforms, using prior knowledge of ECG structure. An attribute grammar evaluator based on Floyd's parser was used to match templates and encoded signals. The evaluator is designer to recognize exact matches, but noisy inputs can be handled by allowing matching of attribute values that are within a specified criterion.

This syntactic recognition technique does not operate on the raw signal, but on a derived symbol string. Thus, the accuracy of the matching step is bounded by the reasonableness of the encoded representation of primitives produced during the pattern extraction step. The primitive extraction and parsing steps tend to be sensitive to erroneous inputs and noisy structures. Koski, et al. introduced a signal preprocessing step to remove high frequency noise, although it is always possible that such techniques will also remove some of the signal component. Input signals are then segmented into lines. Koski based pattern primitives on line slopes, which were nearly flat, moderately positive, negative, or very negative. The slopes of line segments in the actual data clustered into these five categories. The recognition system performed very well on actual data.

3.0 AUTOMATA AND NUMERICAL TIME SERIES PATTERNS

Of all the techniques surveyed in Section 2, it is believed that syntactic pattern recognition techniques offer the most expressive power and robustness for time series classification. They are relatively straightforward, intuitive and well-understood, and easy to implement due to the availability of algorithms for approximate string-grammar matching. However, finite alphabets used in grammars and automata are typically unordered sets of discrete items, such as the letters of the written alphabet. If the data has additional structure, it is difficult to exploit that structure using only the syntactic pattern recognition capability of formal grammars. Some sort of feature extraction or preprocessing is necessary to produce a string of nominal symbols if we want to parse the sequence with grammars/automata as they presently formulated. We have found only one instance in the string matching literature where strings of numbers are used instead of a small finite set of symbols. Wang (1990) defined distance functions and string matching algorithms for strings of numbers.

The analysis of numerical strings is where attribute grammars have additional power over formal languages in the Chomsky hierarchy and automata. The application of attribute grammars depends first on extraction of features from a raw signal. Features may have numerical attributes which characterize the underlying signal at a summary level. But the raw numerical signal itself possesses a rich structure that we would like to capture in a string generator/recognizer. Let's say we have an alphabet that is a finite subset of the reals, (e.g. evenly spaced numbers in a given range.) This alphabet has not only a complete ordering, but a set of relations and operators defined over the subset, such as addition, multiplication, etc. We would like to take advantage of the order relation and other properties to evaluate constraints for recognition of basic patterns that are either monotone increasing, monotone decreasing, upward concave, upward convex, downward concave, downward convex, flat, linear, damped oscillatory or expanding oscillatory, to name a few. Penalties should also be computed using an approximate string-grammar matching paradigm in order to allow for noise in the signal. In addition, template grammars should be capable of recognizing any concatenation or recursive construction (e.g., decreasing oscillation) of these basic templates. An example of concatenation is a simple rise followed by a fall (of the values in the series.)

3.1 Numerical Automata

We prefer to use automata instead of the equivalent formal languages because automata are more intuitive and visual. The section extends the basic definition of automata to handle numerical strings. We begin with a classical definition of finite automata and enrich the structure to exploit the additional information contained in numerical strings. A finite automaton M on the alphabet A = $\{s_1,...,s_n\}$ with states Q = $\{q_1,...,q_m\}$ is given by a transition function δ which maps each pair (q_i,s_i) into a state q_k, together with a set F which is a subset of Q. One state, q_1, is called the initial state, and the set F contains the final or accepting states. If M is in a state contained in F after processing an entire string, then we say that M accepts that string; otherwise M rejects the string. The language accepted by M, written L(M), is the set of all strings accepted by M (Davis and Weyuker 1983.) Automata can be conveniently represented as graphs where the states are shown as nodes, which are connected by directed links or edges.

We can modify the basic definition of an FSA to permit transitions at each stage to either zero, one, or more than one state. Formally this is accomplished by making the values of the transition function δ be sets of states rather than individual elements of Q. These devices are often called nondeterministic finite automata (NDFA), which is the type of automata used in the present context.

We require an automaton that does not cleanly accept or reject an input string, but approximately matches an input string to the L(M) by minimizing some cost. Let us say we have an input string A = $a_1a_2...a_k$ which we wish to test for acceptance by M. There is at least one string B = $b_1b_2...b_k$ which is the best-matching string to the input string accepted by M. Both input and output strings use the same alphabet A. The δ function maps (q_i,b_i) into a state q_k where $d(a_i,b_i)$ is the substitution penalty between the individual elements a_i and b_i. The best-matching string B is the element of L(M) which minimizes the global cost function:

$$C(A,B) = \sum_{i=1}^{n} d(a_i,b_i)$$

Some simple definitions of the distance function are the squared difference and the absolute deviation. In approximate matching to time series data, the best-matching string is the same length as the input string. Insertions and deletions are not allowed; substitutions are the only legal operation.

A simple NDFA requires a large number of states in order to recognize even fairly simple temporal relations in numerical strings. For example, assume a small alphabet A = {1,2,3,4,5,6,7,8,9}. An automaton accepting monotonic increasing strings is given in Figure 7.

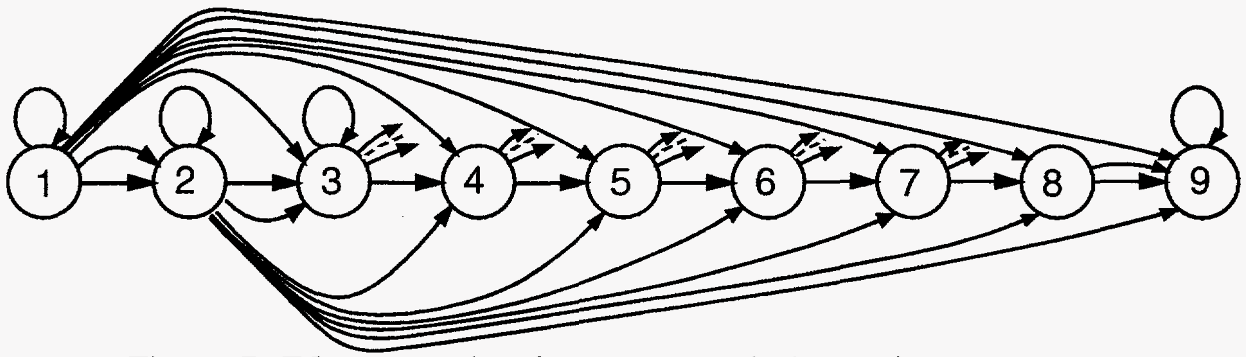


Figure 7. Edge expansion for a monotonic increasing automaton.

A separate state is required for each element of the alphabet A in order to keep track of the value of the previous element in the matching string. The automaton requires (which strictly speaking, recognizes only strings beginning with a "1") 45 edges and 9 states for recognition. In general, N states and N(N+1)/2 edges are required to specify monotonic increasing automata, where N is the resolution, or the cardinality of the set A. More complex automata such as convexity recognizers require additional edges as a function of the cardinality of A.

The simple NDFA is awkward for numerical processing because each possible possible transition must be explicitly encoded in the automaton, thus producing an explosion of the required number of states and edges. A more efficient representation is needed for practical NDFA recognition of numeric string patterns. A compact representation can be achieved by setting preconditions on firing an edge. States correspond to abstract relations instead of individual elements. Instead of associating an edge with a single element from the alphabet A, it is associated with a dynamic subset containing all legal elements derived from application of the preconditions. For example, refer to the two-state monotonic increasing NDFA in Figure 8a. In transitioning from the first state, we select an arbitrary element. The NDFA then cycles back into the second absorbing state, selecting monotonic increasing elements from A that satisfy the precondition: A' = $\{s_i: s_i > t\}$, where *t* is the output symbol arising from the firing of the previous edge. A single edge now represents all allowable transitions from the previous match to the current match. The local best matching element from A' minimizes cost as follows:

$$b_i = \min_j \left[d\left(a_i, b_j\right) \right]$$

It should be emphasized that the best individual match in a local sense may not necessarily be an element of the best-matching string, which has to minimize the global cost function.

Dynamic evaluation of preconditions creates a requirement for the NDFA to maintain a memory of the matching output as it is being generated. After firing an edge that satisfies the preconditions, side effects are executed to update the stored variables used to evaluate the preconditions. After the *i*th transition, at most the NDFA will need to maintain a memory for $b_1b_2...b_i$. For many recognizers, it is sufficient to store the values of the last few outputs, and a few values which mark special features (e.g., peaks and troughs) of the best-matching sequence. The monotonic recognizers use only the previous match, whereas a convexity recognizer requires a memory for the two previous matching outputs. In Figure 8c and 8d, previous matches are stored in Var1, and the next-to-previous match is stored in Var2. The augmented memory for special variables replaces the "state" as the repository for memory in automata.

Two states are sufficient for general monotone trend recognition. The edge leading from the start state is associated with a "don't care" precondition, and the precondition for the self-loop edge merely verifies that the current match is either greater (Figure 8a) or less than the previous value stored in Var1. Two states are similarly used for the more restrictive linear recognizer. The precondition on the self-loop verifies that the current match is exactly a step increment greater than the previous value. The constant K is hardcoded into the automaton definition. Convexity and concavity require at least three states for recognition. The first edge is a "don't care"; the second edge checks only for monotonicity; the third edge evaluates the difference between twice the previous value and the next-to previous value. If the difference is greater than the current match, the string is locally convex; otherwise it is locally concave.

Figure 8e demonstrates a slightly more complex recursive temporal pattern requiring additional memory: the downward oscillation. Strings accepted by this automaton rise and fall alternately, but the global trend is downward. The second and fourth states are respectively associated with monotonic increasing and decreasing subsequences. The third and fifth states are inserted for the sole purpose of marking the peaks and troughs of each rise and fall. Side effects associated with those states store the current value into Var2 (peak) or Var3 (trough) so that subsequent elements can be checked to verify that the global trend is downward.

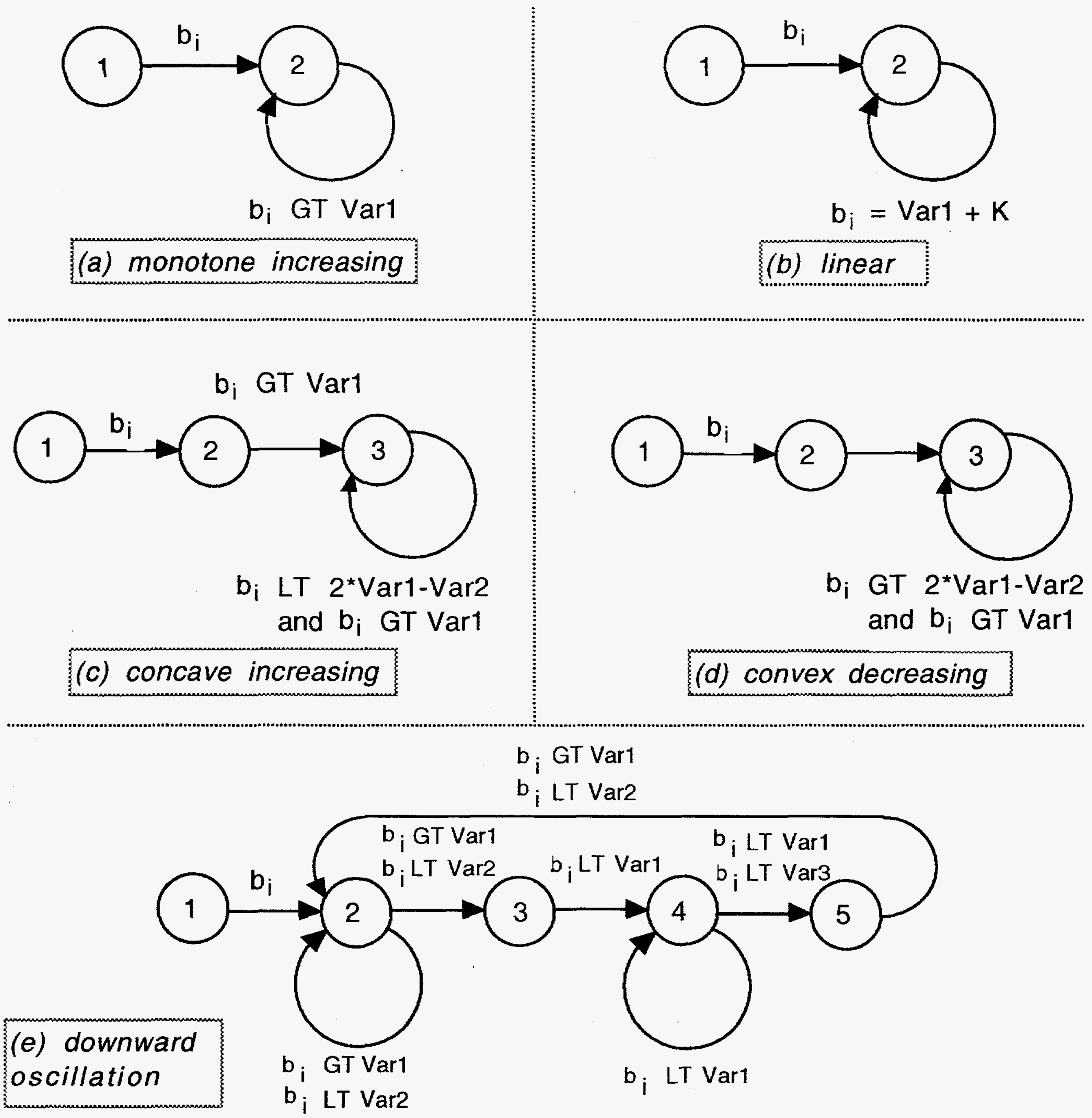


Figure 8. Compact automata with preconditions and side-effects.

3.2 An Approximate String/NDFA Matching Algorithm

Efficient algorithms based on dynamic programming methods have been developed for approximate string-to-CFG matching (Myers 1995) and string-to-regular grammar matching (Myers and Miller 1989; Knight and Myers 1995). Both algorithms use dynamic programming recursions to fill in a cost matrix. Approximate matching scores are computed initially for single elements of the input string, and increasingly scales up to larger substrings of the input string.

The CFG-based algorithm was modified to permit preconditions and side effects to be applied to numerical strings, but it could not easily be adapted to these purposes. The regular expression matching algorithm was considerably simpler, but problems were eventually encountered here too in keeping track of preconditions and side-effects. Eventually a conceptually simpler approach was adopted for finite state automata (or equivalently, regular languages), but one which was less computationally efficient than the Myers and Miller algorithm.

The optimal match between the input string and a given finite automata is computed using a generalization of the Myers and Miller algorithm to take into account the preconditions and side effects. The algorithm consists of two steps. In the first step, the optimal cost table is computed using dynamic programming. The actual matching string is retrieved in a second step.

Let a_1 , a_2 , ... , a_n denote the input string. Let s be the state of the automata which is assumed to have a single start state and a single final state. Let C_{i,s} denote the optimal cost of matching the string a_1 , a_2 , ... , a_i with a string that takes the automaton from the start state to the state s. Note that each state is associated with a fixed set of variables which take values as a result of side effects. A transition or edge of the automaton is fired only if preconditions on the variables of the both states of the edge are satisfied. The recursive equation for the computation of C_{i,s} is given as follows:

```
C_{i,s}= min { min_{t->s, precondition satisfied} { C_{i-1,t} + d(a_i, b_s)},
               C_{i-1,s} + d (a_i ,e),
               min_{t->s, precondition satisfied} { C_{i,t}+d(e,b_s)}
             }
```

where d(a,b) is the cost of mismatch between the input element a and the automata input symbol b, e is empty string, and b_s is the input symbol that is input to automaton in state t.

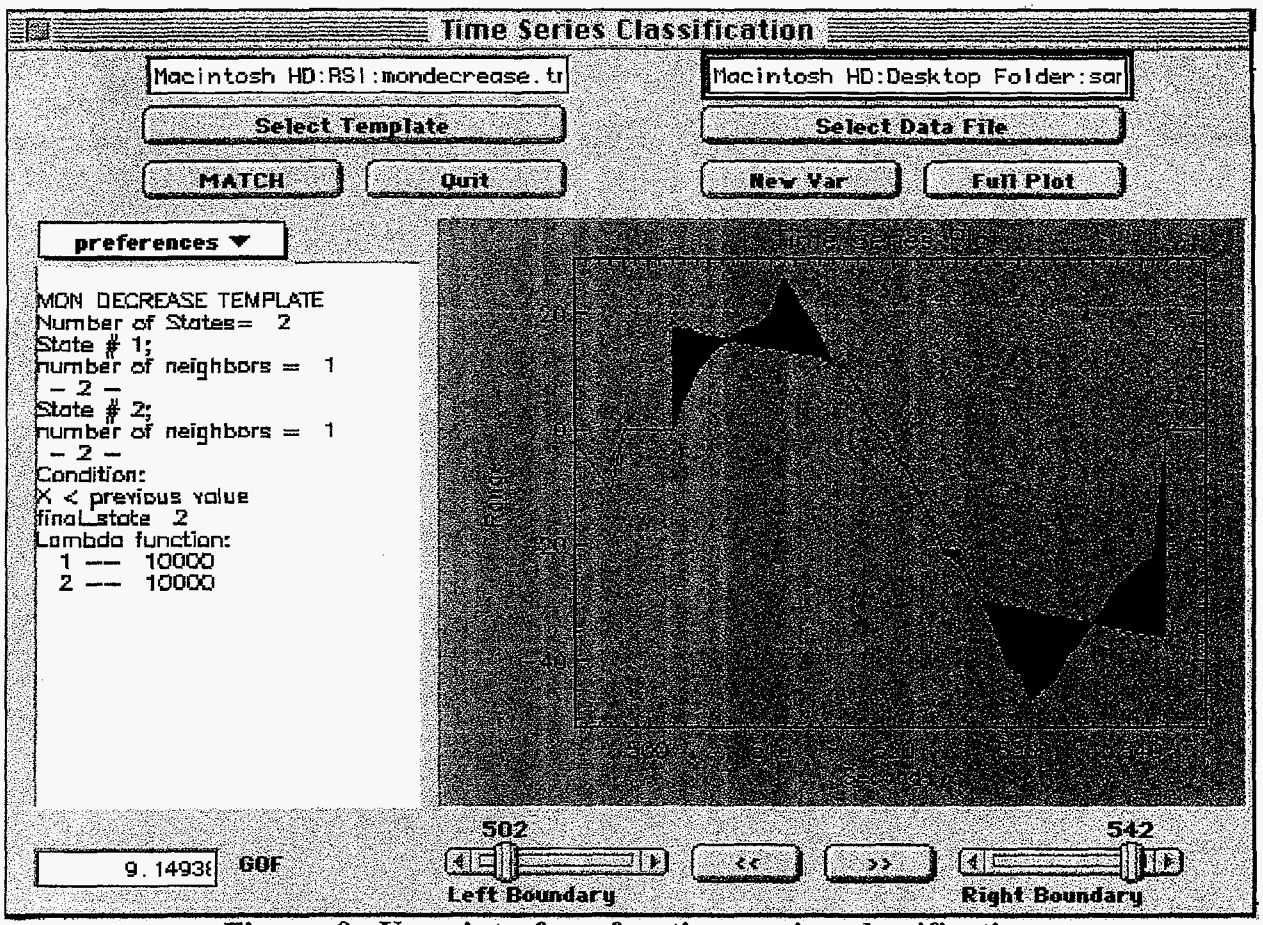
This recursion is evaluated with the increasing values of i. For each values of i, the state s is varied in the topological order of the automaton using the method of Myers and Miller (1989), where the algorithmic details can be found. In their algorithm each transition t -> s is allowed, whereas in our case only those that satisfy the preconditions are allowed. This step is implemented by using a three-dimensional matrix C_{i,s,v} where v denotes the value associated with the state s. Now the transitions of the type t -> s are allowed only if the corresponding preconditions are satisfied. The actual matching string is retrieved by suitably backtracking from the final state with min_{v} C_{i,final_state,v} through the three-dimensional matrix C_{i,s,v}.

4.0 IMPLEMENTATION IN IDL

The purpose of this research is to adapt temporal pattern recognition techniques to time series classification problems in the highway vehicle domain. An interactive tool was desired to provide the analyst some of the capabilities of a CADETS system in order to work with large streams of time series data. One important feature of this prototype system was the ability to browse the time series in a plot window. Another planned feature was a visualization of the fit between the input string and the best-matching string returned by the approximate string-automata matching algorithm for the template selected by the analyst.

IDL was selected as a prototyping environment to establish proof-of-principle of the approximate string-automata matching algorithm, and demonstrate the application of the method to basic time series patterns and actual braking data. IDL is a computing environment and fourth-generation language which integrates array-oriented processing with numerous mathematical analysis and graphical display functions. Using display widgets, a graphical user interface can be prototyped with greater efficiency than is possible in many other languages. The implementation was performed on a Power Macintosh 7100/66AV, although the IDL code is relatively platform independent, and should run in IDL environments installed on Windows and Unix machines.

The prototype environment provides rudimentary aspects of a full-featured CADETS system. The graphical user interface integrates capabilities for time series display and browsing, subsequence selection, approximate string-template matching, and display of goodness of fit of the best matching string and the data string. Figure 9 shows the user interface for control of the entire application.



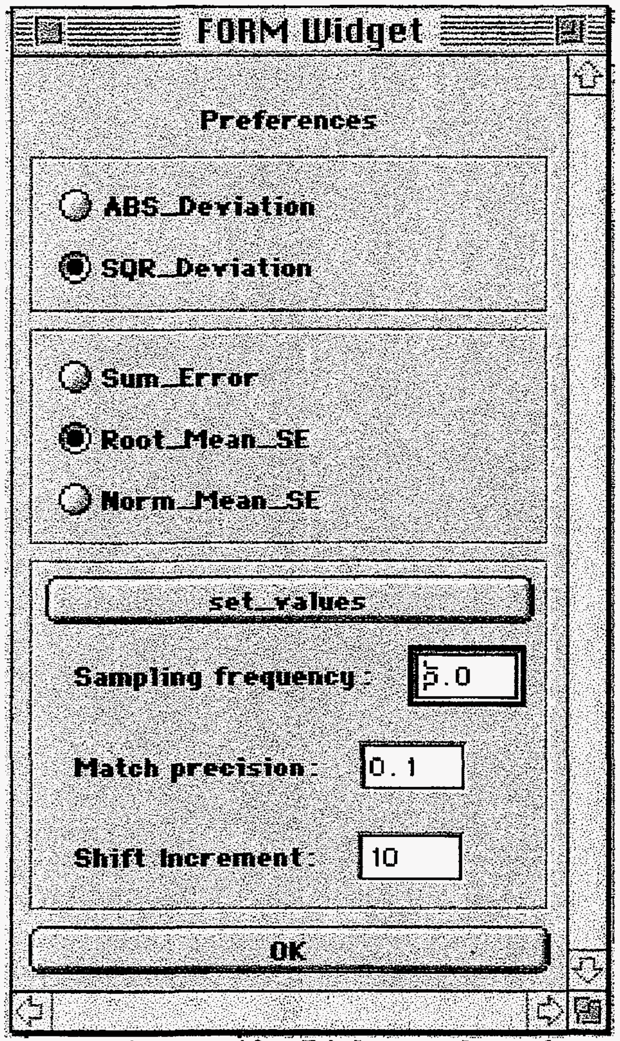
```
MON DECREASE TEMPLATE
Number of States=  2
State # 1;
number of neighbors =  1
 - 2 -
State # 2;
number of neighbors =  1
 - 2 -
Condition:
X < previous value
final_state  2
Lambda function:
  1 --  10000
  2 --  10000
```

Figure 9. User interface for time series classification.

The time series plot window is used for browsing multidimensional time series data. The user first selects a data file from a file browser. Currently only a single data format is supported in the prototype. The software prompts the user for the variable(s) to be plotted in the time series window. A line graph is then drawn in the time series window which displays the entire range of data for the selected variable. The Y-range is selected according to an algorithm that attempts to fit the data intelligently in the window using round numbers as the minimum and maximum. The user can drag the mouse across the graphical window and rubberband a section of displayed data. The result is a zoom on the selected area of the plot. Buttons underneath the time series window can be clicked with the mouse in order to shift the plot to the right or left. Selecting the 'New Var' button brings up a checkbox dialogue to permit the user to change the variable displayed in the plot window. The 'Full Plot' button returns a zoomed/shifted plot window to the initial full-data display.

The user not only selects data but a template file for classification of the data. The data in template files completely specify the required variables for an automaton, and use *.tmp as the extension. A natural language description of the template appears in the template window after selecting a template in the file browser.

In order to match a data segment to a template, the user first needs to specify the endpoints of the segment with the 'left boundary' and 'right boundary' sliders. Dotted vertical red lines demarcate the time series segment defined by the user with the sliders. The segment represents a subseries that the user suspects may encode some meaningful event or behavior, such as curve-following, passing or stopping. Clicking on the 'Match' button activates the matching algorithm using the selected template and time series segment as inputs. The best-matching string is plotted over the segment between the dotted vertical lines. The area showing lack of agreement between input and matching strings is filled in red. A numerical value representing the goodness of fit is displayed in the text box labelled GOF. However, the visual representation of the string-grammar match provides the analyst with additional information about the qualitative fit. The intuition about qualitative fit provided to the analyst is an important supplement to the numerical value; in some instances it may indicate more about the true classification of the time series segment than the quantitative fit. The window labelled 'preferences' is a droplist containing the alternatives: settings, plots and 'change colors'. Selecting 'settings' leads to a dialogue box containing three framed areas pertaining to the distance function, the goodness-of-fit function, and user-specified parameters. Absolute deviation and squared deviation are the two options available for the distance function. The error or distance is summed over the length of the input string and taken as the goodness-of-fit measure. Two other options are to take the square root of the mean summed distance, or normalize the mean summed distance. These choices are indicated the middle frame. The bottom frame contains text boxes where the user can input parameter values for sampling frequency, match precision, and shift increment. Sampling frequency informs the plotting routine on how to properly label the X-axis by specifying the number of data points contained in one second. The value provided in the text box to the right of 'Shift Increment' is the number of data points the plot shifts along the X-axis each time the shift buttons are clicked. New values in the lower frame are not computed unless the 'set-values' button is clicked.



ABS_Deviation [✓ SQR_Deviation]

Sum_Error [✓ Root_Mean_SE] Norm_Mean_SE

Sampling frequency: 5.0

Match precision: 0.1

Shift Increment: 10

Figure 10. Dialogue box for user settings.

The value of 'Match precision' and the Y-range of the input string jointly determine the number of elements in the set A, or the set of possible values that can be assumed by elements of the best-matching string. A large precision leads to coarse matching, but quicker execution time. A small precision approaches optimum performance of the matching algorithm in the limit, but at the cost of up to quadratically increasing execution time. The Y-range searched by the matching algorithm is just the empirical range of the input string which is among the smallest sets containing the best-matching string. It should be noted, however, that there may exist instances where the range of the best-matching string extends beyond the input string range.

The set A is constructed from a discretized input range. An example of the generation of the set A is provided in Figure 11. The Y-range of the input string is indicated by the spread of input string values along a horizontal axis.

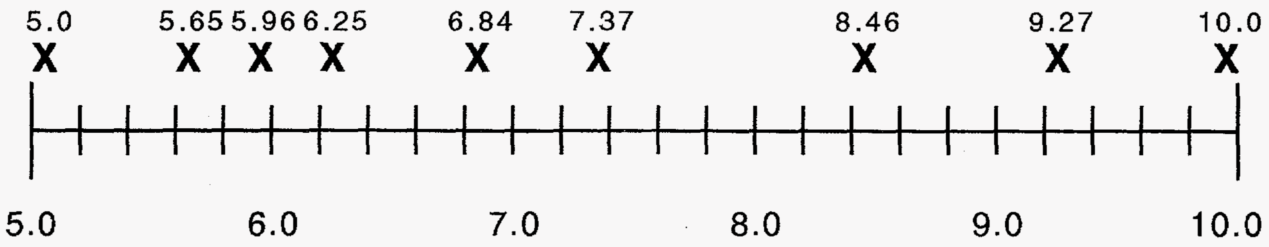


Figure 11. A discrete numeric alphabet.

The number of elements in A is the empirical range (10.0 - 5.0 = 5.0) divided by the precision (0.2) plus one, or 26. Most of the values in the input string (denoted by X's) do not correspond exactly with a member of the set A, but all lie very close to some member of A. Input values cannot therefore be exactly substituted in the best-matching string, but a close substitution is possible. The matching algorithm does not work with the discretized range, but a linear transformation into the integers (in this case, from 0 to 25).

The 'plots' option in the preferences droplist allows the user to select multiple time series plot windows for viewing multidimensional data. The current version of the prototype implementation does not provide for multiple plots. The 'change colors' option will allow the user to choose different colors for plot background, lineplots and template background.

5.0 VALIDATION OF THE STRING-AUTOMATA MATCHING ALGORITHM

This section describes efforts to establish the validity of the new approach to temporal pattern recognition by computing the quantitative goodness-of-fit for several time series segments to a number of predefined templates.

5.1 Validation data

Two types of data were used in the validation study: artificial data and field test data. The artificial data was computed on a spreadsheet using a random number function to generate sequences under appropriate constraints. For example, monotonic increasing sequences were computed by adding a random positive number to the previous value in the sequence. Field test data were acquired with instrumented vehicles in the FOCAS study conducted by James Sayer and Paul Fancher of the University of Michigan Transportation Research Institute. The objective of that study was to evaluate driver braking strategies in order to understand how driver's rates of deceleration relate to headway in dynamic traffic settings. The research emphasis was on characterizing rates of deceleration as a function of velocity and traffic conditions.

Raw data for each trip in the FOCAS study consisted of 18 data fields containing: range, range rate, velocity, acceleration (measured), acceleration (calculated), yaw rate, steering angle, accelerator pedal position, brake pedal pressed, set speed, headway time, valid target, command speed, throttle position, pressure grade, VCR address and tick count. Only calculated forward acceleration was modeled with templates in the present validation study.

Events that were analyzed in the FOCAS study were stops at stop signs, and stops at stoplights. Events at lights were classified according to whether the driver was the first car at the light. Data files were scanned around those times to find the beginning of a braking event, which was defined as when the driver lifted their foot from the accelerator pedal (just before depressing the brake pedal.) The stop event descriptions and timing were summarized in a text file. This information permitted the identification of 'a priori' stopping event classifications which could be used to validate the approximate matching algorithm.

5.2 Template Definition

Automata have considerable power for expressing abstract features of time series while remaining invariant with respect to significant features. Automata and formal languages are much more abstract, for example, than simple numeric string templates such as those discussed by Berndt and Clifford (1996). In string matching, only a single exemplar is used, whereas the automata represent an entire family of strings, i.e., L(M), which all can be considered exemplars of a more abstract prototype.

All required elements for specifying a NDFA with preconditions and side effects are found in the input format for template files. For example, the text block to the left below describes part of the template definition for a oscillation automaton, and the right block shows an automaton which recognizes downward concave patterns.

```
MON REP INC,DEC TEMPLATE
2
2  1 1
     1 0 0 -1 2
   2 1
     1 0 0 -1 1
2  1 1
     1 0 0 -1 2
   2 1
     1 0 0 -1 1
2
```

```
LOCAL DOWNWARD CONCAVE TEMPLATE
3
1  2 1
     1  0 0 -1 0
1  3 1
     1  0 0 -1 1
1  3 2
     1  0 0 -1 1
     2 -1 0 -1 1
3
```

The first line of the input file is a title for the automaton. The second line specifies the number of states. The next line contains three fields which pertain to the first state: (1) The number of successor states; (2) the numeric identifier of the first successor state; and (3) the number of preconditions associated with the first edge leaving the state. Information regarding the first edge is contained in the first indented line. The preconditions are expressed as linear inequalities which contain the value of the matching element on the left side, and a linear combination of three variables on the right side. The five fields for the precondition specify: (1) a multiplier for the first variable in the inequality used to verify the first precondition; (2) a multiplier for the second variable; (3) a multiplier for the third variable; (4) a constant; and (5) a numerical code which determines the type of inequality to insert in the precondition (e.g., greater than, less than, equal to). The next line provides information about the second edge originating in the first state. This edge leads to the second state and has one precondition. Information specifying this inequality is given on the following line. This format is repeated over the next four lines for the second state. The final line identifies the final state.

The text block on the right shows an identical format for three states required by the downward concave template. Notice that the seventh and eighth lines provide specifications for two preconditions associated with the edge joining the third state with itself. The value of negative unity found in the 'constant' field is a special code telling the program to ignore the constant in the evaluation of the inequality precondition.

Stopping events were examined in the FOCAS data files, and a general characterization of stopping event in units of forward acceleration was obtained. The event is initiated as the driver lifts their foot from the accelerator pedal, and acceleration flattens at zero. Acceleration gradually becomes more negative as increasing force is applied to the braking pedal. Maximum braking force is achieved, and the acceleration quickly returns to zero. Just prior to coming to a complete stop, a sudden application of braking force produces a negative spike in acceleration. This sequence is plotted in Figure 12, and an automata that recognizes each element of the sequence is given in Figure 13.

There are many different strategies that drivers use to achieve the necessary braking force to bring the vehicle to a stop. One way to characterize braking strategies is whether the application is smooth, step-like, or pumping. More specific templates can easily be constructed to abstractly characterize various braking strategies. The templates would include more detailed modeling of the gradual monotonic decrease.

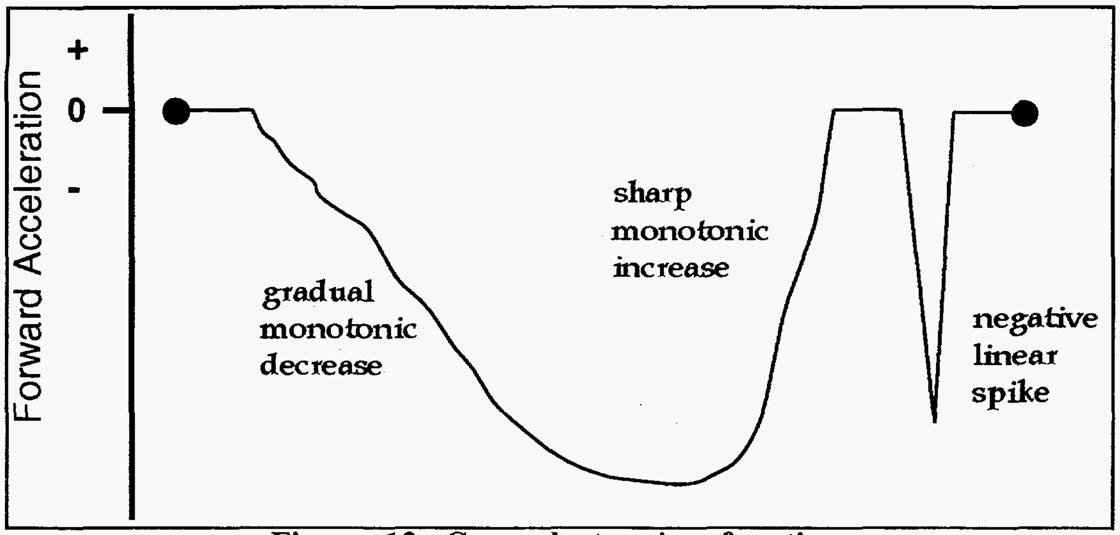


Figure 12. General stopping function.

The flat-zero subsequence is modeled in the self-loops to state 1. The edges into state 2 account for the monotonic decrease, and state 3 allows the monotonic increase up to zero. The second flat section is modeled in edges into state 4. States 5 and 6 are needed to model the downward linear spike; the edge into state 5 selects the initial decline, and edges into state 6 ensures that additional steps are linear by comparing the two previous values. Edges associated with states 7 and 8 model the upward linear spike similarly to states 5 and 6. The edges into the final state model the third flat zero section.

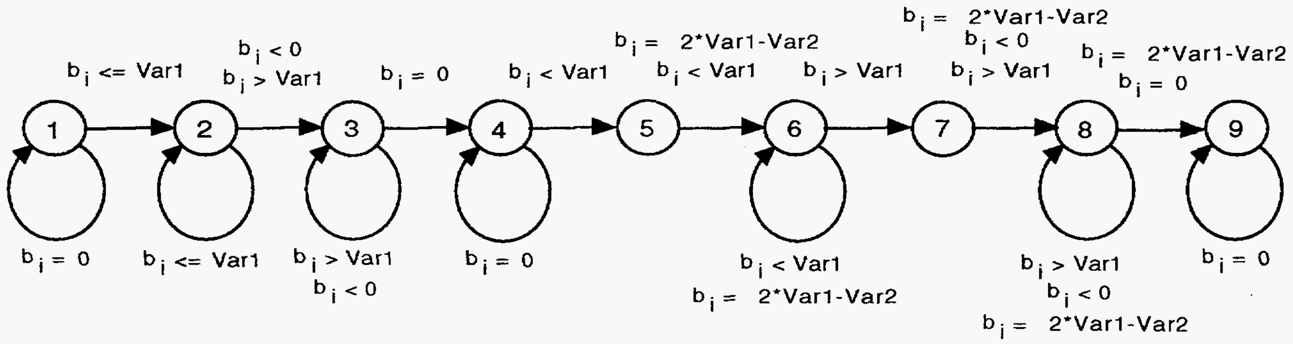


Figure 13. Automaton for general stopping.

5.3 Validation Study

5.3.1 Artificial Data. Templates files were constructed to recognize monotonic increasing, monotonic decreasing, upward convex, downward concave, single rise/fall, double-rise/single-fall and downward oscillation patterns. Three artificial data patterns were generated from each category. A 7 x 7 confusion matrix was generated by filling in the computed distances for all possible template/pattern pairs from the artificial data. In addition, all templates were fit to random walk sequences to assess their flexibility.

The following predictions were made to validate the implemented algorithm for temporal pattern recognition:

- Best fits should be found along the diagonal; i.e., where the template is paired with corresponding artificial data.
- Goodness-of-fit for more general automata should be at least as good as the goodness-of-fit for all more restrictive automata which are contained by general case. For example, a monotonic increasing template is more general than an upward convex template, and should outperform the upward convex template for all data segments .

The raw cost data from matching artificial signals are presented in Table 1. Each cell contains the distance obtained for all three artificial signals. Different resolutions were selected for each artificial signal based on the extent of the signal in both the X-range and Y-range. The algorithm required a substantial amount of time to complete when the amplitudes of the signals were greatly extended, especially if the template had more than a few states. However, in order to avoid bias, the same resolution was used for matching each template with a given signal.

Optimal matching strings were found for nearly every template/signal combination. The input data range was insufficient for fitting upward convex and downward concave patterns to the input string. Rather than increase the range of elements available to fit, piecewise linear automata were used instead of convex or concave automata for several input strings (which are indicated in Table 1 with an asterisk.) These automata were locally linear, in that linear segments were permitted. However, they were globally convex or concave because the linear segments themselves were convex or concave with respect to each other. The piecewise linear automata were able to fit the data within the limits of the input range.

The matching algorithm was unable to generate a matching string using the downward oscillation template (last column) or substitution cost for several signals. The algorithm failed for all the double/rise fall signals, and one random walk signal. There is no apparent explanation for these failures, although it is likely due to a coding error in the implementation of the algorithm.

The goodness of fit or distance scores are not a useful metric for comparing the performance of a single template across signals. Instead of averaging the distances within each cell, the distances were first normalized by dividing each score by the standard deviation of the scores generated by all templates for that particular signal. The normalized score can be interpreted as a kind of z-score in the positive reals including zero. The three scores in each cell were then averaged to produce an overall normalized distance to compare across input signals and templates. The average normalized distances are presented in Table 2.

Inspection of the scores along the diagonal of Table 2 confirm the first hypothesis that templates matched to signals generate very low distances (or excellent goodness-of-fit.) The last two diagonal cells were a little higher than the others, but this was probably due to the low resolution of the alphabet of matching numbers. On the other hand, templates fitted very poorly to signals associated with opposing trends. For example, the fit of the increasing template to the decreasing signal, and the fit of the decreasing template to the increasing signal were nearly two standard deviations above zero.

Table 1. Confusion matrix for raw distances

	Template						
	Increase	Decrease	Upward convex	Downward concave	Rise/fall	Double rise/fall	Rise/fall downward
Increase (prec=0.1)	0.0316 0.0000 0.0000	2.7727 5.6418 8.9403	0.6964 2.6378 0.7169	3.9431 6.6993 10.1302	0.4483 0.6042 1.2300	0.0316 0.0447 0.2236	0.6099 0.8031 2.1121
Decrease (prec=0.1)	3.4186 5.4714 9.3794	0.0000 0.0000 0.0000	4.5979 6.9232 10.5639	0.3286 0.7711 1.0382	0.8106 0.4056 0.8724	0.4472 0.4919 1.2494	0.2025 0.4056 0.6708
Upward convex (prec=0.3)	0.0000 0.0000 0.0000	11.2594 21.1128 22.1456	0.0000 0.0000 0.0949	14.8600 24.7050 25.7588	1.5443 2.4150 3.1536	0.4025 0.8050 0.2121	2.8222 4.7633 6.1723
Downward concave (prec=0.3)	10.5486 19.0895 23.0895	0.0000 0.0000 0.000	15.007 23.516 26.6855	0.0905 0.0905 0.0949	0.2023 0.2558 1.5704	2.3760 5.0597 3.0000	0.2023 0.2558 0.5367
Rise/fall (prec=0.2)	4.2636 6.5301 9.327	1.8193 6.3575 5.6064	4.2636* 8.6034* 10.5095*	1.9794* 6.4075* 5.6303*	0.0447 0.0000 0.0447	0.1095 0.0617 0.5404	0.0447 0.0000 0.0447
Double rise/fall (prec=0.4)	10.2162 17.4971 26.5625	3.9371 4.9344 9.1494	10.2162* 17.4970* 26.5625*	4.4036* 8.0337* 14.5217*	2.6423 4.2389 7.9226	0.1249 0.0632 0.0883	
Rise/fall downward (prec=0.5)	15.5747 16.9203 18.0303	4.8657 4.8755 5.2353	15.5747* 16.9203* 18.0303*	4.8657* 5.0872* 6.3344*	4.6516 4.7697 5.1182	4.6552 4.7732 5.1202	0.1443 0.1118 0.0913

Random (prec=0.1)	0.9524 2.8771 3.2962	0.8142 1.6222 1.7677	0.9609* 2.8771* 3.2962*	0.8145* 2.2597* 2.3632*	0.3505 1.6297 1.4808	0.3519 0.4593 1.3452	0.1069 1.0395

The validity of the second hypothesis was also confirmed in many cells of Table 2. The more general monotone increase and monotone decrease templates respectively fit the upward convex and downward concave signals perfectly, as did their associated templates. The very general downward oscillation template fit the rise/fall signals nearly perfectly. As expected, the converse findings did not hold very well in these cases. The fits of the upward convex automaton to the increasing signals, the downward convex automaton to the decreasing signals, and the rise/fall automaton to the downward oscillation signals were only intermediate at best.

Table 2. Confusion matrix for averaged normalized distances

	Template						
	Increase	Decrease	Upward convex	Downward concave	Rise/fall	Double rise/fall	Rise/fall downward
Increase (prec=0.1)	0.0069	1.9781	0.5252	2.4537	0.2661	0.0296	0.3937
Decrease (prec=0.1)	1.9514	0.0000	2.4227	0.2262	0.2592	0.2309	0.1336
Upward convex (prec=0.3)	0.0000	1.9520	0.0029	2.3719	0.2561	0.0538	0.4914
Downward concave (prec=0.3)	1.8591	0.0000	2.3494	0.0106	0.0641	0.3813	0.0347
Rise/fall (prec=0.2)	2.0329	1.3036	2.3041	1.3382	0.0113	0.0656	0.0113
Double rise/fall (prec=0.4)	2.4590	0.8307	2.4590	1.1784	0.6557	0.0158	0.0000
Rise/fall downward (prec=0.5)	2.5912	0.7696	2.5912	0.8330	0.7467	0.7472	0.0181
Random (prec=0.1)	3.1532	2.0137	3.1614	2.4595	1.4635	0.9926	0.4788

A finding from Table 2 that was not predicted was the robustness of automata in fitting data they were not intended to recognize. These simple automata accepted not only those strings they were built to recognize, but a much larger superset of strings. For example, the double rise/fall and rise/fall templates and signals fit each other quite well. One reason for the successful fitting is that searched features not found in the signal could be applied in minimal fashion, thereby paying only a small cost. The double rise/fall template required the algorithm to append a small downward tail to the end of the rise/fall signal, but it generated only a small deviation from the input string. Further restrictions on automata definitions are probably needed to discourage "feature trivialization" in order to minimize cost. One possible method to force automata to emphasize required features is to require at least N iterations of a given trend such as a rise and fall. Another strategy is to use preconditions to specify least extreme values for peaks and troughs.

5.3.2 FOCAS Data. Four instances of stopping were selected from the FOCAS field test data as exemplars of continuous downward, smooth, step and pump stopping. Automata were constructed to recognize each of the four stopping strategies. The automata or templates resembled the general stopping automaton depicted in Figure 13, but differed only in the way the decrease from zero was modeled. The continuous downward automaton was identical to Figure 13, and it modeled a monotonic downward subsequence. Smooth stopping was modeled as globally convex in the decreasing subsequence, but was permitted to be locally piecewise linear. Step stopping was a downward staircase function, and pump stopping conformed with a cascading series of a strictly decreasing subsequence followed by a strictly increasing subsequence. Table 3 shows the root mean squared error (distance) resulting from matching each time series segment with every stopping template.

Table 3. Confusion Matrix for FOCAS Data

	Template			
	Continuous Downward	Smooth	Step	Pump
Continuous Downward (prec=.005)	.01495	.02437	.01682	.00411
Smooth (prec=.005)	.05237	.05544	.05380	.00403
Step (prec=.005)	.05172	.05982	.05937	.05797
Pump (prec=.005)	.03330	.03998	.03335	.00476

The most significant finding was that every template was able to fit each of the four stopping segments quite well. All sixteen quantitative fits in Table 3 were under 20% of the input data range. To some extent this result was expected, since any stopping template should be able to characterize a stopping event with reasonable accuracy. Accuracy would have been slightly improved if a fine grain resolution were used on the matching alphabet. The success of the templates in fitting all stopping sequences masked any trend toward lower values along the diagonals in Table 3 as compared to off-diagonals.

In nearly every case (except the step stopping segment) the pump stopping template fit the time series data extremely well, in fact much better than the other templates. By allowing the automaton to follow arbitrary rises and falls (i.e., noise) in the raw data, regardless of magnitude or number, the automaton was able to track nearly any subsequence. Further refinement of the pumping template to exclude excessively large numbers of pumping episodes or small episodes would probably aid in preventing good fits to all stopping sequences while still allowing the template to recognize pump-like stopping.

The stopping templates were unable to fit the smooth and step signals as well as the other two stopping sequences. Inspection of the quality of fits, indicated by the mismatch with the best-matching strings generated by the automata, revealed that most of the area of misfit in the smooth sequence was in the spike near the end of the sequence. The spikes were apparently too narrow for the automata to accurately model. The initial segment of the step sequence was substantially above zero, which the automata were not able to track since matching ranges were restricted to nonpositive values.

6.0 FUTURE DIRECTIONS FOR RESEARCH AND DEVELOPMENT

The results from the validation study are an adequate demonstration of proof-of-principle of the approximate string-automata matching algorithm for classification of transportation time series, but much additional work can be done in order to evolve the prototype into a more useful and practical tool for the data analyst.

Two main currents of future research are delineated: improvement of the basic approximate string-automata matching algorithm, and expansion of user interface capability. The most immediate algorithmic improvement that is needed if the system is to have much practical value is to increase efficiency and speed. It may be necessary to demonstrate the prototype application on a platform with a more powerful CPU.

The extension of the algorithm to multivariate time series should be relatively straightforward, at least in a conceptual sense if not in actual lines of additional coding. This modification should be accompanied by a similar change in the user interface to allow the simultaneous display of several variables in multiple plots. The generation of multiple plots in rectangular grid format is especially simple in the IDL language.

Another algorithmic generalization would be to provide parameter ranges for the linear inequality preconditions. Parameter ranges would allow template designers to partition the parameter space into meaningful categories. For example, suppose we wanted separate linear templates to represent small, medium and large slopes. If the interval (U,V) contained the medium-range slopes, then slopes less than U would be small, and large slopes greater than V.

An obvious generalization is to extend the algorithm to include pushdown automata and context-free grammars. The pushdown memory would allow templates to capture symmetries or specific asymmetries in the time series data. It may be possible to modify the Myers algorithm for approximate string-CFG matching with preconditions and side effects.

A number of relatively simple upgrades to the interactive user interface are possible. Support for additional data file formats and user-customization of color in the graphics windows are among the quick modifications. An important feature that would improve usability by the nontechnical analyst would be the availability of a template editor and browser. A point-and-click graphical editor for generating automata would allow a typical user to construct special purpose templates instead of relying solely on a predefined template library. A more understandable description of the template in the template window is also desirable.

Automation of tasks can be done for a variety of important repetitious subtasks. An automated routine called 'selecting a winner' would sequentially match a whole family of templates to a selected input string. The template with the best fit to the input string would be designated as the winner or best classification of the string. Another type of task automation that might be useful is string-automata similarity matching. A template definition is supplied to the routine, and it returns all the predefined segments that match the template within a threshold value.

Segmentation of time series is an important task that precedes classification and visualization. In large-scale time series temporal patterns of interest may be masked by large segments of uninteresting events. Several fast segmentation algorithms (e.g., Faloutsos, et al. 1994, Keogh and Smyth 1997) exist in the data mining literature which could be integrated into the prototype application as part of a two-phase process. Fast segmentation would be followed by a more in-depth phase of classification of segments identified by the first phase.

7.0 CONCLUSIONS

Two related research efforts jointly established a proof-of-principle for a new algorithm which performs time series classification, a key aspect of a CADETS architecture, and demonstrated its application to highway vehicle time series by validating the performance of the algorithm.

The primary technical contribution of this work was the development of a new algorithm for approximate string-NDFA matching with preconditions and side effects that operates on input strings representing raw numerical data. The method is the first known extension of automata to accept and exploit the properties of numerical strings. This algorithm was embedded within an interactive user interface in order to provide the user additional control and insight into the classification algorithm.

The time series classifier is capable of recognizing a wide range of temporal patterns through the careful construction of appropriate NDFA. Automata were constructed for recognition of flat, piecewise linear, monotonic decreasing, monotonic increasing, oscillatory, concave, convex and stopping sequences. Recognition was demonstrated in artificial and field time series data. It was also shown how more complex temporal patterns could be recognized using concatenation and recursion of simpler automata.

This demonstration is not a complete implementation of a CADETS architecture, or even a complete solution to time series classification for transportation time series. Implementation of an integrated package to serve the data analysis requirements of specific user communities necessitates that further research be undertaken.

REFERENCES

Agrawal, R., Faloutsos, C., and Swami, A. (1993). Efficient similarity search in sequence databases. *Proceedings of the FODO Conference*, Evanston, IL.

Agrawal, R. Psaila, G, Wimmers, E. L., and Zait, M. (1995). Querying shapes of histories. *Proceedings of the 21st VLDB Conference*, Zurich, Switzerland.

Bakshi, B. R. and Stephanopoulos, G. (1994a). Representation of process trends - IV. Induction of real-time patterns from operating data for diagnosis and supervisory control. *Computers in Chemical Engineering*, 18(4), 303-332.

Bakshi, B. R. and Stephanopoulos, G. (1994b). Representation of process trends - III. Multiscale extraction of trends from process data. *Computers in Chemical Engineering*, 18(4), 267-302.

Bendat, J. S. & Piersol, A. G. (1986) *Random Data.* New York: John Wiley & Sons.

Berndt, D. J. & Clifford, J. (1996) Finding patterns in time series. pp 229-248 in U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy (Eds.) *Advances in Knowledge Discovery and Data Mining*. Menlo Park, CA: AAAI Press/The MIT Press.

Bottou, L., Soulie, F. F., Blanchet, P., and Lienard, J. S. (1990). Speaker-independent isolated digit recognition: Multilayer perceptrons vs. dynamic time warping. *Neural Networks*, 3, 453-365.

Box, G.E.P., Jenkins, G.M. & Reinsel, G.C. (1994). *Time Series Analysis: Forecasting and Control (3rd Edition).* Englewood Cliffs, NJ: Prentice-Hall, Inc.

Cheung, J. T.-Y. and Stephanopoulos, G. (1990a). Representation of process trends - Part I. A formal representation framework. *Computers in Chemical Engineering*, 14(4/5), 495-510.

Cheung, J. T.-Y. and Stephanopoulos, G. (1990b). Representation of process trends - Part II. The problem of scale and qualitative scaling. *Computers in Chemical Eng.*, 14(4/5), 511-539.

Davis, M. D. And Weyuker, E. J. (1983). *Computability, Complexity, and Languages.* New York: Academic Press.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14(2), 179-212.

Faloutsos, C., Ranganathan, M., and Manolopoulos, Y. (1994). Fast subsequence matching in time-series databases. *Proceedings of the 1994 ACM SIGMOD.*

Keogh, E. And P. Smyth. (1997). A probabilistic approach to fast pattern matching in time series databases. Pp. 24-30 in D. Heckerman, H. Mannila, D. Pregibon, and R. Uthurusamy (Eds.) *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, Menlo Park, CA: AAAI.

Knight, J. R., and Myers, E. W. (1995). Approximate regular expression pattern matching with concave gap penalties. 1-30

Koski, A., Juhola, M., and Meriste, M. (1995). Syntactic pattern recognition of ECG signals by attributed finite automata. *Pattern Recognition*, 28(12), 1927-1940.

Kremer, S. C. (1995). On the computational power of Elman-style recurrent networks. *IEEE Transactions on Neural Networks*, 6(4), 1000-1004.

Lang, K. J., Waibel, A. H., and Hinton, G. E. (1990). A time-delay neural network architecture for isolated word recognition. *Neural Networks*, 3, 23-43.

Makhoul, J. & Schwartz, R. (1994). State of the art in continuous speech recognition. pp. 165-199 in D. B. Roe & J. G. Wilpon (Eds.) *Voice Communication Between Humans and Machines.* Washington, D. C.: National Academy Press.

Myers, G. (1995). Approximately matching context-free languages. *Information Processing Letters*, 54, 85-92.

Myers, E. W., and Miller, W. (1989). Approximate matching of regular expressions. *Bulletin of Mathematical Biology*, 51(1), 5-37.

Rudnicky, A.I., Hauptmann, A.G., & Lee, K. (1994) Survey of current speech technology. *Communications of the ACM*, 37(3), 52-57.

Searls, D. B. (1993). String variable grammar: A logic grammar formalism for the biological language of DNA. *Journal of Logic Programming*, 12, 1-30.

Silverman, H. F. and Morgan, D. P. (1990). The application of dynamic programming to connected speech recognition. *IEEE ASSP Magazine*, 7(3), 6-25.

Stephen, G. A. (1992). *String Search.* Technical Report TR-92-gas-01, Gwynedd, UK: School of Electronic Engineering Science, University College of North Wales.

Trahanias, P. And Skordalakis, E. (1990). Syntactic pattern recognition of the ECG. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7), 648-657.

Wang, Y. P. (1990). Optimal correspondence of string subsequences. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(11), 1080-1087.

Young, T. Y. and Fu, K-S. (1986). *Handbook of Pattern Recognition and Image Processing.* New York: Academic Press, Inc.