Automated Feature Design for Numeric Sequence Classification by Genetic Programming

Dustin Y. Harvey and Michael D. Todd

***Abstract*—Pattern recognition methods rely on maximum-information, minimum-dimension feature sets to reliably perform classification and regression tasks. Many methods exist to reduce feature set dimensionality and construct improved features from an initial set; however, there are few general approaches for the design of features from numeric sequences. Any information lost in preprocessing or feature measurement cannot be recreated during pattern recognition. General approaches are needed to extend pattern recognition to include feature design and selection for numeric sequences, such as time series, within the learning process itself. This paper proposes a novel genetic programming (GP) approach to automated feature design called Autofead. In this method, a GP variant evolves a population of candidate features built from a library of sequence-handling functions. Numerical optimization methods, included through a hybrid approach, ensure that the fitness of candidate algorithms is measured using optimal parameter values. Autofead represents the first automated feature design system for numeric sequences to leverage the power and efficiency of both numerical optimization and standard pattern recognition algorithms. Potential applications include the monitoring of electrocardiogram signals for indications of heart failure, network traffic analysis for intrusion detection systems, vibration measurement for bearing condition determination in rotating machinery, and credit card activity for fraud detection.**

***Index Terms*—Feature design, genetic programming, machine learning, pattern recognition, sequence classification, time series classification, time series data mining.**

I. Introduction

INCREASINGLY, researchers, engineers, and analysts find themselves inundated with numeric sequence data for use in classification tasks. Ever-decreasing costs of sensing, data acquisition, and data storage hardware have led to many applications for which sequential data is abundantly available, but domain knowledge is very limited. Here, sequential data is defined to be a set of ordered numeric samples such as a time series in the time domain or spectral measurement in the frequency domain. Domain knowledge primarily drives the selection and design of preprocessing steps, features, models, and learning algorithms that constitute a pattern recognition system. These decisions are critical to the ultimate performance of the system, but few general guidelines or frameworks exist to guide the inexperienced practitioner. Example applications include the monitoring of electrocardiogram signals for indications of heart failure, network traffic analysis for intrusion detection systems, vibration measurement for bearing condition determination in rotating machinery, and credit card activity for fraud detection. A thorough review of statistical pattern recognition techniques and issues is provided in [1].

Xing *et al.* [2] categorize the various approaches to sequence classification as either model-based, distance-based, or feature-based. Model-based approaches consist of building generative models then applying pattern recognition techniques to the model parameters or predictive errors instead of the sequential data. These approaches are not appropriate for problems where the relevant information is not well represented in the global sequence behavior captured by a generative model. Distance, or similarity, based approaches typically employ a lazy learning approach such as nearest neighbors in conjunction with a Euclidean or similar distance measure between sequences. An experimental comparison of these methods is given in [3]. The main drawback of these approaches is the large computational burden at the operation stage rendering solutions unsuitable for many real-time or embedded applications.

Feature-based approaches represent the sequential data in a feature vector of reduced dimensionality more conducive to the application of statistical pattern recognition algorithms. However, feature design is a challenging task, especially in the absence of sufficient domain knowledge, since numeric sequences have no explicit features and we cannot enumerate all possible features [2]. In a review on the related topic of time series data mining, Fu notes that "the fundamental problem is how to represent the time series data" [4]. Here, we take the position supported by Bagnall *et al.* [5] that the largest performance gains in the area of sequence classification can be achieved through optimal transformation and representation of sequential data in a feature vector as opposed to development of complex classifiers or sequence distance measures.

In this paper, we propose and prove a genetic programming (GP) based approach to automate feature design for numeric sequence pattern recognition systems called Autofead. We refer to the selection of application-specific preprocessing steps and scalar features from sequence data as feature design.

Here, we differentiate between feature design and methods for feature construction, feature extraction, or feature transformation based on the presence of order-dependent operations. As an example, an automated feature construction process using GP with a sequential input is described in [6]. In this paper, the input sequences contain 33 fast Fourier

Manuscript received April 29, 2013; revised August 29, 2013, December 20, 2013, and May 15, 2014; accepted July 16, 2014. Date of publication July 21, 2014; date of current version July 28, 2015. This work was supported by the Department of Defense through the National Defense Science and Engineering Graduate Fellowship Program.

The authors are with the Department of Structural Engineering, University of California, San Diego, CA 92093-0085 USA (e-mail: mdtodd@ucsd.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TEVC.2014.2341451

1089-778X © 2014 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

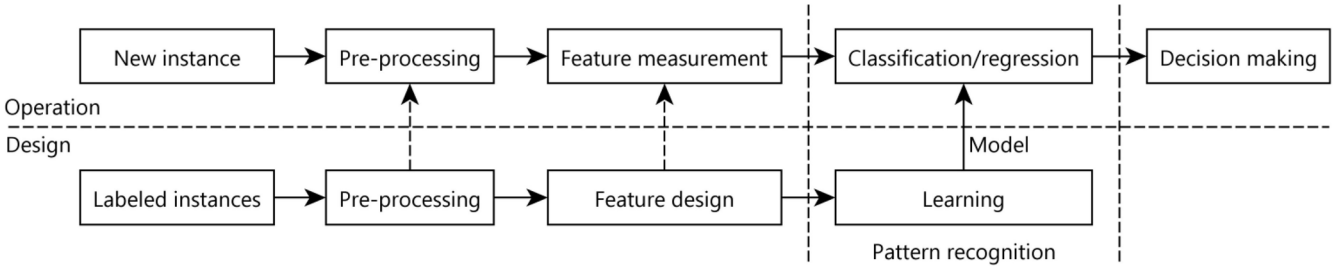

Fig. 1. Statistical pattern recognition paradigm for supervised learning from numeric sequence data.

transform (FFT) bins that are ordered and may contain relevant, dynamic information within their structure. However, no order-based operations are used to construct new feature algorithms; therefore, each FFT bin is treated independently making the system more indicative of feature construction than feature design.

As a usage example, consider a critical machine in a manufacturing process that undergoes periodic bearing failures causing expensive downtime. If maintenance were performed as needed, downtime would be minimized compared to a time- or usage-based maintenance schedule or run-to-failure approach. A damage monitoring system based on pattern recognition would allow for a condition-based maintenance approach. An expert in condition monitoring for rotating machinery could be consulted to design a damage monitoring system based on available literature and past experience with similar machines. Alternatively, an uninitiated engineer or technician could instrument the machine with a vibration sensor and periodically record time series. Records are then labeled as either from the healthy state or just prior to known failures. Finally, an automated approach such as Autofead is used to design optimal preprocessing steps and features for the specific machine and failure mode. The latter approach represents significant potential savings in development time and expense in addition to improved performance of the damage monitoring system.

The remainder of this paper is divided into six sections. The design process for pattern recognition systems is decomposed in Section II into four components. Section III overviews the basic method selected to perform feature design. The Autofead system is detailed in Section IV. Three experiments and detailed results are presented in Section V. Section VI contains an evidence-based discussion of important design choices followed by conclusions from this paper in Section VII.

II. Pattern Recognition System Design

Most pattern recognition algorithms follow the statistical paradigm shown in Fig. 1 wherein each instance is represented by a feature vector describing a single point in the multivariate, scalar feature space. The pattern recognition literature contains numerous methods for performing classification and regression tasks on the feature space as well as improving the feature space through feature selection and feature construction. However, relatively little attention is devoted to the initial feature design process necessary to construct feature vectors from time series, spectra, images, or higher-dimensional measurements. In classification problems, the ideal feature space contains small intraclass variations and large interclass variations to simplify the formation of decision boundaries. For numerical output from regression algorithms, features should be highly correlated with the desired output and robust against any outside effects. The quality of the feature space strongly affects the performance of the overall pattern recognition system. Any relevant information lost during feature measurement cannot be recreated from the feature space. Traditionally, feature designers have relied on domain knowledge and past experience to select features with little assurance of feature optimality. The need is clear for a general, practical, data-driven tool to assist feature designers in minimally-informed pattern recognition applications for sequential data.

Pattern recognition problems are classified as supervised or unsupervised based on whether examples of known patterns are available in the learning process. Here, we pose supervised learning for numeric sequences as a design problem to discover the optimal processing path from sequence inputs to a scalar output or class label relevant to a decision-making process. The processing path begins with design of a preprocessing and feature measurement algorithm for a multivariate set of input sequences. Feature sets are further reduced and transformed to an optimal feature space through feature selection and feature construction. Finally, we train a model to identify the relationship between features and desired outputs in the learning instances. The model can take many different forms dependent on the type of pattern recognition algorithm selected. In operation, the measured feature vector and learned model are used to classify or regress new instances. The design problem can be decomposed into four components as follows, with no single approach optimally suited for each.

Component 1: Feature algorithm.
Component 2: Parameters within feature algorithms.
Component 3: Feature selection and construction.
Component 4: Pattern recognition algorithm.

This paper is primarily concerned with feature algorithm design. Typically, design of features and any data treatment prior to feature measurement is considered part of a preprocessing step to machine learning which deals more directly with components 3 and 4. However, a well-designed feature space eliminates the need for separate feature selection and construction procedures and greatly simplifies the pattern recognition step. Although the four components have very different characteristics, it is difficult to evaluate any one component independently of the system as a whole. Performance of pattern recognition algorithms varies for different feature

choices, and features themselves are best evaluated by their relative performance when input to a pattern recognition algorithm. Therefore, a general approach for data-driven development of numeric sequence pattern recognition systems must handle design of all components simultaneously.

A. Feature Algorithm Design

Here, we define a feature algorithm as a sequence of operations to transform one or more sequential data inputs to a single, scalar feature. Possible operations include, but are not limited to, digital filtering; thresholding; transformations among time, frequency, phase space, or hybrid domains; correlation analysis; principal component analysis; and many more. The algorithm must include a dimension reduction step to produce a scalar such as computing the sum or another statistic from a sequence or selection of the value at a specific sequence index. Feature algorithms can be considered as complex dimension reduction processes with the goal of optimally representing information relevant to a decision process as opposed to the traditional dimension reduction goal of faithfully representing data in a lower dimensional space. In other words, feature design is a dimension reduction process with a performance objective other than self-representation.

Historically, feature design has received the least rigorous treatment of the four system components perhaps because of the abstract nature of features, specifically their representation as algorithms. Typically, features are designed by a domain expert or imposed on the system designer by the available data acquisition and sensing system. Often, a large initial feature set is chosen based on past experience and simple models of the system of interest. The features are selected with little knowledge of their importance to the final decision-making system, leading to a large, difficult problem at the feature selection and feature construction stage.

State-of-the-art general approaches to feature design show promise for moving away from ad-hoc or domain expert approaches. Recently, Deng *et al.* [7] demonstrated improvement over the most popular distance-based approach using the time series forest (TSF) approach. TSF consists of thousands of randomly generated simple features computed over random intervals of the input sequences. Feature selection and classification feature set is performed by a tree-ensemble method. Although the result is still largely uninterpretable, temporal importance curves can be formed indicating critical index regions in the sequence for classification. Similarly, Fulcher *et al.* [8] produced a large, informative feature set by cataloging thousands of features from many different application areas. This paper produced many interesting cross-discipline results but does not naturally lead to a general approach for developing new problem-specific features. Both of these methods produce very large feature sets and a difficult feature selection problem.

A rigorous treatment of feature design as a search for features tailored to a specific decision making process necessitates the development of an algorithm search method that can be incorporated with the other components in the pattern recognition system design problem. Since nearly all modern data processing is performed by computer, we can represent the algorithm search space as the computer program space making the search problem concrete and tractable. In the program space, solutions are composed of functions and arguments where the functions consist of any operations (such as those discussed previously) that may be part of a feature algorithm. Whereas treatment of feature design as a program search has certain advantages, the program space is infinite, making a brute force approach impossible. Some deterministic program search strategies exist such as Levin search [9], Hutter search [10], and the optimal ordered problem solver [11] which provide performance guarantees under certain conditions but can be computationally impractical for even moderately complex problems. Additionally, a random search over an infinite space is unlikely to be efficient. However, evolutionary computation provides a practical, proven search heuristic in the form of GP formally introduced in [12]. Standard implementations are not directly applicable to the feature design problem, but the concept has been successfully adapted to a wide range of problems including feature design for image processing tasks such as edge detection and object recognition [13], [14].

B. Feature Algorithm Parameters

Many feature algorithm operations contain one or more parameters or options that control their behavior. In the feature algorithm design process, optimal features may be overlooked with poor parameter choices. For example, in a filtering operation, proper filter design parameters maximize information gain whereas poor parameter choices can completely obscure the useful information. Numerical optimization techniques are well understood and designed specifically to handle this task. However, standard GP uses predefined or random constants that are recombined simultaneously in conjunction with the overall search. This method does not provide for an efficient numerical search. Some previous work has adopted a hybrid approach including a numerical optimization step prior to fitness evaluation resulting in improved solutions with faster convergence than standard GP [15].

C. Feature Selection and Construction

Often, problems in pattern recognition and data mining start with a large database of features on which a decision is to be based. Feature selection and feature construction operations reduce the feature set to a lower dimension more suitable as input to a pattern recognition algorithm. Feature selection simply identifies a subset of the original features that minimizes information loss. Feature construction attempts to recombine the original features linearly or nonlinearly into a new, smaller feature set. Principal component analysis, factor analysis, and neural networks are popular techniques in the area of feature construction [1]. The optimal, minimal-basis solution retains all the information present in the original data that is relevant to the decision-making process within an orthogonal feature vector. Evolutionary methods have been applied to the problems of feature selection and feature construction with some success [16]. However, these approaches are typically unsuitable for numeric sequence problems where the relevant information is not directly represented in individual samples of the sequence.

D. Pattern Recognition Algorithm

Pattern recognition for multivariate scalar features is a well-developed field in machine learning. Many algorithms are available with varying tradeoffs in accuracy, complexity, and computational cost. The possibility of overfitting must be taken into consideration through selection of a robust method or validation on a data set independent from the data used in the training or learning step. The performance of a poor feature set may be overestimated if overfitting occurs. Conversely, a good feature set may be rejected if the algorithm's assumptions are overly restrictive. For example, the assumption of Gaussian feature distributions greatly reduces computational costs but degrades performance with features that are far from Gaussian. Typically, designers select an appropriate algorithm after basic statistical analysis of the feature set through a trial-and-error process. In this paper, we take the approach that a simple algorithm is sufficient to determine relative performance of features during the feature algorithm design process. Once a good feature set is determined, further studies can be performed to select or develop an enhanced pattern recognition algorithm for the application.

Algorithms can also be inferred directly from the feature set. Espejo *et al.* [16] provide a thorough review on the use of GP for classifier induction. Such methods can evolve decision trees, rule-based systems, discriminant functions, neural networks, and other structures with some advantages over standard algorithms. Classification accuracy of evolved systems is often equivalent or slightly better than traditional algorithms. However, the computational cost to evolve an algorithm is significantly higher than using standard methods from the literature, and the evolved classifiers are often very difficult to interpret or are "black box" in nature [16].

III. Autofead Design Background

A. Introduction to Genetic Programming

GP is a search heuristic originally developed to automate computer programming [12]. Since its introduction, the idea has been adapted to a wide-range of problems including controller design, robotic programming, analog circuit design, and many more [17]. The search process evolves a population of candidate solutions where each individual represents a computer program built from a predefined function set. The most common variant uses a tree-based program representation with functions serving as nodes connected by branches to the input data at the leaves of the tree, called terminals. Individual solutions in each generation are recombined through a breeding process where beneficial code segments (analogous to genetic material) persist in the population and are recombined to find an optimal individual. GP is unique as a search technique because the final size of the solution is not predetermined. The user defines the function and terminal sets, fitness cases (instances in the pattern recognition literature), and provides an interpreter that executes individual programs and computes a fitness measure. Interested readers are directed to [17] for a thorough introduction on the field.

GP was chosen to conduct the feature algorithm design due to its past performance on a wide range of minimally-informed search tasks. In particular, GP-based methods have demonstrated success on problems with the following characteristics [17].

1) Minimal knowledge and understanding of problem domain.
2) Size and structure of desired solution unknown.
3) Large amounts of digital data available or obtainable.
4) Good solutions easily tested but not directly obtained.
5) Analytical solutions unavailable.
6) Optimal solutions desired but approximations acceptable.
7) Small performance improvements worthwhile.

B. GP for Feature Design

As with pattern recognition algorithms, standard GP methods accept multivariate scalar input but cannot take advantage of dynamic information within sequential input data. The literature includes a handful of adaptations for sequence pattern recognition and related problems. The scanning approach is the simplest adaptation wherein standard trees operate on an input sequence in a recursive manner to breed nonlinear digital filters [18]. The Zeus system evolves feature vectors as a forest of strongly-typed trees with each tree producing a single feature [19], [20]. Two systems, GP environment for FIFTH (GPE5) [21], [22] and parallel algorithm discovery and orchestration (PADO) [23], [24], circumvent the need for a strongly-typed system by using a single data stack and functions designed to handle all possible input data types. Table I compares the solution structure for these methods and discusses how each structure addresses the four pattern recognition system design components described in Section II. Very recently, a GP method was proposed for time series event detection, which has many similarities to the current problem. This paper primarily focused on identifying the length of an event in a sequence and finding simple features from multivariate relationships [25].

C. Autofead Design Process

Based on the analysis presented in Section II, the presented review of related literature, and the authors' experiences with various approaches [26], [27], this paper proposes a novel method called Autofead for automated feature design. The overall goal of this paper is to develop a system that can infer from numeric sequence inputs an optimal, minimum-basis feature set for a given decision making process. The approach is designed according to the seven following principles.

1) Mine information from multivariate, sequential inputs.
2) Minimize assumptions on input data.
3) Employ numerical optimization methods appropriately.
4) Use standard pattern recognition methods to evaluate fitness of candidate feature vectors.
5) Avoid black box solutions and strive for easy interpretation of features.
6) Promote feature orthogonality to minimize information redundancy.
7) Minimize required user input.

TABLE I
FOUR GP FEATURE DESIGN ADAPTATIONS

Method, Year	Solution structure	Algorithm and parameter search	Feature selection/ pattern recognition	Comments
Scanning/ recursive trees, 1995 [18]	Single tree evaluated recursively while scanning along input sequence. Trees act as non-linear digital filters.	Standard tree-based genetic programming with simulated annealing to optimize parameters	Single feature output forces feature selection and pattern recognition to be performed within algorithm search.	Solution structure does not naturally lend itself to many common features, but use of numerical optimization within GP search proved beneficial.
PADO, 1995 [22], [23]	Separate system computes confidence for each class. Systems consist of multiple programs represented as graphs with single data stack	Custom genetic operators evolve graphs with function parameters evolved as values on the data stack	Feature selection performed through adjustment of weights in voting procedure for programs to determine system output. Pattern recognition simplified to selecting class based on confidence levels.	Structure and functions designed such that many input types can be handled including sequences, images/matrices, and video data.
Zeus, 2002 [19], [20]	Forest of strongly-typed trees, each tree computes one feature in feature vector	Standard tree-based genetic programming, standard GP constants	Feature selection performed through custom genetic operators with pattern recognition performed on feature vectors using SVM classifier.	Proposes and advocates use of evolved front-end feature extractors with a standard back-end pattern recognition algorithm
FIFTH/ GPE5, 2007 [21], [22]	Linear series of "smart" functions with single data stack	Standard genetic operators with crossover constrained to compatible sequences, standard GP constants	Single feature output forces feature selection and pattern recognition to be performed within algorithm search.	Uses high-level functions such as the FFT which can be calculated from standard optimized libraries and are unlikely to be evolved from basic mathematical operations

IV. AUTOFEAD SYSTEM

A. Overview

This paper represents the first automated feature design system for numeric sequences to leverage the power and efficiency of both numerical optimization and standard pattern recognition algorithms. Autofead uses a GP variant to evolve features for input to standard pattern recognition algorithms through a wrapper approach. Solutions consist of a set of one or more features each computed from a single sequence input through a series of sequence-handling functions, as opposed to a tree structure. For example, a feature that computes energy of a time series sequence would use the two-function algorithm square then sum. Each feature algorithm ends with a summation to ensure scalar output. Unlike traditional GP, the solution structure has no constant terminals; instead, functions have parameters that are numerically optimized during each generation. Feature selection is performed concurrently with feature design by intelligent genetic operators. Fitness of candidate solutions, i.e., feature vectors, is evaluated using any standard classification or regression algorithm and a suitable error-based measure such as classification accuracy. Fig. 2 depicts the search process detailed in the following sections.

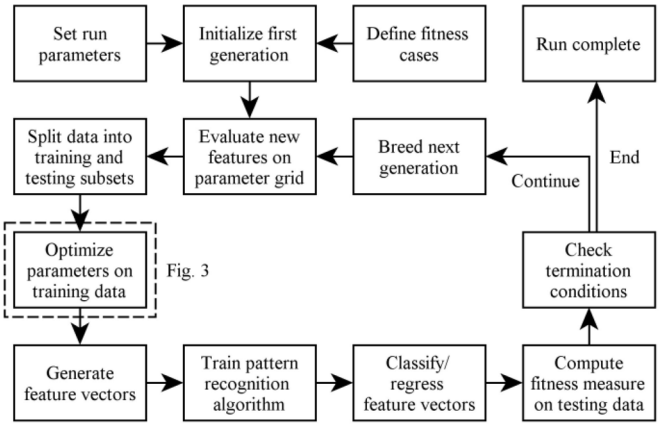

Fig. 2. Autofead search process.

B. Function Set

Autofead features are constrained to relatively short algorithms (typically 2–10 functions) ending in a summation. The solution structure is designed to be far more restrictive than traditional, tree-based GP for two reasons. Firstly, a highly-constrained structure facilitates the analysis and interpretation of results by the user. Secondly, restricting the size of solutions enables use of a large, diverse function set without producing an unreasonably large solution space. However, the inclusion of a carefully designed function set allows for a great variety of possible feature behaviors. The functions are selected to admit features derived from the distributions, dynamics, and transient behaviors of input sequences. In many cases, functions were formed by decomposing common signal processing algorithms into their individual operations then reducing redundancy within the overall function set [28]. Design of the optimal, minimal-basis function set remains an open research topic. The current set of 35 functions and their parameters are described in Table II. Each function, with the exception of sliding windows, takes a single sequence as input and outputs a modified sequence. Dimension-reduction functions reduce the output sequence length to one sample.

The sliding windows function is of particular interest as it provides the primary mechanism for analysis of transients or nonstationarity in a sequence. This function is inspired by the segmentation process used in power spectral estimation via Welch's method [28]. The input sequence is expanded to a matrix of subsequences by extracting consecutive overlapped

TABLE II
AUTOFEAD FUNCTION SET

Function	Output	Parameter (*integer type)
Monotonic element operations		
Cube	Cube of input values	-
Square Root	Square root of magnitude of input values, sign retained	-
Exponential	Exponential function of input values	-
Non-monotonic element operations		
Absolute value	Absolute value of input values	-
Square	Square of input values	-
Sign	Sign of input values, -1 or 1	-
Log10	Base-10 logarithm of magnitude of input values	-
Distribution-altering functions (sample order independent)		
Demean	Input sequence with mean offset removed	-
Normalize	Standard deviations from mean of values in input sequence	-
Set minimum value	Input with values below threshold raised to threshold	Data range threshold (0-1)
Set maximum value	Input with values above threshold lowered to threshold	Data range threshold (0-1)
Control chart	Input with central values of data range set to center of range	Percentage of range kept (0-1)
Sort order	Indices of values in sorted input sequence, ascending order	-
Order-dependent functions		
Difference	First differences of input sequence	-
Cumulative summation	Cumulative summation of input sequence	-
Hanning window	Input sequence with Hanning window applied	-
Low-pass filter	Low-pass filtered input sequence, zero-phase digital filtering by 3rd order Butterworth filter	Normalized cutoff frequency (0-1, upper bound = π rad/s)
High-pass filter	High-pass filtered input sequence, zero-phase digital filtering by 3rd order Butterworth filter	Normalized cutoff frequency (0-1, upper bound = π rad/s)
Auto-correlation function	Biased estimate of input sequence's auto-correlation function, only positive lag values	-
FFT magnitude	Magnitude of FFT of input sequence, [0,π] rad/s bin range	-
FFT phase	Phase of FFT of input sequence, [0,π] rad/s bin range	-
FFT real	Real part of FFT of input sequence, [0,π] rad/s bin range	-
FFT imaginary	Imaginary part of FFT of input sequence, [0,π] rad/s bin range	-
Hilbert magnitude	Magnitude of Hilbert transform of input sequence	-
Hilbert phase	Phase of Hilbert transform of input sequence	-
Hilbert imaginary	Imaginary part of Hilbert transform of input sequence	-
Index-altering functions		
Keep beginning	Input sequence with samples removed from end	*Samples to remove
Keep end	Input sequence with samples removed from beginning	*Samples to remove
Sliding windows	Windowed subsequences extracted from sliding a window along input sequence, number of windows and overlap between adjacent windows determined internally	*Window length
Transpose windows	Swaps subsequence window indices and sample indices	-
Dimension-reduction functions		
Sum (implicit)	Sum of all input values, occurs at end of every feature	
Sum windows	Sequence composed of sum of each subsequence window	-
Select	Sample at index of input sequence that maximizes grade of function output, each subsequence window treated as separate fitness case for grading, brute force search	*Internal index selection
Sorted select	Same as *select* function with input sequence sorted	*Internal index selection
Bisection select	Same as *select* function but using bisection search	*Internal index selection
Sorted bisection select	Same as *sorted select* function but using bisection search	*Internal index selection

windows. To constrain the function to a single parameter and limit growth of data sizes within algorithms, the number of windows constructed and number of overlapping samples in sequential windows is internally determined based on the ratio of the window length parameter to the input sequence length. The total data size of the output windowed subsequence matrix is limited to between 1.5 and 2 times the size of the input sequence. When subsequent functions receive windowed subsequences created by the sliding windows function, they act upon each subsequence individually with the exception of transpose windows. Windowed subsequences are collapsed back to a single sequence through any of the dimension-reduction functions.

The function parameters are divided into continuous type (normalized between 0 and 1) and integer type, which are handled separately within the parameter optimization process. The continuous parameters in the filtering and thresholding operations tend to have strong effects on feature performance without altering the overall feature behavior.

Additionally, all of the continuous parameters have a natural default value that results in minimal or no change to the input. A high cutoff frequency parameter in the low-pass filter function causes minimal filtering effects. In contrast, the integer parameters tend to have coupled relationships leading to a wide range of possible behaviors. For example, the sliding windows function followed by the select function creates a

decimation behavior where the window length parameter controls the new data rate and the index selection parameter varies the first index retained in decimation.

C. Terminal Set

Terminal set selection in Autofead is greatly simplified, as there is no notion of constants. The terminal set simply consists of one or more sequence inputs to the problem. While the system was designed for direct input of raw measurement data such as time series, domain knowledge may indicate that alternative representations are beneficial for a specific application. In this case, transformed input data can be used instead of or in addition to the raw sequence inputs. For multivariate problems, each feature within an individual is associated with a single input sequence. In the breeding process, the input associated with a specific feature can be altered, and features associated with differing inputs are allowed to produce offspring. Scalar inputs are also supported, but are passed directly to the pattern recognition stage.

D. Parameter Optimization

The goal of the parameter optimization scheme within the Autofead search process is not necessarily to find the true optimal parameters but rather to assist the overall search by ensuring that good feature algorithms are not overlooked due to poor parameter choices. The parameters of the final solution's features should be reoptimized post-search by standard global optimization methods before implementation in real systems. However, due to computational expense, a full global optimization of all parameters within each individual is unfeasible within the search. Instead, a sequential global optimization strategy is employed including a number of parameter independence assumptions as described in the following section. Assumption of independence both between features in an individual and between parameters in a feature algorithm greatly reduces the computational requirements for optimization.

To avoid the computational expense of a multifeature objective function, function parameters are optimized for a single feature at a time in an individual based on grades computed by a grading function (objective function for parameter optimization). Grades can be computed as classification accuracy of a 1-D classifier, error rates using a regression function, or from a statistical divergence measure between classes. Computational expense is important in selection of the grading function, as millions to hundreds of millions of grades are required in a single run. Currently, correct classification rate using a linear discriminant analysis classifier is the preferred choice.

Additionally, the optimization of each continuous parameter is performed independently from the set of integer parameters in a given feature based on the observations described in Section IV-B. For a feature containing M integer parameters and N continuous parameters, this decoupling leads to a single, M-dimension optimization followed by N, one-dimension optimizations instead of a single, (M + N)-dimension optimization. For the integer parameter set, the initial grid search is performed using an adaptive grid density based on variance-based total effect sensitivity indices [29]. Default values for the continuous parameters are used while building the integer parameter grid. The grid search for subsequent continuous parameters assumes the best parameter values from previous grid searches. The feature evaluations for all grid points are stored to reduce repeated evaluations on identical feature algorithms later in the search process.

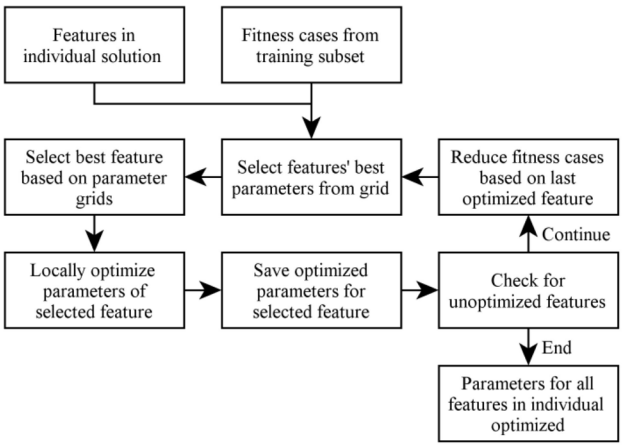

Fig. 3. Autofead parameter optimization process.

After the grid search, local optimization for an individual solution is conducted as shown in Fig. 3. The goal is to locally optimize parameters in each feature starting from the optimal points on the stored parameter grids while simultaneously promoting orthogonality of features to ensure that identical or similar feature algorithms do not generate highly correlated information. First, the parameters within the feature with the highest grade from the grid search are locally optimized. The integer parameter set is optimized through a bisection search with initial step sizes calculated from the adaptive grid density in each dimension. Then, the continuous parameters are optimized via Brent's method. Orthogonality of remaining features in the individual is promoted by reducing the set of fitness cases used in subsequent local optimizations to those with highest error measures using the current feature alone to perform the task at hand. In other words, the second feature is optimized on the data instances misclassified or poorly regressed by the first feature. For classification problems, the current method of fitness case reduction generates equi-probable bins over the feature's value range then retains fitness cases that fall in the half of the bins with largest class parity.

The following process summarizes all optimization steps for a single feature containing two integer parameters, I1 and I2, and two continuous parameters, C1 and C2.

1) Set C1 and C2 to default values.
2) Build adaptive parameter grid for I1 and I2 and set to the best values from grid.
3) Build parameter grid for C1 and set to the best value from grid.
4) Build parameter grid for C2 and set to the best value from grid.

5) Locally optimize I1 and I2 using bisection search.
6) Locally optimize C1 by Brent's method.
7) Locally optimize C2 by Brent's method.

As noted in Table II, the family of select functions has internally-optimized index parameters. Because these functions tend to occur at the end of a feature, the feature output can be graded for different index selections without reevaluating the entire feature from the beginning. Additionally, the index parameter for the select function tends to have high sensitivity and a sparse optimization surface. These characteristics necessitate a brute force search, which would be prohibitively computationally expensive if reevaluating long features for each index. For the sorted select function, the index parameter surface tends to be much smoother, thus admitting a bisection search. Hence, it is recommended the select and sorted bisection select functions be used during search in place of bisection select and sorted select.

E. Fitness

Fitness of candidate feature vectors is calculated as an error measure for the selected classification or regression algorithm. In each generation, the fitness cases are divided into independent training and testing sets to avoid overfitting. More advanced evaluation methods such as cross-validation could be implemented as well but with added computational cost. In accordance with the goal of producing orthogonal features, Naïve-Bayes type classifiers are an appropriate selection due to their assumption of feature independence; however, a Gaussian Naïve-Bayes classifier is not appropriate for many problems as the feature distributions are often far from being normally distributed. Here, class-conditioned probability density functions are estimated by an adaptive-bandwidth kernel method via Scott's rule [30].

F. Breeding

Breeding is a two-step procedure of parent selection and application of a genetic operator to selected parents to produce offspring. Any standard selection mechanism can be used such as tournament selection or fitness-proportional roulette. The Autofead genetic operators are simplifications of the standard crossover, mutate, and reproduce operators in tree-based GP supplemented with operators that intelligently perform feature selection. The usage frequency of each operator is controlled through assigned genetic operator probabilities. Crossover and mutate modify a single feature algorithm within the first parent. Crossover, the most often used operator, replaces a random segment of functions within one feature algorithm with a random algorithm segment from one of the second parent's features. The mutation operator randomly removes, inserts, or swaps out a single function in a feature algorithm or changes the feature's associated input sequence index for multivariate problems. Mutation helps maintain function diversity within the population as it converges and enacts small changes to locally search the solution space. Reproduction clones a single parent with no modification of the features. Although the feature algorithms are identical for the clone, parameters may change as optimization is performed on a new training subset of the fitness cases.

Feature selection is performed through the remove feature and add feature operators inspired by the architecture-altering operations in [31]. The grading function from the parameter optimization process is used to identify the relative value of features in an individual. The remove feature operator deletes the feature with the worst grade on all fitness cases from a single parent. Add feature begin by reducing the fitness case to the subset with largest error using all features from the first parents by the same method of fitness case reduction used in parameter optimization. Then, the features from the second parent are graded on the subset of fitness cases to determine which feature is most beneficial to transfer to the first parent. Additionally, the maximum number of features allowed in an individual is constrained to control bloat.

G. Post-Search Refinement

Autofead includes many simplifications of and restrictions to the solution structure, function set, and parameter optimization scheme, which make the search computationally tractable. Many of these constraints can be removed post-search to improve the final feature set. In particular, individual features should be refined based on an understanding of the behaviors produced by certain patterns of functions. Some of these improvements take the form of simple program rewrite rules that should always be followed to reduce the computation required to generate a feature. Other guidelines recommend preferred forms to create certain feature behaviors or point out certain function design choices that may be limiting feature performance. In these cases, the suggestions should not be followed blindly but taken as a guide for feature refinement. The refinement process could be automated with determination of which refinements to follow based on fitness of the resulting solution. Table III presents a partial list of feature refinement rules and recommendations, respectively.

The last three guidelines provide examples of function design decisions such as the filter choice in the filtering functions, use of a Hanning type window, and restrictions on the sliding windows function that should be explored to determine if fitness can be further improved. Additionally, some feature behaviors can be produced with a variety of function patterns. Using an alternate form of the behavior can improve fitness and/or reduce computational expense. For example, the sequence sliding windows followed by sum windows is an inefficient representation of a moving average filter with decimated output. Replacing this segment of a feature with the low-pass filter function may improve performance. Finally, a global reoptimization of all the parameters in an individual should be performed after any feature modifications.

V. Experiments

Three numerical experiments were performed to test various aspects of the proposed approach. In each problem, we seek a feature set to differentiate between two classes of time series signals. Experiment 1 requires Autofead to generate three simple but very different features and perform feature selection to combine the three features in a single individual. Experiments 2 and 3 require only a single feature and thus focus on the feature

TABLE III
AUTOFEAD POST-SEARCH FEATURE REFINEMENT RULES AND RECOMMENDATIONS

Rule	Applicable Functions
If repeated sequentially, remove all but one instance.	*Absolute value, sign, log10, exponential, demean, normalize, Hilbert magnitude, set minimum value, set maximum value, control chart, keep beginning, keep end*
Remove invalid functions for single sequences (not windowed subsequences).	*Transpose windows, sum windows*
Remove *sliding windows* function when applied to windowed subseqeunces.	*Sliding windows*
Replace with *absolute value*.	*Square* then *square root, square root* then *square*
Remove *demean* function.	*Demean* then *normalize, normalize* then *demean*
Remove *absolute value* when following functions with positive output.	*Exponential, square, sort order, FFT magnitude, Hilbert magnitude*
Remove sequential inverse operations.	*Cumulative summation* and *difference, exponential* and *log10*
Remove functions from end of feature that have no effect.	*Sum windows, transpose windows*
Remove functions from end of feature that result in constant.	*Sort order, demean, normalize*
Recommendation	**Applicable Functions**
Replace bisection search functions in select family with brute force versions.	*Select, sorted select*
Remove if optimal parameter value causes function to have little effect.	*Set minimum value* (p=0), *set maximum value* (p=1), *control chart* (p=1), *low-pass filter* (p=1), *high-pass filter* (p=0), *keep beginning* (p=0), *keep end* (p=0), *sliding windows* (p=1)
Remove each function individually from feature to detect detrimental bloat.	All functions
Explore different filtering options and optimize filter order.	*Low-pass filter, high-pass filter*
Explore different window options including rectangular window formed by keep beginning and keep end functions.	*Hanning window*
Remove restrictions on *sliding windows* function and optimize window length, number of windows, and overlapping samples between windows.	*Sliding windows*

TABLE IV
KOZA TABLEAU FOR EXPERIMENTS 1–3

Parameter	Setting
Objective	Find feature set with best classification rate for two-class signal problem
Structure	Set of features each composed of a series of sequence-handling functions
Function set	33 functions from Table 1 (*Sorted select* and *bisection select* omitted)
Terminal set	Single numeric input sequence
Fitness	Percentage correctly classified signals using Kernel-based Naïve-Bayes classifier
Fitness case sampling	100 training and 2,400 testing cases randomly selected from each class for each generation
Population initialization	Ramped to initial maximum size of 3 features (1 feature for experiments 2 and 3) and up to 5 functions and 3 parameters per feature
Population size	500
Selection	Tournament (tournament size 4), no elitism
Genetic operators	Crossover, 80%; Mutate, 5%; Reproduce, 5%; Add Feature, 5%; Remove Feature, 5%
Individual size limit	5 features (1 feature for experiments 2 and 3), each limited to 15 functions and 8 parameters
Termination	20 generations

algorithm design and parameter optimization components. All reported fitness levels are calculated through 10-fold cross-validation on an independent test set, and each problem has a known analytic solution for performance evaluation.

It is important to note that derivation of the optimal, analytic solutions for each experiment requires complete knowledge of the signal generation process for both classes. Autofead is designed to discover optimal and near-optimal solutions with minimal knowledge. No application-specific information is provided to the system in any of the experiments other than restricting experiments 2 and 3 to single feature solutions. The Koza tableau in Table IV summarizes run options used for the three experiments. Run parameters are currently based on common practice with similar GP systems and the authors' engineering judgment.

A. Experiment 1

The first problem involves separating a zero-mean, unit-variance, white, Gaussian noise (WGN) process in class 0 with one of three equally-likely subclasses in class 1. Subclass 1a is generated by passing WGN through a weighted, 2-sample moving average filter. Subclass 1b has a slight variance increase halfway through the signal, and a white, uniformly distributed noise process produces subclass 1c. Each subclass involves a slight modification to the WGN process of class 0, but in each case the expected mean value and signal energy are unaltered. All signals consist of 100 samples with index 0 to 99. The best Autofead solution from a single run contained five features. Two features were redundant and have been removed with no loss of fitness. The remaining three features target detection of a specific subclass of class 1.

Subclass 1a has altered dynamics from a moving average filter with weights $\sqrt{0.8}$ and $\sqrt{0.2}$ for the 0 and 1 sample lags, respectively. The optimal feature to detect this change is the value of the auto-correlation function at 1 sample lag. The Autofead feature took a different approach by calculating an energy measure for the high-frequency band of the signal. The feature algorithm is high-pass filter (p = 0.70), cube, absolute value, and ends with summation. Because the moving average filter is a low-pass filter, class 0 contains higher energy than subclass 1a in the high-frequency band. The sequence cube, absolute value, and then sum computes an energy measure using the third power instead of the second power.

Subclass 1b is nonstationary with a variance change from $\sqrt{0.5}$ to $\sqrt{1.5}$ at sample 50. The global dynamics and distribution of subclass 1b are identical to a WGN process requiring

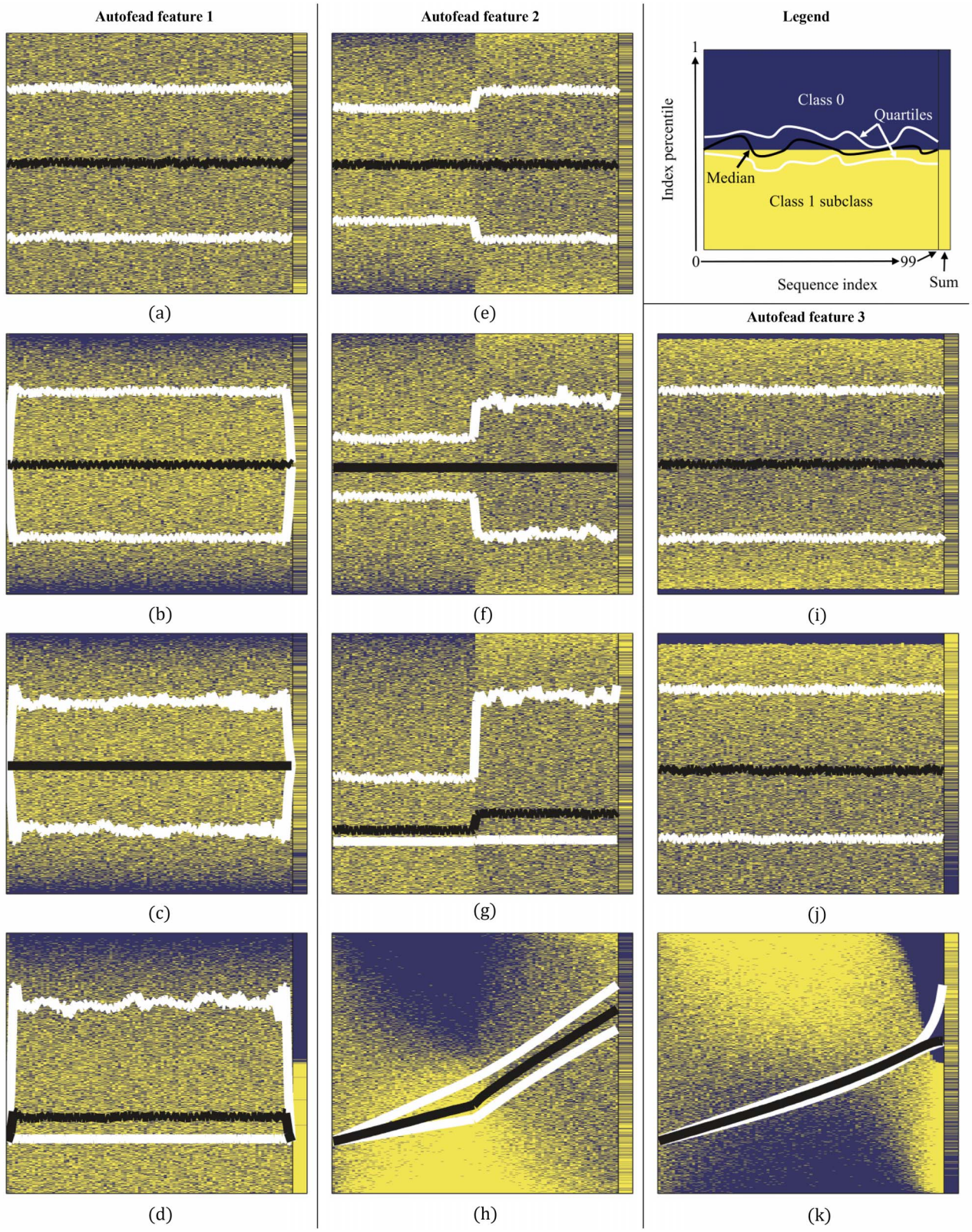

Fig. 4. The processing images for Experiment #1 for three features in the best Autofead solution. Autofead feature 1. (a) Input (class 0 and subclass 1a). (b) High-pass filter (p = 0.70). (c) Cube. (d) Absolute value, end of feature is sum. Autofead feature 2. (e) Input (class 0 and subclass 1b). (f) Cube. (g) Absolute value. (h) cumulative summation, end of feature is select (p = 46). Autofead feature 3. (i) Input (class 0 and subclass 1c). (j) Absolute value. (k) Sort (from sorted select), end of feature is sorted select (p = 99).

the development of a feature that can detect a non-stationary change in the signal. The difference in the variance of the two signal halves provides the optimal discriminator between class 0 and subclass 1b. Autofead accomplishes the task through the feature algorithm cube, absolute value, cumulative summation, and select (p = 46). Again, the sequence cube,

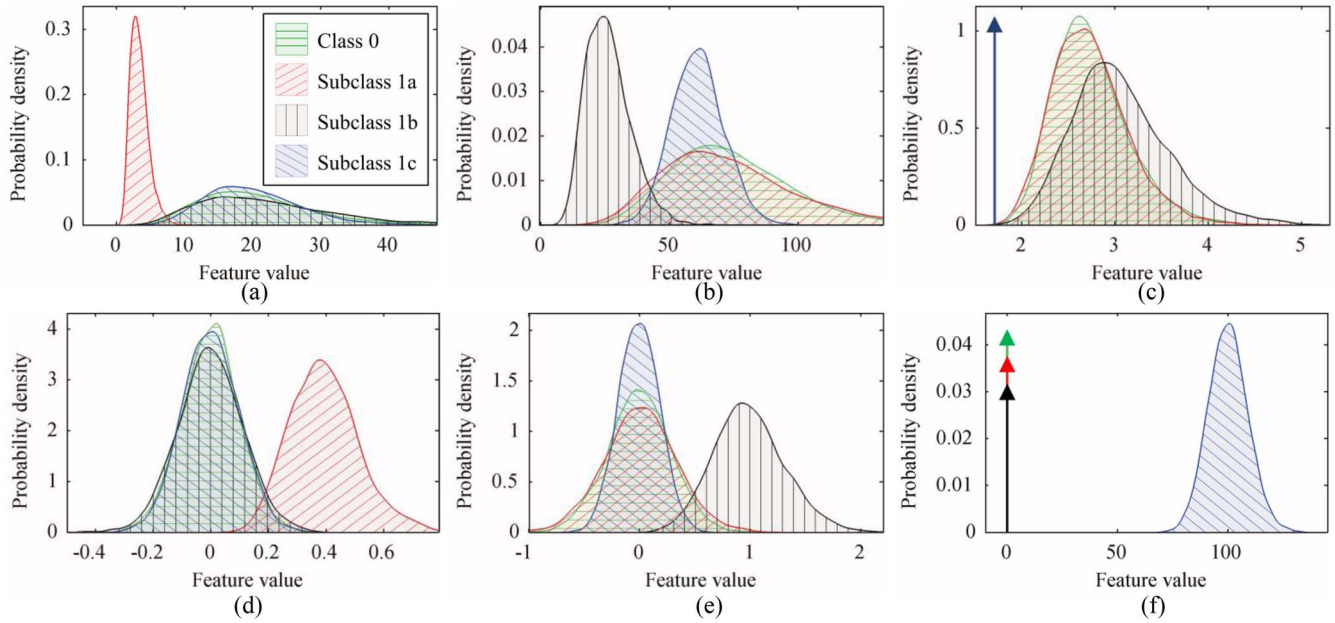

Fig. 5. Experiment 1 comparison of feature probability density functions for optimal and Autofead solutions. (a) Autofead, subclass 1a. (b) Autofead, subclass 1b. (c) Autofead, subclass 1c. (d) Optimal, subclass 1a. (e) Optimal, subclass 1b. (f) Optimal, subclass 1c.

absolute value, and then sum from the cumulative summation computes an energy measure. By taking a cumulative summation and selecting index 46 as the feature value, only samples from the low variance section of the signal are included in the feature. Here, the optimal parameter choice is index 49 to capture the entire first half of the signal, but index 46 was selected as optimal for the particular subset of training fitness cases used during optimization.

Lastly, a white, uniformly-distributed noise process is used to generate subclass 1c. Bounds of $-\sqrt{3}$ to $\sqrt{3}$ for the uniform distribution are selected to produce zero-mean, unit-variance signals. For this subclass, the optimal feature is simply to check for any values outside the bounds of the uniform distribution. Consequently, the associated feature in the Autofead solution finds the largest magnitude sample using the algorithm absolute value and sorted select (p = 99). The sorted select function, using index 99, selects the maximum value.

Fig. 4 illustrates how class 0 and each subclass are identified by the associated three features in the Autofead solution. Each column provides a series of images corresponding to the processing steps in a single feature. For each image, the horizontal axis indicates samples of the sequences at the given processing step and is supplemented with the sum of the current sequences on the right end. The vertical axis represents the entire data range for individual sequence indices. At each sequence index, the values from all fitness cases are sorted and replaced by their known class labels. Then, the background images are generated by assigning a color or gray-level value to each class. In the foreground, median values and quartiles at each sample are shown by black and white lines, respectively.

This visualization technique provides a number of advantages for feature algorithm analysis and design in conjunction with Autofead. Although absolute values and distribution shapes are admittedly obscured, class separability is only dependent on the relative distribution of the classes over the value range that is explicitly presented. For example, Fig. 4(a) shows a completely heterogeneous image indicating that the two class distributions are highly overlapped across the entire sequence. In terms of feature design, a fully heterogeneous image dictates application of an order-dependent function as any element operations or distribution-altering, index-altering, or dimension-reduction functions would act equally on both classes.

In contrast, the final image for each feature shows clear regions of homogeneity indicating a difference between classes that can be exploited. After the absolute value operation in feature 1, Fig. 4(d) shows the largest values across the sequence belong to class 0 and the smallest values fall in subclass 1a. The sum column on the right clearly shows that summing the sequences at this point generates a feature with excellent class separability.

Features 2 and 3 end with different dimension-reduction functions than feature 1 corresponding to specific characteristics apparent in the processing images. Fig. 4(h) contains columns in the middle of the sequence with large class separability even though the sum column only shows small class differences. Consequently, feature 2 ends by selecting index 46 as the feature output. Lastly, Fig. 4(j) shows that the largest values across all indices belong unanimously to class 0. The presence of a percentile level (row in the image) composed of largely a single class suggests terminating the feature through the sorted select function. The image in Fig. 4(k) after the sort operation alone clearly shows excellent class separation for the largest indices corresponding to the maximum values in the unsorted sequences.

The feature distributions for the optimal solution and Autofead solution are given in Fig. 5 conditioned on class 0 and the three subclasses of class 1. The arrows in Fig. 5(c) and (f) indicate impulse or near-impulse distributions. For each feature, one, and only one, subclass shows large separation from class 0 verifying that the problem requires at least three

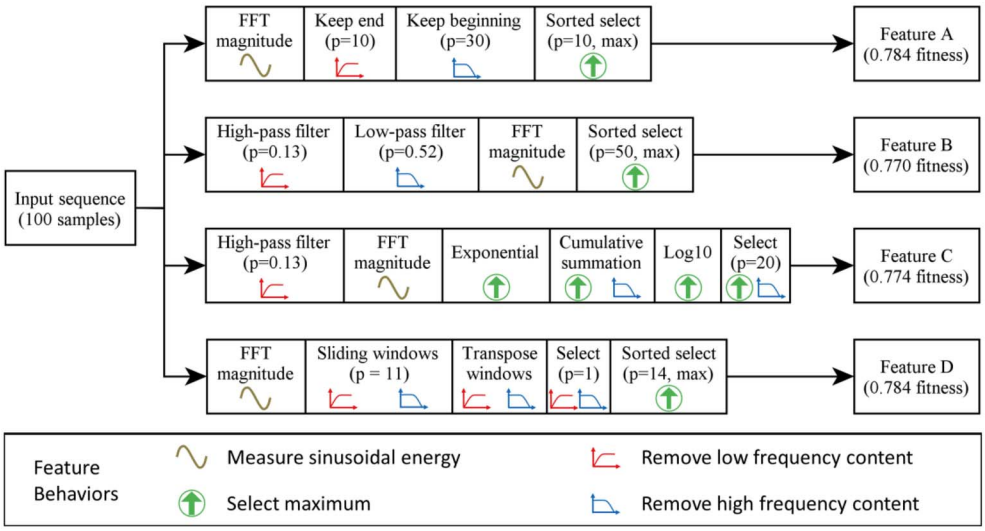

Fig. 6. Experiment 2 feature behavior analysis for four Autofead solutions.

features to construct a near-optimal solution. In this case, the fitness of the discovered solution computed as classification accuracy is identical to the optimal fitness level to within 0.1%. These results demonstrate that the approach can design simple features and perform feature selection within its population.

B. Experiment 2

The second experiment is a classic problem from detection theory of determining the presence of a sinusoidal signal obscured by WGN. Class 0 contains noise alone, and class 1 includes a -10 dB signal-to-noise ratio sinusoid with random phase and frequency. The phase is uniformly distributed over $-\pi$ to π radians. The frequency is uniformly distributed between $\pi/5$ and $2\pi/5$ rad/s. With 100 sample input signals, the frequency range endpoints correspond to the bin-centers of FFT bins 10 and 20.

The optimal feature for experiment 2 is the maximum magnitude of the FFT from bins 10 through 20. Four separate behaviors are necessary to compute the optimal feature. Low- and high-frequency content outside the admissible frequency range must be removed. Next, the amount of sinusoidal energy present at each frequency is measured. Finally, the maximum energy level is selected as the feature. Fig. 6 depicts four solutions found by Autofead that perform these behaviors in different ways. Feature A is the true optimal feature in minimal form. Feature B uses simple filtering operations to control the frequency content with a slight reduction in fitness due to the filter rolloff in the filtering functions. Function C utilizes an interesting pattern to select the maximum over the first 20 FFT bins with the low-frequency content removed by high-pass filtering. Lastly, feature D extracts the FFT bin 10 to 20 segment using the sliding windows function resulting in optimal fitness. Here, the construction of the second of 3, 11-sample subsequence windows corresponding exactly with indices 10 through 20 is purely coincidental. With a different range of admittable frequencies, the same feature would have suffered a small performance loss by missing useful, or retaining unneeded, FFT bins.

Experiment 2 represents a significantly more challenging feature algorithm design problem than experiment 1. Here, the optimal solution requires at least four functions including three parameters. Autofead again performed excellently finding a variety of optimal and near-optimal features. Furthermore, the search discovered unconventional function patterns to effectively perform needed behaviors such as in feature C. For all of these features, the optimized parameter values selected in the optimization scheme represented the true, global optimal values.

C. Experiment 3

Experiment 3 adds an additional level of signal parameter variation to experiment 2 by the obscuring the sinusoidal signal in a longer WGN sequence with a random signal arrival time. Each signal contains 500 samples with class 1 including a 100-sample long sinusoid with starting index uniformly distributed over sample range 0 to 399. The optimal solution involves calculation of the optimal feature from experiment 2 for every 100-sample subsequence and selecting the maximum value. The sliding windows function is designed to permit this type of behavior; however, the restrictions on number of windows and overlap between sequential windows make the optimal solution unrealizable. The best Autofead solution as-found is shown in Fig. 7 along with the optimal solution utilizing an all sliding windows function not present in the function set. The classification accuracy of the as-found solution is still good at 85.8% compared to the optimal level of 90.8%, and the solution is accomplished with only three parameters as opposed to five in the optimal feature.

Although Autofead did not find an optimal-fitness solution, the as-found feature is sufficient as a starting point for refinements to reach the optimal feature. Fig. 7 shows three refinement steps based on understanding of the behaviors within the as-found feature. First, the auto-correlation function prior to an FFT operation is removed as it only acts as a triangular windowing process due to the biased

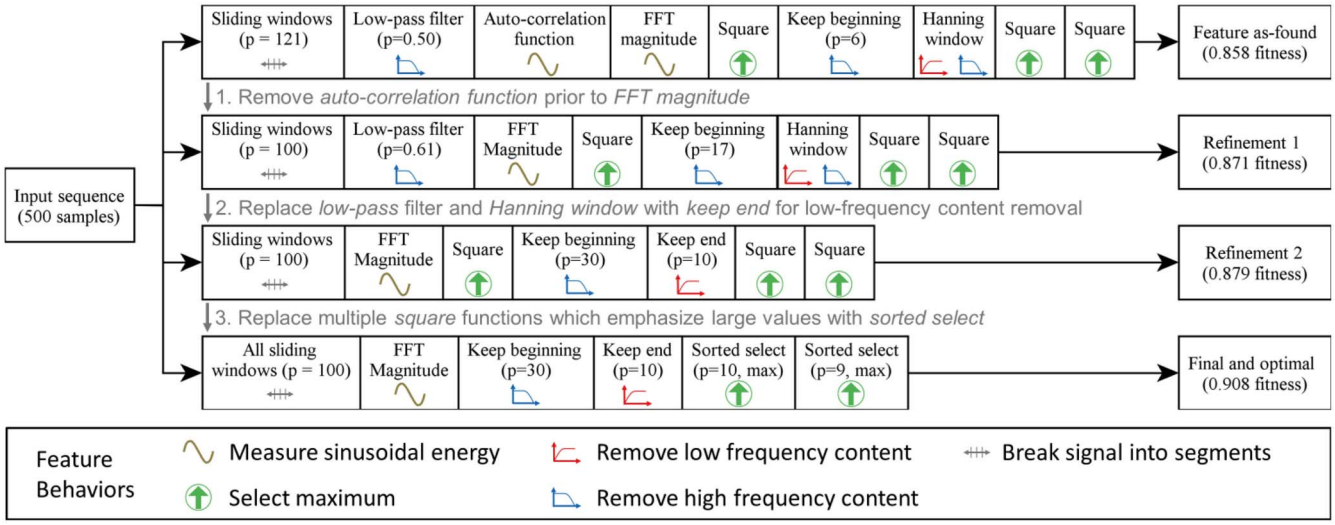

Fig. 7. Experiment 3 feature behavior analysis for Autofead solution, two intermediate levels of refinement, and final solution.

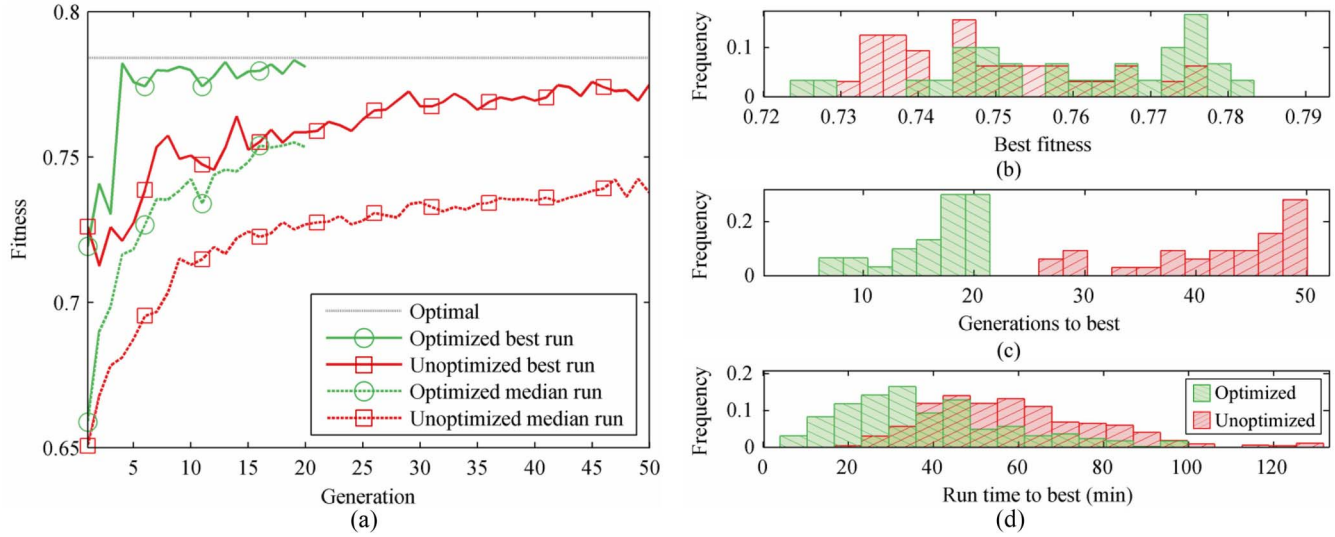

Fig. 8. Autofead performance and run time comparison for experiment 2 with and without parameter optimization. (a) Best and median run fitness by generation. (b) Solution fitness comparison. (c) Solution convergence comparison. (d) Computational effort comparison.

estimation process. For refinement 2, we recognize that the Hanning window and keep beginning functions act together to weight a range of sequence indices in the summation. Replacing Hanning window with keep end creates a rectangular window sized by the parameter optimization process. Lastly, repeated power functions and exponentials before a summation tend to emphasize the largest values, so we replace the three square functions with sorted select resulting in the final and optimal feature. Hence, the Autofead as-found solution, while useful on its own, is best treated as a starting point for the post-search refinement and improvement process. Currently, refinement must be performed manually but could be automated as described in Section IV-G.

VI. Discussion

The three experiments in Section V demonstrate optimal and near-optimal performance of features designed by Autofead for a range of signal classification tasks. In this section, we take a closer look at various aspects of the search process using examples from experiment 2 starting with a discussion of computational effort in Section VI-A. Section VI-B provides evidence supporting design decisions in the parameter optimization scheme.

A. Computational Performance

The computational expense of traditional GP is immense for even moderately large problems. The addition of numerical optimization procedures and training of thousands of pattern recognition algorithms was unrealistic on desktop workstations as recently as a decade ago. On a single, modern desktop workstation utilizing six processor cores, a typical Autofead run time is measured on the order of hours. As with many GP methods, the search process can be heavily influenced by the random initialization of the first generation. Therefore, 10s to 100s of runs are necessary to provide a measure of confidence that the best possible solution is found

requiring days to a few weeks for multiple runs on large problems.

In Autofead, feature evaluations and grades required during parameter optimizations account for 80%–90% of the necessary computational effort. A study was carried out to determine if the inclusion of parameter optimization is beneficial to the search process. Thirty runs of experiment 2 were performed with and without parameter optimization using a population size of 100 individuals. For the unoptimized runs, the optimization module was replaced with a random number generator using a uniform distribution over the parameter bounds, and the runs were carried out to 50 generations per the rule-of-thumb for traditional GP. In each generation of each run, the fitness of the best individual was saved for analysis.

Fig. 8 presents results from the parameter optimization study. In Fig. 8(a), the optimized runs converge in far fewer generations than the unoptimized runs and reach a higher median and best fitness level. Fig. 8(b) and (c) confirm that the inclusion of parameter optimization results in improved solutions found in fewer generations. The convergence rate in terms of generations is admittedly misleading as optimized runs require far more computational effort and run time per generation. By sampling from the distributions in Fig. 8(c) and the measured run times, the distributions of run time to reach best fitness are estimated and presented in Fig. 8(d). From these results, it is clear that the inclusion of parameter optimization in Autofead results in better solutions in less overall run time.

B. Analysis of Parameter Surfaces

Many design decisions were made in development of the Autofead parameter optimization scheme based on observation of parameter behaviors and fitness surfaces for various features. Fig. 9 shows four example parameter surfaces for the features from experiment 2 presented in Fig. 6. Three of the features end with a sorted select function whose parameter has been omitted for visualization purposes. The behavior of the sorted select parameter in these cases is always to select the maximum value. Each parameter surface is shown for the entire bounded range of each its parameters. Features A and D in Fig. 9(a) and (b), respectively, contain an invalid parameter region shown as 0.5 fitness in black.

In general, the parameter surfaces for Autofead features tend to be smooth and unimodal or bimodal in a single region containing the optimal point. Outside this region, the surface contains a rough landscape full of local minima. These observations provide justification for the choice of global optimization instead of a local method. The initial grid search locates the smooth fitness region then local optimization is initiated to reach optimal fitness. For all four features shown in Fig. 9, the parameter optimization scheme found the true, global optimum in less than a hundred feature evaluations.

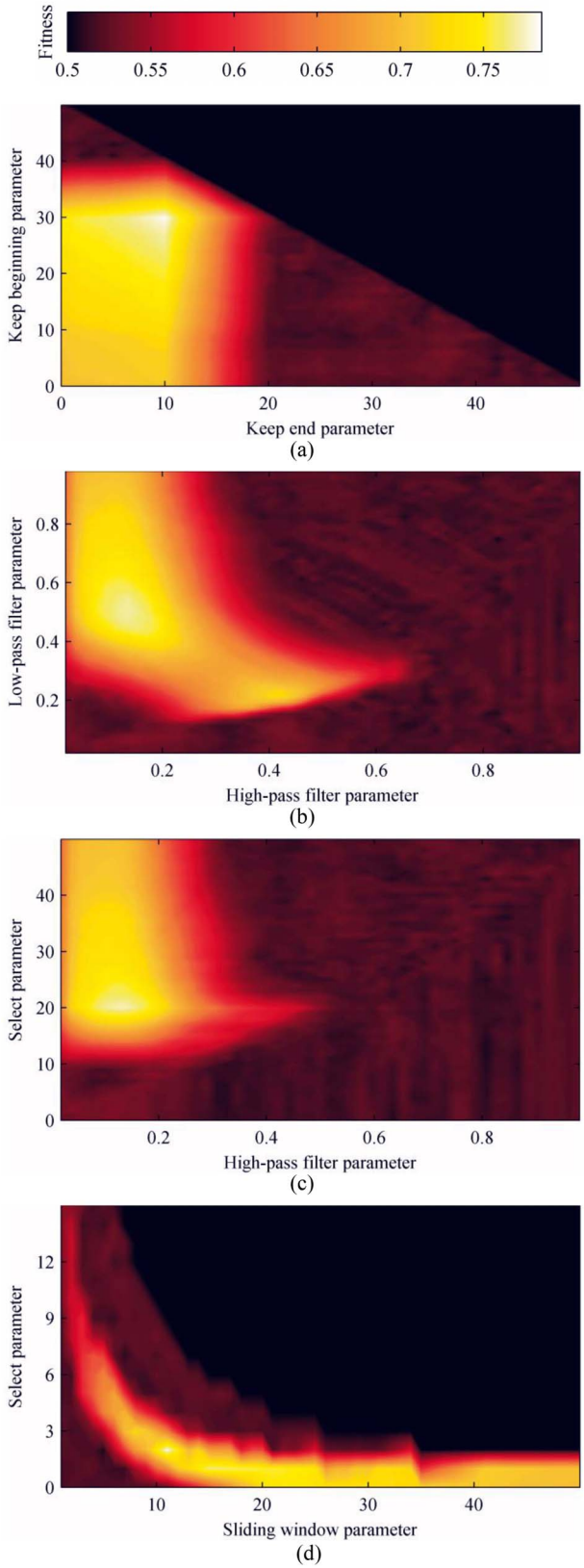

Fig. 9. Parameter surfaces for Autofead features (a)–(d) from Fig. 6(a)–(d).

VII. Conclusion

In this paper, a novel GP based method is proposed to automate feature design for numeric sequence classification

through a fully data-driven system called Autofead. Results are presented for multiple time series classification experiments. In each case, the approach produced optimal and near-optimal feature vectors in a compact, human-interpretable form without any prior knowledge relevant to the problem. The method includes a broad, general function set capable of developing a wide range of feature behaviors pertinent to a multitude of applications. Additionally, problem-specific domain knowledge can easily be incorporated within the function and terminal sets. Autofead has successfully demonstrated the ability to reduce large databases of numeric sequences into minimal-form features relevant to a specific decision-making process.

VIII. Future Work

Currently, the primary limitation in Autofead is its inability to compute features from multiple input sequences. The restricted solution structure only allows for a single input to each feature. Therefore, a large class of features derived from cross-correlation analysis and transfer function estimates are inaccessible in the present initial version. Future versions will circumvent the structural limitations by including a family of functions to perform convolutions and element-wise operations on multiple input sequences.

The efficiency of grading functions is another critical area of Autofead that needs improvement. The parameter optimization scheme relies on feature grades to evaluate relative performance of parameter values. Feature grading is performed more than any other operation in the search, so the computational cost must be minimal to keep run times reasonable. The current choice of classification accuracy from linear discriminant analysis has relatively low computational expense but cannot handle multimodal feature distributions requiring multiple decision boundaries. Gaussian Naïve-Bayes classification accuracy, area under the receiver operating characteristic's curve, the Kolmogorov–Smirnov two-sample test statistic, and a variety of statistical divergence measures have been identified as candidate grading functions.

Further benchmarking needs to be conducted on real world datasets with a variety of input sequence types. The UCR time series repository [32] provides a wide variety of numeric sequence classification datasets. An extensive comparison to the methods presented in [3] and [5] will be beneficial to identify the classes of problems for which Autofead outperforms distance-based classification and other approaches. Additionally, a parametric study of run parameters including population size, number of generations, and run repetitions is needed to optimize the configuration for general performance on a wide range of applications.

Lastly, an automated post-processing module for feature refinement is planned for development. After an Autofead search, the top solutions will go through a simplification and improvement process following the guidelines in Section IV-G. First, the features are checked for the patterns identified in the post-search refinement rules from Table IV. Next, functions with restricted designs such as the sliding windows function and filtering functions are replaced with unrestricted versions. Finally, a global reoptimization of each of the solution's parameters is performed.

References

[1] A. K. Jain, R. P. W. Duin, and J. Mao, "Statistical pattern recognition: A review," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 1, pp. 4–37, Jan. 2000.

[2] Z. Xing, J. Pei, and E. Keogh, "A brief survey on sequence classification," *ACM SIGKDD Explor. Newslett.*, vol. 12, no. 1, pp. 40–48, 2010.

[3] X. Wang *et al.*, "Experimental comparison of representation methods and distance measures for time series data," *Data Mining Knowl. Discov.*, vol. 26, no. 2, pp. 275–309, Mar. 2013.

[4] T. Fu, "A review on time series data mining," *Eng. Appl. Artif. Intell.*, vol. 24, no. 1, pp. 164–181, 2011.

[5] A. Bagnall, L. M. Davis, J. Hills, and J. Lines, "Transformation based ensembles for time series classification," in *Proc. SIAM Int. Conf. Data Mining (SDM)*, 2012, pp. 307–318.

[6] H. Guo, L. B. Jack, and A. K. Nandi, "Automated feature extraction using genetic programming for bearing condition monitoring," in *Proc. 2004 14th IEEE Signal Process. Soc. Workshop Mach. Learn. Signal Process.*, Sao Luis, Brazil, pp. 519–528.

[7] H. Deng, G. Runger, E. Tuv, and M. Vladimir, "A time series forest for classification and feature extraction," *Inform. Sci.*, vol. 239, pp. 142–153, 2013.

[8] B. D. Fulcher, M. A. Little, and N. S. Jones, "Highly comparative time-series analysis: The empirical structure of time series and their methods," *J. R. Soc. Interface*, vol. 10, no. 83, Apr. 2013, Art. ID 20130048.

[9] L. A. Levin, "Universal sequential search problems," *Probl. Peredachi Inf.*, vol. 9, no. 3, pp. 115–116, 1973.

[10] M. Hutter, "The fastest and shortest algorithm for all well-defined problems," *Int. J. Found. Comput. Sci.*, vol. 13, no. 3, pp. 431–443, Jun. 2002.

[11] J. Schmidhuber, "Optimal ordered problem solver," *Mach. Learn.*, vol. 54, no. 3, pp. 211–254, Mar. 2004.

[12] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, 1st ed. Cambridge, MA, USA: MIT Press, 1992.

[13] D. Andre, "Automatically defined features: The simultaneous evolution of 2-dimensional feature detectors and an algorithm for using them," in *Advances in Genetic Programming*. Cambridge, MA, USA: MIT Press, 1994, pp. 477–494.

[14] N. R. Harvey *et al.*, "Image feature extraction: GENIE vs conventional supervised classification techniques," *IEEE Trans. Geosci. Remote Sens.*, vol. 40, no. 2, pp. 393–404, Feb. 2002.

[15] G. R. Raidl, "A hybrid GP approach for numerically robust symbolic regression," in *Proc. 3rd Annu. Conf. Genet. Program.*, Madison, WI, USA, 1998, pp. 323–328.

[16] P. G. Espejo, S. Ventura, and F. Herrera, "A survey on the application of genetic programming to classification," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 40, no. 2, pp. 121–144, Mar. 2010.

[17] R. Poli, W. B. Langdon, and N. F. McPhee, *A Field Guide to Genetic Programming*. Barking, U.K.: Lulu Enterprises, 2008.

[18] K. C. Sharman, A. I. Alcazar, and Y. Li, "Evolving signal processing algorithms by genetic programming," in *Proc. First Int. Conf. Genet. Algorith. Eng. Syst. Innov. Appl. (GALESIA)*, Sheffield, U.K., 1995, pp. 473–480.

[19] D. Eads *et al.*, "Genetic algorithms and support vector machines for time series classification," *Proc. SPIE*, vol. 4787, Dec. 2002, Art. ID 75.

[20] D. R. Eads *et al.*, "A multimodal approach to feature extraction for image and signal learning problems," *Proc. SPIE*, vol. 5200, pp. 79–90, Dec. 2004.

[21] K. L. Holladay and K. A. Robbins, "Evolution of signal processing algorithms using vector based genetic programming," in *Proc. 15th Int. Conf. Digital Signal Process.*, Cardiff, U.K., 2007, pp. 503–506.

[22] K. L. Holladay, K. Robbins, and J. Von Ronne, "FIFTH TM: A stack based GP language for vector processing," in *Proc. 10th Eur. Conf. Genet. Program. (EuroGP)*, vol. 4445. Valencia, Spain, 2007, pp. 102–113.

[23] A. Teller and M. Veloso, "Program evolution for data mining," *Int. J. Expert Syst. Res. Appl.*, vol. 8, pp. 213–236, Jan. 1995.
[24] A. Teller and M. Veloso, "PADO: A new learning architecture for object recognition," in *Symbolic Visual Learning*. Oxford, U.K.: Oxford Univ. Press, 1996, pp. 81–116.
[25] F. Xie, A. Song, and V. Ciesielski, "Event detection in time series by genetic programming," in *Proc. 2012 IEEE Congr. Evol. Comput. (CEC)*, Brisbane, QLD, Australia, pp. 1–8.
[26] D. Y. Harvey and M. D. Todd, "Automated selection of damage detection features by genetic programming," in *Proc. Int. Modal Anal. Conf. XXXI*, Garden Grove, CA, USA, 2013.
[27] D. Y. Harvey and M. D. Todd, "Automated extraction of damage features through genetic programming," *Proc. SPIE*, vol. 8695, Apr. 2013, Art. ID 86950J.
[28] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2009.
[29] T. Homma and A. Saltelli, "Importance measures in global sensitivity analysis of nonlinear models," *Reliab. Eng. Syst. Safety*, vol. 52, no. 1, pp. 1–17, Apr. 1996.
[30] D. W. Scott, *Multivariate Density Estimation: Theory, Practice, and Visualization*, 1st ed. Hoboken, NJ, USA: Wiley, 2009.
[31] J. R. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs*, 1st ed. Cambridge, MA, USA: MIT Press, 1994.
[32] E. Keogh *et al*. (2011). *The UCR Time Series Classification/Clustering Homepage* [Online]. Available: http://www.cs.ucr.edu/∼eamonn/time_series_data/

Dustin Y. Harvey received the B.S. degree in mechanical engineering from Rose-Hulman Institute of Technology, Terre Haute, IN, USA, in 2009. He is currently working toward the Ph.D. degree in structural engineering from University of California San Diego, San Diego, CA, USA.

His research interests include structural health monitoring, machine learning, evolutionary computation, and digital signal processing.

Mr. Harvey was awarded research fellowships by the National Science Foundation and National Defense Science and Engineering Graduate Fellowship programs in 2011.

Michael D. Todd received the B.S., M.S., and Ph.D. degrees from the Department of Mechanical Engineering and Materials Science, Duke University, Durham, NC, USA, in 1992, 1993, and 1996, respectively.

He joined the faculty of the Structural Engineering Department, University of California San Diego, San Diego, CA, USA, in 2003, where he is a Professor and Vice Chair. His research interests include time series models for structural health monitoring applications, statistical signal processing, uncertainty quantification, and nonlinear dynamical processes in structural and mechanical systems.