Unsupervised Temporal Rule Mining with Genetic Programming and Specialized Hardware

Pål Sætrom and Magnus Lie Hetland

***Abstract*— Rule mining is the practice of discovering interesting and unexpected rules from large data sets. Depending on the exact problem formulation, this may be a very complicated problem. Existing methods typically make strong simplifying assumptions about the form of the rules, and limit the measure of rule quality to simple properties, such as confidence. Because confidence in itself is not a good indicator of how interesting a rule is to the user, the mined rules are typically sorted according to some secondary interestingness measure. In this paper we present a rule mining method that is based on genetic programming. Because we use specialized pattern matching hardware to evaluate each rule, our method supports a very wide range of rule formats, and can use any reasonable fitness measure. We develop a fitness measure that is well-suited for our method, and give empirical results of applying the method to synthetic and real-world data sets.**

***Index Terms*— Data mining, rule discovery, time series, genetic programming, pattern matching hardware.**

I. Introduction

TEMPORAL sequence data are ubiquitous in many fields, and lately there has been an increase in the interest for methods that can extract useful information from large sequence databases [1]. One specific problem is that of rule mining: Extracting interesting and unexpected regularities, or rules, from the data.

The rule mining problem consists of finding patterns that satisfy certain criteria in a sequence database. These patterns (or rules) may have a form such as "if we encounter element x, then we will encounter element y within t time units." Here, x is the *antecedent*, and y is the *consequent*. The quality of such rules may be measured by how frequently they occur (support), their predictive power (confidence), and by measures of how interesting they are, described numerically by so-called *interestingness measures*.

The general approach taken by several authors (for example, [2], [3]) is to scan the sequential data and to count every occurrence of a legal rule, as well as the occurrences of every legal antecedent and consequent. This counting makes it possible to calculate the frequencies and confidences of each rule.

However, this approach limits the format of the rules. Even moderately complex rule formats will make the task of counting all occurrences unfeasible. Also, existing methods (such as [4]) have focused on finding rules that are frequent and have high confidence, and only subsequently have sorted the resulting rules using an interestingness measure, which is meant to measure the true quality of the rule.

The approach taken in this paper is based on the method of [5]: with the aid of specialized pattern matching hardware we find sequential rules using genetic programming. In [5] the task was one of simple sequence learning and prediction. In this paper we show that by using general interestingness measures as fitness functions, our method can be used to mine unknown rules of relatively high quality.

A. Related Work

Previous attempts at solving the problem of mining predictive rules from time series can loosely be partitioned into two types. In the first type, supervised methods, the rule target is known and used as an input to the mining algorithm. Typically, this can be specific events in (or possibly extrinsic to), the time series. Thus the goal is to generate rules for predicting these events based on the data available before the event occurred. The papers [5], [6], [7], [8] fall in this category. All of these use some form of evolutionary computation; [5] uses genetic programming, while the others use genetic algorithms.

In the second type, unsupervised methods, the only input to the rule mining algorithm is the time series itself. The goal is to automatically extract informative rules from the series. In most cases this means that the rules should have some level of preciseness, be representative of the data, easy to interpret, and interesting (that is, novel, surprising, useful, and so on), to a human expert [9]. This is the approach we take in this paper. Of the existing attempts to tackle this problem, many rely on scanning the data and counting the occurrence of every legal antecedent and consequent (for example, [2], [3], [10]). The rules are then ranked according to some measure of interestingness. This approach does, however, place some limitations on the rule format in order to make the task of counting all occurrences feasible. Others have focused on specific mining problems, such as detecting unusual movements [11], or finding unusual temporal patterns in market basket data [12].

Unlike these approaches, we try to tackle the core problem directly, that is, mining interesting rules. This is done by defining some formal interestingness measure and using genetic programming to search the rule space for the most interesting rules. Thus, unlike other methods, the interestingness measure is used directly in the mining process and not as a post-processing ranking function. This allows for a much more flexible rule format than the existing method.

Pål Sætrom is employed at Interagon AS, Medisinsk-teknisk senter, Olav Kyrres gt. 3, NO–7489 Trondheim, Norway (email: paalsat@interagon.com, fax: +47 45594458).

Magnus Lie Hetland is at the Department of Computer and Information Science, Norwegian University of Science and Technology, Sem Sælands vei 9, NO–7491 Trondheim, Norway (email: mlh@idi.ntnu.no).

B. Structure of This Paper

The rest of this paper is structured as follows: Section II describes the preprocessing scheme used to discretize the time series data used in the experiments, Section III describes how genetic programming is used to evolve temporal rules, Section IV describes in detail how rules are evaluated, Section V describes our experiments and empirical results, and finally Section VI summarizes and concludes the paper.

II. Preprocessing

The rule mining strategy presented in this paper works on discrete sequences of symbols. To transform the time series data of our empirical application to such a symbolic sequence, we use a simple method used, among other places, in [1]. It extracts all windows of width w, and for each such window a real-valued feature is calculated. This feature may be, for example, the average value or signal to noise ratio. In our experiments we have used the slope of a line fitted to the data points of the window with linear regression.

After such a feature sequence has been constructed, a copy is made, which is sorted and divided into a (approximately) equal-sized intervals. Each interval is assigned an integer from 1 to a, and the limits of the intervals are used to classify the values in the original feature-sequence. By following this procedure, we are guaranteed that the symbols (that is, the integers, which easily map to characters in some alphabet) all have approximately the same frequency.

Our experiments require us to use both training sets, validation sets (for early stopping, or model selection), and test sets. Since the discretization process uses information about "the future" when classifying a single point, it cannot be used directly on the validation and testing sets. Instead, the normal procedure was used on the training set, and the limits found there were used when classifying the features of the validation and testing sets.

Note that by allowing the windows to overlap when classifying the positions we avoid unneeded data reduction, but we also introduce spurious correlations between adjacent symbols. For most time series, two windows that overlap in $w-1$ positions will be quite likely to have similar feature values, which means they are more likely to be assigned the same symbol. How we deal with this problem is described in Section IV-A.

This discretization method is by no means unique. In [13] a method is described, which uses the slope and signal to noise ratio for segments of the series. Other usable methods of discretization include those used to simplify time series for indexing purposes. See [14] for a survey.

III. Evolving Rules

The evolutionary computation strategy used in this paper is genetic programming, as described in [15]. The algorithm uses subtree swapping crossover, tree generating mutation and reproduction as genetic operators. Individuals are chosen for participation in new generations using tournament selection. Each individual in the population is a program tree, representing an expression in some formal language. In our experiments, we use several such languages, each representing a format for the rules we wish to discover. Expressions in the chosen rule languages may be evaluated by the specialized pattern matching hardware described in Section IV-B, and rule fitness is calculated by searching the time series data for rule occurrences.

A. Rule Languages

The basic rule format that will be used throughout this paper is the simple and well known: "If *antecedent* then *consequent* within T time units". In its simplest form, as used in [4], both the antecedent and consequent are single symbols in the discretized alphabet, A, while T is a constant. This results in a rule language of the form: $x \overset{T}{\Rightarrow} y$ for $x, y \in A$.

Several extensions to this simple language are possible and have been investigated by others:

1) *Sequential patterns* [3]: If x_1 and x_2 and $\ldots$ and x_n occur in a window of width w, then y occurs within T time units. Here $x_i, i \in \{1, n\}$ and y are symbols in A.
2) *Regular expressions/episode rules* [2]: If the sequence $x_1, x_2, \ldots, x_n$ can be found within in a window of width w, then y occurs within T time units. Here, $x_i, i \in 1 \ldots n$ can either be a symbol in A, or a set of symbols $X \subseteq A$ for which any $x \in X$ can be a legal match; y is a symbol in A. These rules are a simple form of regular expressions; for example, the antecedent in the episode rule $\mathrm{a}, \{\mathrm{c}, \mathrm{d}\} \overset{t}{\Rightarrow} \mathrm{y}$ can be written as $\mathrm{a}.^{*}(\mathrm{c}|\mathrm{d})$, with the added requirement that the maximum length of the string matched is w.

Note that all of these rule languages share the same basic format. What differs is how the antecedent is defined, that is, the language used to generate the antecedent. In general, all rules of this type can be described by the three parameters: the antecedent language, L_a, the consequent language, L_c, and the maximum temporal distance, T.

Most previously investigated rule languages make a distinction between L_a and L_c: L_a varies in complexity, while L_c usually is a single character from A.[1] In the following no such limitation will be made: unless otherwise noted $L_a = L_c$.

B. Rule Representation

The mining algorithm works by using genetic programming to search the space of possible rules defined by L_a, L_c and T. More specifically, each individual in the population is a syntax tree in the language $L_a \overset{T}{\Rightarrow} L_c$. This is implemented by using three separate branches; One branch for each of L_a, L_c, and T.

In the antecedent and consequent branches, the internal nodes in the parse tree are the syntactical nodes necessary for representing expressions in the corresponding languages. If for example, the considered language is regular expressions, the syntactical nodes needed are *union*, *concatenation* and *Kleene*

[1]One notable exception is [16], which defines a rule language where both L_a and L_c are sequences of characters separated by wildcards, that is, episode rules without parallel episodes.

closure. The leaf nodes in these branches are the symbols from the antecedent and consequent alphabets (Σ_a and Σ_c).

The maximum distance branch defines the maximum distance t of the rule. This branch is constructed by using arithmetic functions (typically $+$ and $-$) as internal nodes, and random integer constants as leaf nodes. The final distance t is found by computing the result of the arithmetic expression r_T, and using the residue of r_T modulo $T+1$.

C. Confidence, Support, and Interestingness

Given a rule $R = R_a \overset{t}{\Rightarrow} R_c$ in the rule language $L_a \overset{T}{\Rightarrow} L_c$ (such that $t \leq T$) and a discretized sequence $S = (a_1, a_2, \ldots, a_n)$, the frequency $F(R_a)$ of the antecedent is the number of occurrences of R_a in S. This can be formalized as

$$F(R_a) = |\{i \mid H(R_a, S, i)\}|, \tag{1}$$

where $H(R_a, S, i)$ is a hit predicate, which is true if R_a occurs at position i in S and false otherwise. The relative frequency, $f(R_a)$, is simply $F(R_a)/n$, where n is the length of S.

The *support* of a rule is defined as:

$$F(R_a, R_c, t) = |\{i \mid H(R_a, S, i) \wedge H(R_c, S, j) \wedge i{+}1 \leq j \leq i{+}t\}| \tag{2}$$

This is the number of matches of R_a that are followed by at least one match of R_c within t time units.

The *confidence* of a rule is defined as:

$$c(R) = \frac{F(R_a, R_c, t)}{F(R_a)} \tag{3}$$

In most existing methods, candidate rules with high confidence and support are selected. This approach usually generates a lot of rules, many of which may not be particularly interesting. As an aid in investigating these rules, *interestingness measures* have been developed (see [17] for a survey). These measures may, for instance, be used to sort the rules in descending order of interest.

One measure of interestingness that has proved to be robust for identifying surprising rules is the J-measure ([18]). This is defined as:

$$J(R_c^t, R_a) = p(R_a) \cdot \Big(p(R_c^t|R_a) \log_2 \frac{p(R_c^t|R_a)}{p(R_c^t)} + (1 - p(R_c^t|R_a)) \log_2 \frac{1 - p(R_c^t|R_a)}{1 - p(R_c^t)}\Big) \tag{4}$$

Here, $p(R_a)$ is the probability of $H(R_a, S, i)$ being true at a random location i in S. $p(R_c^t)$ is the probability of $H(R_c, S, i)$ being true for at least one index i in a randomly chosen window of width t. Finally, $p(R_c^t|R_a)$ is the probability of $H(R_c, S, i)$ being true at for at least one index i in a randomly chosen window of width t, given that $H(R_a, S, j)$ is true and that j is the position immediately before the chosen window. The J-measure combines a bias toward more frequently occurring rules (the first term, $p(R_a)$), with the degree of surprise in going from a prior probability $p(R_c^t)$ to a posterior probability $p(R_c^t|R_a)$ (the second term, also known as the cross-entropy).

An alternative to the J-measure is the Piatetsky-Shapiro rule-interest measure, ***RI***, described in [19]. This measure quantifies the degree of correlation between the antecedent and consequent. Rules with high correlation are then seen as more interesting. In the context of sequence rules, the rule interest function can be defined as[2]

$$\textbf{\textit{RI}}(R_c^t, R_a) = p(R_c^t|R_a) - p(R_a) \cdot p(R_c^t), \tag{5}$$

with the same definitions for the probabilities as for the J-measure. As can be seen from (5), if R_c^t and R_a are statistically independent then ***RI*** $= 0$. If $H(R_c, S, i)$ is more (less) frequently true in a window of length t when $H(R_a, S, i)$ is true and i is the position immediately to the left of the window, then ***RI*** > 0 (***RI*** < 0).

IV. Rule Evaluation

Consider the problem of mining interesting rules from a sequence S, given a rule language L, defined by (L_a, L_c, T), and an interestingness function f. In order to use genetic programming to perform this rule mining, we must be able to compute the value of f for every possible rule in L. In the case that f is one of either J or RI from Section III-C, this amounts to estimating the probabilities $p(R_a)$, $p(R_c^t)$ and $p(R_c^t|R_a)$. In the interest of simplicity, we will use the maximum likelihood estimates for these probabilities. That is, for a given rule $R = R_a \overset{t}{\Rightarrow} R_c$, the estimators are:

$$\widehat{p}(R_a) = f(R_a) \tag{6}$$
$$\widehat{p}(R_c^t) = f(R_c^t) \tag{7}$$
$$\widehat{p}(R_c^t|R_a) = c(R) \tag{8}$$

This amounts to counting the following:

- The number of occurrences of R_a in S (from the definition of $f(R_a)$.)
- The number of windows of length t where $H(R_c, S, i)$ is false at every position (as $p(R_c^t) = 1 - p(\neg R_c^t)$, where $p(\neg R_c^t)$ is the probability that $H(R_c, S, i)$ is false for all positions in a random window of length t in S.)
- The number of hits from R_a where $H(R_c, S, i)$ is true at least once within time t.

A. Handling Correlations Caused by the Discretization Method

The discretization process described in Section II introduces correlations between consecutive symbols in the discretized sequence. This results in that rules with low distances t will have high confidence. Since these rules are artifacts of the discretization process, we do not consider them interesting.

To account for these induced correlations, the number of occurrences of the rule $R = R_a \overset{t}{\Rightarrow} R_c$ in a sequence S, discretized with a window length of w, is defined as ([4]):

$$F(R_a, R_c, t) = |\{i \mid H(R_a, S, i) \wedge H(R_c, S, j) \wedge i{+}w \leq j \leq i{+}w{+}t{-}1\}| \tag{9}$$

[2]Note that this is an adaptation of the definition in [19], where the function is defined for simple classification rules where R_a and R_c are single symbols.

Thus, only occurrences of R_a that are followed by a hit from R_c after $w-1$ units of time are counted.[3]

B. Counting Hits

One important feature of our method is the relative lack of restrictions placed on the allowed rule languages. To allow for such flexibility, we cannot perform any general occurrence counting—the probabilities of each rule must be estimated individually, in the course of calculating their fitness. Each such estimation requires a complete pass through the data.

To speed up these calculations to the level where they are usable as components in a fitness function, we use a specialized search chip ([20], [21]) for hit counting. This pattern matching chip (PMC), is able to search 100 MB/s and can handle from 1 to 64 parallel queries, depending on query complexity.[4] The queries are specified in a special-purpose query language ([22]). This language supports such language features as regular expressions, latency (distance), Boolean combinations, and alpha-numerical comparisons.

As described in Section IV, the process of evaluating a rule consists of counting the occurrences of three different patterns. The PMC can be used for this purpose in the following way:

The number of occurrences of R_a in S: This amounts to counting all hits of R_a in S.

The number of windows of length t where $H(R_c, S, i)$ is false at every position: This can be found by looping through the hits $H_c = \{h_1, \ldots, h_n\} = \{i \mid H(R_c, S, i)\}$ of R_c in S and incrementing a counter by $h_i - h_{i-1} - t$ if $h_i - h_{i-1} > t$ ($h_0 = 0$).

The number of hits from R_a where $H(R_c, S, i)$ is true at least once within time t: This proved difficult to calculate as this expression cannot be directly evaluated by the PMC. The PMC is, however, capable of finding all occurrences where $H(R_c, S, i)$ is true and is preceded by a hit from R_a at a maximum distance of t. This process can be summarized by the *pattern before* operator, with the syntax R_a *PBEFORE*(t) R_c.

As long as the length of the substring matched by R_a and R_c is 1, $F(R_a, R_c, t)$ can be evaluated by using the *PBEFORE* operator in the following way: Construct from S the reverse sequence S^r. $F(R_a, R_c, t)$ is given by counting the number of hits from the expression R_c^r *PBEFORE*(t) R_a^r in S^r. If, however, this is not the case (that is, either R_a or R_c does not match a single symbol), this procedure cannot be used. There are several reasons why it fails, but the most important reason is that the distances are distorted.

Consider, for instance, the rule where $R_a = \mathrm{ab}$, $R_c = \mathrm{c}$ and $t = 1$, and the sequence $S = (\mathrm{a}, \mathrm{b}, \mathrm{c})$. In order for R_a to match the same sub-sequences in S^r as in S, it must be reversed. It should be evident that in this case the reverse of R_a is $R_a^r = \mathrm{ba}$. Searching for R_a^r in the reverse sequence, $S^r = \{\mathrm{c}, \mathrm{b}, \mathrm{a}\}$, will result in a hit at position 3, while R_c will report a hit at position 1. So while the distance between hits from the antecedent and consequent is 1 in S, it has increased to 2 in S^r. Although in this case it is trivial to account for the distance distortion, this is not so in the general case (consider, for instance, $R_a = (\mathrm{a}|\mathrm{bc})$).

These problems can be solved by using another method for evaluating $F(R_a, R_c, t)$: Store the hit locations from R_a and R_c in two arrays sorted by the hit position (this is trivial when using the PMC, as hits are reported sequentially in an array). Iterate through the antecedent array and increment a counter whenever a hit in this array has a hit in the consequent array that is within the desired distance. This can be done in $O(n_a + n_c)$ time, where n_a and n_c is the number of hits from the antecedent and consequent, respectively (or, in other words, in $O(n)$ time, where n is the number of symbols in S, that is, the worst case when R_a matches every position in S.)

Note that both methods can be used for evaluating the modified frequency function from Section IV-A. The only added requirement when evaluating this function is that there must be least $w-1$ symbols between hits from R_a and R_c. The *PBEFORE* method solves this by adding $w-1$ wild-cards (that is, symbols matching any symbol) at the start of R_a^r or at the end of R_c^r. For the hit processing method, this amounts to only considering hits from the consequent that have at least a distance of $w-1$ symbols from a hit from the antecedent.

V. Experimental Results

In our experiments we used the following five rule languages:

- L_1 Single symbols.
- L_2 Single symbols and concatenations of single symbols.
- L_3 Sequential patterns.
- L_4 Regular expressions with the limitation that skips and repetitions cannot be recursive (for example, expressions of the type: $a(b^*c)^*d$, $a(b?c)?d$ and $a(b?c)^*d$ are not allowed.)
- L_5 L_4 with the addition of alpha-numerical comparisons and Boolean operations (for example, rules like $\geq$ alpha & $\leq$ beta, matching all strings that are alpha-numerically between *alpha* and *beta*.)

As can be seen from the description, only rules generated from L_1 can be evaluated using the *PBEFORE* method. (Recall that this method can only be used when the antecedent and consequent both match only a single symbol.)

The system was first tested on two different synthetic datasets with known rules embedded in the sequence. Then it was tested on a data set containing ECG measurements, taken from the UCR Time Series Data Mining Archive [23]. All our results were generated by running the genetic programming system with a population size 5000 for a maximum of 20 generations. Crossover, mutation, and reproduction were used with probabilities 0.9, 0.01, and 0.09, respectively, while the tournament size was 5.

For each data set, the genetic programming algorithm was run several times, with different rule languages and interestingness measures. In addition to the J-measure and the rule interest function *RI*, confidence ($c(R)$) and confidence times support ($c(R) \cdot F(R_a, R_c, t)$) were used as interestingness measures.

[3]Note that this differs from the definition in [4], where the lower range was defined as $i + w + 1$. However, in the limiting case, where $w = 1$ (that is, a single time point), this formula should be equal to the original frequency definition in (2).

[4]The prototype used in these experiments searches 33 MB/s and handles 1 to 4 parallel queries.

A. Synthetic Data

The synthetic data were constructed by repeatedly drawing symbols from a subset of the lowercase Latin alphabet ($a-y$) with uniform probability. The symbol z, used for representing the consequent, was inserted into the sequence when some predefined antecedent pattern was found.

Two different antecedent types were used:

1) The regular expression $o[\wedge o \wedge n]^{*}n$.
2) The symbols a, b, c, d and e occurring in any order within a window of width 10.

The two sets consisted of 100 kB sequence data with about 2000 and 160 occurrences of antecedent type 1 and 2, respectively.

Table I summarizes some typical results produced by the J-measure and *RI* function on the synthetic datasets. In addition, the table presents some typical results from using the confidence and confidence times support as interesting measures. The rule notation is explained in the appendix. Note that the languages $L_5 \overset{1}{\Rightarrow} L_2$ and $L_3 \overset{1}{\Rightarrow} L_1$ were used for generating the rules from dataset 1 and 2, respectively.

As can be seen from this table, both the confidence and rule interest measures produce rules having high confidence but minimal support. Thus neither of these measures are particularly useful as a fitness function for mining interesting rules (unless spurious or "rare" rules are desired). Using confidence times support as a fitness measure rectifies some of these problems. The system is able to partially recover the embedded pattern from set 1. It is, however, unable to recover the pattern from set 2, as its combined support and confidence ($0.0012 \cdot 0.62 = 0.000744$) is lower than that of the random pattern detected ($0.041 \cdot 0.041 = 0.001681$). Another serious shortcoming with this fitness measure may be observed in sets having an uneven symbol distribution. There the rules generated most often involve the most frequently occurring antecedent, as this determines the frequency of the rule, and thus the rule support (data not shown).

B. Modifying the J-measure

Some of the initial results generated by mining the different datasets using the J-measure had a confidence far below 50% (data not shown). This inspired the following modification to the fitness measure: Multiply the J-measure with a confidence correcting function $F(c(R))$. Recall that $c(R)$ is the rule confidence. $F(c(R))$ should be a monotonically increasing function that is close to 1 for values of $c(R)$ larger than some limit c_{min} and close to 0 for values below c_{min}. One function that satisfies these requirements is the sigmoid function:

$$F(c(R)) = \frac{1}{1+e^{-(c(R)-c_{min})\cdot g}} \tag{10}$$

Here g is a parameter regulating how sharp the cutoff at c_{min} should be. In the following sections, the value $g = 20$ was used.

Using the modified J-measure as fitness function, the system was able to fully recover the rule embedded in set 2. With this setup, however, the system was unable to fully recover the rule from set 1. Instead, an approximation was found, using the IQL *PBEFORE*(t) operator. Table II lists two of the expressions generated, along with the rule languages used in the generation process.

TABLE II
SUMMARY OF RESULTS ON SYNTHETIC DATASET USING THE MODIFIED J-MEASURE.

Type	Language	Rule
1	$L_5 \overset{1}{\Rightarrow} L_2$	o $\overset{27}{\longleftarrow}$ n $\overset{1}{\Rightarrow}$ z
2	$L_3 \overset{1}{\Rightarrow} L_1$	$\{a \wedge b \wedge c \wedge d \wedge e : 9\} \overset{1}{\Rightarrow} z$

C. Real-World Data

The system was tested on the ECG dataset from the UCR Time Series Data Mining Archive [23]. The series was split into 10 partially overlapping folds, and each fold was then further divided into a training set, and smaller validation and test sets. The validation set was used for early stopping (model selection). Each training set was then discretized using the procedure from Section II with a window size of 2 and alphabet size of 15. The corresponding validation and test set were then discretized using the limits and symbols from the training set. The 10 folds were then mined using 4 different rule languages. Some of the results are presented in Table III. Note that the results listed in this table are the results produced by using early stopping, that is, those among the "best of generation" results having the highest fitness when applied to the validation set. Also note that the modified support from Section IV-A with $w = 2$ was used.

As can be seen from this table, some rules generated by the system were both highly complex and had an accuracy close to 1, in both the training and test set. Further analysis revealed that these rules actually exploited a feature in the underlying pattern matching hardware: When occurring, both antecedent and consequent match the same pattern, but the hardware reports that the antecedent occurs one or two bytes earlier than what is the actual case.

As a comparison, the other rules generated were fairly simple. This is probably due to the highly regular pattern in the sequence. Thus, to circumvent the problem of complex but invalid rules, and to test the system on a more difficult problem, the system was run on the ECG data with the minimum distance parameter w set to 10. Table IV lists two of the rules generated from the $L_5 \overset{10}{\Rightarrow} L_2$ language.

Figure 1 shows a plot of a subsequence of the ECG set. The figure also shows the hits for the antecedent of the second rule from Table IV in the sequence.

As can be seen, the system has successfully generated a rule for identifying the highly regular pattern in the ECG signal.

D. Random Data

The rule generation method was also tested on a random set without any embedded rules. In this set all characters from the $a-z$ alphabet were drawn with uniform probability. Thus no patterns should be prevalent in the data. For mining this set, the four fitness measures from Table I were again used, in addition to the modified J-measure. The results from

TABLE I

TYPICAL RESULTS PRODUCED BY DIFFERENT INTERESTING MEASURES ON THE SYNTHETIC DATASETS.

Set	Fitness measure	Rule	Supp.	Conf.	J-mea.	*RI*
1	J-measure	g $\xleftarrow{184}$ n $\overset{1}{\Rightarrow}$ z	0.019	0.51	0.072	0.51
1	Rule interest	ywvh | wvhy $\overset{1}{\Rightarrow}$ n	10^{-5}	1.0	$4.7 \cdot 10{-}5$	1.0
1	Confidence	fieg | egif $\overset{1}{\Rightarrow}$ k	10^{-5}	1.0	$4.7 \cdot 10{-}5$	1.0
1	Conf. · Supp.	n $\overset{1}{\Rightarrow}$ z	0.019	0.51	0.072	0.50
2	J-measure	$\{$a $\wedge$ b $\wedge$ c $\wedge$ d $\wedge$ e : 9$\}$ $\overset{1}{\Rightarrow}$ z	0.0012	0.62	0.0099	0.63
2	Rule interest	$\{$z $\wedge$ k $\wedge$ m $\wedge$ s : 4$\}$ $\overset{1}{\Rightarrow}$ x	10^{-5}	1.0	$4.7 \cdot 10{-}5$	1.0
2	Confidence	$\{$s $\wedge$ m $\wedge$ z : 3$\}$ $\overset{1}{\Rightarrow}$ k	10^{-5}	1.0	$4.6 \cdot 10{-}5$	1.0
2	Conf. · Supp.	$\{$h $\wedge$ g : 151$\}$ $\overset{1}{\Rightarrow}$ r	0.041	0.041	0.00	$2.0 \cdot 10{-}4$

TABLE III

EARLY STOPPING RESULTS ON ECG DATA SET EVALUATED ON THE TEST SET.

Language	Rule	Supp.	Conf.	J-mea.	*RI*
$L_2 \overset{10}{\Rightarrow} L_2$	b $\overset{1}{\Rightarrow}$ b	0.042	0.68	0.12	0.68
$L_4 \overset{10}{\Rightarrow} L_2$	n$^+$fhjjcg$^+$jc | n$^+$fhjng$^+$jc | n$^+$fhng$^+$jc | ac | oc | o $\overset{1}{\Rightarrow}$ o	0.074	1.0	0.28	0.99
$L_3 \overset{10}{\Rightarrow} L_1$	$\{$o $\wedge$ c $\wedge$ g $\wedge$ i $\wedge$ e : 57$\}$ $\overset{9}{\Rightarrow}$ o	0.065	0.93	0.18	0.92
$L_5 \overset{10}{\Rightarrow} L_2$	o $\overset{1}{\Rightarrow}$ o	0.061	0.84	0.19	0.84
$L_5 \overset{10}{\Rightarrow} L_2$	a $\overset{1}{\Rightarrow}$ a	0.031	0.83	0.12	0.83

TABLE IV

RESULTS FROM THE ECG SET WITH $w = 10$ EVALUATED ON THE TEST SET.

Rule	Supp.	Conf.	J-mea.	*RI*
e $\xleftarrow{88}$ ($\geq$ lkkl) $\overset{9}{\Rightarrow}$ m	0.11	0.64	0.10	0.60
($\geq$ klhjlj)((a $\xleftarrow{52}$ ($\geq$ lf)) | cnf | bnf) | ((a $\xleftarrow{52}$ ($\geq$ lf)) | cnf | bnf)($\geq$ klhjlj) $\overset{9}{\Rightarrow}$ m	0.12	0.86	0.19	0.83

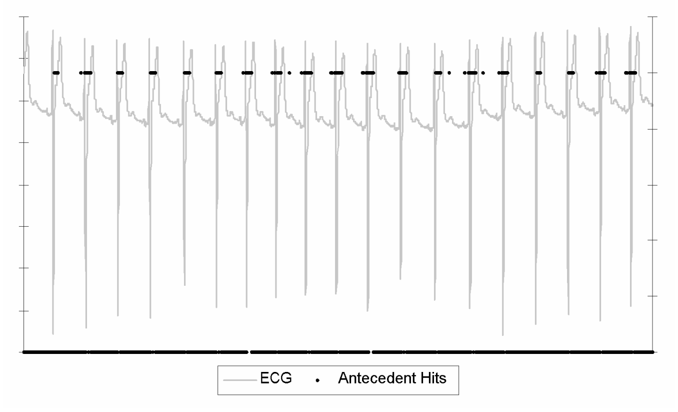

Fig. 1. Hit locations of antecedent in ECG sequence.

these tests confirm the observations from the runs on the synthetic data (see Section V-A), concerning the different fitness measures (data not shown). In addition, the same effect as observed on the ECG data concerning the hardware feature exploitation was again observed in this data set (data not shown).

Several rule languages were tried. This showed that certain language combinations for the antecedent and consequent may result in spurious rules that fit the random data (including the separate test set) well. For example, the language $L_5 \overset{10}{\Rightarrow} L_5$ (with $w = 10$), generated the following rule: !(e($\leq$ i)yk | kye($\leq$ i)) $\overset{1}{\Rightarrow}$!(($\leq$ i)($>$ i)). This rule had support and confidence of ≈ 1.0, and J-measure and Rule Interest measure of 0.37 and 0.23, respectively, when tested on a random set different from the training set. The intuition behind this is that by letting both the antecedent and the consequent be sufficiently general, it is possible to achieve 100% in both confidence and support. In general, however, fixing the consequent (that is, restricting it to be generated from either L_1 or L_2), prevents this from occurring.

VI. SUMMARY AND CONCLUSIONS

In this paper we have examined a novel method for unsupervised mining of rules in time series data. Unlike previous methods, the method places few constraints on the rule representation and the quality measure that is being optimized.

The method works by evolving rules through genetic programming, and uses specialized hardware to calculate the fitness (interestingness) of each candidate rule.

For our experiments, we used synthetic data, a discretized real-world dataset (ECG), and a random data set. We ran experiments using several different rule languages of differing complexity, including support for regular expressions. To our knowledge, no existing methods can accommodate similarly flexible rule formats. The method was able to recover or approximate the rules embedded in the synthetic sequence. In addition, it was able to produce rules recognizing the periodicity in the ECG sequence.

The method described in this paper is still new, and there is still much research to be done in examining various rule formats and interestingness measures. The primary fitness measure used in our experiments is based on the J-measure, which has been found to be robust and useful in ranking rules, but several other interestingness measures exist, and many of these may be useful as fitness measures when evolving rules.

Appendix
Rule Language Syntax

This appendix describes the notation used in the rules presented in Section V.

$R*$: The Kleene closure operator. Signifies that the R is repeated 0 or more times.

$R?$: The optional operator: The R is optional and can be skipped.

$\{x_1 \wedge \ldots \wedge x_n : w\}$: Sequential patterns. Signifies that characters x_1 to x_n will be found in a window consisting of w characters.

$R_i \mid R_j$: This is the alternative operator, meaning that either sub-expression R_i or R_j should match.

$!R$: The expression gives a match whenever R does not (that is, the negation of R).

$R_i \xleftarrow{t} R_j$: Shorthand for the $PBEFORE(t)$ operator. Reports a match whenever R_j reports a match and R_i reported a match at most t symbols before.

$\geq R$: Reports a match whenever the current substring is alpha-numerically (lexically) greater or equal to R (R must be a string.)

$\leq R$: Reports a match whenever the current substring is alpha-numerically (lexically) less than or equal to R (R must be a string.)

$R_i \& R_j$: The conjunction operator: Both R_i and R_j must match at the same location.

References

[1] E. J. Keogh, S. Lonardi, and B. Chiu, "Finding surprising patterns in a time series database in linear time and space," in *Proc. KDD*, 2002, pp. 550–556.

[2] H. Mannila, H. Toivonen, and A. I. Verkamo, "Discovery of frequent episodes in event sequences," *Data Mining and Knowledge Discovery*, vol. 1, num. 3, pp. 259–289, Jan. 1997.

[3] R. Agrawal and R. Srikant, "Mining sequential patterns," in *Proc. ICDE*, 1995, pp. 3–14.

[4] G. Das, K. Lin, H. Mannila, G. Renganathan, and P. Smyth, "Rule discovery from time series," in *Proc. KDD*, 1998, pp. 16–22.

[5] M. L. Hetland and P. Sætrom, "Temporal Rule Discovery using Genetic Programming and Specialized Hardware," in *Proc. 4th Int. Conf. on Recent Advances in Soft Computing* Nottingham, 2002.

[6] G. M. Weiss and H. Hirsh, "Learning to predict rare events in event sequences," in *Fourth International Conference on Knowledge Discovery and Data Mining (KDD'98)* R. Agrawal, P. Stolorz and G. Piatetsky-Shapiro, Eds. publisher = "Menlo Park, CA: AAAI Press, ", 1998, pp. 359–363.

[7] S. Zemke, "Nonlinear Index Prediction," in *Proceedings of the International Workshop on Econophysics and Statistical Finance* Palermo, Italy, September 1998, Physica A Vol. 269, no. 1, Elsevier Science, R. N. Mantegna, Ed., pp. 177–183.

[8] R. J. Povinelli, "Using Genetic Algorithms to Find Temporal Patterns Indicative of Time Series Events," in *GECCO 2000 Workshop: Data Mining with Evolutionary Algorithms*, pp. 80–84, 2000.

[9] A. A Freitas, *Data Mining and Knowledge Discovery with Evolutionary Algorithms*, Berlin: Spinger-Verlag, 2002.

[10] F. Höppner and F. Klawonn, "Finding Informative Rules in Interval Sequences," in *Lecture Notes in Computer Science*, 2189, 125–??, 2001.

[11] R. D. Martin and V. Yohai, "Data Mining for Unusual Movements in Temporal Data," in *KDD Workshop on Temporal Data Mining*, 2001.

[12] S. Chakrabarti, S. Sarawagi and B. Dom, "Mining surprising patterns using temporal description length," in *Twenty-Fourth International Conference on Very Large databases VLDB'98*, New York, NY: Morgan Kaufmann, A. Gupta, O. Shmueli, and J. Widom, Eds., pp. 606–617, 1998.

[13] M. Last, Y. Klein, and A. Kandel, "Knowledge Discovery in Time Series Databases," *IEEE Trans. on Systems, Man, and Cybernetics* vol. 31B, no. 1, pp. 160–169, Feb. 2001.

[14] M. L. Hetland, "A Survey of Recent Methods for Efficient Retrieval of Similar Time Sequences," in *Data Mining in Time Series Databases*, M. Last, A. Kandel, and H. Bunke, Eds. Singapore: World Scientific, to be published.

[15] J. R. Koza, *Genetic Programming: On the programming of Computers by Means of Natural Selection*. Cambridge, MA: The MIT Press, 1992.

[16] M. Spiliopoulou, "Managing Interesting Rules in Sequence Mining," in *Proc. PKDD*, 1999, pp. 554–560.

[17] R. J. Hilderman and H. J. Hamilton, "Knowledge discovery and interestingness measures: A survey," Department of Computer Science, University of Regina, Saskatchewan, Canada, Tech. Rep. CS 99-04, Oct. 1999.

[18] P. Smyth and R. M. Goodman, "Rule induction using information theory," in *Knowledge Discovery in Databases*, G. Piatetsky-Shapiro, W. J. Frawley, Eds. Cambridge, MA: MIT Press, 1991, pp. 159–176.

[19] G. Piatetsky-Shapiro, "Discovery, analysis and presentation of strong rules," in *Knowledge Discovery in Databases* G. Piatetsky-Shapiro, W. J. Frawley, Eds. Cambridge, MA: MIT Press, 1991, pp. 229–248.

[20] Fast Search & Transfer ASA, "Digital processing device," European patent specification EP1125216B1, deriving from international published patent application WO 00/22545.

[21] Fast Search & Transfer ASA, "Søkeprosessor," Norwegian patent 309169, also filed as international published patent application WO 00/29981 titled "A processing circuit and a search circuit."

[22] Interagon AS. (2002, Aug.). The IQL Language. Interagon AS. Trondheim, Norway. [Online]. Available: http://www.interagon.com/pub/whitepapers/IQL.reference-latest.pdf

[23] E. Keogh and T. Folias, "The UCR Time Series Data Mining Archive," [Online] Available from `http://www.cs.ucr.edu/~eamonn/TSDMA/index.html`, Riverside CA. University of California, Computer Science & Engineering Department, 2002.