Machine Learning, 58, 107–125, 2005

Evolutionary Rule Mining in Time Series Databases

MAGNUS LIE HETLAND mlh@idi.ntnu.no
Norwegian University of Science and Technology

PÅL SÆTROM paalsat@interagon.com
Interagon AS

Editor: Eamonn Keogh

Abstract. Data mining in the form of rule discovery is a growing field of investigation. A recent addition to this field is the use of evolutionary algorithms in the mining process. While this has been used extensively in the traditional mining of relational databases, it has hardly, if at all, been used in mining sequences and time series. In this paper we describe our method for evolutionary sequence mining, using a specialized piece of hardware for rule evaluation, and show how the method can be applied to several different mining tasks, such as supervised sequence prediction, unsupervised mining of interesting rules, discovering connections between separate time series, and investigating tradeoffs between contradictory objectives by using multiobjective evolution.

Keywords: sequence mining, knowledge discovery, time series, genetic programming, specialized hardware

1. Introduction

As data are available in ever increasing amounts, there is a need for automated knowledge discovery. One recent development is the use of evolutionary methods to create flexible and powerful mining solutions. Methods for knowledge discovery in sequence data are well developed, but this is one domain where evolution has not been thoroughly examined yet. In our work we use specialized hardware for sequence searching to enable the evolution of rules from sequence data, or, more specifically, time series data. In this study we have used various real-world time series data sets as well as random time series data as a baseline. We attempt to make predictions about whether the time series will go up or down (in the cases where this is feasible) and look for regularities in the data that might be useful to a human expert. Although methods exist for doing this sort of thing (see Section 1.1), our method has two main contributions: (1) it is highly flexible and configurable, in that one can decide which rule format and objective measure to use, and (2) the resulting models and rules are (at least in principle) human-readable.

The foundation for this work has been laid in several previous publications (Hetland & Sætrom, 2002, 2003a, 2003b; Sætrom & Hetland, 2003a, 2003b). In this paper we have collected our main results, compared them with other machine learning methods (Section 3.1) and extending them by using our method for finding relations between several time series (Section 3.2.4).

We do not contend that our method is better at time series prediction or classification than existing methods, although we do show that it is quite robust and has considerable predictive power. Instead, our main contribution lies in the flexibility of our method and the freedom it affords the data miner, in that he or she can specify a rule format and quality function suited for the application at hand. Also, the basic components of our method (that is, the discretization method and genetic programming) are well known; it is the application of these methods to the domain of sequence rule mining that is novel.

Our method is based on the availability of special-purpose pattern matching hardware (Halaas et al., 2004). While this may be seen as a limitation, the new functionality that this hardware offers has, in fact, been one of the driving forces behind our work. In the interest of wider applicability, we have in the past done some work on using more readily available software-based retrieval techniques with our method (Hetland & Sætrom, 2003a). That line of inquiry is not pursued in this paper.

1.1. 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 (Hetland & Sætrom, 2002; Weiss & Hirsh, 1998; Zemke, 1998; Povinelli, 2000) fall in this category. All of these use some form of evolutionary computation; (Hetland & Sætrom, 2002) 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 (Freitas, 2002). 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, Mannila, Toivonen, & Verkamo, 1997; Agrawal & Srikant, 1995; Höppner & Klawonn, 2001). 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 (Martin & Yohai, 2001), or finding unusual temporal patterns in market basket data (Chakrabarti, Sarawagi, & Dom, 1998).

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 methods.

Others have tried this direct approach in the context of standard database mining. For instance, Noda, Freitas, and Lopes (1999) used a genetic algorithm to find interesting association rules in two different relational databases. The algorithm evolved the antecedent and then selected the most accurate consequent from a pre-specified set of consequents. This approach is, however, new to time series mining.

1.2. Structure of this paper

This section outlines the structure of the rest of the paper. As an introduction to the presentation of our method in Section 2, Section 2.1 gives a more precise description of the problem we are trying to solve, that is, discovering human-readable rules from time series data. Then, Sections 2.2 through 2.6 deal with the specifics of rule evolution, fitness calculation, time series discretization and our data management method, the Interagon Pattern Matching Chip. Section 3 describes our experimental set-ups and results for both supervised and unsupervised mining (Sections 3.1 and 3.2, respectively). Section 4 summarizes the paper, and gives some directions for future work. A brief description of the rule notation used in this paper may be found in Appendix A.

2. Method

This section outlines the specifics of the problem we are trying to solve, and describes the variations of our data mining method in more detail.

2.1. Problem definition

In this paper we will look at two different versions of the problem of mining rules from time series. Our *rules* have the following simple and well known rule format: "If *antecedent* then *consequent* within T time units". The first version of the problem can loosely be described as the problem of finding rules to predict specific events in a time series. In the second version, the user has no previous knowledge about the data and simply wants to extract useful information (rules) from the time series. We refer to the first version of the problem as the supervised mining problem and to the second version as the unsupervised mining problem. We will now give more formal definitions of the two versions and give an extension to the second version that considers the problem of mining a *set* of time series.

Definition 1 (Supervised rule mining). Given a sequence S, a predicate p over all the indices of S, and a delay δ, find a rule that estimates the value of $p(i+\delta)$ from the prefix $s_1, \ldots, s_i$ of S.

The estimated predicate is written $\hat{p}$. In the terminology of Sun and Giles (2000) this is a problem of sequence recognition, although by letting the predicate p represent a type of event that may or may not occur at time $i+\delta$, the rules can also be used to make predictions.

Note that we take all available history into consideration by using a full prefix of S, rather than a fixed-width sliding window.

Definition 2 (Unsupervised rule mining). Given a sequence S and a rule language $L = L_a \overset{T}{\underset{w}{\Rightarrow}} L_c$, where L_a and L_c are the antecedent and consequent languages, w is a minimum distance, and T is a maximum distance, find rules $R \in L$ that optimize some objective function $f(R)$.

Definition 3 (Mining multiple series). Given a set $\mathcal{S}$ of m sequences S_i ($S_i \in \mathcal{S}$) and a rule language $L = L_a^i \overset{T}{\underset{w}{\Rightarrow}} L_c^j$, $i, j \in [1, m]$ and $i \neq j$, find rules $R \in L$ that optimize some objective function $f(R)$.

Definition 4 (Multiobjective rule mining). Given a sequence S, a rule language L, and a set $F = \{f_i\}$ of objective functions, find a diverse set of rules $R = \{r_i\}$ with high objective values $f_i(r_j)$, such that no rule $r_i \in R$ dominates any other rule $r_j \in R$. One rule dominates another if it is as good or better in terms of all objectives, and strictly better in terms of at least one objective.

We give more details about various practical aspects of these definitions later in this paper.

2.2. *Evolving rules*

The evolutionary computation strategy used in this paper is genetic programming, as described in Koza (1992). 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. Although the basic mining algorithm remains the same in both the supervised and unsupervised setting, how the rules are represented differs slightly.

In the supervised setting, each individual in the population is simply a syntax tree in the antecedent language L_a. This is because in this setting, the consequent and distance between the antecedent and consequent are inputs to the mining algorithm.

In the unsupervised setting, however, the consequent and distance are evolved along with the antecedent. More specifically, each individual in the population is a syntax tree in the language $L_a \overset{T}{\underset{w}{\Rightarrow}} L_c$. This is implemented by using three separate branches, one 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 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 $T + 1$.

By adding two additional branches, the system can be used to mine sets of m time series. The additional branches identify the series in the set to which the antecedent and consequent belong. These branches are constructed in the same way as the maximum distance branch, with the exception that we now use the size of the set m instead of $T + 1$ in the modulo calculation.

Expressions in the chosen rule languages are evaluated by the specialized pattern matching hardware described in Section 2.6, and rule fitness is calculated by searching the time series data for rule occurrences.

2.3. *Multiobjective evolution*

Multiobjective optimization is the problem of simultaneously optimizing a set F of two or more objective functions. The objective functions typically measure or describe different features of a desired solution. Often these objectives are conflicting in that there is no single solution that simultaneously optimizes all functions. Instead one has a *set* of optimal solutions. This set can be defined using the notion of *Pareto optimality* and is commonly referred to as the *Pareto optimal set* (Coello Coello, 2001).

Assuming that the functions in F are to be maximized, a solution $\mathbf{x}$ is *Pareto optimal* if no other solution $\mathbf{x}'$ exists such that $f_i(\mathbf{x}') \geq f_j(\mathbf{x})$ for all $f \in F$ and $f_i(\mathbf{x}') > f_j(\mathbf{x})$ for at least one $f \in F$. Informally, this means that $\mathbf{x}$ is Pareto optimal if and only if there does not exist a feasible solution $\mathbf{x}'$ which would increase some objective function without simultaneously decreasing at least one other objective function.

The solutions in the Pareto optimal set are called *non-dominated*. Given two solutions, $\mathbf{x}'$ and $\mathbf{x}$, $\mathbf{x}'$ *dominates* $\mathbf{x}$ if $f_i(\mathbf{x}') \geq f_j(\mathbf{x})$ for all $f \in F$ and $f_i(\mathbf{x}') > f_j(\mathbf{x})$ for at least one $f \in F$. In other words, $\mathbf{x}'$ is at least as good as $\mathbf{x}$ with respect to all objectives and better than $\mathbf{x}$ with respect to at least one objective.

The goal in multiobjective optimization is to find a diverse set of Pareto optimal solutions. In evolutionary multiobjective optimization this is typically found by producing a set of solutions from a single evolutionary algorithm run. Several different algorithms for evolutionary multiobjective optimization exist (see Coello Coello, 2001 for an introduction and Coello Coello, 2000 for a survey).

The algorithm used here is based on the SPEA2 (Zitzler, Laumanns, & Thiele, 2001), which uses a fixed size population and archive. The population forms the current base of possible solutions, while the archive contains the current solutions. The archive is constructed and updated by copying all non-dominated individuals in both archive and population into a temporary archive. If the size of this temporary archive differs from the desired archive size, individuals are either removed or added as necessary. Individuals are added by selecting the best dominated individuals, while the removal process uses a heuristic clustering routine in objective space. The motivation for this is that one would like to try to ensure that the archive contents represent distinct parts of the objective space. The fitness of an individual

is based on both the strength of its dominators (if dominated) and the distance to its k-nearest neighbor (in objective space). See Zitzler, Laumanns, and Thiele (2001) for further details.

In this work, the SPEA2 algorithm has been modified as follows: When selecting individuals for participation in the next generation, both the archive and the main population were used. The SPEA2 approach of only selecting from the archive was tried, but this resulted in premature convergence, and the results in the final generation were simple variations of the first archive contents. In addition, to prevent further convergence of the archive contents, only individuals having differing objective values were selected in the initial archive filling procedure. If two or more individuals shared the same objective values, one of these was randomly selected to participate in the archive.

In our experiments, the population size was typically 100 times larger than the archive size. Subtree swapping crossover was used 99% of the time, while tree generating mutation was used 1% of the time.

2.4. *Fitness*

We have used different fitness measures in the supervised and unsupervised setting.

2.4.1. Fitness in supervised mining. In the supervised setting the fitness measure is the *correlation r* between the occurrences of a rule and the *desired* occurrences of the rule, as given by the event to be predicted. Using the elements from the *confusion matrix* of a rule (true and false positives and negatives, with the obvious abbreviations) this correlation may be expressed as:

$$r(p, \hat{p}) = \frac{\mathrm{TP}\cdot\mathrm{TN} - \mathrm{FP}\cdot\mathrm{FN}}{\sqrt{(\mathrm{TN}+\mathrm{FN})(\mathrm{TN}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})(\mathrm{TP}+\mathrm{FP})}}, \tag{1}$$

where p and $\hat{p}$ are predicates describing the desired and actual hits of the rule, respectively.

2.4.2. Fitness in unsupervised mining. In the unsupervised setting, we consider several different fitness functions. All of these functions are based on the concepts of confidence, support, and interestingness.

Given a rule $R = R_a \overset{t}{\underset{w}{\Rightarrow}} R_c$ in the rule language $L_a \overset{T}{\underset{w}{\Rightarrow}} L_c$ (such that $t \leq T$) and a discretized sequence $S = (s_1, s_2, \ldots, s_n)$, the frequency $F(R_a)$ of the antecedent is the number of occurrences of R_a in S. This can be formalized as

$$F(R_a) = |\{i \mid H(R_a, S, i)\}|, \tag{2}$$

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:

$$F(R_a, R_c, t) = |\{i \mid H(R_a, S, i) \wedge H(R_c, S, j) \wedge i+w \leq j \leq i+w+t-1\}|. \tag{3}$$

This is the number of matches of R_a that are followed by at least one match of R_c within t time units. Note that only occurrences of R_a that are followed by a hit from R_c after $w-1$ units of time are counted. The reason for introducing this minimum distance is that the discretization process described in Section 2.5 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. Also note that if no such correlations are present, w can be ignored (that is, set to zero).

The *confidence* of a rule is defined as:

$$c(R) = \frac{F(R_a, R_c, t)}{F(R_a)} \tag{4}$$

This is basically an estimate of the conditional probability of observing the consequent within t time units, given that the antecedent has just been observed.

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 Hilderman & Hamilton, 1999 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 (Smyth & Goodman, 1991). This is defined as:

$$\begin{aligned} J\left(R_c^t, R_a\right) = p(R_a) \cdot \Bigg(& p\left(R_c^t \middle| R_a\right) \log_2 \frac{p\left(R_c^t \mid R_a\right)}{p\left(R_c^t\right)} \\ & + \left(1 - p\left(R_c^t \middle| R_a\right)\right) \log_2 \frac{1 - p\left(R_c^t \middle| R_a\right)}{1 - p\left(R_c^t\right)} \Bigg) \end{aligned} \tag{5}$$

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, and, finally, $p(R_c^t \mid R_a)$ 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, 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 $p(R_c^t \mid R_a)$ (the second term, also known as the *cross-entropy*).

Another important rule quality is comprehensibility, that is how easily the rules can be interpreted by a human user. One of the most important principles of mining comprehensible rules is using a rule representation that in itself is intelligible. In addition, one often tries

to limit the size of the rules. This is motivated by the fact that larger rules usually are harder to interpret. When using genetic programming (GP) as the rule induction method this becomes even more important. This is because GP tends to create large programs that contain semantically irrelevant parts. This tendency toward large programs is known as *bloat*.

Equation (6) gives a definition of *rule simplicity*, which is used in the following experiments.

$$simplicity(R) = \frac{1}{nodeCount(R) + maxDepth(R)} \tag{6}$$

Here the functions $nodeCount(R)$ and $maxDepth(R)$ return the number of nodes and the maximum depth of R, respectively.

2.5. *Discretization algorithm*

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 described, among other places, in Keogh, Lonardi, and Chiu (2002). 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 most likely have similar feature values, which means they are more likely to be assigned the same symbol. We deal with this by using a minimum distance between the antecedent and consequent of at least w (see Section 2.4.2).

This discretization method is by no means unique. In Last, Klein, and Kandel (2001) a method is described that 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 Hetland (2004) for a survey. A recent discretization method with several interesting properties is the SAX method (Lin et al., 2003).

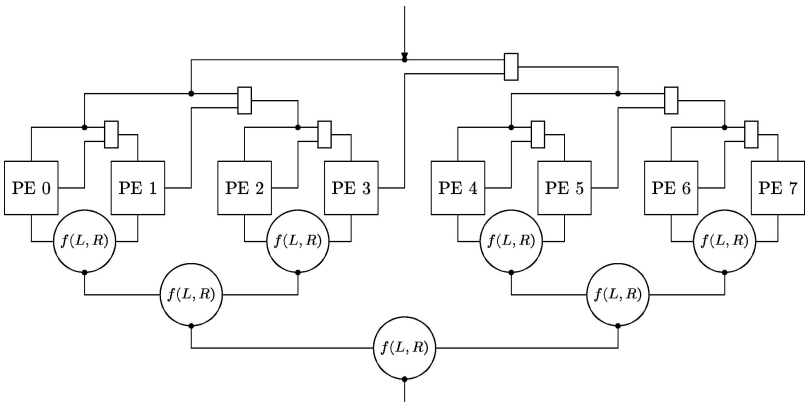

Figure 1. A data distribution tree with eight leaf nodes (PEs), and the corresponding result gathering tree with $f(L, R)$ calculated from the above left (L) and right (R) results.

2.6. *Search hardware*

To make it possible to perform a full search in the training data for each fitness calculation, we used a specialized pattern matching chip (PMC). The PMC is a special purpose co-processor designed for locating complex patterns in unstructured data (Halaas et al., 2004). We will in the following give a brief description of its architecture and main functionality; please consult the original work (Halaas et al., 2004) for additional details.

The PMC consists of three functional units, as illustrated in figure 1: A data distribution tree (top), a set of processing elements, or PEs (middle), and a result gathering tree (bottom). The PEs monitor the data flowing through the chip. They receive the data from the data distribution tree, which can be configured so that single PEs and groups of PEs receive data either sequentially or in parallel. In figure 1, this is illustrated by the tree of 2:1 multiplexers at the top.

Each PE is configured with one byte (character) and a comparison operator, which it uses to look for bytes in the data stream. By reconfiguring the comparison operator, the PEs can look for bytes that are equal, unequal, greater than or equal, or less than or equal to their given byte. Matches are reported to the result gathering tree, which combines them through a configurable function and produces the final result: a Boolean value representing a hit or miss for the entire pattern. If the result is a hit, the hit location is reported back to the host computer by direct memory access (DMA).

The PMC is capable of performing searches at the rate of 100 MB/s and can handle several complex queries in parallel (from 1 to 64 depending on query complexity.) The interface to the PMC is the special purpose query language IQL. This language supports such functionality as regular expressions, latency (distance), alpha-numerical comparisons, hamming distance filtering, and generalized Boolean operations. As an illustration, the PMC supports queries requiring, for example, the strings “William” and “Shakespeare” each to match with a maximum hamming distance of two and separated by a maximum distance of

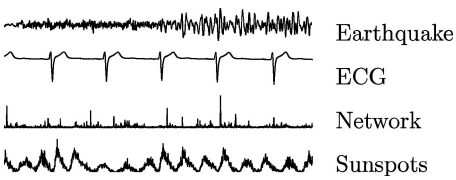

Figure 2. The time series analyzed.

twenty characters. The regular expression syntax is similar to that of UNIX tools such as `grep`. A detailed description of the language is available online (Interagon AS, 2002).

3. Experiments

In our experiments we use four data sets from the UCR Time Series Data Mining Archive (Keogh & Folias, 2002): ECG measurements from several subjects, concatenated; earthquake-related seismic data; monthly mean sunspot numbers from 1749 until 1990; and network traffic as measured by packet round trip time delays. Figure 2 shows plots of parts of the different time series analyzed.

In addition, in our unsupervised mining experiments we used stock series available from Yahoo finance.[1] The series were the adjusted daily closing price of ten U.S. companies listed on the New York Stock Exchange. More specifically, the companies were Amgen, Dell, Disney, General Electric, General Motors, IBM, Microsoft, Oracle, Sun, and Walmart.

3.1. Supervised mining

For each of the four data sets, the target prediction was when the series moved upward, that is, when the next value was greater than the current.

The rules that were developed by our system had their performance tested on a validation set (through a form of k-fold cross-validation). In the simple case, a single resolution and alphabet size is used, and the rule that had the best performance on the validation set is selected and tested on a separate test set. Instead of this simple approach, we use several resolutions and alphabet sizes, run the simple process for each parameter setting, and chose the rule (and, implicitly, the parameter setting) that has the highest validation rating. This way we could discover the resolution and alphabet size that best suited a given time series, without having to arbitrarily set these beforehand. To demonstrate the effect of this procedure, we also selected the rule (and parameter setting) that had the lowest validation score, as a baseline.

We also used two ensemble methods for combining rules from different discretizations: majority vote ensembles and naive Bayesian ensembles. To construct the ensembles, we selected the alphabet size that gave the best single-predictor performance for each resolution, and these rules were then used to construct the ensembles.

We performed experiments with alphabet sizes 2 (the minimum non-trivial case), 5, 10, and 20, as well as window sizes (resolutions) 2, 4, 8, 16, and 32. This was done with 10-fold cross-validation[2] and early stopping; that is, rules were selected based on their performance on a separate validation set before they were tested.

To evaluate our method further and put the results in perspective, we have compared it to three other machine learning algorithms: naive Bayesian classification, which is a simple and, in many cases, effective algorithm; support vector machines, which are considered the state of the art in statistical machine learning; and decision trees (C4.5), which produce results that are comparable to our method in that they are much easier to interpret than the Bayesian and support vector machine classifiers.[3] (Note that as a practical matter, we had to run the SVM on a subset of the ECG data, as it failed to terminate for days on the full data set. The reason for this was, in all likelihood, the high memory demands and superlinear running time of the method.) All these methods used 10-fold cross-validation.

3.1.1. Results. The rules in the ensembles were developed individually, that is, only their individual performance was used in calculating their fitness. The performance of the ensemble was then calculated by performing a simple majority vote among the rules for each position (for the majority vote ensembles) or by choosing the most likely prediction using Bayes's theorem (for the naive Bayesian ensembles). The results are summarized in figure 3. The percentages (predictive accuracy) are averaged over the ten folds. For the ECG, sunspot, network, and earthquake data sets, the difference between the worst single classifier and the best single classifier is statistically significant ($p < 0.01$ with Fisher's exact test), while the differences between the best single classifier, the ensemble, and the Bayesian classifier are not statistically significant ($p > 0.05$ for all except the difference between the best single classifier and the Bayesian combination for the Network data, where $p = 0.049$). For the random data set there are no significant differences, as expected.

In other words, we found that simply selecting the discretization that gave the best validation results was probably the best tradeoff between simplicity and predictive power.

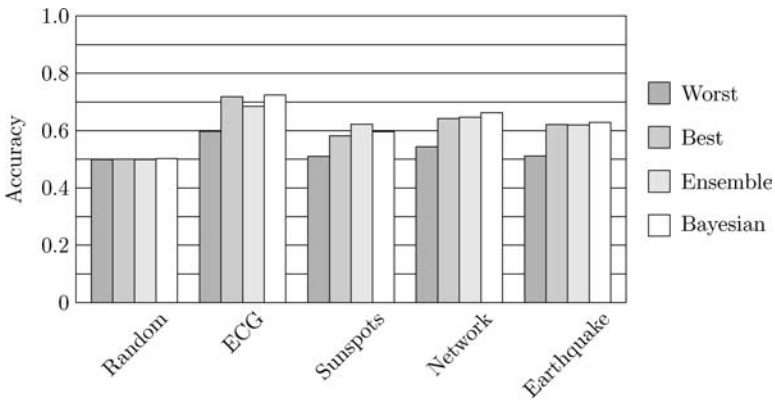

Figure 3. Performance comparison.

Table 1. Comparison of the accuracy (percent correctly predictions) of our supervised method (PMC-GP) to that of existing methods for sequence learning.

	PMC-GP (%)	Bayes (%)	C4.5 (%)	SVM (%)
Random	49.9	50.0	50.2	50.2
ECG	71.8	53.0	80.2	51.8
Sunspots	58.2	51.6	58.5	52.0
Network	64.2	53.8	66.3	69.6
Earthquake	62.0	76.6	63.3	71.1

A comparison of the results obtained by our method and the other machine learning methods used for comparison (as described previously, in Section 3.1) is shown in Table 1. The C4.5 algorithm was run on discretized data, while the Bayesian and SVM classifiers were run on fixed-width sliding window vectors extracted from the time series. It is clear that, although our method achieves accuracy levels comparable to the other methods, it never beats all of them for any of the data sets.

3.2. Unsupervised mining

This section describes our unsupervised experiments, both for single series, multiple series, and using multiobjective optimization.

3.2.1. Mining a single time series. We performed three sets of unsupervised single-series experiments. First, the four time series were mined by using our single-objective genetic programming algorithm. As fitness, this algorithm uses a modified version of the J-measure, which has a confidence correcting term to ensure that all rules generated by the algorithm have a confidence above some minimum threshold. That is, we multiply the standard J-measure with the sigmoid function

$$F(c(R)) = \frac{1}{1 + e^{-(c(R)-c_{\min})\cdot g}}, \tag{7}$$

where g is a parameter regulating how sharp the cutoff at the confidence threshold $c_{\min}$ should be.

Second, we mined the series by using the occurrence counting algorithm described in Das et al. (1998) (from now on referred to as the *Das*-algorithm). We then compared the rules produced by this algorithm to the rules produced by our genetic programming approach.

Third, we mined the series using the multiobjective genetic programming (MOGP) algorithm described in Section 2.3. Here we used the confidence (4), J-measure (5), and rule simplicity (6) as simultaneous objective functions.

In the experiments, the single-objective genetic programming algorithm used a population size of 5000 and ran for 20 generations. The MOGP algorithm used a population size of 1000, an archive size of 10, and ran for 100 generations. Both algorithms used the IQL

language (Interagon AS, 2002) for generating rules. The antecedents were IQL expressions, while the consequents were single characters or concatenations thereof. The Das-algorithm used a minimum confidence of 0.5, a minimum support of 0.01, and the maximum distances $T = 10$ and $T = 5$.

The following sections summarize the results of the experiments. All the results presented in the following sections were generated on series discretized with a window size $w = 2$. (We used 10 for both the minimum and maximum distances, so that the consequent could occur at most 20 positions after the antecedent.) In Sætrom and Hetland (2003a), we showed that increasing the window size makes the rule mining algorithm more noise tolerant. That is, we showed that even if artificial Gaussian noise was added to the time series, we were still able to get good rules if we increased the window size in the discretization procedure. Here we will, however, focus on the short term features of the time series, which are best described using small window sizes.

3.2.2. Single objective optimization. Tables 2 and 3 show the best rules mined by GP and the Das algorithm on the four data sets. The tables show that the GP-based mining algorithm finds rules that have higher J-measure for all the four time series. Even when the best rule has the simple structure of the rules generated by the Das-algorithm (the *Sunspot* rule), GP finds a higher ranked rule by optimizing the rule distance.

To better evaluate the rules mined by the algorithms, figure 4 compares the hits from the rules generated by the two algorithms on the earthquake, ECG, and network series. The figure shows that on the earthquake and ECG series, the GP-based algorithm has captured slightly different aspects of the time series, which results in better rules both in terms of J-measure and confidence.

Table 2. The highest ranked rules mined by GP. The rules are the best of run rules from a single run on each of the four time series.

Series	Rule	J-mea.	Conf.	Supp.
Earthquake	$\mathrm{r} \xleftarrow{28} \geq \mathrm{r} \xrightarrow{8} \mathrm{b}+$	0.030	0.58	0.07
ECG	$\mathrm{t} \xleftarrow{45} \mathrm{t} \xrightarrow{4} \mathrm{s}$	0.066	0.71	0.03
Network	!l ↔ (b | (≥ j ↔!l ↔ [bt] ↔!l ↔ (b | ([^bt] ↔!l) |!r) ↔ ((!r ↔ [bt]) | t))) $\xrightarrow{9}$ s	0.012	0.54	0.07
Sunspot	$\mathrm{j} \xrightarrow{9} \mathrm{j}$	0.019	0.66	0.05

Table 3. The highest ranked rules mined by the Das algorithm.

Series	Rule	J-mea.	Conf.	Supp.
Earthquake	$\mathrm{a} \xrightarrow{10} \mathrm{t}$	0.012	0.52	0.03
ECG	$\mathrm{a} \xrightarrow{5} \mathrm{t}$	0.058	0.58	0.03
Network	$\mathrm{b} \xrightarrow{10} \mathrm{s}$	0.002	0.52	0.03
Sunspot	$\mathrm{j} \xrightarrow{10} \mathrm{j}$	0.017	0.66	0.05

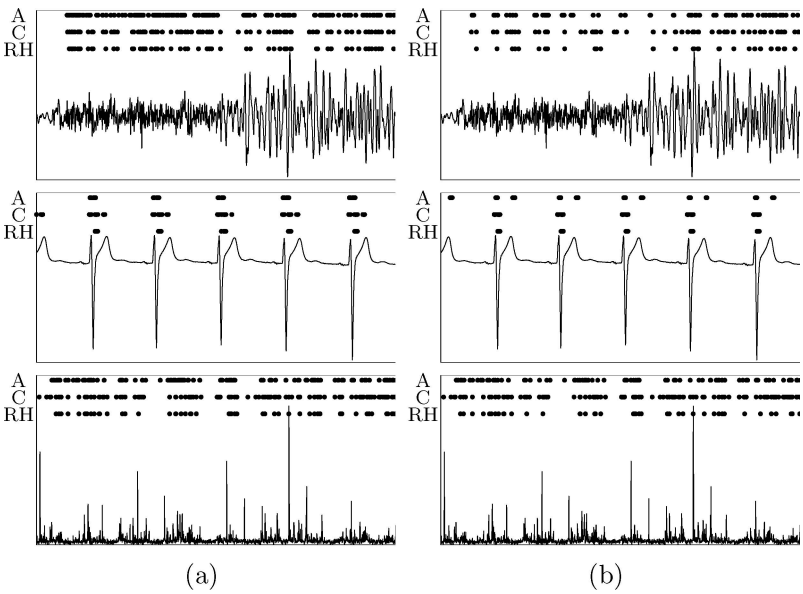

Figure 4. Hit locations of the highest ranked rules from GP (Panel (a)) and the Das algorithm (Panel (b)). The dots following the *A* and *C* labels on the *y*-axis indicate the hit locations of the antecedent and consequent, respectively. The dots following the *RH* label indicate the positions where the rule hits, that is where the consequent follows the antecedent within the desired distance.

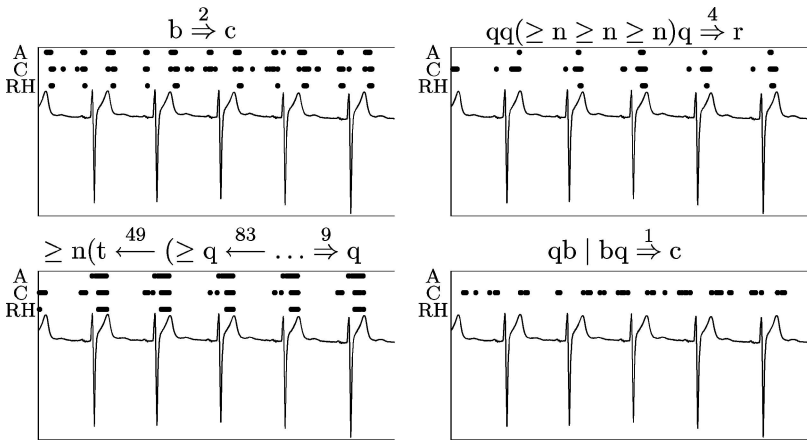

Figure 5. Hit locations in a subsequence of the ECG series of the rules from Table 4. The dots following the *A* and *C* labels on the *y*-axis indicate the hit locations of the antecedent and consequent, respectively. The dots following the *RH* label indicate the positions where the rule hits, that is where the consequent follows the antecedent within the desired distance.

Table 4. An excerpt of the contents of a typical archive at algorithm termination for the ECG dataset.

Rule	J-mea.	Conf.	Supp.	Compl.
$b \overset{2}{\Rightarrow} c$	0.058	0.58	0.029	0.20
$qq(\geq n \geq n \geq n)q \overset{4}{\Rightarrow} r$	0.017	0.92	0.0063	0.042
$\geq n(t \overset{49}{\longleftarrow} (\geq q \overset{83}{\longleftarrow} \geq n \geq c \geq n \geq n)) \geq c \mid$ $\geq c(t \overset{49}{\longleftarrow} (\geq q \overset{83}{\longleftarrow} \geq n \geq c \geq n \geq n)) \geq n \overset{9}{\Rightarrow} q$	0.18	0.77	0.13	0.028
$qb \mid bq \overset{1}{\Rightarrow} c$	0.000059	1.0	0.000014	0.13

The plot for the network data, however, illustrates one of the problems with GP. The GP rule for the network data is more complex than the rule generated by the Das algorithm, but as figure 4 shows, the rules are essentially the same. What has happened is that the GP rule contains subexpressions that do not contribute to the rule—the rule has become *bloated*. The MOGP algorithm (results presented in the next section), tries to solve this problem by including rule parsimony as one of the objective functions to be optimized.

3.2.3. Multiobjective optimization. Table 4 lists a subset of the results from a run on the ECG data set discretized with a window size of 2. In addition, the archive contained

Table 5. Summary of connections between series. The table shows the number of times the best of run rule connected pairs of stock series in the set for different discretization windows w.

		Frequency		
Ant. series	Con. series	$w = 2$	$w = 8$	$w = 16$
0	1	1	0	0
1	3	1	0	0
3	1	1	0	0
1	9	1	0	0
3	6	0	1	0
6	1	1	0	3
6	9	0	0	2
9	6	0	0	1
7	1	0	4	2
7	6	0	1	0
7	8	0	0	1
8	7	0	2	0
8	0	0	2	0
8	3	5	0	0
9	3	0	0	1

Table 6. Selected rules from the multi sequence mining algorithm.

w	Rule	J-mea.	Conf.	Supp.
2	$a \xleftarrow{21} s*^{8} \overset{9}{\Rightarrow} a+^{3}$	0.269	0.88	0.19
8	$\geq r^{3} \overset{9}{\Rightarrow} s+^{6}$	0.084	0.61	0.09
16	$\geq s^{6} \overset{9}{\Rightarrow} t^{9}$	0.098	0.58	0.06

three versions of the first rule having differing maximum distance and three versions of the third rule having small variations in the antecedent. The hits of the rules from Table 4 in a subsequence of the ECG series are plotted in figure 5.

As Table 4 and figure 5 show, the MOGP algorithm generates rule sets that (at least partially) contain distinct rules that capture different aspects of the time series. Also, comparing figure 5 and the ECG results in figure 4 shows that the MOGP algorithm generates other rules than those generated by the algorithms that only optimize the rule interestingness. Thus, the algorithm works as intended in that it presents the user with a choice of several rules for further study and evaluation.

3.2.4. Mining multiple time series. In these experiments we tested the multi-sequence mining algorithm on a set of ten stock quote series. We discretized the data by using window sizes 2, 8, and 16, and mined each of the resulting sets. We used these window sizes because we wanted to investigate both short ($w = 2$), middle ($w = 8$), and long term ($w = 16$) trends in the data. The antecedents and consequents were forced to match different series by assigning a fitness of zero to the rules that mapped them to the same series.

We used the basic single-objective genetic programming algorithm, with the same fitness measure and parameters as in Section 3.2.1. We ran a total of 10 runs on each window size; Table 5 shows the number of times a rule connected the different series for the different

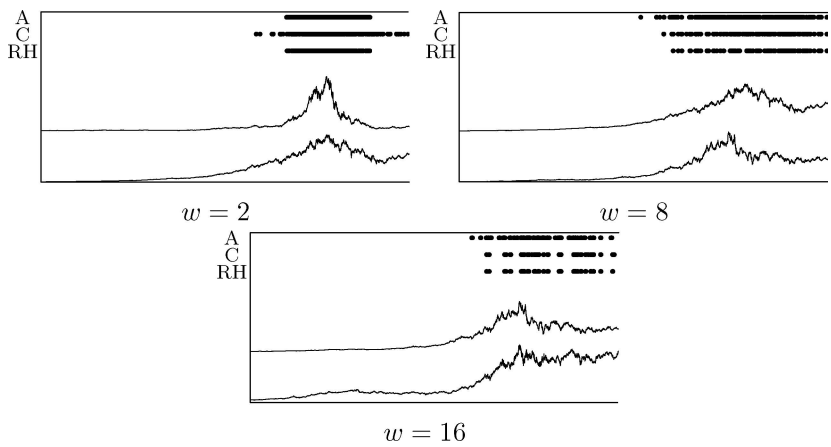

Figure 6. Hit locations of the rules from Table 6. The time series at the top is the antecedent series; the series at the bottom is the consequent series. See figure 4 for the definitions of *A*, *C*, and *RH*.

window sizes. When we identified the corresponding companies we found that for $w = 2$, one of the rules joined the computer companies; but for $w = 8$, seven of the rules joined the computer companies; and for $w = 16$, six of the rules joined the computer companies. This indicates that the larger window sizes ($w = 8$ and $w = 16$) enables the mining algorithm to capture the longer trends common to companies from the same industry. The smaller window size, on the other hand, can also capture short term trends and fluctuations that can be common to companies from unrelated sectors.

Table 6 and figure 6 detail one of the rules mined for each window size. Although all three rules join companies that are unrelated (Sun and GE, GE and Microsoft, and Microsoft and Walmart), the rules have managed to capture aspects of the series that make them similar. The results suggest that there are some aspects in the antecedent series that can be used to make predictions about the future behavior of the consequent series. For instance, the rule connecting Microsoft and Walmart ($w = 16$ in Table 6 and figure 6) has captured that a large decrease in Microsoft stock value has a high chance of being followed by a large decrease in the Walmart stock.

4. Summary and discussion

In this paper we have examined the use of artificial evolution in mining rules from time series. Our method is based on discretizing the time series and using a specialized pattern matching hardware for locating rule occurrences for the purpose of rule fitness calculation. We have developed several variations of this scheme, for supervised mining of predictive rules, for unsupervised mining of interesting rules in a single series, for mining rules that connect related series from a set of candidates, and multi-objective mining for exploring tradeoffs between conflicting objectives such as confidence and parsimony.

We have shown empirically that our method is capable of producing rules with good predictive power and with a high level of interestingness (as measured by existing interestingness measures). We have also shown that it can discover real-world relationships between time series. In comparing our method with other machine learning methods, it would seem that its main strength lies not in the task of prediction itself, but rather in its flexibility, both in terms of rule format and objective function, as a tool for explorative rule mining.

Appendix A: Rule notation

This appendix describes the rule notation used in this paper.

$R*$	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 (i.e. the negation of R).
$R_i \overset{t}{\leftarrow} R_j$	Reports a match whenever R_j reports a match and R_i reported a match at most t letters 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.
$R_i \leftrightarrow R_j$	The adjacency operator: Is equivalent to $(R_i R_j \mid R_j R_i)$. Note that $R_i \leftrightarrow R_j \leftrightarrow R_k$ is equivalent to $(R_i R_j R_k \mid R_k R_j R_i)$.

Notes

1. http://finance.yahoo.com
2. Note that full cross-validation was not used, due to the temporal nature of the data. The validation was constrained so that the elements of the training data occurred at an earlier time than the elements of the testing set.
3. The software used was *Weka* (Witten & Frank, 2000) and *libsvm* (Chang & Lin, 2001).

References

Agrawal, R., & Srikant, R. (1995). Mining sequential patterns. In P. S. Yu & A. S. P. Chen (Eds.), *Eleventh International Conference on Data Engineering* (pp. 3–14). Taipei, Taiwan: IEEE Computer Society Press.

Chakrabarti, S., Sarawagi, S., & Dom, B. (1998). Mining surprising patterns using temporal description length. In A. Gupta, O. Shmueli, & J. Widom (Eds.), *Proc. 24th Int. Conf. on Very Large databases. VLDB* (pp. 606–617). New York, NY: Morgan Kaufmann.

Chang, C.-C. & Lin, C.-J. (2001). LIBSVM: A library for support vector machines. Software available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm.

Coello Coello, C. A. (2000). An updated survey of GA-based multiobjective optimization techniques. *ACM Computing Surveys, 32:2*, 109–143.

Coello Coello, C. A. (2001). A short tutorial on evolutionary multiobjective optimization. In E. Zitzler, K. Deb, L. Thiele, C. A. C. Coello, & D. Corne (Eds.), *First International Conference on Evolutionary Multi-Criterion Optimization* (pp. 21–40). Springer-Verlag, Lecture Notes in Computer Science No. 1993.

Das, G., Lin, K., Mannila, H., Renganathan, G., & Smyth, P. (1998). Rule discovery from time series. In, *Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining. KDD* (pp. 16–22).

Freitas, A. A. (2002) *Data Mining and Knowledge Discovery with Evolutionary Algorithms.* Springer-Verlag.

Halaas, A., Svingen, B., Nedland, M., Sætrom, P., Snøve, O., & Birkeland, O. (2004). A recursive MISD architecture for pattern matching. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 12:7*, 727–734.

Hetland, M. L. (2004). A survey of recent methods for efficient retrieval of similar time sequences. In M. Last, A. Kandel, & H. Bunke (Eds.), *Data Mining in Time Series Databases.* World Scientific (2004).

Hetland, M. L., & Sætrom, P. (2002). Temporal rule discovery using genetic programming and specialized hardware. In *Proc. 4th Int. Conf. on Recent Advances in Soft Computing. RASC.*

Hetland, M. L., & Sætrom, P. (2003a). A comparison of hardware and software in sequence rule evolution. In *Proceedings of the Eighth Scandinavian Conference on Artificial Intelligence. SCAI.*

Hetland, M. L., & Sætrom, P. (2003b) The role of discretization parameters in sequence rule evolution. In *Proc. 7th Int. Conf. on Knowledge-Based Intelligent Information & Engineering Systems. KES.*

Hilderman, R. J. & Hamilton, H. J. (1999). Knowledge discovery and interestingness measures: A survey. Technical Report CS 99-04, Department of Computer Science, University of Regina, Saskatchewan, Canada.

Höppner, F., & Klawonn, F. (2001). Finding informative rules in interval sequences. In *Lecture Notes in Computer Science*, vol. 2189 (pp. 125–134).

Interagon A.S. (2002). The interagon query language: A reference guide. http://www.interagon.com/pub/whitepapers/IQL.reference-latest.pdf.

Keogh, E., & Folias, T. (2002). The UCR, time series data mining archive. http: //www.cs.ucr.edu/∼eamonn/TSDMA.

Keogh, E. J., Lonardi, S., & Chiu, B. (2002). Finding surprising patterns in a time series database in linear time and space. In *Proc. 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining. KDD* (pp. 550–556).

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: The MIT Press.

Last, M., Klein, Y., & Kandel, A. (2001). Knowledge discovery in time series databases. *IEEE Trans, on Systems. Man. and Cybernetics, 31B:1*, 160–169.

Lin, J., Keogh, E., Lonardi, S., & Chiu, B. (2003). A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*. San Diego, CA.

Mannila, H., Toivonen, H., & Verkamo, A. I. (1997). Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery, 1:3*, 259–289.

Martin, R. D. & Yohai, V. (2001). Data mining for unusual movements in temporal data. In *Proc. KDD Workshop on Temporal Data Mining*.

Noda, E., Freitas, A. A., & Lopes, H. S. (1999). Discovering interesting prediction rules with a genetic algorithm. In P. Angeline (Ed.), *Proc. Conference on Evolutionary Computation (CEC-99)* (pp. 1322–1329). Washington DC, USA: IEEE.

Povinelli, R. J. (2000). Using genetic algorithms to find temporal patterns indicative of time series events. In *GECCO 2000 Workshop: Data Mining with Evolutionary Algorithms* (pp. 80–84).

Smyth, P. & Goodman, R. M. (1991). Rule induction using information theory. In G. Piatetsky-Shapiro & W. J. Frawley (Eds.), *Knowledge Discovery in Databases* (pp. 159–176). Cambridge, MA: The MIT Press.

Sun, R. & Giles, C. L. (Eds.) (2000) *Sequence Learning: Paradigms. Algorithms, and Applications*, No. 1828 in Lecture Notes in Artificial Intelligence. Springer-Verlag.

Sætrom, P. & Hetland, M. L. (2003a). Multiobjective evolution of temporal rules. In *Proc. 8th Scandinavian Conf. on Artificial Intelligence. SCAI.* IOS Press.

Sætrom, P. & Hetland, M. L. (2003b). Unsupervised temporal rule mining with genetic programming and specialized hardware. In *Proc. 2003 Int. Conf. on Machine Learning and Applications. ICMLA.*

Weiss, G. M. & Hirsh, H. (1998). Learning to predict rare events in event sequences. In R. Agrawal, P. Stolorz, & G. Piatetsky-Shapiro (Eds.), *Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining. KDD* (pp. 359–363). New York, NY: AAAI Press, Menlo Park, CA.

Witten, I. H. & Frank, E. (2000), *Data Mining: Practical Machine Learning Tools with Java Implementations*. San Francisco: Morgan Kaufmann. Software available at http://www.cs.waikato.ac.nz/∼ml/weka.

Zemke, S. (1998). Nonlinear index prediction. In R. N. Mantegna (Ed.), *Proc. Int. Workshop on Econophysics and Statistical Finance*, Vol. 269, pp. 177–183. Palermo, Italy: Elsevier Science.

Zitzler, E., Laumanns, M., & Thiele, L. (2001). SPEA2: Improving the strength pareto evolutionary algorithm. Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, Switzerland.

Received April 1, 2004
Revised October 6, 2004
Accepted October 6, 2004
Final manuscript October 6, 2004