Magnus Lie Hetland

Evolving Sequence Rules

Dr.ing.-thesis 2003:99

Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Computer and Information Science

NTNU

Evolving Sequence Rules
Magnus Lie Hetland

Norwegian University of Science and Technology
Dr.ing.-thesis 2003:99
ISBN 82-471-5653-9
ISSN 0809-103X
Department of Computer and Information Science
IDI report 2003:09
ISSN 1503-416X

Contents

List of Figures

Preface

> "Seize the moment of excited curiosity on any subject to solve your doubts; for if you let it pass, the desire may never return, and you may remain in ignorance." — William Wirt (1772–1834)

This dissertation is submitted to the Norwegian University of Science and Technology (NTNU) in partial fulfillment of the requirements for the degree Doktor Ingeniør.

The work contained herein has been performed at the Department of Computer and Information Science, NTNU, Trondheim, under the supervision of Professor Arne Halaas.

The dissertation is divided into two parts. The first part contains a brief introduction to the topics of the study, providing context for the reader. The second part is the main contribution of the study: a collection of six research papers. The papers have been slightly modified from their original form, mainly for the purpose of correcting orthographic and typographic errors. Their contents have not been modified.

Acknowledgements

This work would not have been possible without the help and support of many people; you all have my deepest gratitude. Although those worthy of my thanks are too many to list, I would still want to mention some. I would like to thank my supervisor, Arne Halaas, for sticking with me through the tumultuous four-year journey. I would also like to thank Pål Sætrom, co-author of five of the six papers in this collection; without his efforts I would never have been able to finish in time. I would like to thank Interagon AS for making their hardware available. I would like to thank the many helpful people in the administration and technical support here at the institute for helping me out of many troublesome situations. I would like to thank all of my co-workers for creating a stimulating and interesting

work-environment. I would especially like to thank Amund Tveit for many a lofty discussion on research in general and our work in particular. I would like to thank my friends, including all the members of Teaterlaget i BUL i Nidaros, for providing a counterweight to my academic work, and for cheering me up when I needed it. I would like to thank Eamonn Keogh, Kaushik Chakrabarti, and Ricardo Baeza-Yates for taking the time to answer my questions. Finally, I would like to thank my family: my Grandmother, Audhild Lie, for her interest in my work, and for the welcome distraction of many a late-night phonecall; my parents, Kjersti Lie and Tor M. Hetland, for knowing that I could do it and that I didn't really need to; and my sister, Anne Lie Hetland, for coming to my rescue when everything seemed hopeless. Thank you all.

Magnus Lie Hetland
Trondheim, May 8, 2003

List of Papers

Paper I Magnus Lie Hetland. 'A Survey of Recent Methods for Efficient Retrieval of Similar Time Sequences'. In Mark Last, Abraham Kandel, and Horst Bunke, editors, *Data Mining in Time Series Databases*. World Scientific, 2004. To appear.

Paper II Magnus Lie Hetland and Pål Sætrom. *Temporal Rule Discovery using Genetic Programming and Specialized Hardware*. In *Proceedings of the Fourth International Conference on Recent Advances in Soft Computing*, RASC, 2002.

Paper III Magnus Lie Hetland and Pål Sætrom. *The Role of Discretization Parameters in Sequence Rule Evolution*. In *Proceedings of the Seventh International Conference on Knowledge-Based Intelligent Information & Engineering Systems*, KES, 2003.

Paper IV Pål Sætrom and Magnus Lie Hetland. *Unsupervised Temporal Rule Mining with Genetic Programming and Specialized Hardware*. In *Proceedings of the 2003 International Conference on Machine Learning and Applications*, ICMLA, 2003.

Paper V Magnus Lie Hetland and Pål Sætrom. *A Comparison of Hardware and Software in Sequence Rule Evolution*. In *Proceedings of the Eighth Scandinavian Conference on Artificial Intelligence*, SCAI, 2003.

Paper VI Pål Sætrom and Magnue Lie Hetland. *Multiobjective Evolution of Temporal Rules*. In *Proceedings of the Eighth Scandinavian Conference on Artificial Intelligence*, SCAI, 2003.

Context

Chapter 1

Introduction

"Let us watch well our beginnings, and results will manage themselves."
— Alexander Clark

"Computer: Let the science begin!" — Dexter, *Dexter's Laboratory*

This chapter gives a brief overview of some of the main concepts that form the basis for the material in the rest of the thesis, that is, time series and data mining, followed by a short survey of the current state of the art in the area of sequence mining. After this, the aims of this study are outlined, the structure of the thesis is summarized, and, finally, the abstracts of the papers are reproduced, for the convenience of the reader. These, along with the full papers, can also be found in the second part of the thesis.

1.1 An Introduction to Time Series

This thesis deals with sequence mining in general, but has a bias toward time series. A *time series* (or, equivalently, a *time sequence* or *temporal sequence*) can be defined as a set of observations x_t, each one being recorded at a specific time t. The observations may be categorical, but are often assumed to be real numbers. The time series is an abstraction with wide applicability, from geology to medicine, from economics to psychology. It occurs whenever some measurement or observation is repeated over a period of time.

One well-known example is stock prices; here, the set T of time stamps might be the set of days in the year, and each value x_t would represent the stock price of a given company at time t. Another example is the *electrocardiogram* (ECG or

EKG). An ECG is a recording of the electrical activity in the heart, and is used in the investigation of heart disease. Figure 1.1 shows a segment of a typical ECG series.

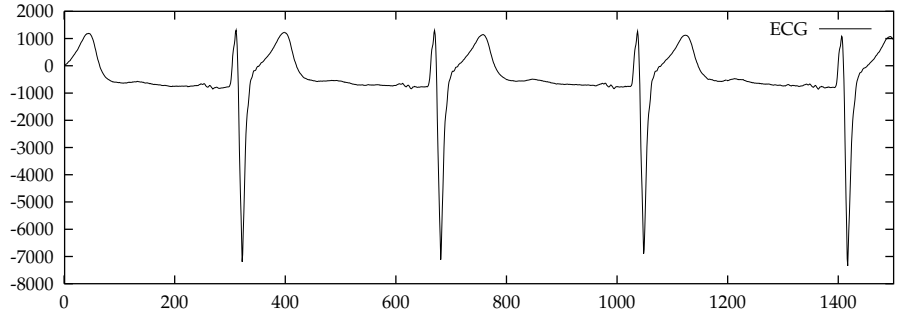

Figure 1.1: An ECG time series

This plot illustrates general behaviour that is common to many real-world time series: It seems to consist of one relatively stable and slow-moving component (which, in this example, happens to be highly periodic) and one less influential, highly random component. It is often useful to model the time series as a sum of these components, that is,

$$x'_t = x_t + \varepsilon_t \ ,$$

where x'_t is the observed value at time t, x_t is the "actual" or "ideal" value, and ε_t is simply *noise*. Sequence models are discussed in more detail in Chapter 2.

As computerized support for gathering and storing time series data is becoming more and more advanced, the need for analysis techniques is growing. Such techniques should be able to extract useful, and hopefully surprising, knowledge from the raw data. This is the aim of data mining, and the broader field of knowledge discovery.

1.2 The Basic Principles of Data Mining

Data mining is the specific operation of extracting some form of structured knowledge from data. It is often viewed as a part of a larger process, called *knowledge discovery*, which can incorporate such stages as data preprocessing and presentation of the results to the user (see Figure 1.2).

The goal of data mining is to discover *useful* and *unknown* (possibly even surprising) relationships in the data. If the relationships aren't useful, the endeavour

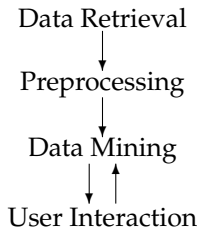

Figure 1.2: The structure of the knowledge discovery task

serves no real purpose, and if the relationships aren't unknown, there is little point in the exercise to begin with. In the following, our method will be placed in the context of data mining methods in general. Various specific aspects of the method (as well as other methods) will be discussed in greater detail in the following chapters.

Hand et al. [2001] describe a reductionist taxonomy of data mining methods. According to this taxonomy, a data mining method consists of the following key components:

Problem. What is the purpose of the data mining? Two broad problems are *description* and *prediction*.

Potential solutions. The structures we are looking for. To broad categories are *models*, that is, structures that describe all of the data, and *patterns*, structures that describe only parts of the data.

Score function. A function that measures how useful or interesting a structure (model or pattern) is.

Search method. The general method used for optimizing the score functions. In some cases the optimization problem may be solved directly, but in most cases some form of search is performed.

Data management method. If primary memory is available to hold all of the data, or the mining method can be applied incrementally with few passes through the data, data management may not be a problem. Otherwise, issues of storage and retrieval may be quite relevant.

It may be instructive to analyze the method presented in this thesis according to this model. Our problem is to find temporal rules from time series. In other words, the problem is mainly predictive, and we are seeking patterns, rather than models. The score function may vary, according to the specifics of the application. We may use a score function that describes the predictive power of a rule (see

Papers II and III) or how interesting or surprising a rule is (see Paper IV). It is even possible, through multiobjective optimization, to use several score functions at once (see Paper VI). Note that whether the the focus is on predictive power or interestingness, rule validation is key. For prediction, techniques such as using an independent test set are appropriate; however, for interestingness, a human expert must be employed for validation.

The two aspects of our method that probably most clearly differentiate it from other, related methods are the search method and the data management method.

To search for solutions we use the form of heuristic search known as *evolutionary computation*. Although this method has been gaining popularity in the data mining community in recent years, to our knowledge, it has not previously been applied to sequence rule mining. One reason for this may be the lack of appropriate data management solutions. When using heuristic search to find useful patterns in a data set, some form of retrieval mechanism must be in place, to locate the occurrences of potential patterns. Relational databases are well suited for this task when looking for so-called *association rules* [Hipp et al. 2000], but no data base system has been available that is equally suited to searching for sequential patterns or sequence rules.[1] Our contribution lies in combining the heuristic search for sequence rules with the proper data management techniques, to form a full-fledged data mining method.

We use two data management techniques: Signature based similarity retrieval, and pattern based retrieval, using both a specialized pattern matching chip and existing software. Although the former kind of sequence retrieval has attracted a great deal of research interest, we have not been able to find any research on using it in in evolutionary rule mining. The pattern matching chip is a relatively recent invention [Interagon AS 2000], and has, until now, not been used for data mining purposes. It has, however, been used to evolve patterns for purposes such as classification. See Chapter 3 for details on these retrieval methods.

1.3 The State of the Art

This section describes briefly some of the more central work that has been done in the areas of sequence rule mining and evolutionary data mining. The ideas touched upon here will be expanded on later: Chapter 4 discusses some of the data mining techniques that are most relevant to this work, and Chapter 5 describes existing methods in evolutionary data mining.

Rule mining has mainly been tackled as a problem of combinatorial optimization, and solved with purely algorithmic techniques. Hipp et al. [2000] give a readable

[1] We consider the terms 'temporal rule' and 'sequence rule' to be equivalent for the purposes of this study.

overview of the basic principles underlying one of the common approaches. Simply put, the approach relies on constraining a rule property called *support*. The support of the rule is roughly the relative number of occurrences of the rule (its relative frequency), or, in other words, the estimated probability that the rule will apply for any given data item. It is assumed that for a rule to be of any use, its support must exceed a given minimum, which may often be supplied by the user. Although support in itself does not by any means guarantee useful or interesting rules, it does allow for a powerful pruning of the search space, as most possible rules will not have sufficient support.

For some rule formats, the pruning may be performed directly. However, this simple approach does not easily generalize to more complex rule formats. The idea that makes it possible to generalize the counting method is the structure of the rule space, that is, the set of possible rules. For many rule formats, such as those presented by Agrawal and Srikant [1995] and Mannila et al. [1997], this space is structured by a generalization relation. In other words, there is a partial order "is more general than" defined on the rule space. This order forms a lattice structure, which makes it possible to prune large segments of the search space without having to explicitly count all the rules. If a given rule has too low support, all the rules that are more specific must necessarily also have too low support. This method was first used to find association rules in relational data, but its application to sequential data has since been refined. Adamo [2001] gives a description of several algorithms.

The use of evolutionary algorithms in data mining is relatively recent, and described thoroughly by Freitas [2002]. The method is based on using rules (or sets of rules) as individuals in an evolutionary algorithm, and describing desirable qualities such as accuracy or interestingness by means of a fitness (or objective) function. Existing work (see Freitas [2001] for a survey) focuses on a traditional data mining context, especially tabular data.

1.4 Aims of the Study

It is clear that the traditional data mining approach has limitations when it comes to rule complexity. These limitations can, in principles, be overcome with evolutionary data mining, provided that an information retrieval method exists that can process the rules as queries. This study aims to examine the use of evolutionary data mining on sequence data. In much of our work we have used a piece of specialized pattern matching hardware, the *pattern matching chip* (PMC), but we also show how the methods may be used with software based retrieval methods.

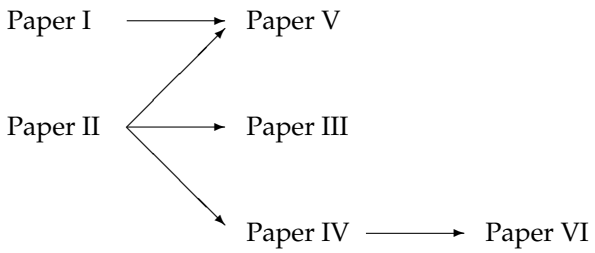

Figure 1.3: How the papers relate to each other

1.5 Thesis Structure

This thesis is structured as follows: The first part (Chapters 1 through 6) gives a context for the main contribution of the thesis, that is, the papers in the second part. Chapter 2 gives an overview of sequence learning, sequence prediction, and the related issue of validation; Chapter 3 describes some basic mechanisms for sequence retrieval; Chapter 4 outlines the process of data mining, specifically focusing on pattern and rule mining; Chapter 5 is a short introduction to the use of heuristic search, and, in particular, evolutionary computation, in data mining; Chapter 6 summarizes the method developed and used in the second part; and, finally, Chapter 7 gives some concluding remarks. The third part contains appendices, with technical details that are slightly peripheral to the main text. Appendices A and B have been taken from Papers I and IV, respectively.

Figure 1.3 shows how the papers relate to each other: Paper I gives a survey of methods for similarity based sequence retrieval; Paper II describes how specialized hardware can be used in evolutionary data mining to perform basic prediction tasks; Paper III is based on Paper II, and investigates the effect of the discretization process on the prediction results; Paper IV presents a novel method for evolving interesting rules using a method similar to that in Paper II, but using a new fitness measure; Paper V is based on Paper I, and shows how it is possible to evolve rules using existing techniques for time series indexing and regular expression matching; finally, Paper VI examines the applicability of multiobjective evolution to rule mining. The following section collects the paper abstracts.

1.6 Paper Abstracts

Paper 1. Time sequences occur in many applications, ranging from science and technology to business and entertainment. In many of these applications, searching through large, unstructured databases based on sample sequences is often desirable. Such similarity-based retrieval has attracted a great deal of attention in recent years. Although several different approaches have appeared, most are based on the common premise of dimensionality reduction and spatial access methods. This paper gives an overview of recent research and shows how the methods fit into a general context of signature extraction.

Paper 2. Discovering association rules is a well-established problem in the field of data mining, with many existing solutions. In later years, several methods have been proposed for mining rules from sequential and temporal data. This paper presents a novel technique based on genetic programming and specialized pattern matching hardware. The advantages of this method are its flexibility and adaptability, and its ability to produce intelligible rules of considerable complexity.

Paper 3. As raw data become available in ever-increasing amounts, there is a need for automated methods that extract comprehensible knowledge from the data. In our previous work we have applied evolutionary algorithms to the problem of mining predictive rules from time series. In this paper we investigate the effect of discretization on the predictive power of the evolved rules. We compare the effects of using simple model selection based on validation performance, majority vote ensembles, and naive Bayesian combination of classifiers.

Paper 4. 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.

Paper 5. Sequence rule mining is an important problem in the field of data mining. Many algorithms have been devised that are based on counting candidate rules and excluding those with low support. Recently, the techniques of heuristic search, and evolutionary algorithms in particular, have been applied to various data mining problems, including sequence mining. In our previous work we have used specialized hardware to make mining certain rule formats feasible. In this paper we compare the performance of this hardware with realistic software alternatives.

Paper 6. In recent years, the methods of evolutionary computation have proven themselves useful in the area of data mining. For rule mining, several objective functions have been used, relating to both accuracy and interestingness in general. However, when searching for rules or patterns in a data set, several conflicting objectives will often be present. As the ultimate goal of data mining is to discover unexpected, useful knowledge, it may not be feasible to prioritize these objectives *a priori*. Simply constructing an aggregate fitness function in these cases could be seen as a more or less *ad hoc* solution. In this paper we propose an alternative: Using well-established multiobjective evolutionary algorithms to evolve a Pareto optimal set of rules.

1.7 Other Publications

Other works authored or co-authored while working on this thesis include the papers listed in the bibliography as Tveit and Hetland [2003] and Tveit et al. [2003], and the book listed as Hetland [2002].

Chapter 2

Sequences and Sequence Learning

"Learning is not compulsory ... neither is survival."
— W. Edwards Deming (1900–1993)

"One pound of learning requires ten pounds of common sense to apply it."
— Persian Proverb

Sequences are ubiquitous, in nature as in science. Whether they are modelled as a fully ordered sets of objects, or as partial maps from the set of integers (positions) to a set of symbols (the alphabet), the basic interpretation is the same. The values in a sequence are inherently ordered, and it is quite natural to interpret them as being ordered in time. If each value is given a time stamp (that is, in a way the indices have become real values rather than integers), the result is a *temporal sequence*. The values themselves may be taken from a discrete set of possible events, giving rise to an *event sequence*, or they may be the (real) values of a variable at the given point in time, leading to a *time series*.

In this study, we have in many cases worked with time series, although the nature of our method has made it necessary to discretize these, with the result being an event sequence emulating the behaviour of the time series. The method is, of course, also applicable directly to event sequences.

2.1 Sequence Models

The task of modeling a sequence can be compared to lossy compression. One wants a relatively simple model that describes the behaviour of the sequence, while accepting some loss of precision in the simplification process. Such models have several uses, including gaining a better understanding of the system underlying the sequence. This section will give a brief description of two broad model types: the iterative statistical type and the syntactic type.

An general iterative model has the form

$$x_t = f(x_1, \ldots, x_{t-1}, \varepsilon_t) \ , \tag{2.1}$$

where ε_t is a (random) noise term. In other words, each value in the sequence is stochastically dependent on the earlier values. If the value only depends on the previous value, that is, x_t, as well as the noise term, we have what is known as a Markov model. If the observed value is actually a stochastic variable with its distribution conditioned on x_t, we have a so-called *hidden* Markov model (HMM). Efficient methods exist for fitting HMMs to sequence data, and these methods have been quite successful in many application areas.

For the general machine learning problem of classifying vectors, which can be viewed as a special case of Equation 2.1, many methods exist that perform well, for example, support vector machines [Rüping 2001]. Section 2.2 discusses this in more detail.

Although the second type of model, the syntactic one, can in many cases be seen as equivalent to the iterative statistical one, the perspective is different. Instead of making each value a stochastic function of previous ones, a syntactic model describes a set of legal sequences, a *language*. Algorithms can be constructed for recognizing legal sequences, and syntactic models can thus be used for classification. A well-known class of grammars is *regular expressions*, which describe so-called regular languages. Regular expressions are built recursively from the basic operations shown in Table 2.1: Atomic values, concatenation of expressions, alternative expressions, and closure, that is, indefinite repetition of an expression. Such expressions can capture sequence patterns that can be described by a finite automata, and are the basis for the rules we deal with in our work. Our rule format is much richer than the basic regular expression format, but the difference is mainly one of efficiency and convenience, not of formal expressiveness.

Syntactic models can include uncertainty, in the form of stochastic grammars. Although such models have met with success we will chiefly be concerned with deterministic syntactic models in the following, mainly because it is what we use in the work presented in the second part. The reason for this is that the pattern matching hardware we use currently has no support for stochastic models directly. Instead, uncertainty is incorporated into the syntactic model, as alterna-

Table 2.1: Definition of regular expressions

Description	Examples
Atomic values	a, b, c
Concatenations	ab, cbc
Alternatives	$a \mid b$, $ab \mid bc$
Closure	ab^*, $a(b \mid c)^*a$

tives.

2.2 Principles of Sequence Learning

Before tackling the particular problems of sequence learning, we will devote a few paragraphs to machine learning in general. The following broad definition gives a clear picture of what machine learning is all about:

Definition 2.1 (Mitchell 1997). *A computer program is said to* learn *from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*

In other words, a machine learning method consists of a generic kind of computer program that is able to improve its performance at some task when supplied with example data.

A collection of machine learning techniques especially geared toward sequences can be found in Sun and Giles [2000]. Sun [2000] outlines four main problems in sequence learning:

Sequence prediction. Given a sequence history, predict what the next sequence element will be.

Sequence generation. Given a sequence history, generate the next element in the sequence.

Sequence recognition. Given a sequence, determine whether it is legitimate or not.

Sequential decision making. Given a sequence, determine what action should be taken.

Depending on their detailed formulations, these tasks may have significant overlaps. For example, prediction and generation are essentially the same task. In our

work, the application is prediction, but in many cases we only wish to predict the *trend* of a time series — whether it will go up or down. In other words, this is essentially the same as sequence recognition.

If we assume that an iterative statistical sequence model with a fixed-length history applies to our problem, we may use many existing machine learning methods, that is, any method that will allow us to classify real vectors. While this includes most of the well-known machine learning methods [Mitchell 1997], a majority of papers on time series prediction seem to use some form of nonlinear discriminant method, such as support vector machines or artificial neural networks.

Because our goals go beyond sheer predictive power (see Section 2.4), we work with a rule format that is readable to humans. That is, we produce rules whose antecedents are syntactic classifiers. These syntactic classifiers may be expressed as simple grammars in a grammar specification language, such as regular expressions. In our case, this language is IQL [Interagon AS 2002]. The learning technique we use is random heuristic search, or, more specifically, genetic programming. This is discussed in more detail in Chapter 5.

2.3 Bias, Variance and Validation

For the sake of the current discussion, and without loss of generality, assume that our machine learning task can be defined as follows: Given a finite *training set* $T = \{(\boldsymbol{x}, y)\}$, find a function f such that $f(\boldsymbol{x}) = y$. This formulation is flawed, because there is an infinite number of arbitrary functions that satisfy this criterion, and as we already have access to the data, the function may not be of much use to us. If the function definition is in some way simpler than the data set itself, this could be seen as a form of compression, but for our models to be of any use in real-world prediction applications, the problem must be reformulated.

A more useful formulation is the following: Given a training data set $T = \{(\boldsymbol{x}, y)\}$ drawn from a probability distribution $P(\boldsymbol{x}, y)$, construct a function f such that $f(\boldsymbol{x}')$ is as close as possible to y' for all elements $(\boldsymbol{x}', y')$ of a *validation* (or *testing*) data set $V = \{(\boldsymbol{x}', y')\}$, also drawn from P. If y is taken from a discrete set of *classes*, this is called *classification*; if it is taken from an arbitrary set of possible values (such as $\mathbb{R}$), it is called *regression*.

This reformulation has two important implications for the learning task: It affects the way models are constructed and the way they are tested.

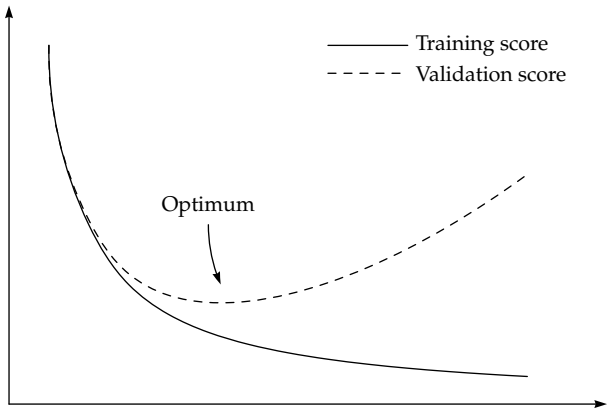

Figure 2.1: Illustration of early stopping

2.3.1 Model Selection

When data are seen as the result of a stochastic process of some kind, it may no longer be desirable for a model to completely reproduce the behaviour of the training data — the model should also perform satisfactorily on other data produced by the same process. In other words, the model should embody a *generalization* of the behaviour observed in the training data.

For some machine learning methods (such as support vector machines) it is possible to formulate and satisfy certain optimality criteria for generalization. This is, however, not possible for machine learning methods in general. One widely used alternative is a form of model selection, which resembles the technique of *early stopping*, often used when training artificial neural networks.

Early stopping works as follows: During the process of training a neural network, its performance is regularly validated against a separate data set. This data set is not used in the training itself, and can therefore not contribute to overfitting (explained below). If this validation error is plotted alongside the training error, something similar to Figure 2.1 may be observed. While the training error continues to decrease as the model adapts to the peculiarities of the training data, the validation error actually *increases* after a while. As the model becomes fine-tuned (overfitted) to the training data, it no longer generalizes well, and its performance on other data sets drawn from the same distribution as the training data deteriorates.

The danger of overfitting is present for all models that are sufficiently flexible. In the gradient descent algorithm often used to train an artificial neural network, early stopping is usually implemented by keeping the previous copy of the net-

work, and stopping the training process when the validation error starts to increase. However, this model selection scheme can be generalized to any situation where you have several candidate models. In particular, in evolutionary algorithms one might keep the best individual from each generation (or the k best individuals, for some fixed number k), and select among them, based on the validation score.

Note, however, that while this method tends to favour general models, it is not fool-proof. If enough candidates are generated, model selection may result in a model that is overfitted to the validation data.

2.3.2 Testing

Using the training error (or any model score on the training data) as a measure of rule quality will give overly optimistic results. For a sufficiently flexible model type, it is possible to get perfect results on the training data, even though the model may not generalize to other data at all. Model selection based on independent validation helps, but even using this validation score will have a positive bias — after all, the model was chosen among the candidates using this score as the criterion. To actually test the validity (or quality) of a model, a *third* data set must be used: the testing data set. This data set must be independent of both the training data and the validation data.

The simplest approach is to use (for example) one third of the data for training, one third for validation and model selection, and one third for testing. This approach may be satisfactory in many cases, but it does have two weaknesses:

1. We only get to test one model, and the quality of this model may not be representative of what the method produces in general.
2. If the amount of data is small, the quality estimate may not be accurate.

One of the most well-known techniques that addresses these issues is *cross-validation*. In a k-fold cross-validation, the data are partitioned into k equal parts (where 10 is a typical value for k), and each of these is, in turn, used as the test data set for a model developed on the rest of the data. (If model selection is used, each of the k folds is partitioned into a training set and a validation set.) The model performance is then averaged over the k folds.

This means that there will be k models produced, and the quality estimate is their average. This will give a more representative estimate of the quality of the method itself, rather than of one specific model. The entire data set will also be used for testing, meaning that the there will be less variability in the estimate. In addition, by using only a small portion of the data for testing in each fold, more

data can be used in training. This will typically result in better models. A version of cross-validation that takes this to the extreme is *leave-one-out* cross-validation, where $k = N$, for N data items. In other words, only one data item is used for testing in each fold.

2.4 Learning Versus Mining

There is considerable overlap between the areas of machine learning and data mining. Many of the techniques used in machine learning can also be used in data mining. There are be many ways of drawing a dividing line between the two, but in this study we will use the following distinction:

- Machine learning tries to create a model that describes the full behaviour of a system.
- Data mining searches for patterns of behaviour that may account for only parts of the system behaviour.

Note that this distinction stands in contrast to the description of data mining given by Hand et al. [2001]. Hand et al. include model-building techniques such as linear regression as part of the field of data mining. We do not object to their definition, but believe that the distinction drawn here may put the data mining techniques introduced later in a clearer context.

Data mining is described in more detail in Chapter 4.

Chapter 3

Sequence Retrieval: Similarity Based and Pattern Based

> "We're about as similar as ... two completely dissimilar things in a pod." — Blackadder III, Episode 6

This chapter describes two important forms of sequence retrieval. The first, similarity based retrieval, is the dominant approach for time series databases. For textual or discrete data, such as biological sequences, the second form, pattern based retrieval, is often more applicable. Both forms of retrieval can be used in evolutionary sequence mining.

Baeza-Yates and Ribeiro-Neto [1999, p. 23] describe the following framework for formally characterizing information retrieval systems:

Definition 3.1 (Baeza-Yates and Ribeiro-Neto). *An information retrieval* model *is a quadruple* $(\mathbf{D}, \mathbf{Q}, \mathcal{F}, R(q_i, d_j))$ *where*

(1) $\mathbf{D}$ *us a set composed of logical views (or representations) for the documents in the collection.*

(2) $\mathbf{Q}$ *is a set composed of logical views (or representations) for the user information needs. Such representations are called* queries*.*

(3) $\mathcal{F}$ *is a framework for modeling document representations, queries, and their relationships.*

(4) $R(q_i, d_j)$ *is a ranking function which associates a real number with a query* $q_i \in \mathbf{Q}$ *and a document representation* $d_j \in \mathbf{D}$*. Such ranking defines an ordering among the documents with regard to the query* q_i*.*

In the following, this framework will be used to describe the main features of, and differences between, similarity based and pattern based retrieval.

3.1 Similarity Based Sequence Retrieval

Similarity based sequence retrieval is discussed more thoroughly in Paper I; the following section is merely a brief overview.

Similarity based retrieval (also called *content based retrieval*) may be defined in terms of Definition 3.1 as follows:

Definition 3.2. *A* similarity based retrieval model *is an information retrieval model where*

(1) *The query objects in* $\mathbf{Q}$ *are of the same form as the document objects in* $\mathbf{D}$.

(2) *The ranking function* $R(q_i, d_j)$ *takes the form of a* similarity function.

Here are some example applications of similarity based retrieval systems:

- A system where the user hums or whistles a tune and similar tunes are retrieved from a music database [Arentz et al. 2003].
- A system where the user draws a rough sketch of a picture and similar pictures are retrieved from a database of photographs [Del Bimbo 1999].
- A system where the user can specify a word and the positions of similar words in a text database are returned [Navarro 2001a].

Definition 3.2 states that the query objects are of the same form as the document objects. This simply means that it is possible to measure a degree of similarity between them. For example, in a query-by-humming system, the hummed query may not seem to be of the same form as the music in the database. Still, the hummed tune may be compared for similarity to the tunes in the stored music. Also, in many systems, the query may be compared to *parts* of the stored objects. For example, a textual query may be matched to similar substrings of stored textual documents. Whether this is allowed or not has significant practical effects on the implementation of the system, but the main concepts remain the same.

The rank function $R(q_i, d_j)$ used in similarity based retrieval is some form of *similarity function*. Some common examples are described in Section 3.1.1. Although ranking may be important, the main mechanism of many similarity based retrieval systems is an index that enables them to filter out objects whose similarity to the query object falls beneath a certain threshold. By inverting the notation, and

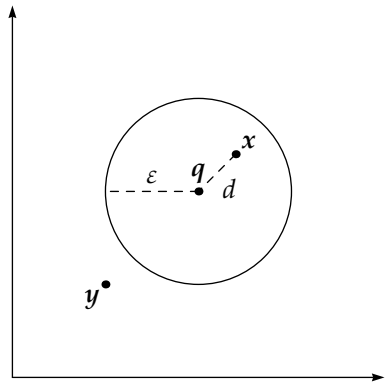

Figure 3.1: Similarity retrieval

using *dissimilarity functions* (some non-increasing transform of a similarity function) instead of similarity functions, this query form has an intuitive, geometric interpretation, as Figure 3.1 shows. (Figure 3.1 is taken from Paper I, page 50, and is repeated here for convenience.)

In this figure, it is assumed that the query and document objects are points in two-dimensional Euclidean space (that is, the Euclidean plane), and that the dissimilarity function is Euclidean distance, L_2. The circle in the figure encloses the subspace described by the inequality

$$d(\boldsymbol{q}, \boldsymbol{x}) < \varepsilon \ .$$

In general, if $\mathbf{Q} = \mathbf{D} = \mathbb{R}^k$ and $R(q, d) = L_2(q, d)$ the objects returned by the system will be all the points falling inside the *hypersphere* with the query at its centre, and the query threshold as its radius.

Euclidean distance, however, is not the only dissimilarity measure, by far. The following section briefly describes some common measures.

3.1.1 Similarity Measures

As the retrieval approach is called *similarity based*, it is fitting that the quality measure used is called a *similarity measure*. However, the interpretation of the method is easier if we use *dissimilarity measures*, which are simply non-increasing transforms of similarity measures. Intuitively, they are the inverse of similarity measures.

The most well-known and well-defined dissimilarity measures are the so-called *distance measures* or *metrics*. They are nonnegative, symmetric real-valued func-

tions which satisfy the *triangle inequality*, that is,

$$d(\boldsymbol{x}, \boldsymbol{y}) + d(\boldsymbol{y}, \boldsymbol{z}) \geq d(\boldsymbol{x}, \boldsymbol{z}) \ . \tag{3.1}$$

In addition, a metric satisfies the conditions that $d(\boldsymbol{x}, \boldsymbol{y}) = 0$ if and only if $\boldsymbol{x} = \boldsymbol{y}$. Indexing methods such as those described in Section 3.1.3 often place similar restrictions on the dissimilarity measures used, although they may, in fact, be slightly less strict (see Section I.2.3). For further details on some distance measures (or, more accurately, dissimilarity measures) often used when comparing sequences, such as *edit distance* and *dynamic time warping*, see Appendix A.

In the following, the word *distance* will be used loosely, encompassing dissimilarity measures in general.

3.1.2 Signature Based Retrieval

In many cases, calculating the distance between two objects can be costly. For example, to calculate the Euclidean distance between two sequences of length n, $O(n)$ operations are needed. In a database of m such sequences, a similarity query (with full-sequence matching) would require $O(mn)$ operations. At first glance, it might seem that this work is unavoidable. However, if some preprocessing is permitted, the search may be speeded up considerably by the use of *signatures*.

Consider what would happen if we could extract some new object, a signature, from each sequence, where the signatures have the following properties:

- Calculating the distance between two signatures is less costly than calculating the distance between two original objects (sequences).
- The distance between two signatures never overestimates the true distance between the original objects.

Because the distance measures are not overestimated, and we use an upper distance limit when retrieving objects, we are guaranteed that we won't have any *false dismissals*, that is, objects that are not retrieved but that should have been retrieved. If the distance between signatures closely approximates the distance between original objects, we will also have few *false alarms*, that is, objects that are retrieved but that should not have been retrieved. Since the signature distance can be calculated more efficiently than the original distance, we will most likely experience a speedup.

For example, Euclidean distance is preserved in the frequency domain (Parseval's Theorem [Shatkay 1995]). Therefore, by replacing each sequence with its Fourier transform, we will not have modified the search process appreciably. However, if we stem the Fourier transform, and keep only the strongest coefficients in the

power spectrum, we will have relatively short signatures that may represent the original sequences quite well. Also, we are guaranteed that the distance is underestimated, because we have only removed positive terms. If the Fourier signatures have k elements, the query time will be $O(mk)$ instead of $O(mn)$, if we ignore the time needed for calculating the Fourier spectrum of the query itself. Although a second step is needed to weed out the false alarms, this may be a significant speedup.

The main advantage of signature extraction, however, is not this speedup, since the query time complexity is still linear in the database size. The real power of this approach comes with the ability to *index* the signatures.

3.1.3 Signature Indexing

The core idea of similarity based sequence indexing is that of using *spatial access methods* to index signatures of fixed dimensionalities. Spatial indexing is a research field in its own right, and many indexing methods exist, some of the most commonly known being the simple multidimensional binary search tree (the *kd*-tree), the *R*-tree [Guttman 1984] and its descendants, the R^+-tree [Sellis et al. 1987] and the R^*-tree [Beckmann et al. 1990], and the *hB*-tree and its descendant the hB^Π-tree [Evangelidis et al. 1997]. A relatively recent tree structure with good performance for higher dimensionalities is the Hybrid Tree [Chakrabarti and Mehrotra 1999]. For a thorough survey of the basic methods, see Gaede and Günther [1998]. In general, such indexing methods give sublinear running times for similarity-based queries. However, existing methods do not deal well with high dimensionalities. Therefore, it is not feasible to index time series directly, with each timestamp as a separate dimension.

3.1.4 Signature Types

Many signature types have been proposed for similarity based retrieval of time series. Some important ones are:

- Spectral signatures, with coefficients taken from the Fourier transform of the sequence.
- Piecewise constant approximations, where segments of the time series are approximated by constant values, such as the average. The signature is a vector consisting of one value for each of these segments.
- Landmark signatures, where *landmarks*, or points of significant change (or some other significance) are encoded as a fixed-dimensional vector.

These, and other signature types, are discussed in more detail in Paper I. Signature indexing methods have been developed to deal with many distance measures, including the various L_p metrics and dynamic time warping, for example (see Appendix A for details on these distance measures).

3.2 Pattern Based Sequence Retrieval

Pattern based sequence retrieval may be defined as follows:

Definition 3.3. *A* pattern based retrieval model *is an information retrieval model where*

(1) *The query objects in* **Q** *are general* patterns, *which describe properties of some of the documents in* **D**.

(2) *The ranking function* $R(q_i, d_j)$ *is a predicate which accepts those documents that satisfy the query pattern.*

In other words, in a pattern based retrieval system, the queries Q are not sample objects that are compared to those in the database. Rather, the queries are *descriptions* or *specifications* of what the objects (sequences) may look like. These queries are normally specified in some form of *query language*. The basic task of the ranking function R will be to separate those sequences that satisfy the query criterion from those that don't. Any additional ranking will be highly application dependent.

The query language can be seen as a a *meta language*, since the expressions of the query language specify a grammar for the language that contains the documents the user wants. Simple query languages simply specify segments of the desired documents, that is, certain words they must contain. Boolean operators may be used to specify various permitted combinations of such words. Other operators may specify their relative positions; for example, one might specify that one word be close to another. More complex patterns can be specified with regular expressions, as described in Section 2.1.

There is no common paradigm for pattern based retrieval, such as there is for similarity based retrieval. There are basic indexing methods for word and phrase search and for the use of Boolean operators, but when it comes to approximate search, regular expressions and more complex patterns, no general method is available. There are, however, basic tools, such as the *suffix tree* and the *suffix array*, both of which are used in several specialized pattern matching algorithms, to avoid redundant calculations. Some systems for matching regular expressions are discussed in Paper V.

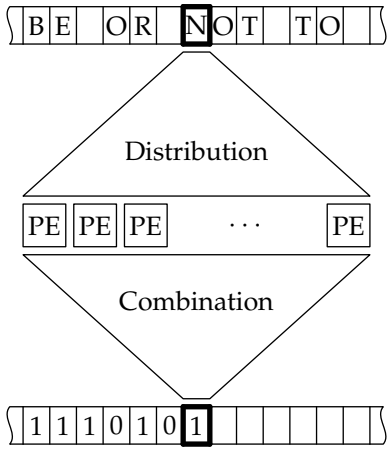

Figure 3.2: High-level architectural view of the PMC

3.2.1 The Pattern Matching Chip

In our work we have used a recent solution to the pattern based retrieval problem: The Interagon Pattern Matching Chip [Interagon AS 2000]. It has a richness of functionality beyond any comparable software solutions, and most of this functionality is available through the query language IQL [Interagon AS 2002]. This functionality includes substring matching, generalized Boolean operations, ordering requirements, textual distance requirements, regular expressions (with some minor restrictions), alphanumerical range matches, and approximate matches. These queries may be performed in massive parallel at a rate of about 100 MB/s.

The basic structure of the PMC is shown in Figure 3.2. A more detailed figure (Figure II.1) can be found in Section II.2.1. The PMC receives binary data as a sequence of bytes, which are distributed to a set of *processing elements*. This distribution may be done either sequentially or in parallel at all levels in a *distribution tree*, which makes a multitude of configurations possible. The processing elements themselves are configured with single bytes and compare these with the incoming data bytes in each clock cycle. They can be configured with several comparison operators, such as equality, inequality, and ordinal comparison. If the comparison for a processing element succeeds, the element reports a hit. Each element may be configured to keep reporting such a hit for a number of clock cycles. The hits are reported to the *result gathering tree*, where each node is a function (a simple Boolean or arithmetic function) that combines the results from its children, until a single Boolean value is reported at the root (at the bottom of the figure). Thus, for a given configuration (representing a query pattern), the

PMC will report *true* (a hit) or *false* (a miss) for each location in the incoming data stream.

Chapter 4

Finding Patterns in Data

> "The most exciting phrase to hear in science, the one that heralds new discoveries, is not 'Eureka!' (I found it!) but 'That's funny . . . '."
> — Isaac Asimov (1920–1992)

> "The beginning of knowledge is the discovery of something we do not understand." — Frank Herbert (1920–1986)

The material in this chapter builds on the basic principles laid out in Section 1.2 and go into a bit more detail about the specifics of mining patterns, especially in sequential data.

4.1 Rules

As mentioned in Chapter 1, there are two main problems in data mining: description and prediction. The structures one looks for may also be divided into two types: models and patterns. Models describe the features and behaviours of the entire data set, whereas patterns describe features or behaviours of *parts* of the data set. In our case, we are interested in *predictive patterns*, that is, *rules*.[1]

Assume we are working with relational data, more specifically, a table of n rows with p attributes. Then our rules might have the following form:

if: property p_i takes the value x
then: property p_j takes the value y.

[1] The rules in Paper II may be seen as models, because they describe the entire data set.

For continuous data, or for discrete data with a large value range, we may create *bins* or intervals as a preprocessing step.

This simple rule format is easily extended to include the use of arbitrary Boolean expressions in the condition.

Rules in sequential data work in the same manner. A simple rule format, which is used by Das et al. [1998], is the following:[2]

if: the symbol at position x is a
then: the symbol at some position y is b,
where $y - x \leq t$.

More complex rule formats are discussed in Section 6.3.

Once we have a rule format that may or may not apply for various data instances (rows in the tabular data or positions in a sequence) we will want to find rules that are somehow interesting or useful. Section 4.2 describes two very basic quality measures, support and confidence. A few more are mentioned in Section 5.3, while the specific quality measures used in our study are described in Section 6.4. Section 4.3 describes the mainstream method for mining rules: counting occurrences.

4.2 How Good is a Rule?

When looking for rules, one will often be interested in those that occur more frequently. The *support* of a rule is a measure of just that: the relative number of data instances where the rule applies and is correct, that is, the fraction of the data where both the antecedent (the predictor condition) and the consequent (that which is to be predicted) of the rule are true.

However, even a rule with relatively high support might be quite inaccurate. For example, it might have a support close to 0.5 and still give rise to wrong predictions more than half of the time. This motivates another quality measure, *confidence*, which is the conditional probability of the consequent being true, given that the antecedent is true. The confidence is usually a maximum likelihood estimate, that is, the conditional relative frequency.

These quality measures, and many others (see, for example, Sections 5.3 and 6.4) can be calculated from four statistics (counts): The number of true and false positives and negatives. A true positive is a data item where both antecedent and con-

[2] See Section 6.2 for a discussion on the flawed discretization method used by Das et al. [1998].

	$\neg\hat{p}(t)$	$\hat{p}(t)$
$p(t)$	FN	TP
$\neg p(t)$	TN	FP

Figure 4.1: The confusion matrix

sequent are true. Similarly, a false positive is a data item where the antecedent is true, while the consequent is false, and so forth. Collectively, these four statistics are often called the *confusion matrix*. If we denote a given data item by t (alluding to the *timestamp* of a location in a time sequence), the antecedent by $\hat{p}(t)$ (an estimate of a predicate describing the data), and the consequent by $p(t)$ (the true predicate), the confusion matrix can be drawn up as shown in Figure 4.1.

Using these counts, support may be calculated can be calculated as

$$s(p, \hat{p}) = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} \ , \tag{4.1}$$

and confidence as

$$c(p, \hat{p}) = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{FP} + \mathrm{FN}} \ . \tag{4.2}$$

The following section describes how support, in particular, may be used to bound the rule search.

4.3 Counting Candidates

A basic operation in rule mining is counting. As shown in the previous section, many quality measures can be computed from only the four counts in the confusion matrix. In evolutionary rule mining, as described in Chapter 5, this counting is done directly, for a specific candidate rule. The more common non-heuristic mining method, however, calculates (in principle) the quality measure for *all* viable rules. For simple rule formats, this scheme may be implemented directly, but for more complex rules, more clever techniques must be employed.

Consider the simple rule format for relational data described in Section 4.1. Assuming that each of the p properties can take on a different values, the number of

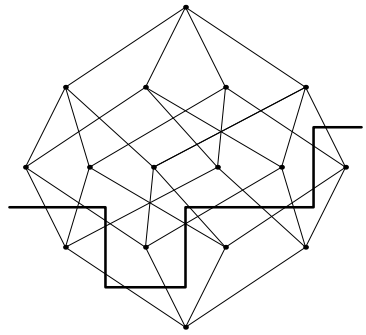

Figure 4.2: Pruning of the rule space

possible rules would be a^2p^2. Assuming that we know the attribute whose value we wish to predict, there would be only a^2p rules, and other constraints could bring this number down even further. Following the same logic, using the simple sequence rule format from Section 4.1 with a possible symbols on a sequence of length n, there would be a^2T possible rules (where T is the maximum value for the rule parameter t).

For many realistic applications, the number of rules would not be prohibitive, and it would be possible to count the the occurrences of all antecedents and consequents (giving us the confusion matrix) with a single pass through the data. The work for the simple relational rule format would be $O(p^2n)$, while for the sequence rule format it would be $O(Tn)$.

The problem with this simple approach appears when the rule antecedent is complicated beyond the single-property single-symbol form. To take a relatively simple example, consider Boolean expressions: there are 2^{2^k} possible Boolean functions of k parameters. If each attribute in a tabular data set may be used several times in the antecedent, with different value assignments, the number of possible rules is even greater. Direct counting of the occurrences is then clearly intractable.

The insight underlying the approach used in such landmark papers as Mannila et al. [1997] and Agrawal and Srikant [1995] and many others since (see [Adamo 2001]) is that there is a relation of *generalization* between the rules. In other words, given two rules, it may be that one is more general than the other (or, equivalently, that the second is more specific than the first). This relation (given some assumptions about the rule format) forms a lattice, as illustrated in Figure 4.2.

By placing a constraint on the support (or frequency) of the mined rules, it is possible to prune away large parts of the search space. The reason for this is that if a given rule has too low support, then all rules that are more specific will *also* have too low support, because their support must be even lower. So, for certain

rule formats it is possible to find all rules that exceed a certain support threshold. These *frequent rules* may then be ranked, using some more useful quality measure.

Although this general method is very powerful, it does have some limitations. Most notably, it limits the possible rule formats, and it makes the assumption that rules must have high support to be interesting. Evolutionary data mining, which is described in Chapter 5, is one approach that tries to lift these restriction, at the cost of not being able to guarantee that *all* interesting rules are found.

Chapter 5

Evolutionary Computation and Data Mining

> "I think that the distinction between accident and design is clear, in principle if not always in practice, but this chapter will introduce a third category of objects which is harder to distinguish. I shall call them *designoid*. . . . Designoid objects *look* designed, so much so that some people — probably, alas, most people — think that they *are* designed. These people are wrong. But they are right in their conviction that designoid objects cannot be the result of chance. . . . No, the true explanation — Darwinian natural selection — is very different. . . . "
>
> — Richard Dawkins, *Climbing Mount Improbable*

This chapter outlines the principles underlying the method of heuristic search in general, and evolutionary computation in particular. Finally, the use of heuristic search in data mining is discussed.

5.1 Heuristic Search and Optimization

Optimization is needed in all areas of computer science. For reasonably well-behaved or well understood functions, techniques exist for efficient and predictable numerical optimization [Nocedal and Wright 1999]. However, if the objective function or the structure of the parameters, or both, become sufficiently complex, these methods may no longer be of any use. In these cases, the general class of techniques known as *heuristic search* may be applicable.

In the realm of numerical optimization, the term *search* might be more appropriate. However, the techniques are often applied to problems where the function being optimized is only a heuristic gauging how close one is to a true solution. Many of the heuristic search methods involve some degree of randomness, and may therefore also be referred to as *stochastic search methods*.

Algorithm 5.1 gives a high level overview of how heuristic search works. First, the search algorithm either receives or generates an initial state. This state is usually a single candidate solution, such as a vector of real numbers for numerical optimization. For a simple (deterministic) example of a search algorithm, see Algorithm 5.2, the basic *hill-climbing algorithm*. Each state is a single solution, new candidates are chosen from a close neighbourhood of the current state (as defined by a *neighbourhood function n*) and the next state is always the candidate that has the lowest objective score (as measured by the objective function). It is assumed here that a low objective score is desirable, that is, that the optimization is one of minimization. The name "hill-climbing" comes from maximization applications, where the algorithm "climbs" the peaks of the objective function. For minimization problems, this method is often called the method of steepest descent (or steepest gradient descent, for differentiable objective functions, where the gradient is used to find the next state).

1: generate or receive initial state
2: **repeat**
3: generate new candidate states
4: choose new state using heuristic (objective) function
5: **until** the current state satisfies halting criteria

Algorithm 5.1: Outline of a heuristic searching algorithm

1: $y \leftarrow s$
2: **repeat**
3: $x \leftarrow y$
4: $y \leftarrow \arg\min_z \{f(z) \mid z \in n(y)\}$
5: **until** $x = y$

Algorithm 5.2: Basic hill-climbing algorithm with objective function f, starting point s, and neighbourhood function n

The state may, however, be a *set* of candidate solutions. When the current state is a solution set, the search is called *beam search*. Evolutionary algorithms (see Section 5.2) are a form of beam search [Banzhaf et al. 1997, p. 27].

Heuristic search methods are so called because they use a heuristic function (the objective function) to guide them toward solutions. (In other words, this function may be used to evaluate solutions that are only partially correct.) The general

search method — in many ways a heuristic in its own right — is often called a *metaheuristic*. The following section describes some such metaheuristics.

5.1.1 Some Existing Metaheuristics

Many distinctly different metaheuristics have been developed, and variations of existing ones sometimes branch off to become the focus of new fields of research. For example, *memetic algorithms* are a close relative of genetic algorithms (or evolutionary algorithms) which adds localized search (learning) to the general search of evolution.

Probably the most well-known metaheuristics are:

- Evolutionary algorithms, including genetic algorithms and genetic programming, which uses an abstract simulation of natural selection to evolve populations of fit solutions [Goldberg 1989; Koza 1992].
- Simulated annealing, which simulates the process of annealing (cooling) to gradually move from a random search to a deterministic hill-climbing search [Laarhoven and Aarts 1987].
- Tabu search, in which the most recent candidate solutions (or the moves between them) are taboo (or tabu), that is, they are not considered viable [Glover and Laguna 1998].

The motivation behind these metaheuristics is that simpler search methods, such as hill-climbing algorithms, tend to get stuck in local optima. For relatively simple objective functions, it may be possible to escape local minima by restarting the search algorithm several times. Many real-world objective functions are, however, too complex for this simple approach to work.

It has been proven that no search method can be better than another for all objective functions, the so-called "no free lunch" theorems [Wolpert and Macready 1995; Droste et al. 2002]. While much empirical work exists on the application of each of the metaheuristics to practical problems, no definite comparison has been performed, and the choice between them may be based on such factors as experience, convenience, or special purpose empirical comparisons.

Although heuristic search has not been widely used as the main search strategy in data mining, it seems that evolutionary algorithms have been used more widely than the other metaheuristics [Freitas 2002]. This is also the metaheuristic we have used in our research. The following section describes evolutionary computation in more detail.

5.2 Evolution as Search Strategy

Evolution occurs in any process where mechanisms of variation, selection, and reproduction are present. These mechanisms may be explicitly simulated, resulting in a powerful optimization method.

The state in an evolutionary algorithm consists of a set of candidate solutions, or *individuals*. This set of individuals is called the *population*. As with other forms of search, the state is updated incrementally, but not all candidate solutions contribute to the updated state; instead, eligible *parent solutions* are chosen using a (stochastic) *selection function*. New solutions are created from these parent solutions using various hereditary (genetic) operators, such as cloning, crossover (swapping of components between parents), and mutation (random modifications). The basic process is demonstrated in Algorithm 5.3.

1: $Y \leftarrow S$
2: **repeat**
3: $\quad X, Y \leftarrow Y, \emptyset$
4: $\quad$ **repeat**
5: $\quad\quad p_1, p_2 \leftarrow s(X, f)$
6: $\quad\quad y_1, y_2 \leftarrow n(p_1, p_2)$
7: $\quad\quad Y \leftarrow Y \cup \{y_1, y_2\}$
8: $\quad$ **until** $|Y| \geq |S|$
9: **until** $\min\{f(y) \mid y \in Y\} \leq \varepsilon$

Algorithm 5.3: Basic generational evolutionary algorithm with fitness function f, initial generation S, selection function s, and reproduction function n

One important factor is hard to discern in Algorithm 5.3, namely how evolution is affected by the objective, or *fitness* function. The core of evolution is the principle of *natural selection*, or, in the words of Charles Darwin, "survival of the fittest" [Darwin 1859]. Artificial selection usually works in a similar fashion; individuals with a high fitness value (that is, candidate solutions with a high heuristic value) have a higher chance of being selected for reproduction. (There are exceptions to this, one recent example being the FUSS selection scheme of Hutter [2001], where the various fitness levels are selected with uniform probability.) Eventually, the population will be dominated by (or at least contain) solutions with high fitness.

Many names have been given to specific variants of evolutionary algorithms, two of the more prominent being genetic algorithms [Goldberg 1989] and genetic programming [Koza 1992]. Originally, the candidate solutions of genetic algorithms were fixed-length binary strings, while those of genetic programming were executable computer programs, represented as syntax trees. The divisions between such methods become increasingly blurred, and in some sense meaningless, as

different kinds of genomes (that is, the candidate solutions) are used. Banzhaf et al. [1997] argue that the term genetic programming be used about the discipline as a whole, as its functionality is a superset of that of other methods — a superset which may well be restricted using domain knowledge when solving specific problems.

In the papers in the second part, we have used this terminology, even though the language used to define our candidate solutions is not Turing complete (that is, not really a programming language). Still, the genomes are syntax trees, and the genetic operators are those used by Koza [1992], such as subtree swapping and random subtree generation.

5.3 Heuristic Data Mining

The goals and principles of data mining are described in Chapter 4. This section describes the particulars of data mining methods which use heuristic search mining as their optimization process. The other components of the methods, that is, the problem, solution structure, score function, and data management method, do not necessarily differ from the more common data mining methods.

There are several reasons why heuristic search, or evolutionary algorithms in particular, may be useful in data mining. Freitas contrasts evolutionary algorithms and more common rule induction methods as follows:[1]

> First, rule induction algorithms typically use deterministic operators. In addition, these operators usually perform a kind of local search in rule space. . . . In contrast, evolutionary algorithms typically use stochastic operators. Some of these operators . . . usually perform a more global search in rule space. . . .
>
> Second, rule induction algorithms typically construct and evaluate a candidate rule in an incremental fashion. . . . In contrast, evolutionary algorithms typically evaluate a complete candidate rule or rule set as a whole.
>
> Third, unlike most rule induction algorithms, evolutionary algorithms work with a population of candidate rules or rule sets, rather than with a single rule or rule set at a time. [2002, pp. 9–10]

According to Freitas, the characteristics of evolutionary algorithms make them deal more effectively with problems such as attribute (or condition) interaction,

[1] It should be noted that Freitas's comments about local search are more applicable to the construction of, for example, tree classifiers than to the counting approach described here.

which may confound mainstream methods. An interesting, although not essential, argument in favour of evolutionary algorithms is that they easily admit multiobjective optimization, a fact which is exploited in Paper VI.

One disadvantage of using existing evolutionary algorithms for data mining is that their only source of domain knowledge is the objective function. In contrast, special purpose algorithms may use operators such as generalization and specification, which exploit domain knowledge for more efficient convergence. As Freitas shows, it is possible to extend evolutionary algorithms to use such operators as modified forms of the common genetic operators. The direction (generalization or specification) can be probabilistically weighted according to the coverage of the rule or pattern (low support or high support, respectively). Although such operators have not been used in the work presented in the second part of this work, the possibility should not be ruled out that they might lead to faster convergence.

One of the advantages of heuristic data mining observed in this study is the flexibility of the method, as opposed to special purpose methods. In heuristic search, one is free to choose any objective function, and any rule format that can meaningfully be used as a search query in the data. Special purpose mining methods normally exploit the specific structure and properties of the objective function and rule structure.

The application of heuristic search to data mining is relatively straightforward. The candidate solutions are either single rules (the so-called Michigan approach) or sets of rules (the Pittsburgh approach), and their fitness is calculated based on the occurrences of the rules in the data.

The most common case is the *supervised* case, where the attribute that is to be predicted is given as part of the problem formulation. In this case, fitness measures are often calculated from the confusion matrix, which consists of the number of true and false positives and negatives. A true positive is a hit correctly reported by a rule, while a false positive is an incorrectly reported hit. Similarly, a true negative is a position where a rule correctly does not report a hit, while a false negative is an incorrect omission. Some basic quality measures that may be calculated based on these numbers are *precision* $(\mathrm{TP}/(\mathrm{TP}+\mathrm{FP}))$, *true positive rate* $(\mathrm{TP}/(\mathrm{TP}+\mathrm{FN}))$, *true negative rate* $(\mathrm{TN}/(\mathrm{TN}+\mathrm{FP}))$, and *accuracy* $((\mathrm{TP}+\mathrm{TN})/(\mathrm{TP}+\mathrm{FP}+\mathrm{FN}+\mathrm{TN}))$. Two possible fitness measures described by Freitas are precision times true positive rate and true positive rate times true negative rate [2002, p. 129]. The fitness measures used in our work are described in Section 6.4.

Chapter 6

Evolving Rules from Sequences

> "A specification that will not fit on one page of 8.5×11 inch paper cannot be understood." — Mark Ardis

This chapter describes some of the specifics of the method used in the second part. The general principles of sequence retrieval, rule mining, and evolutionary algorithms are outlined in Chapters 3, 4, and 5, respectively. In this chapter, the specifics of our use of these methods will be discussed briefly.

6.1 Method Structure

Using the taxonomy introduced in Section 1.2, our method can be analyzed as follows:

Problem. Our problem is one of prediction. Given a history of behaviour, we want to predict the behaviour in the near future.

Potential solutions. Our potential solutions are patterns rather than models. More specifically, we are looking for *rules*. In most of our experiments, these rules are defined in terms of the language IQL, although in Paper V we also use rules defined in terms of spatial signatures and simple regular expressions.

Score function. We use several score functions, but they fall in two main categories: functions measuring *predictive power* and functions measuring *interestingness*.

Search method. To find rules of high quality we use genetic programming, as described in Chapter 5.

Data management method. The data management (or retrieval) method we primarily use is the Interagon *pattern matching chip* (PMC). In addition, in Paper V, multidimensional retrieval and software tools for regular expression retrieval are used. In the following, the focus is on pattern retrieval and the use of the PMC, rather than spatial indexing.

As part of the knowledge discovery process, the time series were discretized. This process is discussed in the following section. Section 6.3 summarizes the rule format, while Section 6.4 describes two of the most important fitness measures used.

6.2 Preprocessing

In order for the pattern matching chip to be able to deal with the time series, they must be preprocessed. The real-valued sequences are discretized to form discrete sequences with elements drawn from a finite alphabet. (Note that this does not apply to the synthetic data sets, which were generated in a discrete form.)

We have used two discretization methods in our work. The first one was introduced by Das et al. [1998]. We applied this to the supervised rule learning experiments in Paper I. In short, the method works as follows: A set of vectors is created by using a sliding window of fixed width on the time series. These vectors are then clustered to find characteristic shapes, and all positions in the time series are labelled according to the cluster to which the vector at that position belongs.

This method has been used in several publications since its appearance. However, Lin et al. have recently shown it to be meaningless when applied to rule mining in general, because the "interesting" rules it leads to are completely spurious.[1] While this should not detract from the results presented in Paper I, where the predictive power of the rules was tested on a separate data sets, we have not used this method in the subsequent papers. Also, when trying to find rules with high measures of interestingness, we have used random data as a baseline (Papers III, IV, and VI). Finding interesting rules in random data would clearly indicate that there is something wrong with the method.

The other method is used, among other places, by Keogh et al. [2002] (and is discussed more thoroughly in Section IV.2). It also involves using a sliding window to create a set of vectors. However, rather than clustering these, a real-valued

[1] J. Lin, E. Keogh, and W. Truppel, *Clustering of Streaming Time Series is Meaningless: Implications for Previous and Future Research*. Manuscript in preparation.

feature, such as the slope or average, is extracted from each. Then, the vectors are sorted according to their feature value, and divided into bins of equal size (where the number of bins is determined by the desired alphabet size for the discretized sequence). Finally, the limiting values are found for each bin, and these are used to assign a bin number (or symbol) to each position in the sequence.

Note that this method guarantees that each symbol is used with equal frequency when discretizing the time series used to find the bins. If the same limits are used to discretize another time series drawn from the same distribution (such as a test data set), the symbol distribution will still be close to uniform.

6.3 The Rules

The rules consist of two parts, the *antecedent* and the *consequent*. In addition to this, a *time limit* between the match of the antecedent and the occurrence of the consequent is imposed. Our antecedents may belong to one of several *pattern languages* (see below), while the consequent describes a sequence position, either dictating the occurrence of specific symbol or a more general sequence property, such as having a positive slope.

In other words, the rules are of the following form:

if: the antecedent matches at position x
then: the sequence has a property p at position y,
where $y - x \leq t$.

The antecedent pattern may in principle be drawn from the full Interagon Query Language, or IQL [Interagon AS 2002], although some restrictions have been necessary, due to practical considerations. Also, we have experimented with several smaller languages, which are simply subsets of IQL. Paper IV (Section IV.3.1 in particular) contains some discussion on this issue. See also Appendix B for a summary of the notation used here, which differs in some ways from the IQL syntax. (This discrepancy is motivated mainly by typographical convenience.)

6.4 Rule Fitness

We have mainly used two fitness functions, both of which can be calculated from the confusion matrix (see Section 5.3). The first one, used to measure predictive power, is *correlation*, which is defined as follows:

$$r(p, \hat{p}) = \frac{\mathrm{TP} \cdot \mathrm{TN} - \mathrm{FP} \cdot \mathrm{FN}}{\sqrt{(\mathrm{TN} + \mathrm{FN})(\mathrm{TN} + \mathrm{FP})(\mathrm{TP} + \mathrm{FN})(\mathrm{TP} + \mathrm{FP})}} \tag{6.1}$$

Here, p is a predicate representing the behaviour of the sequence, and $\hat{p}$ is the behaviour of the predictor rule.

Correlation measures the degree to which a two-dimensional data set falls along a straight line; that is, how close two variables are to being linearly dependent. In the case of binary classification, each variable (the prediction and the actual event) may only take two values, *yes* or *no*. If $r(p, \hat{p}) = 1$, the prediction is perfect; if $r(p, \hat{p}) = -1$, the prediction is completely wrong.

The other fitness measure we have used is the *modified J-measure*, which is introduced in Paper IV. It is based on the existing J-measure [Smyth and Goodman 1991], which is defined as follows:

$$J(R_c^t, R_a) = p(R_a) \cdot \Big(p(R_c^t|R_a) \log_2 \frac{p(R_c^t|R_a)}{p(R_c^t)} + (1 - p(R_c^t|R_a)) \log_2 \frac{1 - p(R_c^t|R_a)}{1 - p(R_c^t)} \Big) \quad (6.2)$$

It is shown in Paper IV that the J-measure used by itself as a fitness function leads to rules with very low confidence. Our modification to the J-measure is inspired by the approach taken in mainstream rule mining algorithms. There, a constraint is usually placed on confidence or support, requiring it to fall above a certain threshold for a given rule to come into consideration at all. Only after this condition has been met is the interestingness measured. To preserve a smooth objective function, which could also measure differences in fitness between rules with low confidence, we wanted to avoid a sharp cutoff. Instead, we used a *sigmoid multiplier*, with the sigmoid function defined as follows:

$$F(c(R)) = \frac{1}{1 + e^{-(c(R) - c_{min}) \cdot g}} \quad (6.3)$$

Here, $c(R)$ is the confidence of the rule, c_{min} is the confidence threshold, g is a *sharpness parameter*, and $F(c(R))$ yields a number between zero and one, which is multiplied with the J-measure. Thus, for rules with high confidence, the fitness is approximately equal to its interestingness (J-measure), but for rules with confidences below the threshold, the fitness will drop away sharply from the J-measure. The shape of the sigmoid function and the effect of the sharpness parameter g are shown in Figure 6.1.

In addition to these two main fitness measures, in Paper VI we used *multiobjective evolution*, which means that we used several objective functions at once. These objective functions were confidence, support, plain J-measure, and comprehensibility (inverse of rule size). For details, see Section VI.2.4.

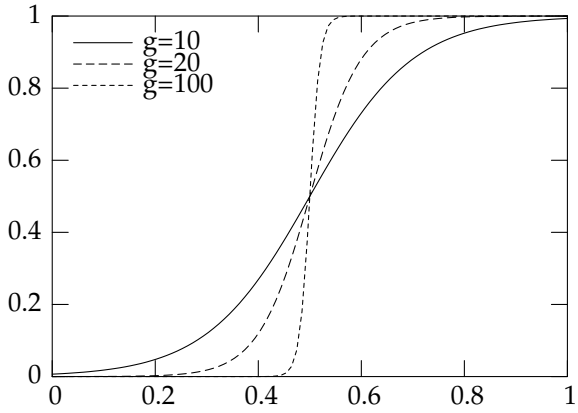

Figure 6.1: The sigmoid function

Chapter 7

Concluding Remarks

> "The world is round and the place which may seem like the end may also be only the beginning." — Ivy Baker Priest (1905–1975), *Parade*

The aim of this study has been to explore the use of evolutionary data mining in sequence rule mining. The main motivation behind this is the flexibility gained, both in the possible rule formats and the quality measures allowed. Little work has been done in this area previously. Also, a contribution has been using specialized search hardware (the Interagon Pattern Matching Chip [Interagon AS 2000]) in evaluating the rules. This is a factor that has afforded us a flexibility in the choice of rule format beyond any other known method. The results of our empirical studies may be summed up as follows:

- Supervised evolutionary rule learning can re-discover rules with complexity comparable to those discovered in existing methods, and can achieve high prediction rates on real-world datasets (Paper II).
- The discretization process is important to the predictive power of the discovered rules. Choosing the right discretization parameters can give greater improvements than using techniques such as ensembles or Bayesian classifiers to improve the matches (Paper III).
- By designing proper objective functions, it is possible to use the method for unsupervised mining of rules with high interestingness scores and high predictive power (Paper IV).
- Although we have mainly used the method with specialized hardware, we have shown that for simpler rule formats, and somewhat longer running times, it is completely feasible to use software based retrieval methods (Papers I and V).

- The use of evolutionary mining makes it possible to perform multiobjective rule mining, something which, to our knowledge, has not been done before. This makes it possible to use several mutually exclusive quality measures and receive a diverse set of solutions for expert evaluation (Paper VI).

While our goals in investigating the use of evolutionary computation and specialized hardware in sequence mining are reached, there are still many potential directions for further research. One obvious topic of research is investigating the applicability of our rule format for various practical problems. Another would be to apply the lessons learned here to other forms of pattern mining or learning. This might include existing applications, such as discovering concise classifiers for biological sequences.

Papers

Paper I

A Survey of Recent Methods for Efficient Retrieval of Similar Time Sequences

Abstract: *Time sequences occur in many applications, ranging from science and technology to business and entertainment. In many of these applications, searching through large, unstructured databases based on sample sequences is often desirable. Such similarity-based retrieval has attracted a great deal of attention in recent years. Although several different approaches have appeared, most are based on the common premise of dimensionality reduction and spatial access methods. This paper gives an overview of recent research and shows how the methods fit into a general context of signature extraction.*

I.1 Introduction

Time sequences arise in many applications—any applications that involve storing sensor inputs, or sampling a value that changes over time. A problem which has received an increasing amount of attention lately is the problem of *similarity retrieval* in databases of time sequences, so-called "query by example." Some uses of this are [Agrawal et al. 1993]:

- Identifying companies with similar patterns of growth.
- Determining products with similar selling patterns.
- Discovering stocks with similar movement in stock prices.

- Finding out whether a musical score is similar to one of a set of copyrighted scores.
- Finding portions of seismic waves that are not similar to spot geological irregularities.

Applications range from medicine, through economy, to scientific disciplines such as meteorology and astrophysics [Faloutsos et al. 1994; Yi and Faloutsos 2000].

The running times of simple algorithms for comparing time sequences are generally polynomial in the length of both sequences, typically linear or quadratic. To find the correct offset of a query in a large database, a naive *sequential scan* will require a number of such comparisons that is linear in the length of the database. This means that, given a query of length m and a database of length n, the search will have a time complexity of $O(nm)$, or even $O(nm^2)$ or worse. For large databases this is clearly unacceptable.

Many methods are known for performing this sort of query in the domain of strings over finite alphabets, but with time sequences there are a few extra issues to deal with:

- The range of values is not generally finite, or even discrete.
- The sampling rate may not be constant.
- The presence of noise in various forms makes it necessary to support very flexible similarity measures.

This chapter describes some of the recent advances that have been made in this field; methods that allow for indexing of time sequences using flexible similarity measures that are invariant under a wide range of transformations and error sources.

The chapter is structured as follows: Section I.2 gives a more formal presentation of the problem of similarity-based retrieval and the so-called *dimensionality curse*; Section I.3 describes the general approach of signature based retrieval, or *shrink and search*, as well as three specific methods using this approach; Section I.4 shows some other approaches, while Section 5 concludes the chapter. Finally, Appendix A gives an overview of some basic distance measures.[1]

I.1.1 Terminology and Notation

A time sequence $\boldsymbol{x} = \langle x_1{=}(v_1, t_1), \ldots, x_n{=}(v_n, t_n)\rangle$ is an ordered collection of elements x_i, each consisting of a value v_i and a timestamp t_i. Abusing the notation

[1] The term "distance" is used loosely in this chapter. A distance measure is simply the inverse of a similarity measure and is not required to obey the metric axioms.

Table I.1: Notation

$\boldsymbol{x}$	A sequence
$\tilde{x}$	A signature of $\boldsymbol{x}$
x_i	Element number i of $\boldsymbol{x}$
$x_{i:j}$	Elements i to j (inclusive) of $\boldsymbol{x}$
$\lvert\boldsymbol{x}\rvert$	The length of $\boldsymbol{x}$

slightly, the value of x_i may be referred to as x_i.

For some retrieval methods, the values may be taken from a finite class of values [Mannila and Ronkainen 1997], or may have more than one dimension [Lee et al. 2000], but it is generally assumed that the values are real numbers. This assumption is a requirement for most of the methods described in this chapter.

The only requirement of the timestamps is that they be nondecreasing (or, in some applications, strictly increasing) with respect to the sequence indices:

$$t_i \leq t_j \Leftrightarrow i \leq j \tag{I.1}$$

In some methods, an additional assumption is that the elements are *equi-spaced*: For every two consecutive elements x_i and x_{i+1} we have

$$t_{i+1} - t_i = \Delta \ , \tag{I.2}$$

where Δ (the *sampling rate* of $\boldsymbol{x}$) is a (positive) constant. If the actual sampling rate is not important, Δ may be normalized to 1, and t_1 to 0. It is also possible to resample the sequence to make the elements equi-spaced, when required.

The *length* of a time sequence $\boldsymbol{x}$ is its cardinality, written as $|\boldsymbol{x}|$. The contiguous subsequence of $\boldsymbol{x}$ containing elements x_i to x_j (inclusive) is written $x_{i:j}$. A *signature* of a sequence $\boldsymbol{x}$ is some structure that somehow represents $\boldsymbol{x}$, yet is simpler than $\boldsymbol{x}$. In the context of this chapter, such a signature will always be a vector of fixed size k. (For a more thorough discussion of signatures, see Section I.3.) Such a signature is written $\tilde{x}$. For a summary of the notation, see Table I.1.

I.2 The Problem

The problem of retrieving similar time sequences may be stated as follows: Given a sequence $\boldsymbol{q}$, a set of time sequences X, a (non-negative) distance measure d, and a *tolerance threshold* ε, find the set R of sequences closer to $\boldsymbol{q}$ than ε, or, more precisely:

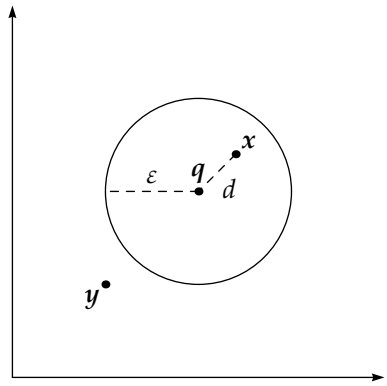

Figure I.1: Similarity retrieval

$$R = \{\boldsymbol{x} \in X \mid d(\boldsymbol{q}, \boldsymbol{x}) \leq \varepsilon\} \tag{I.3}$$

Alternatively, one might wish to find the k nearest neighbours of $\boldsymbol{q}$, which amounts to setting ε so that $|R| = k$. The parameter ε is typically supplied by the user, while the distance function d is domain-dependent. Several distance measures will be described rather informally in this chapter. For more formal definitions, see Appendix A.

Figure I.1 illustrates the problem for Euclidean distance in two dimensions. In this example, the vector $\boldsymbol{x}$ will be included in the result set R, while $\boldsymbol{y}$ will not.

A useful variation of the problem is to find a set of *subsequences* of the sequences in X. This, in the basic case, requires comparing $\boldsymbol{q}$ not only to all elements of X, but to all possible subsequences.[2]

If a method retrieves a subset S of R, the wrongly dismissed sequences in $R - S$ are called *false dismissals*. Conversely, if S is a superset of R, the sequences in $S - R$ are called *false alarms*.

I.2.1 Robust Distance Measures

The choice of distance measure is highly domain dependent, and in some cases a simple L_p norm such as Euclidean distance may be sufficient. However, in many cases, this may be too brittle [Keogh and Pazzani 1999b] since it does not tolerate such transformations as scaling, warping, or translation along either axis. Many

[2]Except in the description of LCS in Appendix A, subsequence should be taken to mean contiguous subsequence, or segment.

of the newer retrieval methods focus on using more robust distance measures, which are invariant under such transformations as *time warping* (see Appendix A for details) without loss of performance.

I.2.2 Good Indexing Methods

Faloutsos et al. [1994] list the following desirable properties for an indexing method:

1. It should be faster than a sequential scan.
2. It should incur little space overhead.
3. It should allow queries of various length.
4. It should allow insertions and deletions without rebuilding the index.
5. It should be correct: No false dismissals must occur.

To achieve high performance, the number of false alarms should also be low. Keogh et al. [2001b] add the following criteria to the list above:

6. It should be possible to build the index in reasonable time.
7. The index should preferably be able to handle more than one distance measure.

I.2.3 Spatial Indices and the Dimensionality Curse

The general problem of similarity-based retrieval is well known in the field of information retrieval, and many indexing methods exist to process queries efficiently [Baeza-Yates and Gonnet 1999]. However, certain properties of time sequences make the standard methods unsuitable. The fact that the value ranges of the sequences usually are continuous, and that the elements may not be equispaced, makes it difficult to use standard text-indexing techniques such as suffix-trees. One of the most promising techniques is multidimensional indexing (*R*-trees [Guttman 1984], for example), in which the objects in question are multidimensional vectors, and similar objects can be retrieved in sublinear time. One requirement of such spatial access methods is that the distance measure must be monotonic in all dimensions, usually satisfied through the somewhat stricter requirement of the the triangle inequality ($d(\boldsymbol{x}, \boldsymbol{z}) \leq d(\boldsymbol{x}, \boldsymbol{y}) + d(\boldsymbol{y}, \boldsymbol{z})$).

One important problem that occurs when trying to index sequences with spatial access methods is the so-called *dimensionality curse*: Spatial indices typically work

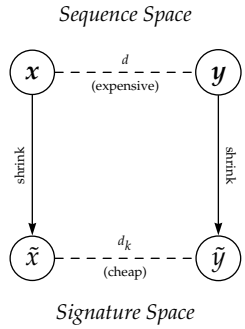

Figure I.2: The signature based approach

only when the number of dimensions is low [Chakrabarti and Mehrotra 1999]. This makes it unfeasible to code the entire sequence directly as a vector in an indexed space.

The general solution to this problem is *dimensionality reduction*: to condense the original sequences into *signatures* in a *signature space* of low dimensionality, in a manner which, to some extent, preserves the distances between them. One can then index the signature space.

I.3 Signature Based Similarity Search

A time sequence $\boldsymbol{x}$ of length n can be considered a vector or point in an n-dimensional space. Techniques exist (spatial access methods, such as the R-tree and variants [Sellis et al. 1987; Chakrabarti and Mehrotra 1999; Wang and Perng 2001]) for indexing such data. The problem is that the performance of such methods degrades considerably even for relatively low dimensionalities [Chakrabarti and Mehrotra 1999]; the number of dimensions that can be handled is usually several orders of magnitude lower than the number of data points in a typical time sequence.

A general solution described by Faloutsos et al. [1994, 1997] is to extract a low-dimensional *signature* from each sequence, and to index the signature space. This *shrink and search* approach is illustrated in Figure I.2.

An important result given by Faloutsos et al. [1994] is the proof that in order to guarantee completeness (no false dismissals), the distance function used in the signature space must underestimate the true distance measure, or:

$$d_k(\tilde{x}, \tilde{y}) \leq d(\boldsymbol{x}, \boldsymbol{y}) \tag{I.4}$$

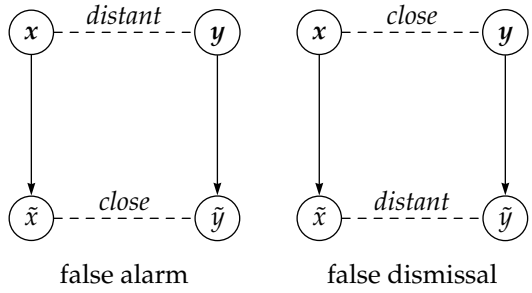

Figure I.3: An intuitive view of the bounding lemma

This requirement is called the *bounding lemma*. Assuming that (I.4) holds, an intuitive way of stating the resulting situation is: "if two signatures are far apart, we know the corresponding [sequences] must also be far apart" [Faloutsos et al. 1997]. This, of course, means that there will be no false dismissals. To minimize the number of false alarms, we want d_k to approximate d as closely as possible. The bounding lemma is illustrated in Figure I.3.

This general method of dimensionality reduction may be summed up as follows [Keogh et al. 2001b]:

1. Establish a distance measure d from a domain expert.

2. Design a dimensionality reduction technique to produce signatures of length k, where k can be efficiently handled by a standard spatial access method.

3. Produce a distance measure d_k over the k-dimensional signature space, and prove that it obeys the bounding condition (I.4).

In some applications, the requirement in (I.4) is relaxed, allowing for a small number of false dismissals in exchange for increased performance. Such methods are called *approximate*.

The dimensionality reduction may in itself be used to speed up a sequential scan, and some methods (such as the piecewise linear approximation of Keogh et al. which is described in Section I.4.2) rely only on this, without using any index structure.

Methods exist for finding signatures of arbitrary objects, given the distances between them [Faloutsos and Lin 1995; Wang et al. 1999], but in the following I will concentrate on methods that exploit the structure of the time series to achieve good approximations.

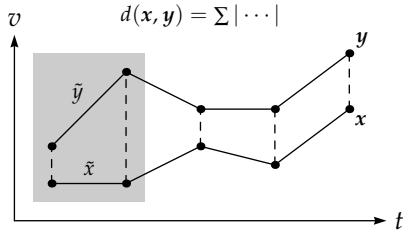

Figure I.4: Comparing two sequences

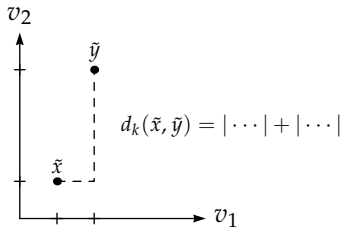

Figure I.5: A simple signature distance

I.3.1 A Simple Example

As an example of the signature-based scheme, consider the two sequences shown in Figure I.4.

The sequences, x and y, are compared using the L_1 measure (Manhattan distance), which is simply the sum of the absolute distances between each aligned pair of values. A simple signature in this scheme is the prefix of length 2, as indicated by the shaded area in the figure. As shown in Figure I.5, these signatures may be interpreted as points in a two-dimensional plane, which can be indexed with some standard spatial indexing method. It is also clear that the signature distance will underestimate the real distance between the sequences, since the remaining summands of the real distance must all be positive.

Although correct, this simple signature extraction technique is not particularly precise. The signature extraction methods introduced in the following sections take into account more information about the full sequence shape, and therefore lead to fewer false alarms.

Figure I.6 shows a time series containing measurements of atmospheric pressure. In the following three sections, the methods described will be applied to this sequence, and the resulting simplified sequence (reconstructed from the extracted

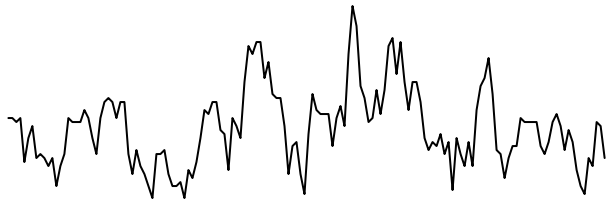

Figure I.6: An example time sequence

signature) will be shown superimposed on the original.

I.3.2 Spectral Signatures

Some of the methods presented in this section are not very recent, but introduce some of the main concepts used by newer approaches.

Agrawal et al. [1993] introduce a method called the *F*-index in which a signature is extracted from the frequency domain of a sequence. Underlying their approach are two key observations:

- Most real-world time sequences can be faithfully represented by their strongest Fourier coefficients.
- Euclidean distance is preserved in the frequency domain (Parseval's Theorem [Shatkay 1995]).

Based on this, they suggest performing the Discrete Fourier Transform on each sequence, and using a vector consisting of the sequence's k first amplitude coefficients as its signature. Euclidean distance in the signature space will then underestimate the real Euclidean distance between the sequences, as required.

Figure I.7 shows an approximated time sequence, reconstructed from a signature consisting of the original sequence's ten first Fourier components.

This basic method allows only for whole-sequence matching. In 1994, Faloutsos et al. introduce the *ST*-index, an improvement on the *F*-index that makes subsequence matching possible. The main steps of the approach are as follows:

1. For each position in the database, extract a window of length w, and create a spectral signature (a *point*) for it.

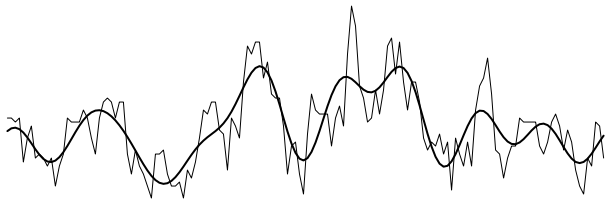

Figure I.7: A sequence reconstructed from a spectral signature

Each point will be close to the previous, because the contents of the sliding window change slowly. The points for one sequence will therefore constitute a *trail* in signature space.

2. Partition the trails into suitable (multidimensional) Minimal Bounding Rectangles (MBRs), according to some heuristic.
3. Store the MBRs in a spatial index structure.

To search for subsequences similar to a query $\boldsymbol{q}$ of length w, simply look up all MBRs that intersect a hypersphere with radius ε around the signature point $\tilde{q}$. This is guaranteed not to produce any false dismissals, because if a point is within a radius of ε of $\tilde{q}$, it cannot possibly be contained in an MBR that does not intersect the hypersphere.

To search for sequences longer than w, split the query into w-length segments, search for each of them, and intersect the result sets. Because a sequence in the result set R cannot be closer to the full query sequence than it is to any one of the window signatures, it has to be close to all of them, that is, contained in all the result sets.

These two papers (Agrawal et al. [1993] and Faloutsos et al. [1994]) are seminal; several newer approaches are based on them. For example, Rafiei and Mendelzon [1997] show how the method can be made more robust by allowing various transformations in the comparison, and Chan and Fu [1999] show how the Discrete Wavelet Transform (DWT) can be used instead of the Discrete Fourier Transform (DFT), and that the DWT method is empirically superior. See Wu et al. [2000] for a comparison between similarity search based on DFT and DWT.

I.3.3 Piecewise Constant Approximation

An approach independently introduced by Yi and Faloutsos [2000] and Keogh et al. [Keogh and Pazzani 2000; Keogh et al. 2001b] is to divide each sequence into k

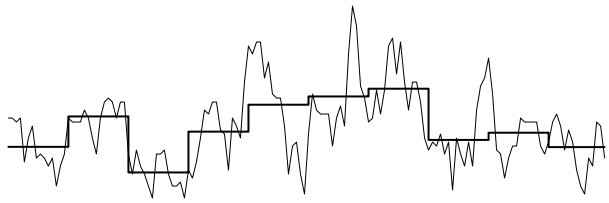

Figure I.8: A sequence reconstructed from a PCA signature

segments of equal length, and to use the average value of each segment as a coordinate of a k-dimensional signature vector. Keogh et al. call the method *Piecewise Constant Approximation*, or PCA. This deceptively simple dimensionality reduction technique has several advantages [Keogh et al. 2001b]: The transform itself is faster than most other transforms, it is easy to understand and implement, it supports more flexible distance measures than Euclidean distance, and the index can be built in linear time.

Figure I.8 shows an approximated time sequence, reconstructed from a ten-dimensional PCA signature.

Yi and Faloutsos [2000] also show that this signature can be used with arbitrary L_p norms without changing the index structure, which is something no previous method (such as Agrawal et al. [1993, 1995], Faloutsos et al. [1994, 1997], Rafiei and Mendelzon [1997], or Yi et al. [1998]) could accomplish. This means that the distance measure may be specified by the user. Preprocessing to make the index more robust in the face of such transformations as *offset translation*, *amplitude scaling*, and *time scaling* can also be performed.

Keogh et al. demonstrate that the representation can also be used with the so-called *weighted Euclidean distance*, where each part of the sequence has a different weight.

Empirically, the PCA methods seem promising: Yi and Faloutsos demonstrate up to a ten times speedup over methods based on the discrete wavelet transform. Keogh et al. do not achieve similar speedups, but point to the fact that the structure allows for more flexible distance measures than many of the competing methods.

Keogh et al. [2001a] later propose an improved version of the PCA, the so-called *Adaptive Piecewise Constant Approximation*, or APCA. This is similar to the PCA, except that the segments need not be of equal length. Thus regions with great fluctuations may be represented with several short segments, while reasonably featureless regions may be represented with fewer, longer segments. The main contribution of this representation is that it is a more effective compression than

the PCA, while still representing the original faithfully.

Two distance measures are developed for the APCA, one which is guaranteed to underestimate Euclidean distance, and one which can be calculated more efficiently, but which may generate some false dismissals. It is also shown that this technique, like the PCA, can handle arbitrary L_p norms. The empirical data suggest that the APCA outperforms both methods based on the discrete Fourier transform, and methods based on the discrete wavelet transform with a speedup of one to two orders of magnitude.

In a recent paper, Keogh [2002] develops a distance measure that is a lower bound for dynamic time warping, and uses the PCA approach to index it. The distance measure is based on the assumption that the allowed warping is restricted, which is often the case in real applications. Under this assumption, Keogh constructs two warped versions of the sequence to be indexed: An upper and a lower limit. The PCA signatures of these limits are then extracted, and together with with Keogh's distance measure form an exact index (one with no false dismissals) with high precision. Keogh performs extensive empirical experiments, and his method clearly outperforms any other existing method for indexing time warping.

I.3.4 Landmark Methods

In 1997, Keogh and Smyth introduce a probabilistic method for sequence retrieval, where the features extracted are characteristic parts of the sequence, so-called *feature shapes*. Keogh [1997] uses a similar *landmark based* technique. Both these methods also use the dimensionality reduction technique of piecewise linear approximation (see Section 4.2) as a preprocessing step. The methods are based on finding similar landmark features (or shapes) in the target sequences, ignoring shifting and scaling within given limits. The technique is shown to be significantly faster than sequential scanning (about an order of magnitude), which may be accounted for by the compression of the piecewise linear approximation. One of the contributions of the method is that it is one of the first that allows some longitudinal scaling.

A more recent paper by Perng et al. [2000] introduces a more general landmark model. In its most general form, the model allows any point of great importance to be identified as a landmark. The specific form used in the paper defines an n-th order landmark of a one-dimensional function to be a point where the function's n-th derivative is zero. Thus, first-order landmarks are extrema, second-order landmarks are inflection points, and so forth. A smoothing technique is also introduced, which lets certain landmarks be overshadowed by others. For instance, local extrema representing small fluctuations may not be as important as a global maximum or minimum.

Figure I.9 shows an approximated time sequence, reconstructed from a landmark

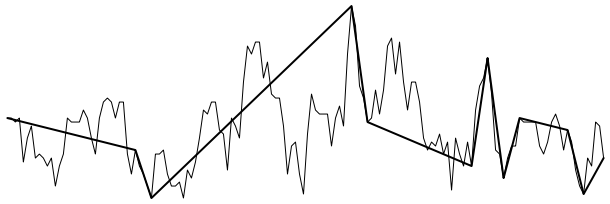

Figure I.9: A landmark approximation

signature.

One of the main contributions of Perng et al. [2000] is showing that for suitable selections of landmark features, the model is invariant with respect to the following transformations:

- Shifting
- Uniform amplitude scaling
- Uniform time scaling
- Non-uniform time scaling (time warping)
- Non-uniform amplitude scaling

It is also possible to allow for several of these transformations at once, by using the intersection of the features allowed for each of them. This makes the method quite flexible and robust, although as the number of transformations allowed increases, the number of features will decrease; consequently, the index will be less precise.

A particularly simple landmark based method (which can be seen as a special case of the general landmark method) is introduced by Kim et al. [2001]. They show that by extracting the minimum, maximum, and the first and last elements of a sequence, one gets a (rather crude) signature that is invariant to time warping. However, since time warping distance does not obey the triangle inequality [Yi et al. 1998], it cannot be used directly. This problem is solved by developing a new distance measure that underestimates the time warping distance while simultaneously satisfying the triangle inequality. Note that this method does not achieve results comparable to those of Keogh [2002].

I.4 Other Approaches

Not all recent methods rely on spatial access methods. This section contains a sampling of other approaches.

I.4.1 Using Suffix Trees to Avoid Redundant Computation

Baeza-Yates and Gonnet [1999] and Park et al. [2000] independently introduce the idea of using suffix trees [Gusfield 1997] to avoid duplicate calculations when using dynamic programming to compare a query sequence with other sequences in a database. Baeza-Yates and Gonnet use *edit distance* (see Appendix A for details), while Park et al. use time warping.

The basic idea of the approach is as follows: When comparing two sequences $\boldsymbol{x}$ and $\boldsymbol{y}$ with dynamic programming, a subtask will be to compare their prefixes $\boldsymbol{x}_{1:i}$ and $\boldsymbol{y}_{1:j}$. If two other sequences are compared that have identical prefixes to these (for instance, the query and another sequence from the database), the same calculations will have to be performed again. If a sequential search for subsequence matches is performed, the cost may easily become prohibitive.

To avoid this, all the sequences in the database are indexed with a suffix tree. A suffix tree stores all the suffixes of a sequence, with identical prefixes stored only once. By performing a depth-first traversal of the suffix tree one can access every suffix (which is equivalent to each possible subsequence position) and backtrack to reuse the calculations that have already been performed for the prefix that the current and the next candidate subsequence share.

Baeza-Yates and Gonnet assume that the sequences are strings over a finite alphabet; Park et al. avoid this assumption by classifying each sequence element into one of a finite set of categories. Both methods achieve subquadratic running times.

I.4.2 Data Reduction through Piecewise Linear Approximation

Keogh et al. have introduced a dimensionality reduction technique using piecewise linear approximation of the original sequence data [Keogh 1997; Keogh and Smyth 1997; Keogh and Pazzani 1998, 1999a,b]. This reduces the number of data points by a compression factor typically in the range 10 to 600 for real data [Keogh 1997], outperforming methods based on the Discrete Fourier Transform by one to three orders of magnitude [Keogh and Pazzani 1999b]. This approximation is shown to be valid under several distance measures, including dynamic time warping distance [Keogh and Pazzani 1999b]. An enhanced representation is introduced by Keogh and Pazzani [1998], where every line segment in the ap-

proximation is augmented with a weight representing its relative importance; for instance, a combined sequence may be constructed representing a class of sequences, and some line segments may be more representative of the class than others.

I.4.3 Search Space Pruning through Subsequence Hashing

Keogh and Pazzani [1999a] describe an indexing method based on hashing, in addition to the piecewise linear approximation. An equi-spaced template grid window is moved across the sequence, and for each position a hash key is generated to decide into which *bin* the corresponding subsequence is put. The hash key is simply a binary string, where 1 means that the sequence is predominantly increasing in the corresponding part of the template grid, while 0 means that it is decreasing. These bin keys may then be used during a search, to prune away entire bins without examining their contents. To get more benefit from the bin pruning, the bins are arranged in a *best-first* order.

I.5 Conclusion

This chapter has sought to give an overview of recent advances in the field of similarity based retrieval in time sequence databases. First, the problem of similarity search and the desired properties of robust distance measures and good indexing methods were outlined. Then, the general approach of signature based similarity search was described. Following the general description, three specific signature extraction approaches were discussed: Spectral signatures, based on Fourier components (or wavelet components); piecewise constant approximation, and the related method adaptive piecewise constant approximation; and landmark methods, based on the extraction of significant points in a sequence. Finally, some methods that are not based on signature extraction were mentioned.

Although the field of time sequence indexing has received much attention and is now a relatively mature field [Keogh et al. 2002] there are still areas where further research might be warranted. Two such areas are (1) thorough empirical comparisons and (2) applications in data mining.

The published methods have undergone thorough empirical tests that evaluate their performance (usually by comparing them to sequential scan, and, in some cases, to the basic spectral signature methods), but comparing the results is not a trivial task—in most cases it might not even be very meaningful, since variations in performance may be due to implementation details, available hardware, and several other factors that may not be inherent in the indexing methods themselves. Implementing several of the most promising methods and testing them

on real world problems (under similar conditions) might lead to new insights, not only about their relative performances in general, but also about which methods are best suited for which problems. Although some comparisons have been made (such as in Wu et al. [2000] and, in the more general context of spatial similarity search, in Weber et al. [1998]), little research seems to have been published on this topic.

Data mining in time series databases is a relatively new field [Keogh et al. 2002]. Most current mining methods are based on a full, linear scan of the sequence data. While this may seem unavoidable, constructing an index of the data could make it possible to perform this full data traversal only once, and later perform several data mining passes that only use the index to perform their work. It has been argued that data mining should be interactive [Das et al. 1998], in which case such techniques could prove useful. Some publications can be found about using time sequence indexing for data mining purposes (such as Keogh et al. [2002], where a method is presented for mining patterns using a suffix tree index) but there is still a potential for combining existing data mining techniques with existing methods for similarity-based sequence retrieval.

Paper II

Temporal Rule Discovery using Genetic Programming and Specialized Hardware

Abstract: *Discovering association rules is a well-established problem in the field of data mining, with many existing solutions. In later years, several methods have been proposed for mining rules from sequential and temporal data. This paper presents a novel technique based on genetic programming and specialized pattern matching hardware. The advantages of this method are its flexibility and adaptability, and its ability to produce intelligible rules of considerable complexity.*

Keywords: *Time series, sequence mining, rule discovery, genetic programming, pattern matching hardware*

II.1 Introduction

Discovering association rules is a well-established problem in the field of data mining, with many existing solutions. In later years, several methods have been proposed for mining rules from sequential and temporal data (see, for example, [Agrawal and Srikant 1995; Das et al. 1998; Mannila et al. 1997]). Quite a few of these methods are based on the common premise of counting the occurrences of viable rules or patterns. While this approach has the advantage of finding all highly frequent patterns, it constrains the set of possible solutions.

One promising alternative is to use evolutionary algorithms, as described in [Fre-

itas 2002]. While this approach places fewer restrictions on the form of patterns and rules that can be discovered, the performance of the method relies heavily on a speedy evaluation of each candidate rule, and such an evaluation typically involves examining the entire training data set. When mining rules in relational databases, existing indexing methods make it possible to efficiently calculate the fitness of each rule. When mining sequences for complex patterns, this evaluation is not quite as straightforward. Efficient indexing methods for some forms of patterns exist, (for example, using Patricia trees, as in [Baeza-Yates and Gonnet 1996], or multigram indices, as in [Cho and Rajagopalan 2002]), but in this paper we use a specialized co-processor designed to perform very advanced, high volume pattern matching.

The paper is structured as follows: Section II.1.1 describes the problem we are trying to solve in more formal terms; Section II.1.2 gives a brief overview of some related work; Section II.2 describes the specifics of our method; Section II.3 contains some empirical results; and, finally, Section II.4 concludes the paper.

II.1.1 Problem Definition

Our problem is as follows: 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.

II.1.2 Related Work

A survey of association rule mining algorithms can be found in [Hipp et al. 2000]. The underlying principle in these algorithms is exploiting the lattice structure of the pattern space when searching for frequent patterns and rules. This principle has also been applied to sequence mining [Agrawal and Srikant 1995; Mannila et al. 1997; Das et al. 1998]. The assumptions about the pattern space makes this general approach unsuitable for mining more complex patterns and rules.

Even though the problem tackled in this paper is closer to that of sequence prediction than that of sequence mining (see [Sun and Giles 2000] for several sequence prediction algorithms), the goal of our method is to find rules that are readable and understandable by a human expert. Since this is one of the fundamental goals of data mining and knowledge discovery, we have chosen to classify our method as a rule discovery method.

A problem similar to ours is tackled in [Giles et al. 2001], where Giles et al. use recurrent neural networks to predict fluctuations in foreign exchange rates. In addition to the prediction task, their method encompasses the extraction of deterministic finite state automata, which are equivalent to regular expressions. Like most current sequence learning methods, the algorithms works with a fixed-width sliding window. We have tested our method on the same data sets as [Giles et al. 2001] in Section II.3.3.

II.2 Method

To evolve our predictor rules we use genetic programming with the rule encoding scheme referred to in [Freitas 2002] as the Michigan approach, that is, each individual in the population represents a single rule. Since the consequent is a part of the problem definition, each rule is represented by its antecedent, an expression in a general query language that can be interpreted by our pattern matching hardware (see Section II.2.1).

The training data consist of a discrete sequence s with elements from some finite alphabet, and predicate p, represented by the set of indices for which p is true. The rules are evaluated by having the hardware interpret their antecedents as queries, and comparing the returned hit positions, that is $\hat{p}$, with the correct positions given by p.

Compared to existing methods for discovering sequential rules, our method has two main advantages: (1) It can produce rules in a very rich rule language, which can be customized to each application as needed, and (2) each rule has the entire history of its sequence available when making a prediction. This means that the method is quite flexible, and that restrictions on the rule format are mainly dictated by the problem, rather than by the method itself.

A similar approach was originally used to evolve patterns for classifying human genomes. This is the subject of an upcoming paper.[1] The hardware and the process of fitness evaluation are discussed in the following subsections.

II.2.1 The Pattern Matching Chip

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 [Interagon AS 2000]. A detailed description is the subject of an upcoming

[1] Svingen, Sætrom, Hetland: "Pattern Evolution" (manuscript in preparation)

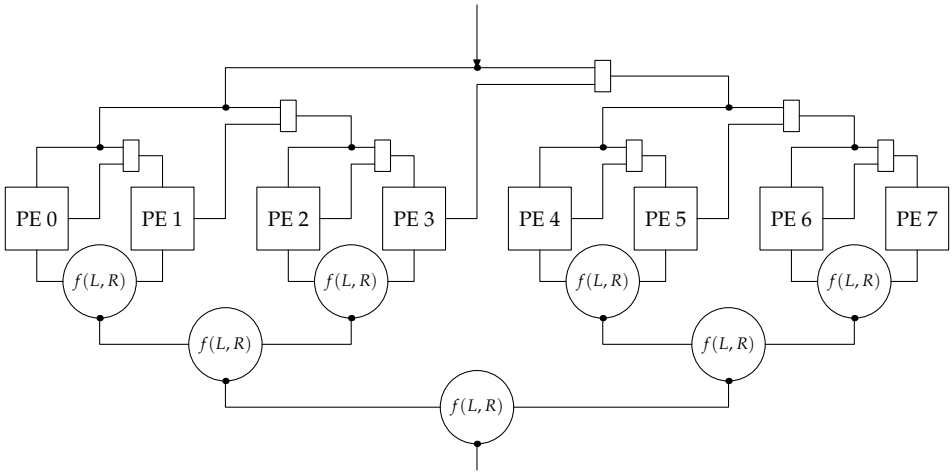

Figure II.1: A data distribution tree with eight leaf nodes (PEs), and the corresponding result gathering tree with $f(L, R)$ calculated from the above left (L) and right (R) results

paper;[2] a brief description will be given here.

The PMC consists of three functional units, as illustrated in Figure II.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. Each PE is configured with one byte (character) and a comparison operator, which it uses to look for bytes in the data stream. Matches are reported to the result gathering tree, which combines them 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.

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.)[3] 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, and generalized Boolean operations. The regular expression syntax is similar to that of UNIX tools such as `grep`. A detailed description of the language is available online[4] and is the subject of an upcoming paper.[5]

[2] Svingen, Halaas, Birkeland, Nedland: "The Pattern Matching Chip" (manuscript in preparation)
[3] The prototype used in our experiments runs at 33 MB/s and handles up to 4 parallel queries.
[4] `http://www.interagon.com/pub/whitepapers/IQL.reference-latest.pdf`
[5] Svingen, Halaas, Hetland: "The Interagon Query Language" (manuscript in preparation)

II.2.2 Rule Evaluation

In evolutionary algorithms (such as genetic programming) each individual of the population must be assigned a fitness score, which describes how well the individual solves the problem at hand. In rule mining there are several possible quality measures, including various measures of interestingness. In this paper we focus on predictive power, because the events to be predicted (given by p) are part of the problem definition, which makes most interestingness measures unsuitable.

Some measures of predictive power (precision, true positive rate, true negative rate, and accuracy rate, as well as some combinations) are described in [Freitas 2002, pp. 129–134]. In this paper we have used another measure, the correlation coefficient of the set of data points given by $(p(i), \hat{p}(i))$, for $1 \leq i \leq n$ (where n is the size of the training sequence). This can be interpreted as the cosine of the angle between two n-dimensional vectors p and $\hat{p}$, which means that we get a fitness of $+1.0$ for perfect prediction and -1.0 for completely erroneous prediction.

By classifying the hits reported by a given rule as true or false positives (correct or incorrect hits), the correlation can be expressed as

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

Here TP and FP are the number of true and false positives, respectively. The number of true and false negatives (TN and FN) can easily be calculated from TP, FP, and the total number of positives and negatives (directly available from the training data).

II.3 Experiments

The method was tested on three data sets: Synthetic data consisting of uniformly random symbols where certain positions were flagged according to predetermined rules; DNA sequence data, where the task was to predict the location of intron/exon splice sites; and foreign exchange rates, where the goal was to predict the trend for the next day (as in [Giles et al. 2001]). In the first two experiments we were mainly interested in the expressive power of our rule format, while in the last experiment we focused on predictive power.

For the first two data sets we used a "fuzzy" version of the problem definition from Section II.1.1: For this fuzzy problem, the prediction predicate is the disjunction $p(i+\delta) \vee \cdots \vee p(i+\delta+\varepsilon)$ for some fixed window size ε. For these two data sets we used $\varepsilon = 10$. The original definition, which was used with the currency data, corresponds to $\varepsilon = 1$. For all the data sets we used $\delta = 1$.

Table II.1: Results on synthetic data set

Type	Antecedent Expression	Corr.
1	`([^n]+.+o[^n]+)n`	1.0
2	`{{a:d=9},{b:d=9},{c:d=9},{d:d=9},{e:d=9}:p=5}`	1.0

For the last two data sets the technique of early stopping was used to prevent overfitting. This simply means that, in addition to a training set and a test set, we used a validation set, and the fitness calculated for this data set was used to select the best rules.

II.3.1 Synthetic Data

The synthetic data were constructed by repeatedly drawing symbols from the lowercase Latin alphabet (`a`–`z`) with a uniform probability distribution. The hit positions (representing the predicate p) were then found according to some predetermined antecedent patterns.

Two different antecedents types were used:

1. The regular expression `o[^n]*n`.

2. The letters `a`, `b`, `c`, `d` and `e` occurring in any order within a window of width 10.

The two sets consisted of 1 MB of sequence data with about 20000 and 1600 occurrences of antecedent type 1 and 2, respectively.

For the second antecedent type we wanted to simulate a search for fixed-width window rules, so we introduced a new operator into the language, composed from existing operators. The function of this operator was to match any number of arbitrary symbols (specified as parameters to the operator) that were all found within a window of width 10.

The system was able to generate expressions that recognized all occurrences of both rules. Table II.1 lists two of the expressions generated. As can be seen from this table, both problems were solved with a perfect correlation (100% prediction rate) for the test set. This rate is certainly a result of the data set being particularly well suited for our rule format.

Table II.2: Results on DNA data set

Type	Antecedent Expression	Corr.
5′	({t[ag]:d=52}{c:c=21}) ({(<=c):d=3}gg)t[ag]ag	0.266
3′	({[ct][ct][ct][cgt][ct][ct][ct][ct]:d=9}[ct]ag) &ag&(>=atctgt)	0.177

II.3.2 DNA Sequence Data

The goal of this experiment was to find rules predicting intron/exon splice sites i DNA sequences. In addition to testing for predictive power, the rules were informally validated by comparing them to well-known splice site patterns.

The DNA sequences and exon locations were retrieved from NCBI.[6] The combined data set consisted of more than 6000 DNA sequences totalling 34 MB and about 20000 splice sites (of types 5′ and 3′, representing the beginning and end of an intron, respectively). The first 10 MB of this set were used in the training process (8 MB for training and 2 MB for early stopping.) The rest of the set was used for testing the generated rules.

Table II.2 lists the rules produced for the 5′ and 3′ splice sites. The results are comparable to previously published splice site patterns (see, for example, [Burge et al. 1999]).

II.3.3 Foreign Exchange Rates

This experiment was performed on 5 sets of foreign exchange rates.[7] Because the number of data points was quite small (fewer than 4000 in total), tenfold cross-validation was used on each data set. The training data were discretized using the clustering method described in [Das et al. 1998], and the resulting clusters were used to discretize the test and evaluation sets.

The average correlation and average prediction rate for each of the five currencies is listed in table II.3. Giles et al. [Giles et al. 2001] report a prediction rate of 52.9% on the same data set.

Due to the sequential nature of the data, the use of cross-validation is problematic. Therefore, we performed the same experiment on a larger data set,[8] with about

[6] http://www.ncbi.nlm.nih.gov

[7] Available from http://www-psych.stanford.edu/~andreas/Time-Series/Data/Exchange.Rates.Daily

[8] The exchange rate of BP to USD from 1971 to 2002, available from http://www.federalreserve.

Table II.3: Results on currency data sets

Set	Avg. corr.	Avg. pred.	Max. pred.	Min. pred.
1	0.0465	54.5%	58.7%	50.5%
2	0.0446	54.3%	60.9%	45.1%
3	0.0845	54.8%	61.4%	48.4%
4	0.0527	56.1%	62.0%	52.2%
5	−0.0164	50.6%	57.6%	43.5%

8000 data points, using the first half as the training set. The experiment was repeated five times, giving prediction rates of 52.4%, 64.8%, 59.4%, 52.9%, and 65.1%.

II.4 Summary and Conclusions

The experiments performed so far seem promising: The method produces rules with good predictive power that are also readable by humans. For the synthetic data, we were able to reproduce the original rules, which contained regular expressions of varying complexity, without restriction on sequence history. For the DNA sequence data, our method produced rules with relatively good predictive power, which resemble well-known intron/exon splice site motifs. For the foreign exchange rates we were able to achieve a prediction rate comparable to that of Giles et al. in [Giles et al. 2001]. In each case, the rules were fairly easy to interpret, which means that it should be possible for a domain expert to find some structure in them, and perhaps modify them to a simpler or more general form, and to test them in real time, using the pattern matching hardware.

gov/Releases/H10/Hist

Paper III

The Role of Discretization Parameters in Sequence Rule Evolution

Abstract: *As raw data become available in ever-increasing amounts, there is a need for automated methods that extract comprehensible knowledge from the data. In our previous work we have applied evolutionary algorithms to the problem of mining predictive rules from time series. In this paper we investigate the effect of discretization on the predictive power of the evolved rules. We compare the effects of using simple model selection based on validation performance, majority vote ensembles, and naive Bayesian combination of classifiers.*

III.1 Introduction

As raw data become available in ever-increasing amounts, there is a need for automated methods that extract comprehensible knowledge from the data. The process of knowledge discovery and data mining has been the subject of much research interest in later years, and one recent subfield is that of sequence mining. In our previous work [Hetland and Sætrom 2002] we have applied evolutionary algorithms to the problem of mining predictive rules from time series. Our method involves discretizing the time series, in order to be able to evaluate our rules on them (which, in turn, allows us to use genetic programming to find such rules). This process of discretization discards some information about the time series, and the parameters chosen (such as the width of the discretization window) will determine which features are available for the mining algorithm.

In this paper we investigate the effect of discretization on the predictive power of the evolved rules. We evolve rules for different discretization parameter settings and design predictors using the following methods:

1. We test rules evolved for different settings on a validation set. We then take the rule with the best performance to be our predictor.
2. We select the best rule for each of five discretization resolutions and combine them to form a predictor ensemble by simple majority vote.
3. We select the best rule for each of five discretization resolutions and combine them by means of a naive Bayesian model.

Majority vote ensembles are a simple way of combining several predictors to achieve increased predictive power. To make a prediction, all the component predictors make their predictions; a simple majority vote is used to decide which prediction "wins".

The naive Bayesian predictor is a simple probabilistic model that assumes conditional independence among the predictor variables. Even though this assumption may be too strong in many cases, there is much empirical evidence suggesting that the method is quite robust.

III.2 Method

In this section we first describe the general approach, as developed in [Hetland and Sætrom 2002], as well as the extensions introduced here, that is, combining predictors for several resolutions. Finally, the discretization method is discussed.

III.2.1 The General Mining Approach

The basic method works by using an evolutionary algorithm (EA) to develop rules that optimize some score (or fitness) function, which describes what qualities we are looking for. Three components are of central importance: The rule format, the mechanism for rule evaluation, and the fitness function.

For the purposes of this discussion, we only consider one rule format: The rule consequent is some given event that may occur at given times, possibly extrinsic to the series itself, while the antecedent (the condition) is a pattern in a very expressive pattern language called IQL [Interagon AS 2002]. This language permits, among other things, such constructs as string matching with errors, regular expressions, shifts (latency), and Boolean combinations of expressions. In our EA system, these expressions are represented as syntax trees, in the standard manner.

In order to calculate the fitness of each rule, we need to know at which positions in the data the antecedent is true. To ascertain this, we use the antecedent as a query in an information retrieval system, the Pattern Matching Chip (PMC) [Interagon AS 2000], for which our pattern language was designed. Such a chip is able to process about 100 MB/s.

There are many possible fitness functions that measure the precision, degree of recall, or interestingness of a query or rule. For the relatively straightforward prediction task undertaken here, simple correlation is quite adequate. We use a supervised learning scheme, in which we mark the positions where want the rule to make a prediction, and then, for each rule, calculate the correlation between its behaviour and the desired responses. This can be calculated from the confusion matrix of the rule (true and false positives and negatives) [Hetland and Sætrom 2002].

It is worth mentioning that even though we focus mainly on predictive power here, one of the strengths of the method is the transparency of its rule format. Rather than a black-box predictor, the user receives a rule in a human-readable language. This can be an advantage in data mining contexts.

III.2.2 Selecting and Combining Predictors

We want to investigate the effect of the parameters of the discretization process, or, more specifically, of the resolution and the alphabet size, on the predictive power of the developed rules. The specific meaning of resolution and alphabet size is given in Section III.2.3; informally, the resolution refers to the width of each (overlapping) discretized segment of the time series, while the alphabet size is simply the cardinality of the alphabet used in the resulting strings.

As mentioned in the introduction, we will compare three methods of exploiting the variability introduced by these parameters: model selection through cross-validation, majority vote ensembles, and naive Bayesian combination of predictors.

The rules that are developed by our system have their performance tested on a validation set (done through a form of k-fold cross-validation). In the simple case, a single resolution and alphabet size is used, and the rule that has 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 choose the rule (and, implicitly, the parameter setting) that has the highest validation rating. This way we can discover the resolution and alphabet size that best suits a given time series, without having to arbitrarily set these beforehand. To demonstrate the effect of this procedure, we also select the rule (and parameter setting) that has the lowest validation score, as a baseline.

The next method is the majority vote ensemble. For this, we run the basic process for each parameter setting, and for each resolution we choose the rule (and, implicitly, the alphabet size) that has the best validation performance. We then combine these rules to form a single predictor, through voting. In other words, when we observe a time series (for example, the test set) we discretize it at all the resolutions, with the given alphabet sizes, (possibly incrementally) and run each rule in its own resolution. The prediction ("up" or "down") that is prevalent (simple majority) among the rules is taken to be the decision of the combined predictor. Ensembles are described in more detail in [Dietterich 2000]. The basic idea behind them is that if each of a set of classifiers (or predictors) is correct with a probability $p \geq 0.5$ and the classifiers are generally not wrong at the same time (that is, they are somewhat independent) then a majority vote will increase the probability of a correct classification.[1] More sophisticated ensemble methods exist; see [Dietterich 2000] for details.

The naive Bayesian predictor is a simple probabilistic model that assumes conditional independence among the predictor variables. Assume that we have a family $\{X_i\}$ of predictor variables, and one dependent (predicted) variable Y. Assume that $P(Y = y)$ is the *a priori* probability of a given value y being observed for Y, and $P(X_i = x_i \mid Y = y)$ is the conditional probability of observing x for X_i, given that $Y = y$. Then, the naive Bayesian model states that our predicted class should be

$$\zeta(\boldsymbol{x}) = \arg\max_y P(Y = y) \prod_i P(X_i = x_i | Y = y) \ . \tag{III.1}$$

The independence assumption may in many cases be quite strong, but the naive Bayesian model can be surprisingly robust, and, as shown in [Domingos and Pazzani 1997], may even be optimal in cases where the independence assumption is violated.

III.2.3 Feature Extraction and Discretization

Many features may potentially be extracted from time series; for the task of prediction, we need to extract features in a way that lets us access the features of a sequence prefix. A natural approach is to divide the sequence into (overlapping) windows of width w, and to extract features from each of these. This discretization approach is described in the following.

Our basic discretization process is a simple one, used, for example, in [Keogh et al. 2002]. It works as follows. A time series $\boldsymbol{x} = (x_i)_{i=1}^n$, where $x_i \in \mathbb{R}$, is discretized by the following steps:

[1] Note that because of this requirement, rules with lower than 50% validation accuracy were excluded from the ensembles.

1. Use a sliding window of width k to create a sequence $\boldsymbol{w}$ of m overlapping vectors. More formally: Create a sequence of windows $\boldsymbol{w} = (w_i)_{i=1}^{m}$, where $w_i = (x_j)_{j=l(i)}^{r(i)}$, such that $l(1) = 1$, $r(m) = n$, $r(i) - l(i) = k - 1$ for $1 \leq i \leq m$, and $l(i) = l(i-1) + 1$ for $1 < i \leq m$. The window width is also referred to as the *resolution*.
2. For some feature function $f : \mathbb{R}^k \to \mathbb{R}$, create a feature sequence $\boldsymbol{f} = (f(w_i))_{i=1}^{m}$.
3. Sort $\boldsymbol{f}$ and divide it into a segments of equal length. We refer to a as the *alphabet size*.
4. Use the limit elements in the sorted version of $\boldsymbol{f}$ to classify each element in $\boldsymbol{f}$, creating a new sequence $\boldsymbol{s}$ where s_i is the number of the interval where f_i is found.

After this discretization process, we have a sequence $\boldsymbol{s}$, where s_i is an integer such that $1 \leq s_i \leq a$, and each integer occurs the same number of times in $\boldsymbol{s}$. For convenience, we map these integers to lowercase letters (so that for $a = 3$ our alphabet is a...c). This discretization is performed on the training set, and the resulting limits are used to classify the features of the test set. (This means that there is no guarantee that each character is represented an equal number of times in the test set.) Many feature functions f are possible, such as average value or signal-to-noise ratio. Since we are interested in the upward and downward movement of the series, we have chosen to use the slope of the line fitted to the points in each window through linear regression.

III.3 Experiments

In our experiments we use six data sets, available from the UCR Time Series Data Mining Archive [Keogh and Folias 2002] (the first five) and the Federal Reserve Board [Release 2002] (the last data set):

Random walk. A standard random walk, where each value is equal to the previous one, plus a random number.

ECG. ECG measurements from several subjects, concatenated.

Earthquake. Earthquake-related seismic data.

Sunspots. Monthly mean sunspot numbers from 1749 until 1990. The series is periodic, with one period of eleven years, and one of twenty-seven days.

Network. Packet round trip time delay for packets sent from UCR to CMU.

Exchange rates. Daily exchange rate of Pound Sterling to US Dollars.

For each of the data sets, the target prediction was when the series moved upward, that is, when the next value was greater than the current. It is to be expected that good predictors can be found for the ECG data (as it is quite regular and periodic), whereas the random walk data, and, to some extent, the exchange rate data, function as baselines for comparison. Finding predictors for random data would clearly indicate that our experiments were flawed, and finding good predictors for exchange rate data would also be surprising, as this is considered a quite difficult (if at all possible) task.

For each of the data sets 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 tenfold cross-validation[2] and early stopping; that is, rules were selected based on their performance on a separate validation set before they were tested. As described in Section III.2.2, results were used to select the alphabet size that gave the best single-predictor performance for each resolution, and these rules were then used in constructing ensembles and Bayesian classifiers.

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. The results are summarized in Figure III.1. 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 and exchange rate data sets there are no significant differences, as expected.

Figure III.2 shows how rule accuracy is related to window and alphabet size. For the random and exchange rate data, no clear trend is discernible. Although there are clearly problem specific differences, for the ECG, sunspot, and network data, there seems to be a rough inverse relationship between window size and accuracy, regardless of alphabet size. For the earthquake data, a large alphabet makes up for poor resolution, giving peak performance for the two largest window sizes and the most fine-grained alphabet.

[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.

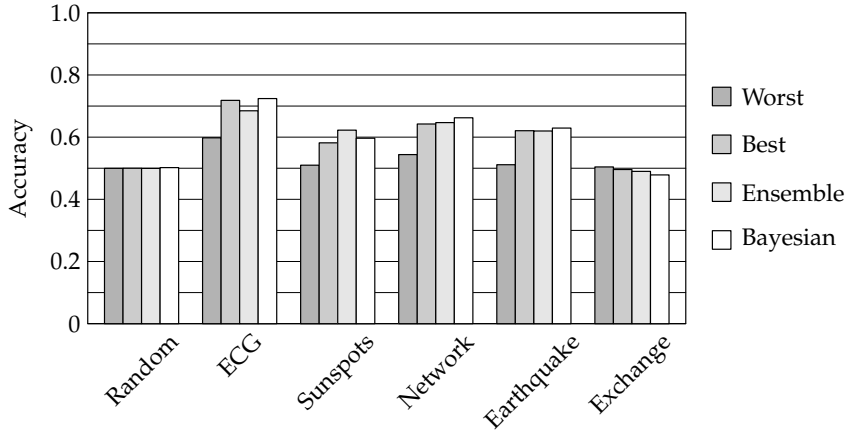

Figure III.1: Performance comparison

III.4 Discussion

In this paper we have examined the role of discretization when evolving time series predictor rules. We used three main techniques to improve the basic evolution presented in [Hetland and Sætrom 2002]: Model selection based on validation performance, majority vote ensembles, and naive Bayesian classifiers. Prior to our empirical study, we expected the ensembles to outperform the simple selection, and the Bayesian classifiers to outperform both of the other methods. As it turns out, on our data, there was no statistically significant difference between the three methods, even though the difference between the result they produced and that produced by unfavorable parameter settings (discretization resolution and alphabet size) was highly significant. This leads to the conclusion that, given its simplicity, plain model selection may well be the preferred method. Our experiments also showed that the relationship between discretization resolution, alphabet size, and prediction accuracy is highly problem dependent, which means that no discretization parameters can be found that work equally well in all cases.

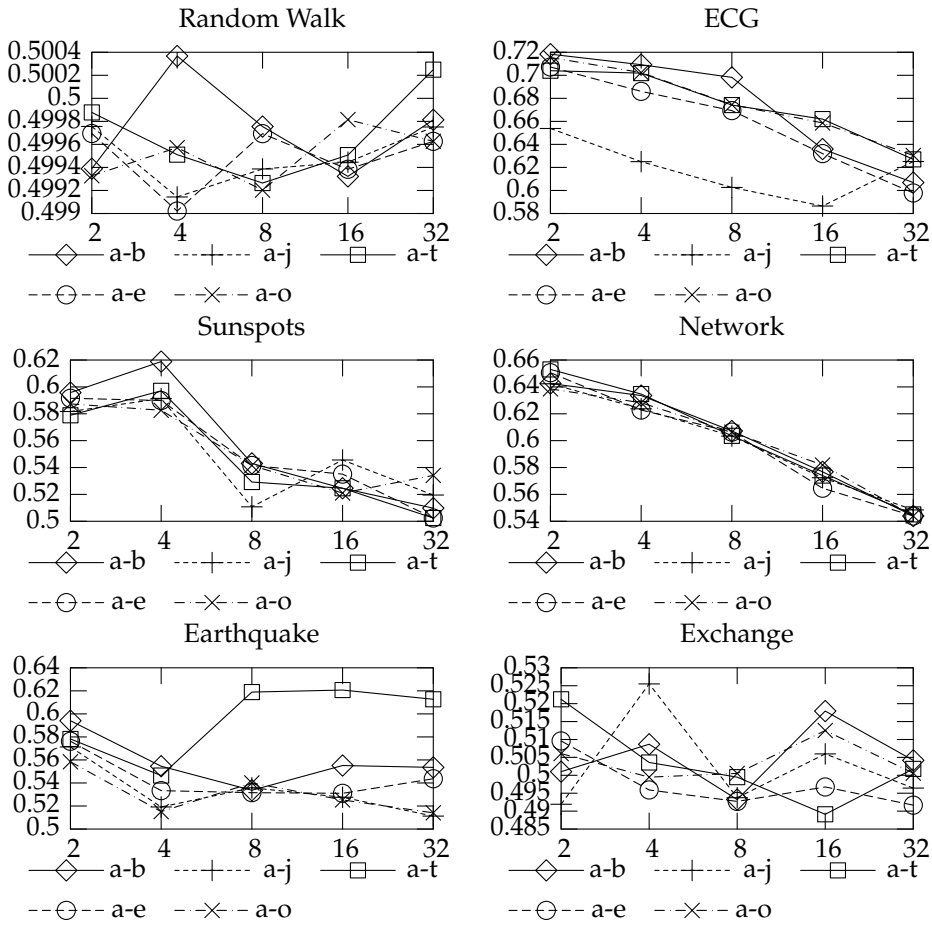

Figure III.2: Accuracy as function of window size and alphabet

Paper IV

Unsupervised Temporal Rule Mining with Genetic Programming and Specialized Hardware

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.*

Keywords: *Data mining, rule discovery, time series, genetic programming, pattern matching hardware.*

IV.1 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 [Keogh et al. 2002]. 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, [Mannila et al. 1997; Agrawal and Srikant 1995]) 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 [Das et al. 1998]) 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 [Hetland and Sætrom 2002]: with the aid of specialized pattern matching hardware we find sequential rules using genetic programming. In [Hetland and Sætrom 2002] 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.

IV.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 and Sætrom 2002; Weiss and Hirsh 1998; Zemke 1998; Povinelli 2000] fall in this category. All of these use some form of evolutionary computation; [Hetland and Sætrom 2002] uses genetic programming, while the others use genetic algorithms.

In the second type, unsupervised methods, the only input to the rule mining al-

gorithm 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 et al. 1997; Agrawal and Srikant 1995; Höppner and 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 and Yohai 2001], or finding unusual temporal patterns in market basket data [Chakrabarti et al. 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 method.

IV.1.2 Structure of This Paper

The rest of this paper is structured as follows: Section IV.2 describes the preprocessing scheme used to discretize the time series data used in the experiments, Section IV.3 describes how genetic programming is used to evolve temporal rules, Section IV.4 describes in detail how rules are evaluated, Section IV.5 describes our experiments and empirical results, and finally Section IV.6 summarizes and concludes the paper.

IV.2 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 [Keogh et al. 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 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.4.1.

This discretization method is by no means unique. In [Last et al. 2001] 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 [Hetland 2003] for a survey.

IV.3 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. Expressions in the chosen rule languages may be evaluated by the specialized pattern matching hardware described in Section IV.4.2, and rule fitness is calculated by searching the time series data for rule occurrences.

IV.3.1 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 [Das et al. 1998], 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: $x \stackrel{T}{\Rightarrow} y$ for $x, y \in A$.

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

1. *Sequential patterns* [Agrawal and Srikant 1995]: If x_1 and x_2 and ... 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* [Mannila et al. 1997]: If the sequence $x_1, \ldots, x_n$ 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 $\mathrm{a}, \{\mathrm{c}, \mathrm{d}\} \stackrel{t}{\Rightarrow} \mathrm{y}$ 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 $L_a = L_c$.

IV.3.2 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 $L_a \stackrel{T}{\Rightarrow} L_c$. 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 closure*. The leaf nodes in these branches are the symbols from the antecedent and consequent alphabets (Σ_a and Σ_c).

[1] One notable exception is [Spiliopoulou 1999], 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.

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 $T + 1$.

IV.3.3 Confidence, Support, and Interestingness

Given a rule $R = R_a \stackrel{t}{\Rightarrow} R_c$ in the rule language $L_a \stackrel{T}{\Rightarrow} L_c$ (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

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

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:

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

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:

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

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 [Hilderman and 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 and Goodman 1991]). This is defined as:

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

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 [Piatetsky-Shapiro 1991]. 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]

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

with the same definitions for the probabilities as for the J-measure. As can be seen from (IV.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.4 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 IV.3.3, 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:

$$\widehat{p}(R_a) = f(R_a) \tag{IV.6}$$

$$\widehat{p}(R_c^t) = f(R_c^t) \tag{IV.7}$$

$$\widehat{p}(R_c^t|R_a) = c(R) \tag{IV.8}$$

This amounts to counting the following:

- The number of occurrences of R_a in S (from the definition of $f(R_a)$.)

[2]Note that this is an adaptation of the definition in [Piatetsky-Shapiro 1991], where the function is defined for simple classification rules where R_a and R_c are single symbols.

- 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.

IV.4.1 Handling Correlations Caused by the Discretization Method

The discretization process described in Section IV.2 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 [Das et al. 1998]:

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

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

IV.4.2 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 [Interagon AS 2000; Fast Search & Transfer ASA] 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 [Interagon AS 2002]. This language supports such language features as regular expressions, latency (distance), Boolean combinations, and alpha-numerical comparisons.

[3] Note that this differs from the definition in [Das et al. 1998], 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 (IV.2).

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

As described in Section IV.4, 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.4.1. 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 match-

ing 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.

IV.5 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 [Keogh and Folias 2002]. 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.

IV.5.1 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

Table IV.1: Typical results produced by different interesting measures on the synthetic datasets

Set	Fitness measure	Rule	Supp.	Conf.	*J*-mea.	*RI*
1	*J*-measure	$g \overset{184}{\longleftarrow} n \overset{1}{\Rightarrow} z$	0.019	0.51	0.072	0.51
1	Rule interest	$ywvh \mid wvhy \overset{1}{\Rightarrow} n$	10^{-5}	1.0	$4.7 \cdot 10{-}5$	1.0
1	Confidence	$fieg \mid egif \overset{1}{\Rightarrow} k$	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	$2.0 \cdot 10{-}4$

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 $o[\hat{}o\hat{}n]^*n$.
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 IV.1 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

Table IV.2: Summary of results on synthetic dataset using the modified J-measure

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

rule, and thus the rule support (data not shown).

IV.5.2 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:

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

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 IV.2 lists two of the expressions generated, along with the rule languages used in the generation process.

IV.5.3 Real-World Data

The system was tested on the ECG dataset from the UCR Time Series Data Mining Archive [Keogh and Folias 2002]. 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 IV.2 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 IV.3. Note that the results listed in this

Table IV.3: Early stopping results on ECG data set evaluated on the test set

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

Table IV.4: Results from the ECG set with $w = 10$ evaluated on the test set

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

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.4.1 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.4 lists two of the rules generated from the $L_5 \overset{10}{\Rightarrow} L_2$ language.

Figure IV.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.4 in the sequence.

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

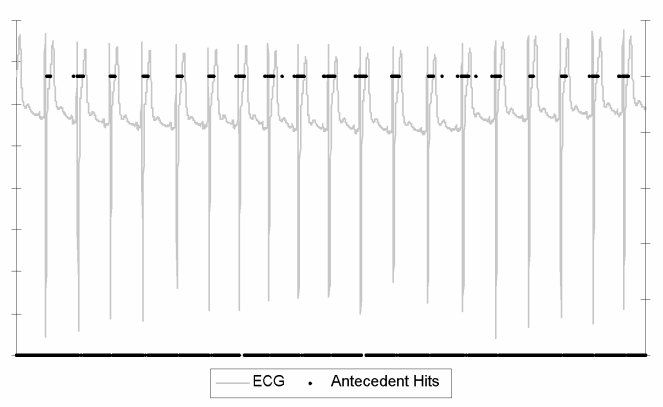

Figure IV.1: Hit locations of antecedent in ECG sequence

IV.5.4 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 IV.1 were again used, in addition to the modified J-measure. The results from these tests confirm the observations from the runs on the synthetic data (see Section IV.5.1), 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: $!(\mathrm{e}(\leq \mathrm{i})\mathrm{yk} \mid \mathrm{kye}(\leq \mathrm{i})) \overset{1}{\Rightarrow} !((\leq \mathrm{i})(> \mathrm{i}))$. 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.

IV.6 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.

Paper V

A Comparison of Hardware and Software in Sequence Rule Evolution

Abstract: *Sequence rule mining is an important problem in the field of data mining. Many algorithms have been devised that are based on counting candidate rules and excluding those with low support. Recently, the techniques of heuristic search, and evolutionary algorithms in particular, have been applied to various data mining problems, including sequence mining. In our previous work we have used specialized hardware to make mining certain rule formats feasible. In this paper we compare the performance of this hardware with realistic software alternatives.*

V.1 Introduction

Sequence rule mining is an important problem in the field of data mining. The basic problem is that of discovering rules that describe the relationship between parts of the data and that have certain desirable qualities, such as confidence and support [Hand et al. 2001], or specialized measures of *interestingness* [Hilderman and Hamilton 1999]. Many algorithms have been devised that are based on counting candidate rules and excluding those with low support. It is assumed that rules with low support (that is, those that occur infrequently) are less interesting.

Recently, the techniques of heuristic search, and evolutionary algorithms in particular, have been applied to various data mining problems [Freitas 2002], includ-

ing sequence mining [Hetland and Sætrom 2002; Sætrom and Hetland 2003c]. These techniques make possible very flexible rule formats and quality measures, but they rely on efficient information retrieval techniques, because one or more queries on the full database must be performed for each evaluation of the heuristic objective function, and this function is typically calculated a very large number of times.

In our previous work we have shown that specialized pattern matching hardware can make it feasible to use evolutionary data mining to mine rules of a format that is a superset of those of most existing approaches, with many added features [Hetland and Sætrom 2002]. In this paper we examine the role of this specialized hardware in our method, and compare its performance, both in terms of rule quality and execution speed, with realistic software alternatives. We focus on two software solutions: (1) similarity based time series retrieval with signature indexing [Hetland 2003] and (2) pattern based sequence retrieval with regular expressions [Manber and Wu 1994; Navarro 2001b].

V.2 Method

This section first describes the general principles of evolutionary sequence mining. Then follows a section about sequence retrieval, outlining the three main retrieval methods used in this paper.

V.2.1 Evolutionary Sequence Mining

There are many forms of heuristic search, including tabu search [Glover and Laguna 1998], simulated annealing [Laarhoven and Aarts 1987], and simple hill-climbing. While we use evolutionary algorithms, other heuristic search methods would quite likely give similar results.

Evolutionary sequence mining is simply an evolutionary search for interesting patterns in data. In practical terms:

1. The individual solutions are patterns, or, in our case rules, and
2. The fitness is related to the occurrences of the individual solutions in the data, found using some form of information retrieval.

Evolution Setup

In our approach, the individuals are rule antecedents, expressed in the query language IQL [Interagon AS 2002] or some subset. They are represented as syntax

(1) $R \rightarrow SS$
(2) $S \rightarrow TT$
(3) $T \rightarrow UU$
(4) $U \rightarrow C\&C$
(5) $U \rightarrow C$
(6) $C \rightarrow \geq A$
(7) $C \rightarrow \leq A$
(8) $A \rightarrow a$, for some $a \in \Sigma$

Figure V.1: PCA rule grammar

trees, and subject to the common crossover and mutation operations used in Genetic Programming (GP) [Koza 1992].[1] The internal nodes in the trees represent the operators in the given language. The leaf nodes are symbols from the alphabet used for discretizing the data (see Section V.3 for information about the discretization).

More specifically, in the regular expressions experiments the syntactical nodes used were *union*, *character sets*, and *concatenation*.

Due to an interaction effect between union and closure when matching with the PMC, certain matches may be reported with a small offset. While this is not critical in most cases, it can be crucial in a predictive setting such as ours. It is possible to circumvent this problem, but for the sake of implementational convenience, we have opted to forego the use of closure. As this is done in both forms of pattern queries (the regular expressions and the larger IQL subset) this omission should not give an unfair advantage to either rule format. Also, preliminary experiments show no gain in predictive accuracy by adding closure to the available operators.

The PCA vector experiments (see Section V.2.2) used *concatenation*, *conjunction*, and *ordinal comparisons* ($\leq$ and $\geq$). Also, in the PCA experiments, the syntactic structure was constrained as shown in Figure V.1. The R, S and T nodes are used for generating a balanced tree of the height needed for matching a vector of (in our case) length 8. In principle, a grammar can be created for matching vectors of any given length. Also note that the grammar in Figure V.1 corresponds to a subset of IQL, so the rules may be used with the PMC.

For all our experiments we used fifty generations. For the regular expression and full IQL experiments (see Section V.3 for details) we used a population size of 100, while for the PCA experiments we used a population size of 1000.

[1] The PCA rule format (see Section V.2.2) is also amenable to simple fixed-length array genome representation and Genetic Algorithm operators [Goldberg 1989].

Fitness

Several fitness measures are possible for rule mining. In [Sætrom and Hetland 2003c], we developed a fitness measure for unsupervised mining of interesting rules, and in [Sætrom and Hetland 2003b] we used multiobjective evolution to combine several fitness measures. In this paper we have chosen to use the correlation measure used in [Hetland and Sætrom 2002], because it makes objective quality comparisons based on predictive power easier.

The fitness measure is the *correlation* between the occurrences of a rule and the *desired* occurrences of the rule, as given by the event to be predicted. In our case, this event is the time series moving up in the next time step. 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:

$$\frac{\mathrm{TP}\cdot\mathrm{TN} - \mathrm{FP}\cdot\mathrm{FN}}{\sqrt{(\mathrm{TN}+\mathrm{FN})(\mathrm{TN}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})(\mathrm{TP}+\mathrm{FP})}}$$

V.2.2 Sequence Retrieval and Rule Formats

The field of information retrieval is broad, and the number of methods for retrieving sequences is vast. In this paper we concentrate on two approaches: (1) similarity based retrieval, using sample sequences as queries, and (2) pattern based retrieval, where the query is a pattern in some query language, for example, a regular expression, a Boolean expression, or some combination.

We look at three specific retrieval methods, signature indexing, pattern matching with software (including pattern indexing), and pattern matching hardware. These methods are described in the following three sections.

Signature Indexing

Signature indexing is a method for effectively performing similarity based sequence retrieval. The basic principles of the method are described in [Hetland 2003].

Simply put, similarity based retrieval is based around the notion of *distance functions* (or, more precisely, a *dissimilarity* functions). A query takes the form of a sample object (or possibly a sample sub-object, such as a subsequence). Given such a query q and a distance function $d(\cdot,\cdot)$, the retrieval system must find all objects in the data base D that fall within a certain distance d_{max}, that is,

$$R = \{r \in D \mid d(q,r) \leq d_{max}\} \ , \tag{V.1}$$

where R is the set of returned objects.

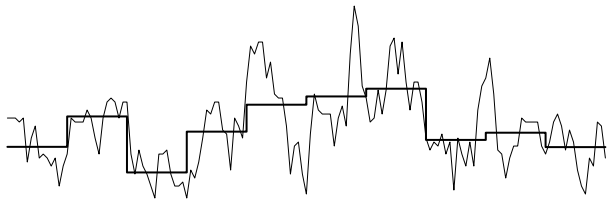

Figure V.2: A time series and its PCA signature

It is typically difficult to create index structures that fit this form of retrieval directly for a given object type, such as a time series. However, methods exist for indexing simple objects such as real vectors of fixed dimensionality (so-called spatial access methods).

To use such indexing methods, we must define a map s, which creates a *signature* vector for an object, and a corresponding distance measure d_s, such that

$$\forall x, y \quad d_s(\tilde{x}, \tilde{y}) \leq d(x, y) \ , \tag{V.2}$$

where $\tilde{x} = s(x)$ and $\tilde{y} = s(y)$. Then, intuitively, if we use $s(q)$ as our query and d_s as our distance function, things seem closer than they really are (unless $s(x) = x$ and $d_s = d$). This requirement guarantees that using d_s will not result in any false dismissals. (It will, however, most likely result in several incorrectly returned objects that may be filtered out at a later stage.) For it to be possible for an index to work with d_s, it must usually obey certain other restrictions. We will not go into that here.

This form of signature indexing for similarity retrieval has received much attention in recent years (see [Hetland 2003] for a survey). This makes this form of sequence database a natural candidate for data mining. In the following, we describe one specific form of signature, and how it may be used in evolutionary data mining. Similar techniques will be possible with other signature forms.

Figure V.2 demonstrates a signature form introduced independently by Yi et al. [Yi and Faloutsos 2000] and Keogh et al. [Keogh et al. 2001b], the Piecewise Constant Approximation, or PCA (also called the Piecewise Aggregate Approximation, PAA). The signature is calculated from a time series by dividing the sequence into k equal-sized non-overlapping segments, and calculating the average value in each segment. The resulting k dimensional vector is the signature. This signature type can be used with any L_p norm, and Keogh has shown that it can be used with more complex distance functions, such as Dynamic Time Warping [Keogh 2002]. It is robust, and simple to implement.

Assume that and index with PCA vectors is available, that the original time series was of length n and that the signatures have been extracted from all (overlapping)

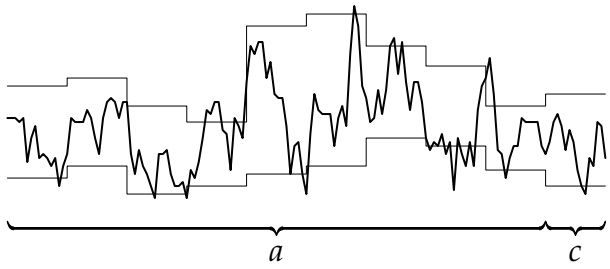

Figure V.3: Intuitive interpretation of a bounding hyperrectangle as a rule with antecedent a and consequent c

windows of length m. Also assume that we have associated a boolean value p with each signature, which indicates whether the given window is immediately followed by an upward movement in the time series. Given this database, our challenge is finding a rule format that may be used as a query. We could use a single signature (as shown in Figure V.2), a distance measure such as $d = L_2$ (Euclidean distance), and a radius d_{max}, which would be equivalent to a hypersphere in k dimensional space. However, an even simpler alternative is possible, which is also easier to visualize: an axis parallel hyperrectangle.

An axis parallel hyperrectangle simply gives an upper and lower limit to the permitted values in each dimension. All common spatial access methods are able to retrieve objects found within such hyperrectangles. (This is commonly known as a multidimensional range search.) As Figure V.3 shows, axis parallel hyperrectangles are (unlike hyperspheres) easy to visualize, which is important when the results are presented to a human expert. The figure also shows how such rules may be used in unsupervised rule mining (as in [Sætrom and Hetland 2003c]), by letting the first $k - 1$ segments function as antecedent (marked a), and the last segment as consequent (marked c). We will not be using the rules in this manner here.

In our experiments we use the Hybrid Tree [Chakrabarti and Mehrotra 1999] to index the PCA signatures.

Pattern Matching and Indexing

Pattern matching is a well-known problem in the field of information retrieval [Baeza-Yates and Ribeiro-Neto 1999]. A *pattern* is some form of specification, often in the form of a grammar or a predicate, that characterizes the requested objects. Well-known examples are *regular expressions* (the kind of grammars that characterize regular languages) and *Boolean* or *set expressions*, using set operations such

as union, intersection, and negation. Such set operations assume that some other form of query has been executed to generate result sets that may be combined. These subqueries may be plain substring searches (for example, for words in text) or other pattern queries, such as regular expressions.

For simple query types such as Boolean combinations of single word occurrences, effective and simple index forms exist (such as, for example, inverted files [Baeza-Yates and Ribeiro-Neto 1999]). There are index methods for more complex patterns such as regular expressions, but much less research has been done in this area.

One theoretically attractive indexing method is that described in [Baeza-Yates and Gonnet 1996]. The basic idea is to use the finite automaton derived from a regular expression to traverse a suffix tree over the data. For a certain class of regular expressions, this gives a logarithmic running time, and for regular expressions in general, it gives a sublinear running time. Whereas [Baeza-Yates and Gonnet 1996] theoretically analyzes the properties of the method in great detail, the method has not (to our knowledge) yet been implemented.

Another indexing scheme is that of [Cho and Rajagopalan 2002], which relies on k-gram indexing. The software system described in [Cho and Rajagopalan 2002] is not, however, publicly available.

In this paper we use two well-known tools for efficiently matching regular expressions:

- The index structure Glimpse [Manber and Wu 1994], which uses agrep for its regular expression matching. Agrep has long been regarded as the state of the art for regular expression searching, both exact and approximate.

- A more recent tool, NR-grep [Navarro 2001b], which in many cases is known to be faster than agrep.

Note that, unlike the suffix tree method in [Baeza-Yates and Gonnet 1996], neither of these actually index the regular expressions (the index structure of Glimpse is designed for roughly locating words in large amounts of data consisting of several files).

Pattern Matching Hardware

In this paper, as well as in our previous work, we use the Interagon Pattern Matching Chip (PMC) [Interagon AS 2000] to localize rule occurrences in the data. This gives us access to the full Interagon Query Language (IQL) [Interagon AS 2002] as rule format.

IQL is a rich query language supporting functionality such as regular expressions, positional offsets, and generalized Boolean expressions. The PMC is able to efficiently parallelize IQL queries, and can process data at about 100 MB/s.

V.3 Experiments

In this paper we have restricted ourselves to using a single data set, the ECG data used in [Hetland and Sætrom 2003] (taken from the UCR time series data mining archive [Keogh and Folias 2002]). This data set is highly regular and periodic, and should yield good results for any viable data mining method. For the pattern rules (regular expressions and IQL), we use the discretization process described in [Hetland and Sætrom 2003], using the best parameters found there (alphabet size 2 and window size 2). For the PCA experiments, the data were handled differently for the spatial index and the PMC.

For the spatial index, a sliding window was used to generate a set of vectors of fixed dimensionality (eight dimensions were used, for ease of representation in the PMC; see Section V.2.2). Each vector then had its contents shifted and scaled (by subtracting the minimum value and dividing by the value span) to have its values fall in the range $[0, 200]$. (The number 200 was chosen to parallel that used in the PMC experiments; see below.)

For the PMC, the values were transformed in a similar manner: Each number was transformed by subtracting the minimum of the last $k = 8$ numbers (the ones leading up to, and including, the number in question), and then dividing by the value span of these k numbers. Finally, these numbers were multiplied by 200 and rounded to get integers in the range $[0, 200]$. This was done to accommodate the need for byte-oriented data in the PMC.

The rule formats used in our experiments are described in Section V.2.2. We have compared the three for predictive accuracy. In addition, for the PCA rules and the simple pattern rules, we have compared the running time when using an index structure to that of using the PMC.

All the experimental results (times and accuracies) are the average over ten separate runs (and, for the accuracies, using separate test sets).

V.3.1 Mining Speed

In the following, the time used by the GP system is not considered, as it is not dependent on the retrieval mechanism used in the fitness computation. It may be interesting, though, to know that the time used by the GP system ranged from about 22 s to about 243 s.

Table V.1: Speed comparison

	PCA	Regex	IQL
Hybrid tree	27.16 s	–	–
Glimpse	–	509.65 s	–
NR-grep	–	542.44 s	–
PMC	0.96 s	0.28 s	2.38 s

Table V.2: Quality comparison

Rule form	Accuracy
PCA	69.0%
Regex	71.5%
IQL	72.2%

The software retrieval experiments (Glimpse, NR-grep and the Hybrid Tree) were performed on a 1 GHz Pentium III with 256 Mb of memory, running Gentoo Linux.

As can be seen in Table V.1, there are great differences in running times. For regular expression rules, the time taken by the PMC is less than 0.05% of that taken by the software (Glimpse and NR-grep). For the PCA data, the differences are less pronounced but still quite clear, with the PMC taking only 3.5% of the time used by the Hybrid Tree.

Please note that these figures should not be taken to indicate that Glimpse is in general faster than NR-grep, or indeed as a thorough empirical test of the speed of the given software solutions. We have modified the software to run multiple queries to avoid the overhead in starting the programs, but may still not have used the software in an optimal manner. Also, since our main goal was to illustrate the great difference in running time between the PMC and the software based solutions, we have not tested the behavior across many different kinds of data and queries. While such an approach might have given more favorable results for the software solutions, it is doubtful whether the differences between them and the PMC would be less marked.

V.3.2 Rule Quality

For comparing rule quality, we use predictive accuracy, that is, the probability that a rule will give a correct prediction at a random position in the test data. The results are shown in Table V.2.

Again, using other data sets would most likely produce different results (see [Hetland and Sætrom 2003] for the application of IQL to other data sets), but the given data still demonstrate that the predictive power of the different rule forms is comparable. Even though the differences are statistically significant ($p \ll 0.01$ with Fisher's exact test), they may well be problem dependent, and may not by themselves be large enough to justify using one method over the others. Other factors, such as rule comprehensibility or ease of implementation, may also influence the choice of rule format.

V.4 Discussion

In this paper we have compared the use of specialized hardware and existing solutions for information retrieval in evolutionary sequence mining. In addition to rule formats used in our previous work, we have also introduced the PCA rule format, which is suited for spatial indexing. We have shown that the software solutions are substantially slower than the PMC, but the running times are still acceptable, indicating that evolutionary sequence mining may well be performed with existing software. By using distributed evolution (that is, parallelization) the running times could be reduced even further. Even so, the superior running time of the PMC indicates that it could be used to mine huge data sets directly, whereas the software solutions most likely would have to be used on random data samples.

The other main advantage of the PMC is the expressiveness of its rule language. Even though this is clearly useful in that it allows the user to specify a domain-specific language, depending on the type of rules desired, it may not be crucial for predictive power, as indicated by the results in this paper. It should be noted, though, that the data chosen for these experiments is not particularly complex, and that for more complex data sets, the expressiveness of full IQL might impact the results.

Acknowledgements

We would like to thank Prof. Ricardo Baeza-Yates for his kind assistance. Also, we would like to thank Kaushik Chakrabarti, Gonzalo Navarro, and the Arizona Board of Regents for making the source code available for the Hybrid Tree, NR-grep, and Glimpse, respectively.

Paper VI

Multiobjective Evolution of Temporal Rules

Abstract: *In recent years, the methods of evolutionary computation have proven themselves useful in the area of data mining. For rule mining, several objective functions have been used, relating to both accuracy and interestingness in general. However, when searching for rules or patterns in a data set, several conflicting objectives will often be present. As the ultimate goal of data mining is to discover unexpected, useful knowledge, it may not be feasible to prioritize these objectives* a priori. *Simply constructing an aggregate fitness function in these cases could be seen as a more or less* ad hoc *solution. In this paper we propose an alternative: Using well-established multiobjective evolutionary algorithms to evolve a Pareto optimal set of rules.*

VI.1 Introduction

In recent years, the methods of evolutionary computation have proven themselves useful in the area of data mining. For the problem of rule mining, several objective functions have been used, relating to both accuracy and interestingness in general [Freitas 2002; Sætrom and Hetland 2003a]. However, when searching for rules or patterns in a data set, several conflicting objectives will often be present. Such objectives might include measures of accuracy and interestingness, as well as readability or parsimony. As the ultimate goal of data mining is to discover unexpected, useful knowledge, it may not be feasible to prioritize these objectives *a priori*. Simply constructing an aggregate fitness function in these cases could be seen as a more or less *ad hoc* solution. In this paper we propose an alternative: Using well-established multiobjective evolutionary algorithms. These

produce an approximation of the Pareto front in the multiobjective search space. An expert user may then inspect the resulting rule set to decide which rules are of potential use. We demonstrate the idea using the SPEA2 algorithm [Zitzler et al. 2001], combined with the temporal rule mining approach described in [Sætrom and Hetland 2003a].

VI.2 Method

In [Sætrom and Hetland 2003a] a method for doing unsupervised data mining in time series is presented. The method is based on genetic programming and generates rules from a prespecified rule language by optimizing a function that measures the (perceived) interestingness of a rule. This is an aggregate function, which combines several aspects of rule quality into a single measure.

In the following sections, a multi objective genetic programming (MOGP) algorithm based on the ideas of [Sætrom and Hetland 2003a] and [Zitzler et al. 2001] will be outlined. In Sect. VI.2.1 we give a brief description of our time series preprocessing. Following that, in Sect. VI.2.2 the rule format and internal rule representation used by the algorithm is described. Section VI.2.3 gives a brief outline of multiobjective optimization and the SPEA2 algorithm. In addition, some small modifications in the SPEA2 algorithm, used in the MOGP algorithm, are presented. Section VI.2.4 describes the objective functions used by the MOGP algorithm. Finally, Sect. VI.2.5 briefly outlines how the objective functions are evaluated.

VI.2.1 Discretization

The time series data are discretized by sequentially extracting a real-valued feature from a sliding window, in this case the slope of a line fitted to the points in the window through linear regression. Following this feature extraction, discretization limits are found for a set of symbols in an alphabet Σ (see Sect. VI.2.2) in a manner that ensures a uniform distribution of the symbols in the resulting data set. For more information about the discretization process, see [Sætrom and Hetland 2003a].

VI.2.2 Rule Representation

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 general, the rule format can be formalized by defining the respective languages L_a and L_c that

the antecedent and consequent can belong to. Several different languages have been used in the literature, ranging from single symbols from a fixed alphabet Σ [Das et al. 1998] to relatively complex pattern languages [Sætrom and Hetland 2003a].

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

The function of the maximum distance branch, T, is to set the maximum distance of the rule. Hence, the branch is constructed by using arithmetic functions (typically $+$ and $-$) as internal nodes, and random integer constants as leaf nodes. The final distance is found by computing the result of the arithmetic expression r_T, and using the residue of $r_T \bmod T + 1$.

VI.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 optimize 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 should be maximized, then a solution $\mathbf{x}$ is *Pareto optimal* if there 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 2 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 et al. 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 et al. 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.

VI.2.4 Objective Functions

Rules generated by an automatic data mining algorithm should often satisfy several requirements. For example, the rules should be accurate, interesting and comprehensible [Freitas 2002]. In the following, formalizations of these notions in the form of real-valued functions are presented. These formalisms are then used as objective measures in the multiobjective evolution.

Accuracy

Given a rule $R = R_a \stackrel{t}{\Rightarrow} R_c$ in the rule language $L_a \stackrel{T}{\Rightarrow} L_c$ (such that $t \leq T$ – see Sect. VI.2.2), and a discretized sequence $S = (a_1, a_2, \ldots, a_n)$, the frequency $F_S(R_a)$ of the antecedent is the number of occurrences of R_a in S. This can be formalized as

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

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, $f_S(R_a)$, is simply $F_S(R_a)/n$, where n is the length of S.

The *support* of a rule is defined as:

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

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:

$$c_S(R) = \frac{F_S(R_a, R_c, t)}{F_S(R_a)} \tag{VI.3}$$

The confidence measures the accuracy of the antecedent at predicting the consequent, while the support gives a measure of how well the rule represents the data. A rule having very low support, typically reflects freak incidents or noise in the data and thus is neither particularly accurate nor interesting.

Interestingness

The term interestingness is one commonly used in the field of data mining to denote the degree of surprise associated with the discovery of a rule. Several different interestingness measures have been developed (see [Hilderman and Hamilton 1999] for a survey). The J-measure ([Smyth and Goodman 1991]), is one particular measure which has already been proven useful in mining time series. This is defined as

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

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).

Comprehensibility

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 (VI.5) gives a definition of *rule complexity*, which is used in the following experiments.

$$complexity(R) = (nodeCount(R) + maxDepth(R))^{-1} \tag{VI.5}$$

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

VI.2.5 Rule Evaluation

As in [Sætrom and Hetland 2003a], a special purpose search chip [Interagon AS 2000; Fast Search & Transfer ASA] is used for finding the support and confidence of each rule. It is also used for estimating the probabilities needed for calculating the J-measure. Spurious correlations introduced by the discretization process are circumvented by setting a minimum distance d_w between the antecedent and consequent in all rules generated for a sequence discretized with window size w. The comprehensibility is computed by a simple traversal of each branch in the rule tree (see Sec. VI.2.2).

VI.3 Experiments

The MOGP algorithm was tested on four data sets from the UCR Time Series Data Mining Archive [Keogh and Folias 2002]: ECG measurements from several

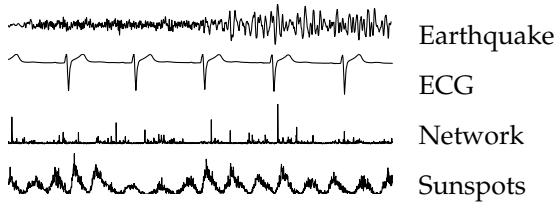

Figure VI.1: The time series analyzed

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 VI.1 shows plots of sub-sequences of the different time series analyzed.

We performed four sets of experiments. First, the four time series were mined using the four objective functions from section VI.2.4: support (VI.2), confidence (VI.3), J-measure (VI.4), and rule complexity (VI.5). Second, the time series were again mined, but now only the confidence, J-measure, and rule complexity were optimized. This was motivated by the fact that the J-measure implicitly rewards frequently occurring rules via the $p(R_a)$ factor (see (VI.4)). Thus, in theory, there should be no need to optimize the support explicitly. Third, the MOGP algorithm was modified so that the final rule set would contain rules having differing consequents and the time series were again mined. Fourth, we performed a set of experiments to determine how the window size in the discretization algorithm influenced the results of the mining algorithm.

The following sections outline the results of the four sets of experiments. All results were generated by running the MOGP algorithm with a population size of 1000 and archive size of 10 for a maximum of 100 generations. We used the IQL language [Interagon AS 2002] as a template for generating rules. The antecedents were IQL expressions, while the consequents were single characters or concatenations thereof.

VI.3.1 Using Support, Confidence, J-measure, and Rule Complexity

Table VI.1 lists the results from a run on the ECG data set discretized with a window size of 2. The hits of the first, second and fifth rules in a subsequence of the ECG series are plotted in Figure VI.2.

Table VI.1 and Figure VI.2 are representative of the solutions obtained on the ECG set. Two observations can be made from these results. First, the archive apparently has converged, as many of the rules are simply minor variations of

Table VI.1: Typical archive contents at algorithm termination for the ECG dataset

Rule	*J*-mea.	Conf.	Supp.	Compl.
$\mathrm{t} \overset{5}{\Rightarrow} \mathrm{s}$	0.057	0.67	0.033	0.20
$\mathrm{b} \overset{9}{\Rightarrow} \mathrm{c}$	0.050	0.75	0.038	0.20
$\mathrm{t} \overset{6}{\Rightarrow} \mathrm{s}$	0.053	0.67	0.033	0.20
$!(\geq \mathrm{rnokoussrfisehznh}) \overset{9}{\Rightarrow} \mathrm{g}$	0.020	0.48	0.41	0.020
$!((\geq \mathrm{o})(\geq \mathrm{nokoussrfisehznh})) \overset{9}{\Rightarrow} \mathrm{g}$	0.037	0.52	0.39	0.019
$\leq \mathrm{rnokoussrfisehznhdgv} \overset{9}{\Rightarrow} \mathrm{g}$	0.021	0.48	0.41	0.017
$\leq \mathrm{rnokoussrfisehznhv} \overset{9}{\Rightarrow} \mathrm{g}$	0.020	0.48	0.41	0.019
$\leq \mathrm{rrnouonequsehznhv} \overset{9}{\Rightarrow} \mathrm{g}$	0.017	0.47	0.41	0.020
$\leq \mathrm{rnoononequsehznhv} \overset{9}{\Rightarrow} \mathrm{g}$	0.020	0.48	0.41	0.020
$\leq \mathrm{rnouonequssrfisehznhv} \overset{9}{\Rightarrow} \mathrm{g}$	0.021	0.48	0.41	0.017

other rules. Second, the results illustrate an effect observed in runs on the other data sets. The archive typically contains two groups of rules: One group with rules having *confidence* $\gg$ *support*, and another group having *confidence* < 0.5 and *confidence* $\approx$ *support*. As the plot of the fifth rule from Table VI.1 in Figure VI.2 shows, the antecedents in the latter group typically match almost every position in the sequence. Because of this, these rules are of little or no value to a human user.[1]

To conclude this section, Figure VI.3 presents some of the rules mined from the different time series. As these plots show, the MOGP algorithm was able to generate rules that recognize and predict the significant features of the different time series. These include a rule for recognizing the increased oscillations occurring during an earthquake, rules that recognize the major peaks in the sunspot and ECG data, and a rule that is partly able to detect the periods of increased network activity.

VI.3.2 Using Confidence, *J*-measure, and Rule Complexity

Table VI.2 lists a subset of the results of the reanalysis of the ECG data discretized with window size 2. In addition, the archive contained 3 versions of the first rule having differing maximum distance, and 3 versions of the third rule having small

[1]Note, however, that the positions where the antecedent of the fifth rule in Table VI.1 does *not* match correspond to the peaks of the ECG sequence. In other words, the inverse of an antecedent may also reveal interesting aspects of a sequence.

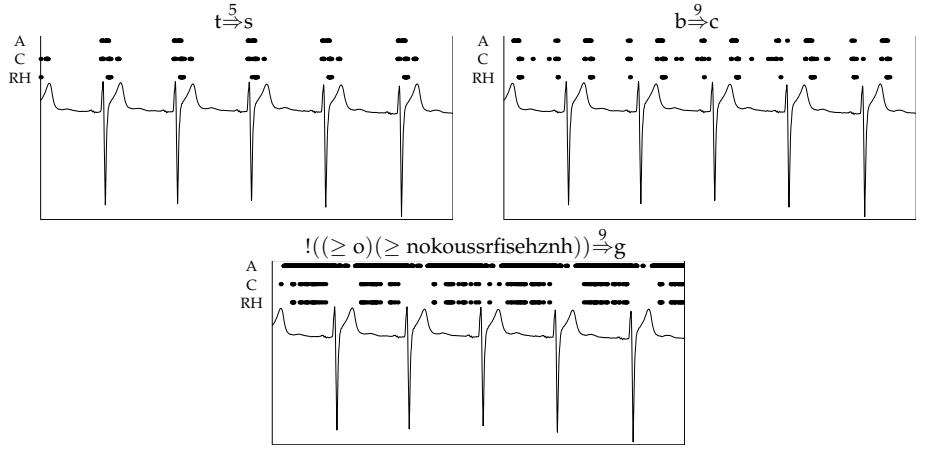

Figure VI.2: Hit locations in a subsequence of the ECG series of selected rules from Table VI.1. The dots following the *A* and *C* labels on the *y*-axis indicate the hit locations of the antecedent and consequent, respectively. The dots following the *RH* label indicate the positions where the rule hits, that is where the consequent follows the antecedent within the desired distance

variations in the antecedent. The hits of the rules from Table VI.2 in a subsequence of the ECG series are plotted in Figure VI.4.

As can be seen, removing the explicit support optimization did not adversely affect the support of the generated rules. Some of the generated rules did have very low support and *J*-measure values (typically corresponding to a single occurrence of the rule in the data set). Rules like these were, however, also generated when the support was optimized directly (data not shown).

VI.3.3 Promoting Differing Consequents

To further stimulate the discovery of rules exploring different properties of the time series, the domination relation in the MOGP was modified slightly: One rule could only dominate another rule if both rules shared the same consequent. In addition, all rules with no support were automatically dominated.

This approach did not however have the intended effect. Even though the resulting rules did have differing consequents, most of the rules produced were either highly specialized (having confidence ≈ 1 and low support and *J*-measure), or had very low confidence (data not shown). Thus it seemed that instead of the multiple almost identical rules produced earlier, the system now produced a few

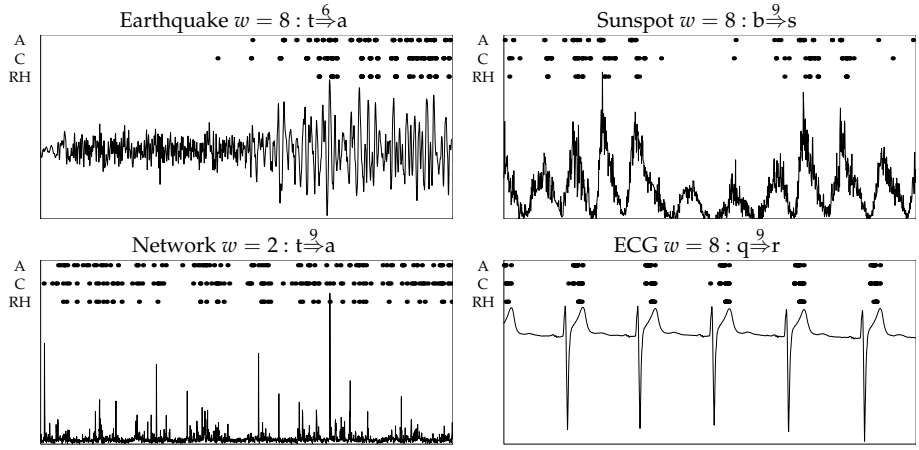

Figure VI.3: Hit locations of selected results on different sequences (see Figure VI.2 for the definitions of *A*, *C*, and *RH*)

Table VI.2: Selected archive contents at algorithm termination for the ECG dataset when not optimizing the support

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

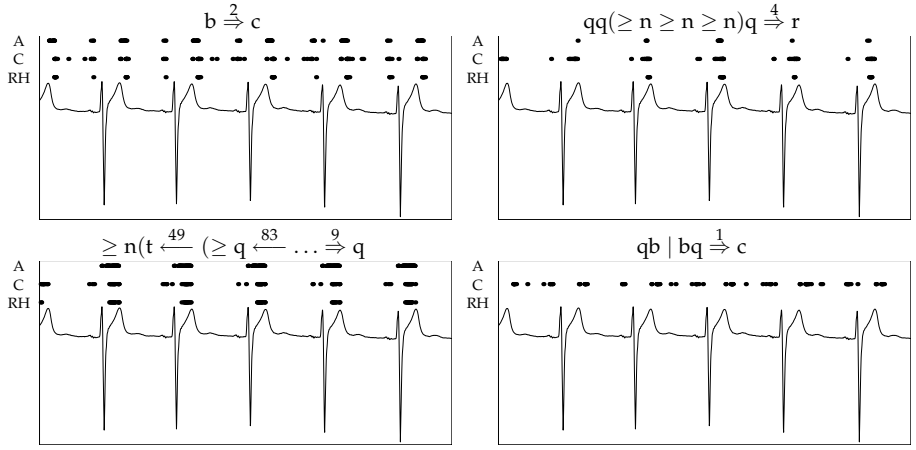

Figure VI.4: Hit locations in a subsequence of the ECG series of the rules from Table VI.2 (see Figure VI.2 for the definitions of *A*, *C*, and *RH*)

relevant rules and several unwanted and uninteresting rules. These results suggest that the MOGP algorithm can find rules involving the most interesting consequents, without using the modification described in this section. As a result, this approach was abandoned.

VI.3.4 Investigating the Window Size Effect

We performed several analyses of the four time series discretized with different window sizes. This revealed that by increasing the window size, the MOGP algorithm was better able to generate rules for recognizing the characteristic features in some of the series. This is illustrated in Figure VI.5, which shows a plot of different rules generated from the earthquake sequence discretized with window sizes 2, 4, 8, and 16. This figure shows that by increasing the window size, the generated rules zoom in on the part of the sequence representing an earthquake.

We observed the same effect in the sunspot set, but it was not as apparent in the ECG series (data not shown). This is most likely because the ECG sequence contains far less noise than the earthquake and sunspot sequences. To test this hypothesis, several versions of the ECG series with increasing noise levels were constructed. These sequences were constructed by adding Gaussian noise with mean zero and a the standard deviations set to 0.1%, 0.5%, 1%, 5%, 10% and 20% of the original value range.

The system was able to generate good rules recognizing the characteristic feature

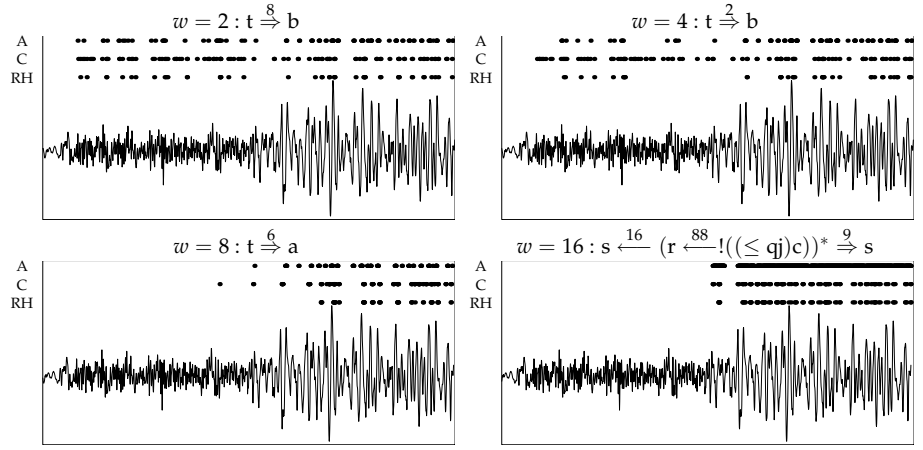

Figure VI.5: Rules generated from the earthquake series with increasing window sizes (see Figure VI.2 for the definitions of *A*, *C*, and *RH*)

of the ECG series for all window sizes for the two lowest noise levels. For 1% error level, the system only generated good rules for sequences discretized with window sizes of 4 or more. For the other series, the minimum window size required for generating good rules was 8, 16 and 32, respectively (in order of increasing noise levels).

When analyzing the network series with increasing window sizes, the same effect was not observed (data not shown). A possible explanation is that the significant feature in the network series is short bursts of increased network activity, represented by isolated series of spikes. Increasing the window size increases the minimum distance between the antecedent and consequent (see section VI.2.5). In addition, large window sizes represent more long term trends in the data (see section VI.2.1). Thus one should expect that rules produced from large window sizes will focus on more long term trends than rules produced from small window sizes. As shown above, when the window size is increased, the long term trends in the sunspot, earthquake, and noisy ECG data are more easily detected, while the short bursts in the network series are not. Thus the results support this proposition.

VI.4 Summary and Conclusion

An algorithm for unsupervised data mining in time series has been presented. The algorithm works by optimizing several, often conflicting, measures of rule

quality. As a result, it is able to generate a set of rules exploring different aspects of the time series analyzed. This has been demonstrated by analyzing several different sequences, and extracting rules detecting the significant features in each sequence. The robustness to noise has also been demonstrated.

The algorithm presented here is an extension of a method presented in [Sætrom and Hetland 2003a], where a single ad hoc rule goodness measure was used. This work follows the more natural approach when dealing with multiple, conflicting objectives, using multiobjective optimization to generate several possible solutions instead of a single result. This leaves the task of evaluating the trade-offs between the different objectives to the human user. In addition, optimizing a single measure may reduce the diversity of the resulting rules, and thereby omit certain potentially interesting results.

Appendices

Appendix A

Distance Measures

Faloutsos et al. [1997] describe a general framework for sequence distance measures (a similar framework can be found in Jagadish et al. [1995]). They show that many common distance measures can be expressed in the following form:

$$d(\boldsymbol{x}, \boldsymbol{y}) = \min \begin{cases} \min_{T_1, T_2 \in T} \{c(T_1) + c(T_2) + d(T_1(\boldsymbol{x}), T_2(\boldsymbol{y}))\} \\ d_0(\boldsymbol{x}, \boldsymbol{y}) \end{cases} \tag{A.1}$$

T is a set of allowable transformations, $c(T_i)$ is the cost of performing the transformation T_i, $T_i(\boldsymbol{x})$ is the sequence resulting from performing the transformation T_i on $\boldsymbol{x}$, and d_0 is a so-called *base distance*, typically calculated in linear time. For instance, L_p norms (such as Manhattan distance and Euclidean distance) result when $T = \emptyset$ and

$$d_0(\boldsymbol{x}, \boldsymbol{y}) = L_p = \sqrt[p]{\textstyle\sum_{i=1}^{l} |x_i - y_i|^p} \tag{A.2}$$

where $|\boldsymbol{x}| = |\boldsymbol{y}| = l$.

Editing distance (or Levenshtein distance) is the weight of the minimum sequence of editing operations needed to transform one sequence into another [Sankoff and Kruskal 1999]. It is usually defined on strings (or equi-spaced time sequences), but Mannila and Ronkainen [1997] show how to generalize this measure to general (non equi-spaced) time sequences. In the framework given above, editing distance may be defined as:

$$d_{ed} = \min \begin{cases} c(del(x_1)) + d_{ed}(x_{2:m}, \boldsymbol{y}) \\ c(del(y_1)) + d_{ed}(\boldsymbol{x}, y_{2:n}) \\ c(sub(x_1, y_1)) + d_{ed}(x_{2:m}, y_{2:n}) \end{cases} \tag{A.3}$$

where $m = |\boldsymbol{x}|$, $n = |\boldsymbol{y}|$, $del(x_1)$ and $del(y_1)$ represent deleting the first elements of $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively, and $sub(x_1, y_1))$ stands for substituting the first element of $\boldsymbol{x}$ with the first element of $\boldsymbol{y}$.

A distance function with time warping allows non-uniform scaling along the time axis, or, in sequence terms, *stuttering*. Stuttering occurs when an element from one of the sequences is repeated several times. A typical formulation is:

$$d_{tw}(\boldsymbol{x}, \boldsymbol{y}) = d_0(x_1, y_1) + \min \begin{cases} d_{tw}(\boldsymbol{x}, y_{2:n}) & (\boldsymbol{x}\text{-stutter}) \\ d_{tw}(x_{2:m}, \boldsymbol{y}) & (\boldsymbol{y}\text{-stutter}) \\ d_{tw}(x_{2:m}, y_{2:n}) & (\text{no stutter}) \end{cases} \tag{A.4}$$

Both d_{ed} and d_{tw} can be computed in quadratic time ($O(mn)$) using dynamic programming [Cormen et al. 2001; Sankoff and Kruskal 1999]: An $m \times n$ table D is filled iteratively so that $D[i, j] = d(x_{1:i}, y_{1:j})$. The final distance $d(\boldsymbol{x}, \boldsymbol{y})$ is found in $D[m, n]$.

Appendix B

Rule Language Syntax

This appendix describes the notation used in the rules presented in section IV.5.

$R*$: 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).

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

Bibliography

J.-M. Adamo. *Data Mining for Association Rules and Sequential Patterns*. Springer-Verlag, 2001.

R. Agrawal, C. Faloutsos, and A. N. Swami. Efficient similarity search in sequence databases. In *Proc. 4th Int. Conf. on Foundations of Data Organization and Algorithms, FODO*, pages 69–84, 1993.

R. Agrawal, K. Lin, H. S. Sawhney, and K. Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series database. In *Proc. 21st Int. Conf. on Very Large Databases, VLDB*, pages 490–501, 1995.

R. Agrawal and R. Srikant. Mining sequential patterns. In P. S. Yu and A. S. P. Chen, editors, *Eleventh International Conference on Data Engineering*, pages 3–14, Taipei, Taiwan, 1995. IEEE Computer Society Press.

W. A. Arentz, M. L. Hetland, and B. Olstad. Rhytm and pitch based music information retrieval system. Manuscript in preparation, 2003.

R. Baeza-Yates and G. H. Gonnet. A fast algorithm on average for all-against-all sequence matching. In *Proc. 6th String Processing and Information Retrieval Symposium, SPIRE*, pages 16–23, 1999.

R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press/Addison–Wesley Longman Limited, 1999.

R. A. Baeza-Yates and G. H. Gonnet. Fast text searching for regular expressions or automaton searching on tries. *Journal of the ACM*, 43(6):915–936, 1996.

W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming, An Introduction: On the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann, first edition, 1997.

N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R^*-tree: An efficient and robust access method for points and rectangles. In *Proc. 1990 ACM SIGMOD Int. Conf. on Management of Data*, pages 322–331, 1990.

C. B. Burge, T. Tuschl, and P. A. Sharp. Splicing of precursors to mRNAs by the spliceosomes. In *The RNA World (Second edition)*, pages 525–560. Cold Spring Harbor Laboratory Press, Cold Spring Harbor, New York, 1999.

K. Chakrabarti and S. Mehrotra. The hybrid tree: An index structure for high dimensional feature spaces. In *Proc. 15th Int. Conf. on Data Engineering, ICDE*, pages 440–447, 1999.

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

K. Chan and A. W. Fu. Efficient time series matching by wavelets. In *Proc. 15th Int. Conf. on Data Engineering, ICDE*, pages 126–133, 1999.

J. Cho and S. Rajagopalan. A fast regular expression indexing engine. In *Proc. 18th Int. Conf. on Data Engineering, ICDE*, 2002.

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

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

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and S. C. *Introduction to Algorithms*. The MIT Press, second edition, 2001.

C. Darwin. *On the Origin of Species*. 1859.

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

A. Del Bimbo. *Visual Information Retrieval*. Morgan Kaufmann, 1999.

T. G. Dietterich. Ensemble methods in machine learning. *Lecture Notes in Computer Science*, 1857:1–15, 2000.

P. Domingos and M. J. Pazzani. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, 29(2–3):103–130, 1997.

S. Droste, T. Jansen, and I. Wegener. Optimization with randomized search heuristics: The (A)NFL theorem, realistic scenarios, and difficult functions. *Theoretical Computer Science*, 2002.

G. Evangelidis, D. Lomet, and B. Salzberg. The hB$^{\Pi}$ tree: a multi-attribute index supporting concurrency. *The VLDB Journal*, 6:1–25, 1997.

C. Faloutsos, H. V. Jagadish, A. O. Mendelzon, and T. Milo. A signature technique for similarity-based queries. In *Proc. Compression and Complexity of Sequences, SEQUENCES*, 1997.

C. Faloutsos and K.-I. Lin. Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proc. 1995 ACM SIGMOD Int. Conf. on Management of Data*, pages 163–174, 1995.

C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proc. 1994 ACM SIGMOD Int. Conf. on Management of Data*, pages 419–429, 1994.

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.".

A. A. Freitas. A survey of evolutionary algorithms for data mining and knowledge discovery. To appear in: Ghosh, A.; Tsutsui, S. (Eds.) Advances in evolutionary computation. Springer-Verlag., 2001.

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

V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

C. L. Giles, S. Lawrence, and A. C. Tsoi. Noisy time series prediction using a recurrent neural network and grammatical inference. *Machine Learning*, 44(1/2): 161–183, July/August 2001.

F. W. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1998.

D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison–Wesley, 1989.

D. Gusfield. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, 1997.

A. Guttman. *r*-trees: A dynamic index structure for spatial searching. In *Proc. 1984 ACM SIGMOD Int. Conf. on Management of Data*, pages 47–57, 1984.

D. J. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining*. The MIT Press, 2001.

M. L. Hetland. *Practical Python*. Apress, 2002. 648 pages.

M. L. Hetland. A survey of recent methods for efficient retrieval of similar time sequences. In M. Last, A. Kandel, and H. Bunke, editors, *Data Mining in Time Series Databases*. World Scientific, Singapore, 2003. To appear.

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, RASC*, 2002.

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

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

J. Hipp, U. Güntzer, and G. Nakhaeizadeh. Algorithms for association rule mining – a general survey and comparison. *SIGKDD Explorations*, 2(1):58–64, July 2000.

F. Höppner and F. Klawonn. Finding informative rules in interval sequences. In *Lecture Notes in Computer Science*, volume 2189, pages 125–134, 2001.

M. Hutter. Fitness uniform selection to preserve genetic diversity. (IDSIA-01-01): 13 pages, Jan. 2001.

Interagon AS. Digital processing device. PCT/NO99/00308, Apr 2000.

Interagon AS. The Interagon query language: A reference guide. `http://www.interagon.com/pub/whitepapers/IQL.reference-latest.pdf`, sep 2002.

H. V. Jagadish, A. O. Mendelzon, and T. Milo. Similarity-based queries. In *Proc. 14th Symposium on Principles of Database Systems, PODS*, pages 36–45, 1995.

E. Keogh and T. Folias. The UCR time series data mining archive. `http://www.cs.ucr.edu/~eamonn/TSDMA`, sep 2002.

E. J. Keogh. A fast and robust method for pattern matching in time series databases. In *Proc. 9th Int. Conf. on Tools with Artificial Intelligence, ICTAI*, pages 578–584, 1997.

E. J. Keogh. Exact indexing of dynamic time warping. In *Proc. 28th Int. Conf. on Very Large Data Bases, VLDB*, pages 406–417, 2002.

E. J. Keogh, K. Chakrabarti, S. Mehrotra, and M. J. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. In *Proc. 2001 ACM SIGMOD Conf. on Management of Data*, pages 151–162, 2001a.

E. J. Keogh, K. Chakrabarti, M. J. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Journal of Knowledge and Information Systems*, 3(3):263–286, 2001b.

E. J. Keogh, S. Lonardi, and B. Chiu. 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*, pages 550–556, 2002.

E. J. Keogh and M. J. Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining, KDD*, pages 239–243, 1998.

E. J. Keogh and M. J. Pazzani. An indexing scheme for fast similarity search in large time series databases. In *Proc. 11th Int. Conf. on Scientific and Statistical Database Management, SSDBM*, pages 56–67, 1999a.

E. J. Keogh and M. J. Pazzani. Scaling up dynamic time warping to massive datasets. In *Proc. 3rd European Conf. on Principles of Data Mining and Knowledge Discovery, PKDD*, pages 1–11, 1999b.

E. J. Keogh and M. J. Pazzani. A simple dimensionality reduction technique for fast similarity search in large time series databases. In *Proc. 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining, PAKDD*, pages 122–133, 2000.

E. J. Keogh and P. Smyth. A probabilistic approach to fast pattern matching in time series databases. In *Proc. 3rd Int. Conf. on Knowledge Discovery and Data Mining, KDD*, pages 24–30, 1997.

S. Kim, S. Park, and W. W. Chu. An index-based approach for similarity search supporting time warping in large sequence databases. In *Proc. 17th Int. Conf. on Data Engineering, ICDE*, pages 607–614, 2001.

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

P. J. M. V. Laarhoven and E. H. L. Aarts. *Simulated Annealing: Theory and Applications*. D Reidel Publishing Company, 1987.

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

S. Lee, S. Chun, D. Kim, J. Lee, and C. Chung. Similarity search for multidimensional data sequences. In *Proc. 16th Int. Conf. on Data Engineering, ICDE*, pages 599–609, 2000.

U. Manber and S. Wu. GLIMPSE: A tool to search through entire file systems. In *Proc. USENIX Winter 1994 Technical Conference*, pages 23–32, San Fransisco, CA, USA, 17–21 1994.

H. Mannila and P. Ronkainen. Similarity of event sequences. In *Proc. 4th Int. Workshop on Temporal Representation and Reasoning, TIME*, pages 136–139, 1997.

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

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

T. M. Mitchell. *Machine Learning*. McGraw–Hill, 1997.

G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001a.

G. Navarro. NR-grep: a fast and flexible pattern-matching tool. *Software Practice and Experience*, 31(13):1265–1312, 2001b.

J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer-Verlag, 1999.

S. Park, W. W. Chu, J. Yoon, and C. Hsu. Efficient search for similar subsequences of different lengths in sequence databases. In *Proc. 16th Int. Conf. on Data Engineering, ICDE*, pages 23–32, 2000.

C. Perng, H. Wang, S. R. Zhang, and D. S. Parker. Landmarks: a new model for similarity-based pattern querying in time series databases. In *Proc. 16th Int. Conf. on Data Engineering, ICDE*, pages 33–42, 2000.

G. Piatetsky-Shapiro. Discovery, analysis and presentation of strong rules. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 229–248. The MIT Press, Cambridge, MA, 1991.

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

D. Rafiei and A. Mendelzon. On similarity-based queries for time series data. *SIGMOD Record*, 2(26):13–25, 1997.

F. R. S. Release. Foreign exchange rates 1971–2002. `http://www.federalreserve.gov/Releases/H10/Hist`, oct 2002.

S. Rüping. SVM kernels for time series analysis. In R. Klinkenberg, S. Rüping, A. Fick, N. Henze, C. Herzog, R. Molitor, and O. Schröder, editors, *LLWA 01: Tagungsband der GI-Workshop-Woche Lernen – Lehren – Wissen – Adaptivität*. University of Dortmund, 2001.

D. Sankoff and J. Kruskal, editors. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. CSLI Publications, reissue edition, 1999.

T. K. Sellis, N. Roussopoulos, and C. Faloutsos. The R^+-tree: A dynamic index for multi-dimensional objects. In *Proc. 13th Int. Conf. on Very Large Database, VLDB*, pages 507–518, 1987.

H. Shatkay. The Fourier transform: a primer. Technical Report CS-95-37, Brown University, 1995.

P. Smyth and R. M. Goodman. Rule induction using information theory. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 159–176. The MIT Press, Cambridge, MA, 1991.

M. Spiliopoulou. Managing interesting rules in sequence mining. In *Proc. 3rd European Conf. on Principles of Data Mining and Knowledge Discovery, PKDD*, pages 554–560, 1999.

P. Sætrom and M. L. Hetland. Mining interesting temporal rules with genetic programming and specialized hardware. In *Proc. 2003 Int. Conf. on Machine Learning and Applications, ICMLA*, 2003a.

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

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

R. Sun. Introduction to sequence learning. In Sun and Giles [2000], pages 1–10.

R. Sun and C. L. Giles, editors. *Sequence Learning: Paradigms, Algorithms, and Applications*. Number 1828 in Lecture Notes in Artificial Intelligence. Springer-Verlag, 2000.

A. Tveit, H. Engum, and M. L. Hetland. Incremental and decremental proximal support vector classification using decay coefficients. In *Proc. 5th Int. Conf. on Data Warehousing and Knowledge Discovery, DaWaK*, 2003.

A. Tveit and M. L. Hetland. Multicategory incremental proximal support vector classifiers. In *Proc. 7th Int. Conf. on Knowledge-Based Intelligent Information & Engineering Systems, KES*, 2003.

H. Wang and C. Perng. The S^2-tree. an index structure for subsequence matching of spatial objects. In *Proc. 5th Pacific-Asia Conf. on Knowledge Discovery and Data Mining, PAKDD*, pages 312–323, 2001.

J. T.-L. Wang, X. Wang, K.-I. Lin, D. Shasha, B. A. Shapiro, and K. Zhang. Evaluating a class of distance-mapping algorithms for data mining and clustering. In *Proc. 1999 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 307–311, 1999.

R. Weber, H. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. 24th Int. Conf. on Very Large Databases, VLDB*, pages 194–205, 1998.

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

D. H. Wolpert and W. G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe, NM, 1995.

Y. Wu, D. Agrawal, and A. E. Abbadi. A comparison of DFT and DWT based similarity search in time-series databases. In *Proc. 9th Int. Conf. on Information and Knowledge Management, CIKM*, pages 488–495, 2000.

B. Yi and C. Faloutsos. Fast time sequence indexing for arbitrary norms. *The VLDB Journal*, pages 385–594, 2000.

B. Yi, H. V. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *Proc. 14th Int. Conf. on Data Engineering, ICDE*, pages 201–208, 1998.

S. Zemke. Nonlinear index prediction. In R. N. Mantegna, editor, *Proc. Int. Workshop on Econophysics and Statistical Finance*, volume 269, pages 177–183, Palermo, Italy, sep 1998. Elsevier Science.

E. Zitzler, M. Laumanns, and L. Thiele. 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, May 2001.