Pål Sætrom

Hardware accelerated genetic programming for pattern mining in strings

Dr.philos. thesis 2005:197

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

NTNU

Hardware accelerated genetic programming for pattern mining in strings
Pål Sætrom

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

Dr.philos. thesis 2005:197

ISBN 82-471-7289-5 (printed version)
ISBN 82-471-7288-7 (electronic version)
ISSN 1503-8181

To Gerd Inger,
Bjørn Magnus,
and Lars Gunnar

Abstract

This thesis considers the problem of mining patterns in strings. Informally, this is the problem of extracting information (patterns) that characterizes parts of, or even the complete, string. The thesis describes a high performance hardware for string searching, which together with genetic programming, forms the basis for the thesis' pattern mining algorithms.

This work considers two different pattern mining problems and develops several different algorithms to solve different variants of these problems. Common to all algorithms is that they use genetic programming to evolve patterns that can be evaluated by the special purpose search hardware.

The first pattern mining problem considered is unsupervised mining of prediction rules in discretized time series. Such prediction rules describe relations between consecutive patterns in the discretized time series; that is, the prediction rules state that if the first pattern occurs, the second pattern will, with high probability, follow within a fixed number of symbols. The goal is to automatically extract prediction rules that are accurate, comprehensible, and interesting.

The second pattern mining problem considered is supervised learning of classifiers that predict whether or not a given string belongs to a specific class of strings. This binary classification problem is very general, but this thesis focuses on two recent problems from molecular biology: i) predicting the efficacy of short interfering RNAs and antisense oligonucleotides; and ii) predicting whether or not a given DNA sequence is a non-coding RNA gene. The thesis describes a genetic programming-based mining algorithm that produce state-of-the-art classifiers on both problems.

Preface

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

The main contributions of this dissertation are the seven research papers that have been, or will be published in peer-reviewed scientific conference proceedings and journals. These papers are collected in the dissertation's second part, and throughout this work, I will refer to the papers as Paper I through Paper VII (see the List of Papers).

The first part of the dissertation introduces the topics covered in the papers and provides a context for the reader. The introduction focuses mostly on the papers' pattern mining aspects, but also gives a brief introduction to the application areas covered by the papers. These are i) unsupervised mining of prediction rules in time series; ii) predicting non-coding RNA genes in DNA sequences; and iii) predicting the efficacy of short interfering RNAs and antisense oligonucleotides. The papers give additional references.

Background

This dissertation is the result of several decades of research on special purpose search hardware. The search hardware described here—the pattern matching chip (PMC)—is the third generation design and was conceived by Arne Halaas, Børge Svingen, and Olaf René Birkeland in the mid nineties. I was introduced to the architecture while working on my master degree. Then, through Børge Svingen's simple idea of using genetic programming to evolve search patterns for the PMC, I was introduced to genetic programming. After completing my master thesis on genetic programming and the PMC in February 2001, I joined the PMC-team at Fast Search & Transfer. On 18 January the following year, we established Interagon AS as a bioinformatics company that was to do complex pattern analysis in biological data. Since then, we have used genetic programming and the PMC in several commercial and academic research projects.

It is hard to put a date on when I started working on this thesis, as it has very much been a continual process since I finished my master thesis. 17 June 2002 is, however, an important date in this respect, as this was when Magnus Lie Hetland and I started our collaboration. This collaboration has so far resulted in six research papers—five of which appeared in Magnus' PhD-thesis and three of which appears in mine.

On joint authorships

Most of this thesis builds on work done in collaboration with others, but as the thesis' title suggests, my main contributions in this work have been systems for hardware accelerated pattern mining in strings. More specifically, in Magnus Lie Hetland's and my work, the solutions for supervised rule mining were his main contributions; the solutions for unsupervised rule mining were mine. Consequently, of the six papers from our collaboration, the three papers included in this thesis focus mostly on unsupervised rule mining. In Interagon, Olaf René Birkeland and Ola Snøve have focused mostly on the search processor and biological applications, respectively; I have focused on machine learning methods for string classification.

Acknowledgements

Most of this work was financed by Interagon AS, as I have been a full-time employee while working on this dissertation. Parts of the work was also supported by the Norwegian Research Council, grant 151899/150, and the bioinformatics platform at NTNU.

I would like to thank all the people that have made this work possible through your contributions, encouragement, help, and support. I would like to thank my co-authors on the different papers; their contributions have been invaluable. I would like to thank Finn Drabløs, Arne Halaas, Magnus Lie Hetland, and Ola Snøve for reading and commenting on this manuscript. I would like to thank Arne Halaas for introducing me to the PMC, for steering me into the FAST milieu, and for his constant encouragements throughout this work. I would like to thank Magnus Lie Hetland for giving me the opportunity to contribute in a fruitful collaboration. I would like to thank Hans E. Krokan for all his support. I would like to thank my colleagues at Interagon, Olaf René Birkeland, Ståle H. Fjeldstad, Thomas

Grünfeld, Håkon Humberset, Magnar Nedland, and Ola Snøve; this has in many ways been a team effort. Finally, I would like to thank my family for being there and for keeping up with me working the many odd hours—without you it would not have been worth it.

Contents

1 Introduction **1**
1.1 Thesis structure 2
1.2 Paper abstracts 3
1.3 Other publications 6

2 Strings and string mining **7**
2.1 Strings defined 7
2.2 String searching 8
2.2.1 String searching defined 8
2.2.2 String searching in software 8
2.2.3 String searching in hardware 9
2.3 String mining 11
2.3.1 Mining rules in strings 11
2.3.2 Classifying strings 12
2.3.3 Overfitting and accuracy 13
2.3.4 Estimating model accuracy 15
2.4 Problem domains 16
2.4.1 Rule mining in time series 16
2.4.2 Classification of DNA sequences 17

3 Evolving string patterns **21**
3.1 Evolutionary algorithms 21
3.2 Genetic programming 23
3.2.1 Solution representation 23
3.2.2 Variation operations 25
3.3 Fitness 27
3.3.1 Fitness in rule mining 27
3.3.2 Fitness in classification 28
3.4 Selection 30
3.4.1 Niching 31
3.4.2 Fitness sharing 31

3.4.3 Species selection 32

4 Ensemble methods to increase classifier performance 35
4.1 Voting 37
4.2 Boosting 38

5 Contributions and further work 43
5.1 Contributions 43
5.2 Further work 44

Bibliography 47

Papers 57

List of figures

1.1 How the papers relate to one another 2

2.1 Basic PMC architecture 9
2.2 Basic PMC configuration 10
2.3 Training without restrictions 14
2.4 Noise in the training set 14
2.5 The double stranded DNA 18
2.6 Antisense oligonucleotides 19

3.1 Evolutionary algorithm 22
3.2 A simple syntax tree 24
3.3 A simple language grammar 24
3.4 Standard subtree swapping crossover disrupts grammar . . 26
3.5 Constrained crossover conserves grammar 26
3.6 Confusion matrix 28

4.1 Averaging reduces variance 38
4.2 General boosting algorithm 39
4.3 Maximizing the margin improves generalization 41

List of papers

Paper I Arne Halaas, Børge Svingen, Magnar Nedland, Pål Sætrom, Ola Snøve Jr., and Olaf René Birkeland. A recursive MISD architecture for pattern matching. *IEEE Trans. on VLSI Syst.*, 12(7):727–734, 2004.

Paper II Pål Sætrom and Magnus Lie Hetland. Unsupervised temporal rule mining with genetic programming and specialized hardware. In *Proceedings of the International Conference on Machine Learning and Applications (ICMLA'03)*, pages 145–151, 2003.

Paper III Pål Sætrom and Magnus Lie Hetland. Multiobjective evolution of temporal rules. In *Eight Scandinavian Conference on Artificial Intelligence*, 2003.

Paper IV Magnus Lie Hetland and Pål Sætrom. Evolutionary rule mining in time series databases. *Mach. Learn.*, 58(2–3):107–125, 2005.

Paper V Pål Sætrom. Predicting the efficacy of short oligonucleotides in antisense and RNAi experiments with boosted genetic programming. *Bioinformatics*, 20(17):3055–3063, 2004.

Paper VI Pål Sætrom and Ola Snøve Jr. A comparison of siRNA efficacy predictors. *Biochem. Biophys. Res. Commun.*, 321(1):247–253, 2004.

Paper VII Pål Sætrom, Ragnhild Sneve, Knut I. Kristiansen, Ola Snøve Jr., Thomas Grünfeld, Torbjørn Rognes, and Erling Seeberg. Predicting non-coding RNA genes in *Escherichia coli* with boosted genetic programming. *Nucleic Acids Res.*, 33(10):3263–3270, 2005.

Chapter 1

Introduction

INFORMATION retrieval in strings is a problem of increasing importance, as the number and size of electronic string collections is ever increasing. This thesis considers two specific problems in two different application domains: i) automatic extraction of prediction patterns in discretized time series, and ii) automatic classification of DNA sequences. Both problems fall into the general bag labeled "pattern mining in strings".

There are numerous methods for mining patterns in strings, including patterns in time series and patterns in protein and DNA sequences, but most of these methods use deterministic search and pruning techniques to find all candidate patterns; see for example Das et al. (1998); Höppner and Klawonn (2001); Mannila et al. (1997); Yang et al. (2003) (time series), Jonassen et al. (1995) (proteins), and Vanet et al. (1999) (DNA). This simple technique is optimal for simple rule formats, but more complex patterns make the search intractable. To mine more complex patterns, one must resort to heuristic search methods, such as evolutionary algorithms (Freitas 2002).

Genetic programming is an evolutionary algorithm that intuitively is attractive in pattern mining because it evolves symbolic expressions (Koza 1992). Several groups have used genetic programming to mine patterns in strings; see for example Hetland (2003) (time series), Heddad et al. (2004); Koza et al. (1999) (proteins), and Howard and Benson (2003) (DNA). Like other evolutionary algorithms, genetic programming uses evolution in a population of candidate solutions to explore the search space. A part of the evolution process is to test the quality of each candidate solution in the current population, and in string pattern mining, this test typically means scanning each candidate pattern against the string database. Consequently, genetic programming can become impractical when the database is too large or when the patterns are too complex.

$$\begin{tikzcd}
 & \text{Paper II} \arrow[r] & \text{Paper III} \arrow[r] & \text{Paper IV} \\
\text{Paper I} \arrow[ur] \arrow[r] & \text{Paper V} \arrow[r] \arrow[dr] & \text{Paper VI} & \\
 & & \text{Paper VII} &
\end{tikzcd}$$

Figure 1.1: How the papers relate to one another.

This work is based on a special purpose hardware—the pattern matching chip (PMC)—which can search for complex patterns in large string databases. The PMC's functionality, performance, and ease of use have been key factors behind this work, as these factors made it possibile to explore solutions that otherwise might have been impractical because of computational limitations. Hetland (2003); Hetland and Sætrom (2002, 2003a,b, 2004) used genetic programming and the PMC to mine patterns in discretized time series. This study further explores the use of genetic programming for pattern mining in strings.

1.1 Thesis structure

This thesis has two main parts. The first part (chapters 1 to 5) gives an introduction to the thesis' main contributions: the papers in the second part. Chapter 2 introduces the problem of mining patterns in strings; Chapter 3 discusses how genetic programming can be used to mine such string patterns; Chapter 4 shows how combining several patterns can improve the performance of the patterns mined by genetic programming; and Chapter 5 summarizes the main contributions and outlines some possible further work.

Each paper in the second part is self-contained. Nevertheless, the papers are related, as some papers build on the work of other papers. You should therefore have Figure 1.1 in mind when reading the papers. Paper I presents the specialized search hardware that forms the basis for the following papers on string mining; papers II, III, and IV develop algorithms for unsupervised mining of prediction rules in strings; Paper V considers the problem of predicting how effective short oligonucleotides will be when they are used in antisense and RNA interference experiments (ODNs and siRNAs), and present a boosted genetic programming algorithm that shows promising results; Paper VI compares the performance of the classifiers created by the boosted genetic programming algorithm with the performance of other recently published siRNA efficacy prediction algorithms; and Paper VII uses the boosted genetic programming algorithm to predict non-coding RNA genes in the bacteria *Escherichia coli*. The following section lists the paper abstracts.

1.2 Paper abstracts

Paper I *A recursive MISD architecture for pattern matching.* Many applications require searching for multiple patterns in large data streams for which there is no preprocessed index to rely on for efficient lookups. An MISD VLSI architecture that is based on a recursive divide and conquer approach to pattern matching is proposed. This architecture allows searching for multiple patterns simultaneously. The patterns can be constructed much like regular expressions, and add features such as requiring subpatterns to match in a specific order with some fuzzy distance between them, and the ability to allow errors according to prescribed thresholds, or ranges of such. The current implementation permits up to 127 simultaneous patterns at a clock frequency of 100 MHz, and does 1.024×10^{11} character comparisons per second.

Paper II *Unsupervised temporal rule mining with genetic programming and specialized hardware.* 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 III *Multiobjective evolution of temporal rules.* 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*. In this paper we propose an alternative to constructing an *ad hoc* aggregate fitness function: using well-established multiobjective evolutionary algorithms to evolve a Pareto optimal set of rules. We apply the method to several real-world data sets and demonstrate how the method is able to evolve a varied set of rules that explore different aspects of the time series in question.

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

Paper V *Predicting the efficacy of short oligonucleotides in antisense and RNAi experiments with boosted genetic programming.* Both small interfering RNAs (siRNAs) and antisense oligonucleotides can selectively block gene expression. Although the two methods rely on different cellular mechanisms, the methods share the common property that not all oligonucleotides (oligos) are equally effective. That is, if mRNA target sites are picked at random, many of the antisense or siRNA oligos will not be effective. Algorithms that can reliably predict the efficacy of candidate oligos can greatly reduce the cost of knockdown experiments, but previous attempts to predict the efficacy of antisense oligos have had limited success. Machine learning has not previously been used to predict siRNA efficacy.

We develop a genetic programming based prediction system that shows promising results on both antisense and siRNA efficacy prediction. We train and evaluate our system on a previously published database of antisense efficacies and our own database of siRNA efficacies collected from the literature. The best models gave an overall correlation between predicted and observed efficacy of 0.46 on both antisense and siRNA data. As a comparison, the best correlations of support vector machine classifiers trained on the same data were 0.40 and 0.30, respectively.

The prediction system uses proprietary hardware and is available for both commercial and strategic academic collaborations. The siRNA database is available upon request.

Paper VI *A comparison of siRNA efficacy predictors.* Short interfering RNA (siRNA) efficacy prediction algorithms aim to increase the probability of selecting target sites that are applicable for gene silencing by RNA interference. Many algorithms have been published recently, and they base their predictions on different features such as duplex stability, sequence characteristics, mRNA secondary structure, and target site uniqueness.

We compare the performance of the algorithms on a collection of publicly available siRNAs. First, we show that our regularized genetic programming algorithm, the GPboost, appear to perform better and more stable than do other algorithms on the collected datasets. Second, several algorithms gave close to random classification on unseen data, and only GPboost and three other algorithms have a reasonably high and stable performance on all parts of the dataset. Third, the results indicate that the siRNAs's sequence is sufficient input to siRNA efficacy algorithms, and that other features that have been suggested to be important may be indirectly captured by the sequence.

Paper VII *Predicting non-coding RNA genes in* Escherichia coli *with boosted genetic programming.* Several methods exist for predicting non-coding RNA (ncRNA) genes in *Escherichia coli* (*E. coli*). In addition to about sixty known ncRNA genes excluding tRNAs and rRNAs, various methods have predicted more than thousand ncRNA genes, but only 95 of these candidates were confirmed by more than one study. Here we introduce a new method that uses automatic discovery of sequence patterns to predict ncRNA genes. The method predicts 135 novel candidates. In addition, the method predicts 152 genes that overlap with predictions in the literature. We test sixteen predictions experimentally, and show that twelve of these are actual ncRNA transcripts. Six of the twelve verified candidates were novel predictions. The relatively high confirmation rate indicates that many of the untested novel predictions are also ncRNAs, and we therefore speculate that *E. coli* contains more ncRNA genes than previously estimated.

1.3 Other publications

Other publications co-authored while working on this thesis include the papers listed in the bibliography as Hetland and Sætrom (2002, 2003a,b, 2004); and Snøve Jr. et al. (2004).

Chapter 2

Strings and string mining

WHETHER we consider written text, DNA, or daily finite precision temperature measurements, a string is an ordered list of symbols from a finite alphabet. Data collections of strings are ever increasing—consider for example the ever growing World Wide Web (Lawrence and Giles 1998; Risvik 2004) or the exponentially increasing number of available DNA sequences (NCBI News 2004). Consequently, effective methods to query and retrieve information from strings are increasingly valuable.

This chapter considers the problems of searching strings and mining strings. After formally defining what a string is, the chapter discusses string searching in general and a high performance hardware solution in particular. The chapter concludes by presenting two string mining problems—unsupervised mining of prediction rules in strings and classifying strings—and outlining some potential application areas.

2.1 Strings defined

Def. 1 *A string s is an ordered set of characters from a finite alphabet Σ, and $|s|$ is the length of the string. $s[i]$ is the character at position i.*

For example, in the DNA alphabet where $\Sigma = \{a, c, g, t\}$, $s =$ *"acgt"* is a string of length 4 and $s[2] = c$.

Def. 2 *$s[i \dots j]$ is the substring from position i to position j in s. In particular, $s[1 \dots i]$ is the i-th prefix of s and $s[i \dots |s|]$ is the i-th suffix of s.*

Thus, in *"acgt"*, *"ac"* is the second prefix and *"cgt"* is the second suffix.

We use S to denote a set of strings, and subscripts to enumerate the strings in a set; that is, $S = \{s_1, \dots, s_{|S|}\}$.

2.2 String searching

Informally, string searching is the problem of finding the occurrences of patterns in a string. The following sections will give some useful definitions of string searching, briefly discuss software-based solutions, and introduce the string searching hardware that form the basis for this work.

2.2.1 String searching defined

Finding the abstracts that contain the word *"siRNA"* in a database of scientific abstracts, or finding all English words that can be transformed to *"fun"* by adding, removing, or substituting one letter, are both examples of string searching problems. In the former problem, we search the string of abstracts for all exact occurrences of the pattern `siRNA`; in the latter problem, we search the string of all English words for all occurrences of the approximate pattern `fun` that allows for one addition, removal, or replacement of a letter. These problems are also known as the exact and approximate string matching problems (Gusfield 1997).

Formally, given a string s and a pattern p, the string search problem is to find all occurrences of pattern p in string s. To determine whether a given pattern matches a given string, we introduce the concept of hit functions on the space of strings $\mathbb{S}$ and patterns $\mathbb{P}$ (Halaas et al. 2004):

Def. 3 *The hit function $H(s,p)\colon \mathbb{S} \times \mathbb{P} \to \{0,1\}$ is a function returning 1 if pattern p matches some suffix of s, and 0 otherwise. For example, if $s =$ "regular" and $p =$ `ar`, then $H(s,p) = 1$.*

Thus, a hit function formalizes the concept of matching a pattern against a string.

2.2.2 String searching in software

There are several algorithms that solve the exact string matching problem and variants of the approximate string matching problem (for example Gusfield (1997); Navarro (2004); Navarro and Raffinot (2002); Witten et al. (1999)). These algorithms find pattern occurrences either by scanning the string or by constructing and querying an index of the string. Index-based algorithms are usually faster than are scanning algorithms and can handle much larger strings than can scanning algorithms, but index-based solutions i) are not available for all search types; ii) can have excessively large space requirements; and iii) are inefficient on volatile data, as building the index is costly (Navarro 2001).

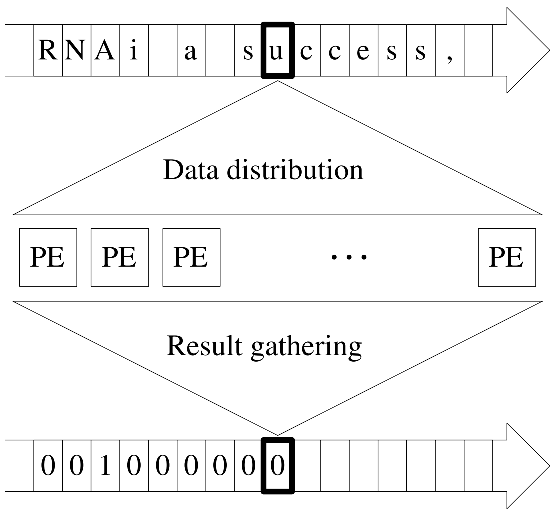

Figure 2.1: Basic PMC architecture (adapted from Hetland (2003)).

2.2.3 String searching in hardware

This work considers a specific class of patterns, namely the patterns that can be handled by a special purpose search hardware called the pattern matching chip (PMC). The patterns have several key properties, including (Halaas et al. 2004)

1) concatenation of basic strings and patterns;
2) alphanumerical comparisons of basic strings;
3) boolean operations on subexpressions;
4) hamming distance filtering;
5) hit lingering (latency); and
6) limited regular expressions.

By combining these properties, we can construct patterns such as for example $\{\texttt{siRNA}:\texttt{p}\geq 4,\texttt{d}=50\}\&\{\texttt{off target}:\texttt{p}\geq 9,\texttt{d}=50\}$. This pattern looks for occurrences of the strings *"siRNA"* and *"off target"* within a distance of fifty bytes, where each string tolerates up to one mismatch. Obviously, this pattern matches *"siRNA off target"* and *"off target siRNA"*, but the pattern also matches *"miRNA off-target"* or *"Is RNAi a success, or do potential off-target effects threaten its status?"*, for example. Nedland et al. (2002) formalized the above pattern characteristics in the Interagon query language (IQL).

Given an IQL pattern and a string, the PMC scans the string for all occurrences of the pattern; that is, the PMC can be seen as a hit function that solves the string searching problem for this specific pattern class. To

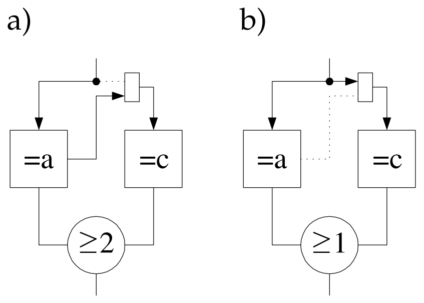

Figure 2.2: Basic PMC configuration of the data distribution tree, processing elements, and result gathering tree to match queries (a) `ca` (*"c"* followed by *"a"*) and (b) `a | c` (*"a"* or *"c"*).

understand how the PMC works, consider Figure 2.1, which shows the PMC's basic architecture. The PMC receives the string as a stream of bytes and, through a binary distribution tree, distributes this stream to a set of basic processing elements (PEs). As the byte stream flows past, each PE compares each byte with a preconfigured value and reports the Boolean result of the comparison to the binary result gathering tree. At each position in the byte stream, the result gathering tree collects the results from the different PEs and, depending on how each node in the tree is configured, reports either yes (1) or no (0) depending on whether the current pattern configuration matches the current string prefix.

Figure 2.2 illustrates how the data distribution tree, PEs, and result gathering tree can be configured to match two simple regular expressions. To match the simple string *"ca"* (Panel (a)), the PMC sequentially sends the data stream to two PEs that compare the bytes in the stream to the respective characters. The node in the result gathering tree returns a match if both PEs reports a match. To match either *"a"* or *"c"*, we change the data flow so that the PEs receive the data stream in parallel, and change the function in the result gathering tree to the `OR` function; that is, the result gathering tree reports a match if at least one of the PEs reports a match.

The above examples illustrates the basic functionality of the PMC; Paper I describes the PMC architecture in further detail and also gives examples of more advanced configurations.

2.3 String mining

The previous section considered the problem of finding the occurrences of a given pattern in a string. The opposite problem is when you have a string (or a set of strings) with some known properties, and you want to create a pattern that characterizes parts of, or even the complete, string (set). The latter problem is more commonly known as data mining.

This work considers two different data mining problems, namely rule mining in strings and classification of strings. In the former problem, we want to find patterns that describe parts of a string; in the latter problem, we want to create patterns that recognize strings with a common property. The following sections present these problems in more detail and consider the problem of overfitting and how to estimate the pattern accuracy.

2.3.1 Mining rules in strings

Prediction rules are well known in the area of relational database mining (for example, Freitas 2002) and generally have the form

$$\texttt{IF}\ \textit{antecedent}\ \texttt{THEN}\ \textit{consequent}. \tag{2.1}$$

Here, the antecedent is some expression that relates the state of one or more attributes to the state of a single goal attribute—the consequent. The expression

$$\texttt{IF}\ \textit{age} \leq 10\ \texttt{THEN}\ \textit{income} \leq \$10.00 \tag{2.2}$$

is one example of a simple prediction rule. This rule format can also be used to describe prediction rules in sequences; for example, Das et al. (1998) used the following rule format to mine prediction rules from discretized time series:

$$\texttt{IF}\ s[i] = a\ \texttt{THEN}\ s[j] = b, 0 < j - i \leq t.$$

These rules state that symbol a will likely be followed by symbol b within t symbols in the discretized time series string s. By using the concept of hit functions (Def. 3), we can extend this rule format to include general patterns:

$$\texttt{IF}\ H(s[1 \ldots i], p_a)\ \texttt{THEN}\ H(s[1 \ldots j], p_c), 0 < j - i \leq t. \tag{2.3}$$

That is, if the antecedent pattern p_a hits at position i then it is likely that the consequent pattern p_c will hit some position within t symbols.

Given a string, our goal is to automatically extract high quality prediction rules from that string. Freitas (2002) lists three key aspects of rule quality: accuracy, comprehensibility, and interestingness. First, our rules model predictive relations in the string, and we therefore want our rules to be accurate; that is, the antecedent should have a high probability of predicting the occurrences of the consequent. Second, as we want our rules to represent high level information about the strings, rules that nobody can understand have little value beyond being simple prediction tools. Third, even if a rule is both accurate and comprehensible, the rule may not be interesting, as it may represent a very obvious relationship, such as the rule in (2.2). As several measures of rule accuracy, comprehensibility, and interestingness are known (see for example Baldi et al. (2000); Freitas (2002); Hilderman and Hamilton (1999)), one can solve the rule extraction problem by optimizing such quality measures within the space of possible rules. We use this approach in this work.

The string prediction rule from equation (2.3) describes relationships between patterns in the string. These rules can be used to model the complete string; for example, by having rules that predict the possible consequents in the string (for example, Hetland and Sætrom (2002, 2003b, 2004); and Paper IV, Section 3.1). Here, the prediction rules are, however, used to model parts of strings. Section 2.4.1 presents several variants of the rule extraction problem in the context of time series mining.

Note that Freitas (2002) refers to the above rule mining problem as dependence modeling. This problem can also be viewed as an unsupervised data mining problem. That is, there is no previous knowledge about the string and the goal is simply to extract useful information from the string. In contrast, supervised string mining are problems where each string has some associated property, and the goal is to create rules or models that predict this property. This is the subject of the following section.

2.3.2 Classifying strings

In the previous section we considered the problem of extracting rules that describe parts of a string. In string classification, each string belongs to a conceptual class—for example, a string is either written in Norwegian or not—and the task is to create a model that correctly predicts the class of an unlabeled string. More specifically, from a training set of labeled strings $(s_1, y_1), \ldots, (s_n, y_n) \in \mathbb{S} \times \mathbb{Y}$, we want to estimate a function $f : \mathbb{S} \rightarrow \mathbb{Y}$ such that the error $L(f)$ when classifying unseen examples is as small as possible (Meir and Rätsch 2003; Shawe-Taylor and Cristianini 2004). One typically assumes that the training set is generated independently and at random from some underlying, but unknown, probability distribution $P(s, y)$.

The above problem is also known as supervised data mining, as each data instance s_i has an associated label y_i that has been assigned by some supervisor (Vapnik 1998). Different feature spaces $\mathbb{Y}$ result in different versions of the supervised mining problem (Shawe-Taylor and Cristianini 2004); for example, the problem where the labels are real numbers ($\mathbb{Y} = \mathbb{R}$) is known as regression. Here, we only consider binary classification, where $\mathbb{Y} = \{-1, 1\}$; that is, the strings belong to one of two classes. Section 2.4.2 gives additional details on two binary classification problems where the strings are DNA sequences.

2.3.3 Overfitting and accuracy

In general, the true performance or generalization error of a model f is

$$L(f) = \int \lambda(f(s), y) dP(s, y), \tag{2.4}$$

where λ is some loss function, such as for example the 0/1-loss

$$\lambda(f(s), y) = \begin{cases} 1 & \text{if } yf(s) \leq 0, \\ 0 & \text{otherwise}, \end{cases} \tag{2.5}$$

which is often used in binary classification (Meir and Rätsch 2003). As we do not know the generating probability distribution $P(s, y)$, we can not directly find the optimal hypothesis that minimizes equation (2.4). Instead, we must use approximations based on the model's performance on the training set, such as for example the empirical risk

$$\hat{L}(f) = \frac{1}{n} \sum_{i=1}^{n} \lambda(f(s_i), y_i). \tag{2.6}$$

Optimizing the empirical risk directly, however, typically leads to poor results on unseen data. This is because most training sets are small, and without restricting the class of hypotheses, one can find infinitely many solutions that have zero empirical risk, as illustrated in Figure 2.3. Thus, to solve the classification problem when faced with limited training sets, one must limit the complexity of the candidate solutions (Meir and Rätsch 2003). This is called regularization.

Figure 2.4 illustrates another problem, which is related to the problem of limited training sets. In this figure, one of the training instances has been

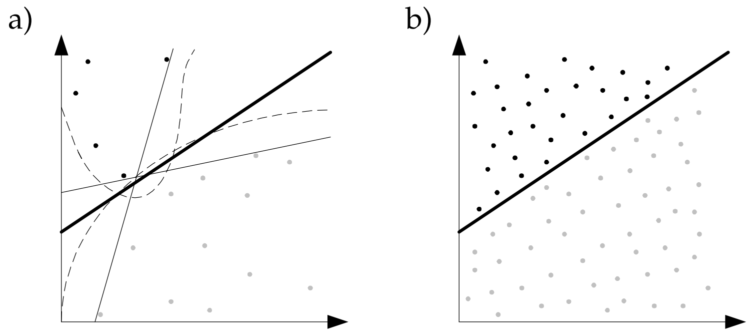

Figure 2.3: Training without restrictions in small training sets can lead to poor results. (a) Not only are there infinitely many linear functions that can perfectly separate the positive and negative samples in the small training set. There are also infinitely many more complex functions that can perfectly separate the samples. (b) Most of these solutions will, however, have poor results on the unseen data.

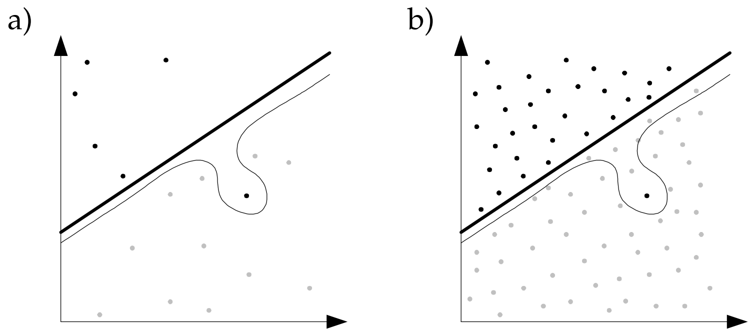

Figure 2.4: (a) Arbitrary complex functions can find perfect solutions in noisy training sets, but (b) this can give poorer results on unseen data.

mislabeled, which is often the case in "real world" data sets; for example, because of typing errors when the data set was collected. Mislabeled instances are noise because they are not true samples from the generating probability distribution $P(s, y)$. Training with arbitrary complex functions will, however, give zero empirical risk even on noisy training sets. Thus, regularization is also necessary when one suspects that the training set contains noise.

Certain machine learning algorithms, such as support vector machines (Schölkopf et al. 2000; Vapnik 1995) and regularized boosting algorithms (Meir and Rätsch 2003; Rätsch 2001; Rätsch et al. 2001, 2000), have regularization as an integral part (Chapter 4 will discuss boosting algorithms in more detail). For other algorithms, such as neural networks, there exist several pruning algorithms that limit the networks' complexity (Reed 1993). Early stopping is a method that limits the model complexity by choosing the model that has the best performance on a validation set, which is a set of labeled data instances that are not used to generate the models (Prechelt 1998). There are several issues with using this method, such as for example how to identify the model having the best validation performance, that the models become overfitted on the validation set, or that using a validation set reduces the size of the training set and may consequently reduce the performance of the generated models. Nevertheless, early stopping is useful because it is a general method that can be used with any machine learning method that iteratively creates increasingly complex models.

2.3.4 Estimating model accuracy

As outlined in the previous section, the goal in classification is to create a model that generalize well to unseen samples. Although certain algorithms, such as support vector machines (SVMs), provably can create models that generalize well, their performance varies depending on the problem and the kernel used to train the SVM (see for example, Müller et al. (2001)). Thus, in general, one wants to estimate the generalization performance of every model created by every machine learning method, to get an idea of the risk involved in using the models in predictive settings. In addition, estimating generalization performance is important when comparing the performance of different models on a given problem.

k-fold cross validation is a general method for estimating the performance of models (Stone 1974). In k-fold cross validation the training set is divided into k equal sized, non-overlapping subsets (or folds). Then, for each fold $i, i \in \{1, \ldots, k\}$, we train a model on the remaining $k-1$ folds and test its performance on the i-th fold. The average of these k test values is known to be a good estimate of the models' true generalization error for $k \geq 10$ (Martin and Hirschberg 1996).

Many machine learning algorithms have several parameters that influence their generalization performance—for example, the regularization parameter in SVMs—and cross validation is often used to select the optimal values of such parameters. As Salzberg (1997) notes, however, one should be very careful when making conclusions based on the performance of such optimized models. Testing multiple parameter settings and choosing the setting that gives the highest cross validation estimated performance is a form of training, as the user identifies the best parameter setting for the current data set. This optimized performance does not, however, represent the true generalization performance of the model. Thus, using such optimized performance measures to compare different algorithms or classifiers will, in general, lead to invalid conclusions about which algorithm or classifier is the best.

Identifying optimal parameters can, however, be very important for some machine learning algorithms. To solve this problem—that is, identifying optimal parameters while getting reliable estimates of the generalization performance—one must test the optimized models on test data not used when training and optimizing the models. More specifically, k-fold cross validation can again be used to estimate the generalization performance. To do this, we use k-fold cross validation to identify the optimal parameters on each of the k training sets, and test the optimized model on the remaining test fold. As a result, the final generalization error estimate is independent of both the model training and parameter optimization.

This double k-fold cross validation procedure is, however, very computationally expensive. Compared to using only k-fold cross validation without optimizing parameters, the number of machine learning runs needed increases with a factor of $k \cdot p$, where p is the number of parameter settings tested. Note, however, that any procedure that does not completely separate the training data from the test data—both for model training and parameter optimization—risks producing biased estimates of the generalization performance. Procedures that mix the training and test data do, however, seem to be common (for example, Müller et al. (2001)).

2.4 Problem domains

2.4.1 Rule mining in time series

A time series is a chronologically ordered sequence of observations. Time series are common, as many different events are continuously measured through time; examples include daily closing prices of stocks, daily rainfall, monthly employment figures, or quarterly profits. Although the observations in general can be real-valued, there exist several methods that transform real-valued time series into a sequence of symbols from a finite alphabet; see for example Paper IV, Section 2.5 and the references therein. Thus, we will here model time series as strings (in accordance with Def. 1).

Section 2.3.1 outlines the general problem of mining prediction rules in strings, which is to automatically extract accurate, comprehensible, and interesting rules. This can be formalized as follows (Paper IV):

Def. 4 (Unsupervised rule mining) *Given a string s and a rule language $L = L_a \overset{T}{\underset{w}{\Rightarrow}} L_c$, where L_a and L_c are the antecedent and consequent languages, w is a minimum distance, and T is a maximum distance, find rules $R \in L$ that optimize some objective function $f(R)$.*

Note that L represents the space of possible prediction patterns, as L_a and L_c specify the legal antecedent and consequent patterns, and w and T specify the minimum and maximum distances between the antecedent and consequent. Thus, any rule $R \in L$ conforms to the general prediction rule format in (2.3).

Def. 4 formulates the rule mining problem as an optimization problem. This definition is the basis for Paper II, which solves the rule mining problem by optimizing different interestingness measures. Paper III extends this work by simultaneously optimizing rule accuracy, comprehensibility, and interestingness; that is (Paper IV):

Def. 5 (Multi-objective Rule Mining) *Given a string s, a rule language L, and a set $F = \{f_i\}$ of objective functions, find a diverse set of rules $R = \{r_i\}, r_i \in L$ with high objective values $f_i(r_j)$, such that no rule $r_i \in R$ dominates any other rule $r_j \in R$. One rule dominates another if it is as good or better in terms of all objectives, and strictly better in terms of at least one objective.*

The final problem we will be considering is to find rules that relate the characteristics of two time series, or formally (Paper IV):

Def. 6 (Mining Multiple Series) *Given a set S of m strings s_i ($s_i \in S$) and a rule language $L = L_a^i \overset{T}{\underset{w}{\Rightarrow}} L_c^j$, $i, j \in [1, m]$ and $i \neq j$, find rules $R \in L$ that optimize some objective function $f(R)$.*

Here, the antecedent and consequent patterns occur in different strings s_i and s_j. Thus, if the antecedent pattern occurs in s_i the rule predicts that the consequent pattern will occur in s_j within a distance window of $[w, T]$.

Papers II, III, and IV give further details on the three problems, including the objective functions used and the results of several simulation experiments.

2.4.2 Classification of DNA sequences

An organism's genetic information is encoded in a large molecule called deoxyribonucleic acid (DNA). DNA consists of two helical chains of four smaller molecules called nucleotides, and these four nucleotides, adenine

$$\begin{array}{c} 5'\,\mathrm{A\,C\,G\,T\,G\,T\,C\,G}\,3' \\ |\;\;|\;\;|\;\;|\;\;|\;\;|\;\;|\;\;| \\ 3'\,\mathrm{T\,G\,C\,A\,C\,A\,G\,C}\,5' \end{array}$$

Figure 2.5: The double stranded DNA. One strand uniquely determines the other strand.

(A), cytosine (C), guanine (G), and thymine (T) form the alphabet of the genetic code. The two DNA chains are arranged in a double helix such that A paires with T and C paires with G, as illustrated in Figure 2.5. Thus, one strand uniquely determines the other DNA strand.

Ribonucleic acid (RNA) is a molecule similar to DNA in that it also encode genetic information as a chain of nucleotides. In RNA, thymine is, however, replaced with uracil (U), and the RNA molecule is (usually) single stranded. Both DNA and RNA are uniquely read from the 5′ to the 3′ end. Thus, DNA can be considered a string from the alphabet $\Sigma_D = \{A, C, G, T\}$, and RNA a string from the alphabet $\Sigma_R = \{A, C, G, U\}$.

A common definition of a gene is a subsequence of DNA that contains the information for making one RNA (Lewin 2000). That is, the gene encodes a sequences that is transcribed to an RNA sequence by the cell. For most genes the RNA sequence functions as a messenger (mRNA) that is translated into a protein. Nevertheless, the RNA sequence of some genes is not translated; the function of these non-coding RNA (ncRNA) genes lies in the RNA sequence itself and not in a protein product. The protein coding genes are, however, the more prevalent and best known gene class. Indeed, the best know ncRNA genes are the transfer RNA and ribosomal RNA that the cell uses to translate RNA into proteins.

There is, however, a growing recognition that ncRNA genes are more diverse and more common than previously believed (Eddy 2001). For example, microRNAs were unknown until Lee et al. (1993) and Wightman et al. (1993) identified the first microRNA gene, and were not identified as a large gene class until 2001 (Lagos-Quintana et al. 2001; Lau et al. 2001; Lee and Ambros 2001). Recently, Lim et al. (2003) estimated microRNAs to constitute about one percent of all predicted genes in the human genome. Because of this growing interest, and because existing tools that predict protein coding genes can not reliably predict ncRNA genes (Eddy 2001), there is a growing need for tools that predict novel ncRNA genes. That is, given a DNA string, one would like to predict whether or not the string contains an ncRNA gene. This is essentially a binary classification problem and Paper VII presents a possible solution.

Many ncRNAs regulate the expression of other genes. This is for example the case for microRNAs, which regulate the expression of target

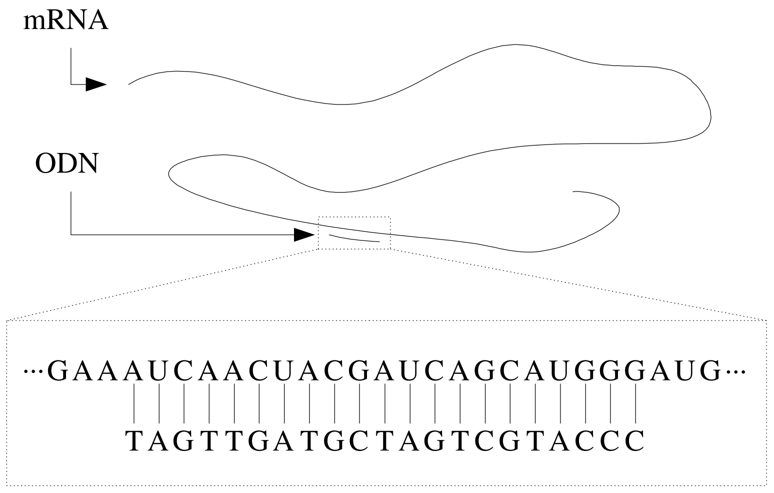

Figure 2.6: The antisense oligonucleotide base-pairs perfectly with its target mRNA.

genes through an antisense mechanism. That is, the $\sim$ 22 nucleotide (nt) microRNA sequence regulates its target genes by base pairing to the target genes' mRNA. Almost perfect base-pairing degrades the mRNA (Yekta et al. 2004; Zeng et al. 2002); less perfect base pairing prevents the mRNA from being translated into a protein (Moss et al. (1997); Olsen and Ambros (1999); see Bartel (2004) for a review). MicroRNAs are a relatively recent discovery, but sequence specific knockdown of genes through antisense mechanisms have been used since the early 1980s (Scherer and Rossi 2003). These synthetic constructs, such as the antisense oligonucleotides (ODNs) and the more recent short interfering RNAs (siRNAs), are designed to effectively knock down the mRNAs of specific genes. That is, the ODNs and siRNAs are designed to base-pair perfectly with their target mRNAs, but not with any other mRNAs. Figure 2.6 illustrates the base-pairing between an ODN and its target mRNA.

Not all ODNs and siRNAs are equally effective, however. For example, if ODNs and siRNAs are designed against random mRNA positions, many of the resulting constructs will not give detectable knockdown. Because lab experiments are both costly and time consuming, one would like to reduce the chance of failure by predicting whether or not a candidate ODN or siRNA will be effective. This is a binary classification problem; that is, given a DNA string, predict whether the resulting ODN or siRNA will give effective knockdown (1) or not (-1). Paper V describes an algorithm that solves this problem and Paper VI compares this algorithm to other recently published algorithms for predicting siRNA efficacy.

Snøve Jr. (2005) discusses siRNAs and microRNAs and how they relate in further detail.

Chapter 3

Evolving string patterns

IN many computer science problems one searches for some solution in a large space of candidate solutions. Specifically, the rule mining and classification problems from Chapter 2 are examples of such problems. A common method to solve such search problems is to recast them as optimization problems; that is, one uses a function to measure how close a candidate solution is to the desired solution. The goal is to find the solution that optimizes this function.

If the objective function is well behaved and its relationship to the search space is well understood, there are methods that can efficiently give exact solutions. More often, however, one must resort to heuristic search methods that find approximate solutions. Although there are many different heuristic search methods (see for example Blum and Roli (2003)), this chapter will focus on one method in particular. The method, called genetic programming, comes from a family of heuristic search algorithms called evolutionary algorithms. The following sections introduce evolutionary algorithms in general and move on to discuss genetic programming and how it can be used to create string prediction rules and string classification models.

3.1 Evolutionary algorithms

The term evolutionary algorithms refer to a family of heuristic search and optimization algorithms inspired by Charles Darwin's ideas of evolution by natural selection. Figure 3.1 gives the pseudocode for the basic evolutionary algorithm. As the pseudocode shows, there are four key elements in evolutionary algorithms:

```
create initial population of solutions
**repeat**
    select individuals based on fitness
    transform selected individuals
    update population with transformed individuals
**until** stopping criterion is satisfied
```

Figure 3.1: Pseudocode for a general evolutionary algorithm.

1) *Solution population*. The algorithm operates on a set of candidate solutions.
2) *Fitness and selection*. The algorithm measures the fitness of each candidate solution and uses the fitness to guide the search for candidate solutions.
3) *Variation*. The algorithm introduce variation in the solution set by using some method to modify the candidate solutions. Typical variation operations are:
    a) *crossover*—selected solutions exchange random parts;
    b) *mutation*—the selected solution is randomly modified; and
    c) *reproduction*—the selected solution remains unchanged.
4) *Repeated selection and variation*. The algorithm repeats the fitness-based selection and solution modification over several generations of candidate solutions.

In short, evolutionary algorithms use simulated evolution in a population of candidate solutions to optimize a fitness measure within some solution space.

As mentioned previously, the term evolutionary algorithms encompasses several different algorithms. Freitas (2002) identifies four main variants: evolution strategies, evolutionary programming, genetic algorithms, and genetic programming (see also Banzhaf et al. (1997)). Originally, these methods were distinguished by how the solutions were represented and which variation operations were preponderant. For example, the candidate solutions in genetic algorithms were originally fixed length strings (Holland 1975), whereas the candidate solutions in genetic programming were computer programs in the form of syntax trees (Koza 1992); and evolution strategies and evolutionary programming mainly used mutation for variation, whereas genetic algorithms and genetic programming emphasized crossover. This distinction has, however, become increasingly blurred as the original algorithms have evolved; for example, O'Neill and Ryan (2003) describe a method that uses variable length binary strings to evolve computer programs.

This thesis focuses on genetic programming, which here will mean methods that use evolutionary computation to evolve symbolic expressions in some solution language. Although this is a broad definition, which for example also encompasses O'Neill and Ryan's grammatical evolution (O'Neill and Ryan 2003), this work is based on Koza's syntax tree representation (Koza 1992). The following section discusses genetic programming in more detail.

3.2 Genetic programming

Section 3.1 outlined four key elements of evolutionary algorithms: a solution population, a fitness measure and selection algorithm, variation operators, and repeated selection and variation. This section presents how the solutions in the genetic programming population can be represented and how the common variation operations—crossover and mutation—can be implemented. The following two sections discuss fitness and selection.

3.2.1 Solution representation

In Koza's genetic programming (Koza 1992, 1994; Koza et al. 1999, 2003), the individuals in the population are syntax trees in some solution language. Others have developed alternative solution representations, such as linear strings or directed acyclic graphs; see for example Banzhaf et al. (1997); O'Neill and Ryan (2003) and references therein.

The solution language defines the structure of the possible solutions, and hence also the possible solution space. The question then is: how can one specify the solution language? Koza (1992) uses the concepts of function and terminal sets, where the terminal set T contains the possible leaf nodes in the tree, and the function set F contains the possible internal nodes in the tree. As each function in F can have a varying number of arguments—represented as children in the tree—each function in F has an associated argument count. For example, the language of Boolean logic can be represented as $F = \{(\vee, 2), (\wedge, 2), (\neg, 1)\}$ and $T = \{0, 1\}$. In this representation, a random solution can be grown by randomly selecting elements from the two sets.

In the function and terminal set representation, each function can have any one of the other functions or terminals as children in the syntax tree. Although this representation is attractive because of its simplicity, it has one major drawback: it is impossible to specify languages where certain functions can only take certain inputs. For example, consider the language

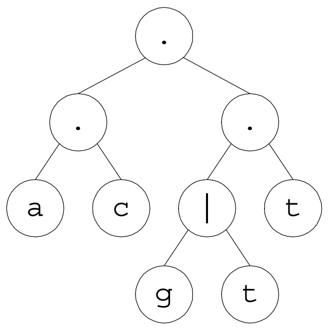

Figure 3.2: A syntax tree for the expression `ac(g | t)t`.

	a)	b)	
(1)	<con> ::= <con> · <con>	$F_0 = \{(\cdot,2),(\mid,2)\}$	$T_0 = \{\texttt{a},\texttt{c},\texttt{g},\texttt{t}\}$
(2)	| <alt>		
(3)	<alt> ::= <alt> | <alt>	$F_1 = \{(\mid,2)\}$	$T_1 = \{\texttt{a},\texttt{c},\texttt{g},\texttt{t}\}$
(4)	| <chr>		
(5)	<chr> ::= `a`	$F_2 = \emptyset$	$T_2 = \{\texttt{a},\texttt{c},\texttt{g},\texttt{t}\}$
(6)	| `c`		
(7)	| `g`		
(8)	| `t`		

Figure 3.3: (a) The grammar of a simple pattern language. The grammar is in Backus Naur form. (b) The grammar implicitly defines a hierarchy of function and terminal sets.

that uses the regular expression functions union and concatenation to create patterns that consists of concatenations of single characters or unions of single characters, such as `ac(g | t)t` (syntax tree in Figure 3.2). Here, the concatenation function $(\cdot)$ can have both itself, the union function $(\mid)$ and the terminals $\{\texttt{a},\texttt{c},\texttt{g},\texttt{t}\}$ as children, but the union function $(\mid)$ can only have itself and the terminals as children. As we want to create solutions with an even more complex structure, such as the different string rule languages in Section 2.4.1, we need an alternative to the function and terminal set representation to specify the solution language.

A more general, and common, method to specify languages is to specify the language's grammar in the form of production rules. For example, Figure 3.3(a) shows the grammar for the previous regular expression language in Backus Naur form (Knuth 1964). The grammar can however, also be represented as a hierarchy of function and terminal sets, as illustrated in Figure 3.3(b). The idea of representing the solution language as several function and terminal sets is the key element in the strongly typed genetic programming of Montana (1995). Although initially developed to solve the problem that certain functions take arguments of different types—for example, an if-then-else function that takes one Boolean and two integers as arguments—this method can also be used to create languages with complex grammars. (Koza et al. (2003) refer to strong typing as a constrained syntactic structure.) In this representation, a random solution can be grown by randomly selecting elements from the function and terminal set corresponding either to the grammar's start production or to each function's different arguments.

3.2.2 Variation operations

The standard method of using one function and terminal set to specify legal solutions can be used to solve problems where one desires a more complex grammar such as the one in Figure 3.3. However, using the standard method comes at the expense of a dramatically increased search space; for example, Montana (1995) reports experiments where only one solution in 2500 was legal. Thus, the computational benefit of ensuring that all solutions are legal should be obvious. To ensure this, the standard variation operations in genetic programming, subtree swapping crossover and tree generating mutation, must be changed to respect the grammatical restrictions.

Figure 3.4 illustrates how standard subtree swapping crossover disrupts the grammar constraints. To ensure that the individuals modified by crossover are legal solutions, the crossover operation must be modified as follows:

1) select a random subtree in parent one—Figure 3.4(a);
2) determine the set of subtrees in parent two that can be replaced with the selected subtree from parent one without creating an illegal expression—in Figure 3.4(b) all the subtrees can be replaced, but in Figure 3.5(b) only the subtrees starting in the dark gray nodes can be selected; and
3) create one offspring by randomly replacing one of the candidate subtrees in parent two with the subtree from parent one—in other words, only the offspring in Figure 3.4(d) is created.

The tree generating mutation operation is easily modified to produce legal expressions. In the standard version, the mutation operation i) selects a random subtree and ii) replaces this subtree with a randomly generated subtree. In the modified version, the mutation operation uses the function and terminal sets of the deleted subtree's parent to ensure that the random

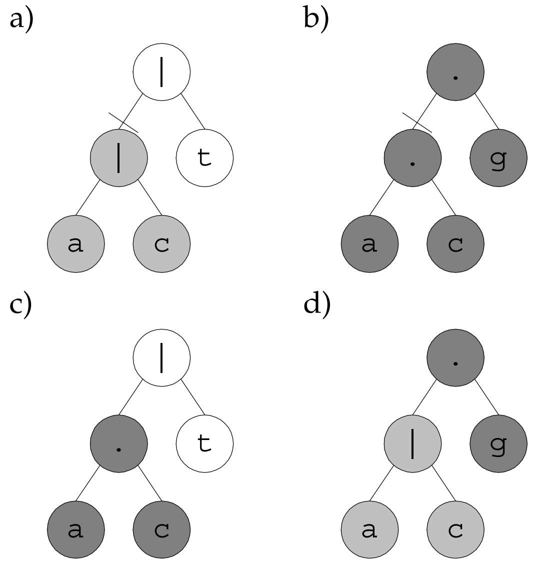

Figure 3.4: Standard subtree swapping crossover disrupts grammar. Standard subtree swapping crossover selects a subtree in each parent (a) and (b) at random and swaps the subtrees to create two offspring (c) and (d). Although both parents are legal expressions in the grammar from Figure 3.3, offspring (c) is not.

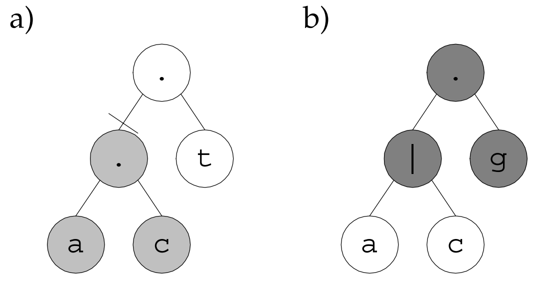

Figure 3.5: Constrained crossover conserves grammar. The subtree selected in parent one (a) can only replace one of the subtrees rooted in the dark gray nodes in parent two (b). Otherwise, the resulting offspring will not be a legal expression in the grammar from Figure 3.3.

replacement is legal. To illustrate, consider the syntax tree in Figure 3.4(a). If the mutation operation in the first step deletes the light gray subtree, it can only create replacement subtrees consisting of the union function | and terminal symbols `a`, `c`, `g`, and `t`.

Note that using multiple function and terminal sets is not the only method to create solutions with specific grammars. O'Neill and Ryan (2003) briefly review other tree-based methods and describe a string-based method to create solutions with specific grammars.

3.3 Fitness

Evolutionary algorithms solve problems by optimizing a fitness function f within the space of candidate solutions. In other words, evolutionary algorithms in general, and genetic programming in particular, try to find the solution that maximizes (or minimizes) f. Thus, both the solution space and the fitness function are important for genetic programming to succeed in solving a given problem. In particular, the fitness function should give a continuous measure of how well the candidate solutions solve the given problem so that (Banzhaf et al. 1997):

> *"...smaller improvements in how well a program has learned the learning domain are related to smaller improvements in the measured fitness of the program, and larger improvements in how well a program has learned the learning domain are related to larger improvements in its measured fitness."*

Section 2.4 presented the string rule mining and string classification problems as optimization problems, but did not present the functions that should be optimized to solve the problems. This section presents two fitness measures that can be used by genetic programming to solve these two problems. Common to both measures is that they use the hits of the pattern to be evaluated as input. Thus, the largest part of the fitness calculation is to search for the occurrences of the candidate solution in the the training data. We use the pattern matching chip from Paper I to accelerate these searches.

3.3.1 Fitness in rule mining

As stated in Section 2.3.1, the goal in rule mining is to find rules that are accurate, comprehensible, and interesting. The J-measure $J(R)$ evaluates the rule R's interestingness (Smyth and Goodman (1991); see Paper II for a definition), but if the J-measure is used directly as the fitness function in

	$y = 1$	$y = -1$
$f(s) = 1$	TP	FP
$f(s) = -1$	FN	TN

Figure 3.6: Confusion matrix for classification model f. y is the string s's actual class.

genetic programming, the resulting rules may suffer from a low confidence (Paper II) (the confidence measures a rule's accuracy; see Paper II for a definition). Because of this, Paper II introduced the modified J-measure:

$$J'(R) = J(R) \cdot F(c(R)), \tag{3.1}$$

where R is a string rule in the solution language $L = L_a \overset{T}{\underset{w}{\Rightarrow}} L_c$, $c(R)$ is the rule's confidence, and F is a function ensuring that the rule's confidence is above some threshold.

As a sharp cutoff function such as

$$F'(c(R)) = \begin{cases} 0 & \text{if } c(R) < c_{min}, \\ 1 & \text{otherwise,} \end{cases} \tag{3.2}$$

changes the J-measure to a discontinuous function, we instead used the sigmoid cutoff function

$$F(c(R)) = \frac{1}{1 + \mathrm{e}^{-(c(R) - c_{min}) \cdot g}}. \tag{3.3}$$

Here, c_{min} is the confidence cutoff and g is a parameter that adjusts the sharpness of the cutoff. This function preserves the smoothness of the J-measure and penalizes solutions with low confidence.

3.3.2 Fitness in classification

In classification, the empirical risk (2.6) is one possible candidate for a fitness measure. In binary classification, with 0/1-loss (2.5), the empirical risk reduces to the accuracy rate

$$Acc = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}, \tag{3.4}$$

where TP, TN, FP, and FN are the number of true positive, true negative, false positive, and false negative classifications (see Figure 3.6). As Freitas (2002) points out, however, the accuracy rate is not a good measure of a rule's predictive accuracy on problems where the class distribution is very unbalanced. To illustrate, consider a problem where the number of negative strings outnumber the positive strings nine to one. The naive classifier that always predicts the negative class will here have an accuracy rate of $9/(9+1) = 0.9$. This is also the case for a random classifier that takes the class distribution into account.

Thus, not only does the accuracy rate depend on the class distribution, which makes it very difficult to assess a classifier's performance based on the accuracy rate alone. The range of good accuracy rates—that is, rates better than the naive classifier—gets more compressed the more skewed the class distribution is. In practice, this results in that the evolutionary algorithm will have more trouble finding solutions that are better than the naive solution. Note that other machine learning algorithms such as neural networks, decision tree induction algorithms, and support vector machines have similar problems on skewed data sets; see for example Japkowicz (2000) (neural networks), Weiss and Provost (2001) (decision trees), and Akbani et al. (2004); Forman and Cohen (2004); Wu and Chang (2003) (support vector machines).

Instead of the accuracy rate, one wants a fitness measure that always gives the same score to the naive classifiers, independent of the training set's class distribution. One possible solution is to modify the accuracy rate to account for the skewness of the class distribution:

$$Acc_{Norm} = \frac{\mathrm{TP} \cdot w_P + \mathrm{TN} \cdot w_N}{n}, \tag{3.5}$$

where $n = \mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}$, and w_P and w_N adjust the relative importance of the positive and negative classes. To ensure that Acc_{Norm} is independent of the class distribution, w_P and w_N must satisfy

$$\frac{(\mathrm{TP} + \mathrm{FN}) \cdot w_P}{n} = \frac{1}{2} \qquad \frac{(\mathrm{TN} + \mathrm{FP}) \cdot w_N}{n} = \frac{1}{2}, \tag{3.6}$$

which gives

$$Acc_{Norm} = \frac{1}{2}\left(\frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} + \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}}\right) = \frac{1}{2}\left(Se + Sp\right). \tag{3.7}$$

Thus, by adjusting the accuracy rate for class skewness, the resulting accuracy measure is simply the average of the rate of correctly classified positives and correctly classified negatives. These rates are also known as the sensitivity Se and specificity Sp.

Note that alternatively, Acc_{Norm} is given by

$$Acc_{Norm} = \frac{1}{n} \sum_{i=1}^{n} d_i \cdot \lambda(f(s_i), y_i), \tag{3.8}$$

where $d_i = w_P$ if s_i is in the set of positive sequences and $d_i = w_N$ otherwise.

Although the normalized accuracy Acc_{Norm} is independent of the class skewness, this measure may still not be optimal when used as a fitness measure in evolutionary computation. The main problem is that Acc_{Norm} depends linearly on the sensitivity and specificity. Hence, both small and large deviations from perfect classification have the same weight. This is not ideal, as the evolutionary algorithms can use error signals that are differentially weighted to guide the learning process (Baldi et al. 2000; Banzhaf et al. 1997). Because of this, the correlation score has been a more popular fitness measure i binary classification problems (for example Hetland and Sætrom (2004); Koza (1994); Koza et al. (1999)), as it is both independent of class skewness and a non-linear function of the sensitivity and specificity (Baldi et al. 2000). Nevertheless, the normalized accuracy Acc_{Norm} has its applications. Chapter 4 uses a a version of (3.8) where each sequence s_i in the training set can have a different weight d_i.

3.4 Selection

Given a population of candidate solutions and a fitness measure to be optimized by the evolutionary algorithm, the question then is how to select the candidate solutions that will form the updated population (see Figure 3.1). Selection algorithms in general select candidate solutions such that fitter solutions have a higher chance of being selected than have less fit solutions (although there are exceptions; see, for example, Hutter (2002)). For example, in fitness-proportional selection, the probability of a candidate solution being selected equals its fitness divided by the population's total fitness. In comparison, tournament selection selects the fittest of t randomly selected candidate solutions. Banzhaf et al. (1997) and Freitas (2002) briefly discuss the advantages and disadvantages of fitness-proportional and tournament selection and variants, and also describe two other common selection algorithms: truncation and ranking selection.

The following subsections discuss niching—a more advanced form of selection—and present two niching algorithms: fitness sharing and species selection.

3.4.1 Niching

The above selection methods share the property that the probability of a candidate solution being selected depends on its fitness relative to the fitness of the other candidate solutions in the population (see for example Banzhaf et al. (1997) and the comparison in Blickle and Thiele (1995)). Using any of these selection methods results in that the evolutionary algorithm converges to a population of similar individuals. In practice this convergence introduces two interrelated problems. First, the evolutionary algorithm may have converged to a local minimum. The probability of this happening depends on the parameter settings (Banzhaf et al. 1997; Freitas 2002) such that given suitable parameters, the evolutionary algorithm can converge to the global optimum (for example, Schmitt (2001, 2004) discuss the asymptotic convergence of a genetic algorithm with a simulated annealing-type selection method). Nevertheless, as for simulated annealing (Kirkpatrick et al. 1983; Metropolis et al. 1953), where asymptotic convergence is guaranteed with a sufficiently slow annealing schedule (Geman and Geman 1984), one often resorts to parameter settings that give faster convergence, but that is not guaranteed to reach the global optimum. Second, many problems have multiple global optima, and one often desires that the evolutionary algorithm should converge on as many of these optima as possible (Freitas 2002).

To prevent premature convergence and promote convergence to multiple optima, one must ensure that the population stays as diverse as possible. This has lead to several different niching methods, where the probability of a candidate solution being selected depends on its similarity to other candidate solutions in addition to its relative fitness. Note that the fitness uniform selection scheme takes this idea to the extreme, as the probability of a candidate solution being selected only depends on the number of similar individuals and the total number of groups of similar individuals (Hutter 2002). Mahfoud (1995) describes several niching methods; here we will concentrate on one method, called fitness sharing (Goldberg and Richardson 1987).

3.4.2 Fitness sharing

In fitness sharing, the fitness of each candidate solution is penalized based on the number of similar individuals. That is, given a population of candidate solutions P and a fitness measure $f(p), p \in P$, the shared fitness $f_s(p)$ is

$$f_s(p) = \frac{f(p)}{\sum_{p' \in P} s(d(p, p'))} \tag{3.9}$$

where $d(p, p')$ measures the distance between individuals p and p', and s is the sharing function

$$s(d(p, p')) = \begin{cases} 1 - \left(\frac{d(p,p')}{\delta}\right)^a & \text{if } d(p, p') < \delta, \\ 0 & \text{otherwise.} \end{cases} \tag{3.10}$$

Here, a is a parameter that determines the shape of the sharing function and δ is the maximum sharing distance. Any of the basic selection algorithms, such as for example tournament selection, can then use the shared fitness to select candidate solutions.

The distance function d can either measure the dissimilarity between the candidate solutions' encoding—the genotype—or the candidate solutions' behavior—the phenotype. One can, for example, use the tree edit distance to measure the genotypic distance between tree encoded candidate solutions. A candidate solution's fitness is a simple encoding of its behavior, so a simple phenotypic distance measure is the difference between two candidate solutions' fitness values. In the case of string classification (see Section 2.3.2), a more complex phenotypic distance measure can, for example, use information about which strings the two candidate solutions recognize.

Note that, strictly speaking, fitness sharing as defined in (3.9) is only a transformation of the underlying fitness measure to penalize highly populated areas of the search space. Thus, it can be argued that fitness sharing is not a separate selection method in itself. Nevertheless, evolutionary algorithms that use fitness sharing still optimize the underlying fitness measure f; the fitness sharing is just a mechanism to cope with premature convergence and multiple optima. Because of this, it is constructive to separate niching algorithms such as fitness sharing from the standard selection algorithms.

3.4.3 Species selection

Fitness sharing, as outlined above, does not preserve the ordering of the original fitness measure. That is, given two candidate solutions p_i and p_j and a fitness measure f such that $f(p_i) < f(p_j)$, it is not generally true that after using (3.9), $f_s(p_i) < f_s(p_j)$. The following section describes a selection scheme that is similar to fitness sharing, but that preserves the ordering of the original fitness function.

Species selection (Grotmol 2002), like fitness uniform selection, partitions the population into groups of similar individuals. That is, given a population P, a distance measure d, and a distance threshold δ, one groups the population into m groups G where $G = \{G_1, \ldots, G_m\}$ and $\bigcup_{i=1}^{m} G_i = P$, such that $d(p, p') \leq \delta$ for all $p, p' \in G_i$ and $G_i \in G$. In the simplest case, $\delta = 0$, the groups only consist of identical candidate solutions, candidate solutions that have identical fitness, or, in the case of string classification, candidate solutions that match the same strings. In addition, for $\delta = 0$, a simple hashing operation gives the groups in linear time.

Given a grouping of the population, species selection then selects individuals in a two step procedure:

1) use a standard selection scheme, such as for example tournament selection, to choose one group of individuals; and
2) randomly select one individual from the chosen group.

Thus, as in fitness uniform selection and fitness sharing, the idea behind species selection is to prevent premature convergence by reducing the influence of large groups of similar individuals. But unlike fitness uniform selection, which selects a group with uniform probability, species selection uses the groups' fitness to select a group. And, unlike fitness sharing, species selection with $\delta = 0$ preserves the ordering of the original fitness measure.

Chapter 4

Ensemble methods to increase classifier performance

FINDING the best classifier on a given problem is essentially a tradeoff between two factors. First, the classifier's performance on the training set should be as high as possible; that is, the empirical risk (2.6) should be as low as possible. Second, as mentioned in Section 2.3.3, given that the classifier comes from a sufficiently complex function class, the classifier's performance in the training set will be perfect and have zero empirical risk. Functions with unrestricted complexity might, however, actually overfit the training data, such that the generalization error is much higher compared to that of less complex functions.

The tradeoff between optimizing the empirical risk and limiting the complexity of the classifier is known in many guises. Geman et al. (1992); Wolpert (1997) presented it as the bias/variance dilemma; that is, the complexity of the function class defines the classifier's bias, such that more complex function classes can better approximate the process that generates the data. These function classes will, however, have a higher variance on different training sets. Correspondingly, function classes with a high bias have low variance on different training sets. Statistical learning theory formalizes this as a bound on the generalization error (2.4), which depends on the empirical risk (2.6) and the complexity of the function class as measured by the VC dimension:

Theorem 1 (Vapnik (1995, 1998)) *Let h be the VC dimension of the function class F and let $\hat{L}(f)$ be the empirical risk (2.6) with 0/1 loss (2.5). For all $\delta > 0$ and $f \in \mathsf{F}$ the inequality*

$$L(f) \leq \hat{L}(f) + \sqrt{\frac{h(\ln \frac{2n}{h} + 1) - \ln(\delta/4)}{n}} \tag{4.1}$$

bounds the generalization error with probability of at least $1 - \delta$ *for* $n > h$*, where* n *is the size of the training set.*

Thus, the generalization error is determined both by the empirical risk and the complexity of the function class. Section 4.2 gives a version of this theorem that uses a different complexity measure to give tighter bounds on the generalization error; see also Meir and Rätsch (2003); Shawe-Taylor and Cristianini (2004) and their references.

This thesis considers a restricted class of functions, namely the class of patterns that can be evaluated by the pattern matching chip (PMC—see Section 2.2.3). Although the PMC has a rich functionality, certain searches are not possible (the PMC is not Turning complete). The PMC can not, for example, handle general edit distances (see for example Gusfield (1997); Navarro and Raffinot (2002) for a definition). Thus, in the string classification problem where the positive set are all strings with edit distance less than two to some string s, the PMC can only give an approximate solution. Correspondingly, on other problems the solution may only need a subset of the PMC's functionality; for example, in the problem where the positive set are strings having a common substring.

Imagine that we want to find PMC patterns that can solve the edit distance and substring problem. Given a representative training set from both problems, we can, for example, use genetic programming to search for the PMC patterns that have the lowest empirical risk on the training set, as outlined in Chapter 3. This approach will, however, lead to two different problems. In the edit distance problem, the PMC patterns will be biased, as the PMC does not have the full functionality to solve the problem. In the common substring problem, only a small fraction of the PMC's functionality is needed, so the PMC patterns may have a high variance—especially if the training set is small, contains noise, or both. Another important factor is that using genetic programming for pattern mining is in itself a source of variance, as genetic programming is a stochastic search method. Consequently, two independent runs of genetic programming on the same training set will generally not create identical solutions.

The following sections present methods that can be used to improve the performance of the PMC patterns. These include voting to reduce the classifier variance, and boosting to reduce the classifier bias. Both approaches achieve increased performance by combining several base classifiers h into a joint classifier f of the form

$$f(s) = \sum_{t=1}^{T} \alpha_t \cdot h_t(s) \tag{4.2}$$

where T is the number of classifiers and α_t is the weight of classifier h_t such that $\alpha_t \geq 0$ and $\sum_{t=1}^{T} \alpha_t = 1$. We call this combined classifier an ensemble and refer to the simple case where all classifiers have the same weight $\alpha_t = \frac{1}{T}$ as voting.

4.1 Voting

The intuition behind using multiple classifiers in predictions is simple: given that each of n classifiers have probability p of making an incorrect classification, the probability that more than half of the classifiers make the same mistake if the classifiers are independent is (Hansen and Salamon 1990)

$$\sum_{k>\frac{n}{2}}^{n} \binom{n}{k} p^k (1-p)^{n-k}. \tag{4.3}$$

The sum (4.3) converges to zero if $p < 0.5$.

Although the above result requires that the classifiers are independent, averaging a diverse set of classifiers reduces the error compared to that of the best individual classifier, even when the classifiers are not independent (for example, Breiman (1996a); Hansen and Salamon (1990); Perrone (1993)). How much the error is reduced depends on how similar and accurate the classifiers are. For example, combining classifiers that make a few mistakes on different instances gives an increased performance; combining classifiers that make many and similar mistakes gives a poorer performance compared to the best individual classifier (Breiman 1996a,b; Friedman 1997).

Breiman (1996a) describes an algorithm that constructs a diverse voting ensemble. The bootstrap aggregated (bagging) algorithm iteratively creates base classifiers by presenting different versions of the training set to the base algorithm. To construct the training sets used in each iteration, bagging draws, with uniform probability and replacement, n random samples from the original training set (Efron and Tibshirani 1993). Here, n is the size of the original training set. The final ensemble classifier is simply the average of the base classifiers created at each iteration.

Figure 4.1 illustrates the effect of averaging a set of diverse and accurate classifiers: the voting ensemble smoothes the predictions of the individual classifiers, reducing their variance. As Figure 4.1(b) shows, however, averaging does not reduce the classifier bias. Bauer and Kohavi (1999); Breiman (1996b); Dietterich (2000); Perrone (1993); Schapire et al. (1998) give both theoretical and empirical evidence that the voting ensemble re-

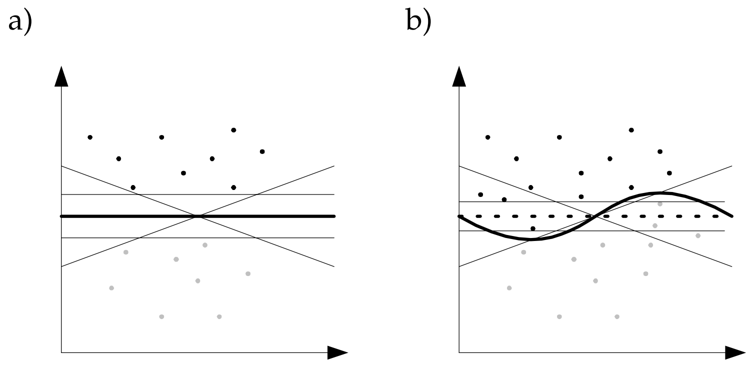

Figure 4.1: Averaging diverse and accurate classifiers reduces variance. In Panel (a) all the decision boundaries are 100% accurate in the training set, but averaging the decision boundaries creates a more optimal solution (the heavy pattern) that will generalize better to unseen examples. In Panel (b) none of the decision boundaries are 100% accurate, but their average (heavy dotted line) will again generalize better than the individual solutions. The average solution is, however, suboptimal, as the correct decision boundary is the heavy, non-linear line. The linear decision lines can only give an approximate solution—they are biased.

duces the variance but not the bias of the individual classifiers. In accordance with these results, averaging classifiers with little variance, such as nearest neighbor classifiers, does not create an improved voting ensemble (Breiman 1996a). Thus, averaging the classifiers can be seen as a form of regularization. Indeed, bagging can be seen as a form of fully regularized boosting (Rätsch 2001).

4.2 Boosting

The previous section showed that although a voting ensemble can have an increased performance compared to its best classifier, this increased performance is because averaging reduces the individual classifiers' variance. If the classifiers are biased, voting does not help. Nevertheless, given enough data, classifiers that each only perform slightly better than random, can be combined to form an arbitrarily good ensemble hypothesis (Kearns and Valiant 1994). In other words, biased classifiers that only have mediocre accuracy on the training set can be boosted to have an arbitrary high accuracy on the training set. The idea is to iteratively construct the boosted ensemble by letting the base algorithm that creates the base classifiers focus its efforts on classifying the difficult examples in the training set (Schapire 1990).

```
**Input:** $B, S, T, G$
$f_0 \leftarrow 0$
$g' \leftarrow \frac{\partial}{\partial f} G$
$d_i^{(1)} \leftarrow g'(0, y_i)$ for all $i \in \{1, \ldots, n\}$
$t \leftarrow 1$
**repeat**
    $h_t \leftarrow B(S, \mathbf{d}^{(t)})$
    $\alpha_t \leftarrow \arg\min_{\alpha \in \mathbb{R}} G(f_{t-1} + \alpha h_t, S)$
    $f_t \leftarrow f_{t-1} + \alpha_t h_t$
    $d_i^{(t+1)} \leftarrow g'(f_t(s_i), y_i)$ for all $i \in \{1, \ldots, n\}$
    $t \leftarrow t + 1$
**until** $t > T$
```

Figure 4.2: Pseudocode for a general boosting algorithm that takes as input a base algorithm B, a training set S, the number of iterations T, and a loss function $G(f, S)$.

Freund and Schapire's adaptive boosting algorithm (AdaBoost) was the first practical boosting algorithm (Freund and Schapire 1997). Given a training set $S = \{(s_1, y_1), \ldots, (s_n, y_n)\}$, AdaBoost assigns a weight d_i to each example s_i and iteratively updates these weights based on the performance of the ensemble constructed so far. At each iteration t, AdaBoost requires that the base algorithm creates a classifier h_t with a weighted empirical error

$$\varepsilon = \sum_{i=1}^{n} d_i^{(t)} \cdot |h_t(s_i) - y_i| \,, \tag{4.4}$$

such that $\varepsilon < \frac{1}{2}$. If the base algorithm can not find such a classifier, AdaBoost terminates, as including a classifier with $\varepsilon \geq \frac{1}{2}$ would reduce the overall performance of the ensemble. Given a classifier h_t with a suitable weighted empirical error, AdaBoost sets the classifier's weight α_t in the ensemble such that the weight minimizes the loss function

$$G(f_\alpha, S) = \sum_{i=1}^{n} \mathrm{e}^{-y_i(\alpha h_t(s_i) + f_{t-1}(s_i))}, \tag{4.5}$$

where f_{t-1} is the ensemble constructed so far. Generally, other loss functions can also be used, which leads to the general boosting algorithm in Figure 4.2 (Meir and Rätsch 2003).

AdaBoost will always create an ensemble with arbitrary low empirical risk (2.6), given that the base algorithm can find classifiers with a good enough weighted empirical error (4.4) (Freund and Schapire 1997). As this is also true for training sets with randomly assigned labels, one might worry that boosted ensembles will generalize poorly; that is, AdaBoost may create an ensemble with no bias but high variance. Empirical studies, however, showed that AdaBoost can reduce both bias and variance (Bauer and Kohavi 1999; Breiman 1996b; Schapire et al. 1998; Webb 2000). The discovery of Schapire et al. (1998) that AdaBoost maximizes the minimal margin $\rho_i(f) = y_i \cdot f(s_i)$ in the training set and the following theorem and corollary explain these counter-intuitive results.

Theorem 2 (Meir and Rätsch (2003)) *Let $R_n(\mathsf{F})$ be the Rademacher complexity of the class F of real valued functions $f\colon \mathbb{S} \to [-1,1], f \in \mathsf{F}$, and let $\theta \in [0,1]$. Given a training set $S = \{(s_1, y_1), \dots, (s_n, y_n)\}$ drawn independently and at random from an underlying probability distribution $P(s,y)$, then for any n, $\delta > 0$, and $f \in \mathsf{F}$ the inequality*

$$L(f) \leq \hat{L}^{\theta}(f) + \frac{4R_n(\mathsf{F})}{\theta} + \sqrt{\frac{\ln(2/\delta)}{2n}}, \tag{4.6}$$

bounds the generalization error with probability at least $1 - \delta$.

Corollary 1 (Meir and Rätsch (2003)) *Let the conditions of Theorem 2 hold and set $\mathsf{F} = co_T(\mathsf{H})$, where*

$$co_T(\mathsf{H}) = \left\{ f\colon f(s) = \sum_{t=1}^{T} \alpha_t h_t(s) \colon \alpha_t \geq 0, \sum_{t=1}^{T} \alpha_t = 1, h_t \in \mathsf{H} \right\}. \tag{4.7}$$

Then, for any n, $\delta > 0$, and $f \in co_T(\mathsf{H})$ the inequality

$$L(f) \leq \hat{L}^{\theta}(f) + \frac{4R_n(\mathsf{H})}{\theta} + \sqrt{\frac{\ln(2/\delta)}{2n}}, \tag{4.8}$$

bounds the generalization error with probability at least $1 - \delta$.

The Rademacher complexity $R_n(\mathsf{F})$, as the VC dimension in Theorem 1, measures the complexity of the function class F and is related to the VC dimension such that $R_n(\mathsf{F}) = O(\sqrt{VC/n})$ (see for example Meir and Rätsch (2003) for a definition). $\hat{L}^{\theta}(f)$ is the empirical margin error

$$\hat{L}^{\theta}(f) = \frac{1}{n} \sum_{i=1}^{n} \varphi_{\theta}(y_i f(s_i)), \tag{4.9}$$

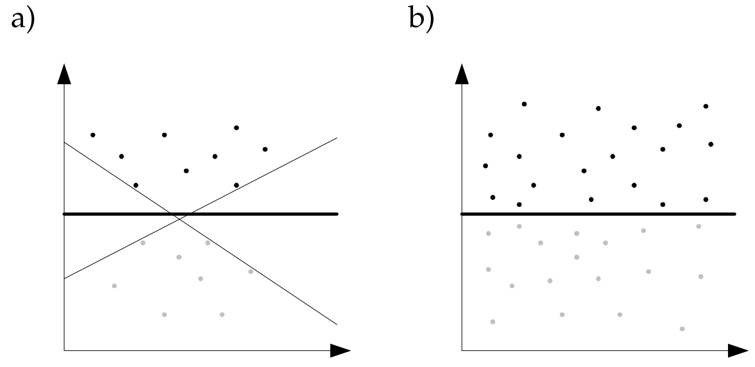

Figure 4.3: Maximizing the margin improves generalization. Panel (a) shows a finite training set where linear curves can perfectly separate the classes. There are infinitely many linear curves that can perfectly separate the data, but only the heavy line of these has the maximal margin (using a particular distance measure). Thus, maximizing the margin limits the number of possible solutions and can therefore be seen as a form of regularization. (b) Given that the training data are representative of the underlying probability distribution, the maximal margin solution has the lowest generalization error.

where

$$\varphi_\theta(z) = \begin{cases} 1 & \text{if } z \le 0, \\ 1 - z/\theta & \text{if } 0 < z \le \theta, \\ 0 & \text{otherwise.} \end{cases} \tag{4.10}$$

Theorem 2 and Corollary 1 have two important implications:

1) functions should have a small empirical margin error $\hat{L}^\theta(f)$ for a large value of the margin parameter θ to generalize well; and
2) the complexity term for the boosting ensembles in (4.8) is independent of the size of the ensemble; thus, a boosting ensemble is not more complex than its individual classifiers.

These observations lead to the following conclusion: as long as the base algorithm returns classifiers that are better than random and the empirical margin error continues to drop for a fixed θ, adding new base classifiers to the boosting ensemble will only improve its performance on unseen data.

As Schapire et al. explain, the margin measures the confidence of the classification. Consequently, maximizing the margin is equivalent with finding the classifier that gives the most confident predictions. Figure 4.3 gives an intuitive illustration of why maximizing the margin gives good generalization.

From the above discussion, one can conclude that AdaBoost will create ensembles that generalize well if the training set contains little or no noise. When the training set contains noise, however, AdaBoost can create ensembles that generalize worse than the base classifier (see for example Bauer and Kohavi (1999); Dietterich (2000); Webb (2000)). The reason is that AdaBoost concentrates all its efforts on classifying the most difficult data points, which in noisy data sets tend to be the mislabeled or noisy data points. AdaBoost maximizes the smallest margin in the training set, but in noisy data sets this may increase the overall margin error. Thus, to get optimal generalization one should accept that some instances are misclassified and have a negative margin, as long as the overall margin is large. In other words, one needs some form of regularization to account for the noise in the training set.

Meir and Rätsch (2003) review regularized boosting algorithms in general; here we will briefly discuss two algorithms: AdaBoost$_{\text{Reg}}$ (Rätsch et al. 2001) and ν-Arc (Rätsch et al. 2000; Rätsch et al. 2000). AdaBoost$_{\text{Reg}}$ reduces the influence of the most difficult data points in the data set, as one expects that these may be noise in the data set. To do this, AdaBoost$_{\text{Reg}}$ defines the mistrust of each data point as

$$\mu_i^{(t)} = \frac{1}{\sum_{r=1}^{t} \alpha_r} \sum_{r=1}^{t} \alpha_r d_i^{(r)}, \tag{4.11}$$

which is the average weight of example i so far in the boosting process. AdaBoost$_{\text{Reg}}$ then uses a regularization constant C to adjust the importance of the example's mistrust compared to its margin. This results in the following loss function

$$G(f^{(t)}, S) = \sum_{i=1}^{n} \mathrm{e}^{-(y_i \frac{f^{(t)}(s_i)}{\sum_{j=1}^{t} \alpha_j} + C\mu_i^{(t)}) \sum_{j=1}^{t} \alpha_j}. \tag{4.12}$$

Note that ignoring the mistrust—that is, setting $C = 0$—recovers AdaBoost's loss function (4.5).

ν-Arc uses the regularization parameter $\nu \in [0,1]$, which represents an upper bound on the fraction of margin errors. As with AdaBoost$_{\text{Reg}}$, setting $\nu = 0$ gives similar results as AdaBoost; setting $\nu = 1$, however, gives the bagging algorithm (Rätsch 2001; Rätsch et al. 2000). Thus, the averaging of diverse classifiers can be seen as a form of fully regularized boosting. Webb (2000) uses this property in his MultiBoosting algorithm, which uses bagging to create a diverse ensemble of AdaBoost classifiers. Paper V shows that this regularization property increases the performance of boosted, genetic programming classifiers.

Chapter 5

Contributions and further work

This thesis has investigated the use of genetic programming in string mining. The driving force behind the work has been the specialized search hardware described in Paper I. The query power and search speed of this hardware gave our unsupervised string mining algorithm a flexibility in the rule format that surpasses any other methods. The search hardware does, however, have restricted functionality. Consequently, the search expressions produced by our genetic programming approach can have a lower accuracy than those produced by other machine learning methods. We see this both in Paper IV and Paper V. This motivated combining boosting algorithms and genetic programming to create more accurate classifiers. Papers V, VI, and VII show that this combination can create classifiers that are better than are the classifiers created by other approaches such as support vector machines and neural networks.

The following section lists the main contributions of each paper; the chapter concludes with a section describing possible further work.

5.1 Contributions

Paper I This paper describes the theoretical background, design, and architecture of the specialized search hardware.

Paper II By directly optimizing a suitable measure of a rule's interestingness, we show that genetic programming can be used in unsupervised mining of accurate and interesting prediction rules.

Paper III We use multi-objective genetic programming to mine sets of prediction rules that represent a tradeoff between rule accuracy, comprehensibility, and interestingness.

Paper IV This paper summarizes and extends the work of Hetland and Sætrom on both supervised and unsupervised mining of prediction rules in time series (Hetland and Sætrom (2002, 2003a,b, 2004); Sætrom and Hetland (2003a,b); see also Hetland (2003)). More specifically, we extend our unsupervised algorithm such that the new algorithm can find prediction rules that relate the characteristics of two time series. The algorithm discovers both the antecedent, consequent, and the time series in which they occur.

Paper V The paper describes a boosted genetic programming algorithm and shows that this algorithm can create classifiers that predict the efficacy of short oligonucleotides used in antisense and RNA interference experiments. The classifiers' accuracy are comparable and slightly better than are the accuracy of classifiers created by other machine learning algorithms such as neural networks and support vector machines.

Paper VI We compare the accuracy of the short interfering RNA (siRNA) efficacy predictors from Paper V to the accuracy of several other recently published algorithms. This comparison shows that many of these algorithms have close to random performance and that only a few algorithms have a consistent high performance. More specifically, our boosted genetic programming predictors had the highest overall performance.

Paper VII Using a version of the boosted genetic programming algorithm from Paper V, we develop classifiers that predict non-coding RNA (ncRNA) genes in *Escherichia coli*. We experimentally test 16 of our predictions and verify that 12 of these are new ncRNA genes. Six of these new ncRNA genes have not previously been predicted by other methods.

5.2 Further work

Although we have shown that the boosted genetic programming algorithm can create very accurate classifiers, these classifiers are much more complex than are the expressions created by genetic programming. In other words, we have traded accuracy for comprehensibility. In some applications, where one is mainly interested in a highly accurate classifier, this may be fine. In other applications, one may be just as concerned with understanding the classifiers. An interesting continuation of this work would therefore be to try to build comprehensible expressions from the boosted genetic programming classifiers.

This work has described two approaches for mining strings and used these approaches to solve problems in a few application areas. We have, for example, only used the unsupervised rule mining algorithm to mine discretized time series, but it would also be interesting to investigate whether the algorithm can be used to mine DNA strings; for example, to find regulatory elements in DNA. Likewise, it would be interesting to investigate how well the boosted genetic programming approach can solve other string classification problems, such as remote protein family recognition.

The boosted genetic programming algorithm used here combines explicit regularization (AdaBoost$_{\mathrm{Reg}}$) with implicit regularization (averaging). It would be interesting to investigate how this approach compares to other regularized boosting algorithms; especially in the context of very skewed data sets.

Naturally, further work also remains on the application areas we have considered; for example, it would be interesting to investigate whether ncRNA gene prediction could be improved by integrating our boosted genetic programming classifiers with RNA secondary structure predictions and predictions of transcription initiation and termination.

Bibliography

Akbani, R., Kwek, S., and Japkowicz, N. (2004). Applying support vector machines to imbalanced datasets. In Boulicaut, J.-F., Esposito, F., Giannotti, F., and Pedreschi, D., editors, *ECML*, pages 39–50. Springer-Verlag.

Baldi, P., Brunak, S., Chauvin, Y., Andersen, C. A., and Nielsen, H. (2000). Assessing the accuracy of prediction algorithms for classification: an overview. *Bioinformatics*, 16(5):412–424.

Banzhaf, W., Nordin, P., Keller, R. E., and Francone, F. D. (1997). *Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann Publishers, San Francisco, CA.

Bartel, D. P. (2004). MicroRNAs: genomics, biogenesis, mechanism, and function. *Cell*, 116(2):281–297.

Bauer, E. and Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Mach. Learn.*, 36(1–2):105–139.

Blickle, T. and Thiele, L. (1995). A comparison of selection schemes used in genetic algorithms. Technical Report 11, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH), Gloriastrasse 35, 8092 Zurich, Switzerland.

Blum, C. and Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308.

Breiman, L. (1996a). Bagging predictors. *Mach. Learn.*, 24(2):123–140.

Breiman, L. (1996b). Bias, variance, and arcing classifiers. Technical Report 460, Statistics Department, University of California, Berkeley, CA.

Das, G., Lin, K., Mannila, H., Renganathan, G., and Smyth, P. (1998). Rule discovery from time series. In *Knowledge Discovery and Data Mining*, pages 16–22.

Dietterich, T. G. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Mach. Learn.*, 40(2):139–157.

Eddy, S. R. (2001). Non-coding RNA genes and the modern RNA world. *Nat. Rev. Genet.*, 2(12):919–929.

Efron, B. and Tibshirani, R. J. (1993). *An Introduction to the Bootstrap*. Chapman & Hall, New York.

Forman, G. and Cohen, I. (2004). Learning from little: Comparison of classifiers given little training. In *PKDD*, pages 161–172.

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

Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. System Sci.*, 55(1):119–139.

Friedman, J. H. (1997). On bias, variance, 0/1-loss, and the curse-of-dimensionality. *Data Mining Knowl. Discov.*, 1(1):55–77.

Geman, S., Bienenstock, E., and Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Comput.*, 4(1):1–58.

Geman, S. and Geman, D. (1984). Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Trans. Pattern Anal. Machine Intell.*, 6(6):721–741.

Goldberg, D. E. and Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the 2nd International Conference on Genetic Algorithms*, pages 41–49.

Grotmol, Ø. (2002). Species selection. Algorithm developed when working for Interagon AS.

Gusfield, D. (1997). *Algorithms on Strings, Trees, and Sequences : Computer science and computational biology*. Cambridge University Press, Cambridge, UK.

Halaas, A., Svingen, B., Nedland, M., Sætrom, P., Snøve Jr., O., and Birkeland, O. R. (2004). A recursive MISD architecture for pattern matching. *IEEE Trans. on VLSI Syst.*, 12(7):727–734.

Hansen, L. K. and Salamon, P. (1990). Neural network ensembles. *IEEE Trans. Pattern Anal. Machine Intell.*, 12(10):993–1001.

Heddad, A., Brameier, M., and MacCallum, R. M. (2004). Evolving regular expression-based sequence classifiers for protein nuclear localisation. In Raidl, G. R., Cagnoni, S., Branke, J., Corne, D., Drechsler, R., Jin, Y., Johnson, C. G., Machado, P., Marchiori, E., Rothlauf, F., Smith, G. D., and Squillero, G., editors, *EvoWorkshops*, volume 3005 of *Lecture Notes in Computer Science*, pages 31–40. Springer-Verlag.

Hetland, M. L. (2003). *Evolving Sequence Rules*. PhD thesis, Norwegian University of Science and Technology.

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

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

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

Hetland, M. L. and Sætrom, P. (2004). Temporal rule discovery using genetic programming and specialized hardware. In Lotfi, A. and Garibaldi, J. M., editors, *Applications and Science in Soft Computing*, Advances in Soft Computing, pages 87–94. Springer-Verlag. Revised version of Hetland and Sætrom (2002).

Hetland, M. L. and Sætrom, P. (2005). Evolutionary rule mining in time series databases. *Mach. Learn.*, 58(2–3):107–125.

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

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI.

Höppner, F. and Klawonn, F. (2001). Finding informative rules in interval sequences. In Hoffmann, F., Hand, D. J., Adams, N. M., Fisher, D. H., and Guimarães, G., editors, *IDA*, volume 2189 of *Lecture Notes in Computer Science*, pages 125–134. Springer-Verlag.

Howard, D. and Benson, K. (2003). Promoter prediction with a GP-automaton. In Raidl, G. R., Meyer, J.-A., Middendorf, M., Cagnoni, S., Cardalda, J. J. R., Corne, D., Gottlieb, J., Guillot, A., Hart, E., Johnson, C. G., and Marchiori, E., editors, *EvoWorkshops*, volume 2611 of *Lecture Notes in Computer Science*, pages 44–53. Springer-Verlag.

Hutter, M. (2002). Fitness uniform selection to preserve genetic diversity. In *CEC-2002*, volume 1, pages 783–788.

Japkowicz, N. (2000). The class imbalance problem: Significance and strategies. In *Proceedings of the 2000 International Conference on Artificial Intelligence (IC-AI'2000)*, volume 1, pages 111–117.

Jonassen, I., Collins, J. F., and Higgins, D. G. (1995). Finding flexible patterns in unaligned protein sequences. *Protein Sci.*, 4(8):1587–1595.

Kearns, M. and Valiant, L. (1994). Cryptographic limitations on learning boolean formulae and finite automata. *J. ACM*, 41(1):67–95.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

Knuth, D. E. (1964). Backus normal form vs. Backus Naur form. *Commun. ACM*, 7(12):735–736.

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

Koza, J. R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts.

Koza, J. R., David Andre, Bennett III, F. H., and Keane, M. (1999). *Genetic Programming 3: Darwinian Invention and Problem Solving*. Morgan Kaufmann Publishers, San Fransisco, CA.

Koza, J. R., Keane, M. A., Streeter, M. J., Mydlowec, W., Yu, J., and Lanza, G. (2003). *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers.

Lagos-Quintana, M., Rauhut, R., Lendeckel, W., and Tuschl, T. (2001). Identification of novel genes coding for small expressed RNAs. *Science*, 294(5543):853–858.

Lau, N. C., Lim, L. P., Weinstein, E. G., and Bartel, D. P. (2001). An abundant class of tiny RNAs with probable regulatory roles in *Caenorhabditis elegans*. *Science*, 294(5543):858–862.

Lawrence, S. and Giles, C. L. (1998). Searching the World Wide Web. *Science*, 280(5360):98–100.

Lee, R. C. and Ambros, V. (2001). An extensive class of small RNAs in *Caenorhabditis elegans*. *Science*, 294(5543):862–864.

Lee, R. C., Feinbaum, R., and Ambros, V. (1993). The *C. elegans* heterochronic gene *lin-4* encodes small RNAs with antisense complementarity to *lin-14*. *Cell*, 75(5):843–854.

Lewin, B. (2000). *Genes VII*. Oxford University Press, Oxford, UK.

Lim, L. P., Glasner, M. E., Yekta, S., Burge, C. B., and Bartel, D. P. (2003). Vertebrate microRNA genes. *Science*, 299(5612):1540.

Mahfoud, S. W. (1995). *Niching methods for genetic algorithms*. IlliGAL report no. 95001, University of Illinois at Urbana-Champaign.

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

Martin, J. K. and Hirschberg, D. S. (1996). Small sample statistics for classification error rates I: Error rate measurements. Technical Report 96-21, ICS Dept., UC Irvine.

Meir, R. and Rätsch, G. (2003). An introduction to boosting and leveraging. In Mendelson, S. and Smola, A., editors, *Advanced Lectures on Machine Learning*, volume 2600, pages 118–183. Springer-Verlag.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equations of state calculations by fast computing machines. *J. of Comp. Phys.*, 21(6):1087–1091.

Montana, D. J. (1995). Strongly typed genetic programming. *Evol. Comput.*, 3(2):199–230.

Moss, E. G., Lee, R. C., and Ambros, V. (1997). The cold shock domain protein LIN-28 controls developmental timing in C. elegans and is regulated by the *lin-4* RNA. *Cell*, 88(5):637–646.

Müller, K.-R., Mika, S., Rätsch, G., and Tsuda, K. (2001). An introduction to kernel-based learning algorithms. *IEEE Trans. Neural Networks*, 12(2):181–201.

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

Navarro, G. (2004). Pattern matching. *J. Appl. Stat.*, 31(8):925–949.

Navarro, G. and Raffinot, M. (2002). *Flexible pattern matching in strings: practical on-line search algorithms for texts and biological sequences*. Cambridge University Press, Cambridge, UK.

NCBI News (2004). Exponential growth of GenBank continues with release 142. [Online]. Available at `http://www.ncbi.nlm.nih.gov/Web/Newsltr/Spring04/gbrel.html`.

Nedland, M., Svingen, B., and Hetland, M. L. (2002). The Interagon Query Language – a reference guide. Available on request: `info@interagon.com`.

Olsen, P. H. and Ambros, V. (1999). The *lin-4* regulatory RNA controls developmental timing in *Caenorhabditis elegans* by blocking LIN-14 protein synthesis after the initiation of translation. *Dev. Biol.*, 216(2):671–680.

O'Neill, M. and Ryan, C. (2003). *Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language*, volume 4 of *Genetic programming*. Kluwer Academic Publishers, Dordrecht, The Netherlands.

Perrone, M. P. (1993). *Improving Regression Estimation: Averaging Methods for Variance Reduction with Extensions to General Convex Measure Optimization*. PhD thesis, Department of Physics, Brown University.

Prechelt, L. (1998). Automatic early stopping using cross validation: quantifying the criteria. *Neural Netw.*, 11(4):761–767.

Rätsch, G. (2001). *Robust Boosting via Convex Optimization: Theory and Applications*. PhD thesis, University of Potsdam. Chapter 4.

Rätsch, G., Onoda, T., and Müller, K.-R. (2001). Soft margins for AdaBoost. *Mach. Learn.*, 42(3):287–320.

Rätsch, G., Schökopf, B., Smola, A., Müller, K.-R., Onoda, T., and Mika, S. (2000). ν-arc: Ensemble learning in the presence of outliers. In Kearns, M. S., Solla, S. A., and Cohn, D. A., editors, *Advances in Neural Information Processing Systems 12: Proc. of NIPS'99*. MIT Press.

Rätsch, G., Schölkopf, B., Smola, A. J., Mika, S., Onoda, T., and Müller, K.-R. (2000). Robust ensemble learning. In Smola, A. J., Bartlett, P. L., Schölkopf, B., and Schuurmans, D., editors, *Advances in Large Margin Classifiers*, pages 207–220. The MIT Press, Cambridge, Massachusetts.

Reed, R. (1993). Pruning algorithms—a survey. *IEEE Trans. Neural Networks*, 4(5):740–747.

Risvik, K. M. (2004). *Scaling Internet Search Engines: Methods and Analysis*. PhD thesis, NTNU, Trondheim, Norway.

Sætrom, P. (2004). Predicting the efficacy of short oligonucleotides in antisense and RNAi experiments with boosted genetic programming. *Bioinformatics*, 20(17):3055–3063.

Sætrom, P. and Hetland, M. L. (2003a). Multiobjective evolution of temporal rules. In *Eight Scandinavian Conference on Artificial Intelligence*.

Sætrom, P. and Hetland, M. L. (2003b). Unsupervised temporal rule mining with genetic programming and specialized hardware. In *Proceedings of the International Conference on Machine Learning and Applications (ICMLA'03)*, pages 145–151.

Sætrom, P., Sneve, R., Kristiansen, K. I., Snøve Jr., O., Grünfeld, T., Rognes, T., and Seeberg, E. (2005). Predicting non-coding RNA genes in *Escherichia coli* with boosted genetic programming. *Nucleic Acids Res.*, 33(10):3263–3270.

Sætrom, P. and Snøve Jr., O. (2004). A comparison of siRNA efficacy predictors. *Biochem. Biophys. Res. Commun.*, 321(1):247–253.

Salzberg, S. (1997). On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining Knowl. Discov.*, 1(3):317–328.

Schapire, R. E. (1990). The strength of weak learnability. *Mach. Learn.*, 5(2):197–227.

Schapire, R. E., Freund, Y., Bartlett, P., and Lee, W. S. (1998). Boosting the margin: a new explanation for the effectiveness of voting methods. *Ann. Stat.*, 26(5):1651–1686.

Scherer, L. J. and Rossi, J. J. (2003). Approaches for the sequence-specific knock-down of mRNA. *Nat. Biotechnol.*, 21(12):1457–1465.

Schmitt, L. M. (2001). Theory of genetic algorithms. *Theor. Comput. Sci.*, 259(1–2):1–61.

Schmitt, L. M. (2004). Theory of genetic algorithms II: models for genetic operators over the string-tensor representation of populations and convergence to global optima for arbitrary fitness function under scaling. *Theor. Comput. Sci.*, 310(1–3):181–231.

Schölkopf, B., Smola, A. J., Williamson, R., and Bartlett, P. L. (2000). New support vector algorithms. *Neural Comput.*, 12(5):1207–1245.

Shawe-Taylor, J. and Cristianini, N. (2004). *Kernel Methods for Pattern Analysis*. Cambridge University Press, Cambridge, UK.

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

Snøve Jr., O. (2005). *Analysis of RNAi intermediates using the pattern matching chip*. PhD thesis, Norwegian University of Science and Technology.

Snøve Jr., O., Nedland, M., Fjeldstad, S. H., Humberset, H., Birkeland, O. R., Grünfeld, T., and Sætrom, P. (2004). Designing effective siRNAs with off-target control. *Biochem. Biophys. Res. Commun.*, 325(3):769–773.

Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(2):111–147.

Vanet, A., Marsan, L., and Sagot, M.-F. (1999). Promoter sequences and algorithmical methods for identifying them. *Res. Microbiol.*

Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag, N.Y.

Vapnik, V. N. (1998). *Statistical Learning Theory*. Wiley-Interscience, New York, NY, USA.

Webb, G. I. (2000). Multiboosting: A technique for combining boosting and wagging. *Mach. Learn.*, 40(2):159–196.

Weiss, G. M. and Provost, F. (2001). The effect of class distribution on classifier learning. Technical Report ML-TR 43, Department of Computer Science, Rutgers University.

Wightman, B., Ha, I., and Ruvkun, G. (1993). Posttranscriptional regulation of the heterochronic gene *lin-14* by *lin-4* mediates temporal pattern formation in *C. elegans*. *Cell*, 75(5):855–862.

Witten, I. H., Moffat, A., and Bell, T. C. (1999). *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishers, Los Altos, CA 94022, USA, 2nd edition.

Wolpert, D. H. (1997). On bias plus variance. *Neural Comput.*, 9(6):1211–1243.

Wu, G. and Chang, E. Y. (2003). Adaptive feature-space conformal transformation for imbalanced-data learning. In *ICML*, pages 816–823.

Yang, J., Wang, W., and Yu, P. S. (2003). Mining asynchronous periodic patterns in time series data. *IEEE Trans. Knowl. Data Eng.*

Yekta, S., Shih, I., and Bartel, D. P. (2004). MicroRNA-directed cleavage of *HOXB8* mRNA. *Science*, 304(5670):594–596.

Zeng, Y., Wagner, E. J., and Cullen, B. R. (2002). Both natural and designed micro RNAs can inhibit the expression of cognate mRNA when expressed in human cells. *Mol. Cell.*, 9(6):1327–1333.

Papers

Paper II

Unsupervised temporal rule mining with genetic programming and specialized hardware

Unsupervised Temporal Rule Mining with Genetic Programming and Specialized Hardware

Pål Sætrom and Magnus Lie Hetland

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

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

I. Introduction

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

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

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

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

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

A. Related Work

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

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

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

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

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

B. Structure of This Paper

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

II. Preprocessing

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

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

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

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

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

III. Evolving Rules

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

A. Rule Languages

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

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

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

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

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

B. Rule Representation

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

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

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

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

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

C. Confidence, Support, and Interestingness

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

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

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

The *support* of a rule is defined as:

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

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

The *confidence* of a rule is defined as:

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

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

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

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

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

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

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

with the same definitions for the probabilities as for the J-measure. As can be seen from (5), if R_c^t and R_a are statistically independent then $\textit{RI} = 0$. 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 $\textit{RI} > 0$ ($\textit{RI} < 0$).

IV. Rule Evaluation

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

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

This amounts to counting the following:

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

A. Handling Correlations Caused by the Discretization Method

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

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

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

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

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

B. Counting Hits

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

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

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

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

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

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

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

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

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

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

V. Experimental Results

In our experiments we used the following five rule languages:

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

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

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

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

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

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

A. Synthetic Data

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

Two different antecedent types were used:

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

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

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

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

B. Modifying the J-measure

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

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

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

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

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

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

C. Real-World Data

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

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

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

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

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

D. Random Data

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

TABLE I

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

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

TABLE III

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

Language	Rule	Supp.	Conf.	J-mea.	*RI*
$L_2 \overset{10}{\Rightarrow} L_2$	$\mathrm{b} \overset{1}{\Rightarrow} \mathrm{b}$	0.042	0.68	0.12	0.68
$L_4 \overset{10}{\Rightarrow} L_2$	$\mathrm{n^{+}fhjjcg^{+}jc} \mid \mathrm{n^{+}fhjng^{+}jc} \mid \mathrm{n^{+}fhng^{+}jc} \mid \mathrm{ac} \mid \mathrm{oc} \mid \mathrm{o} \overset{1}{\Rightarrow} \mathrm{o}$	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

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

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

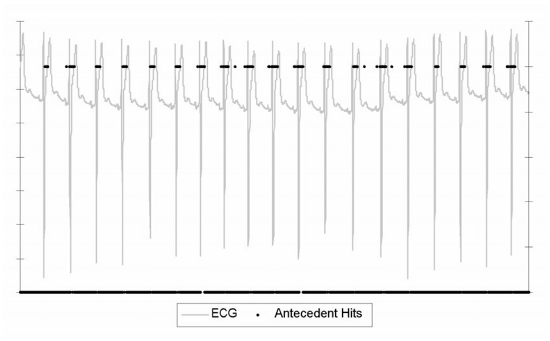

Fig. 1. Hit locations of antecedent in ECG sequence.

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

Several rule languages were tried. This showed that certain language combinations for the antecedent and consequent may result in spurious rules that fit the random data (including the separate test set) well. For example, the language $L_5 \overset{10}{\Rightarrow} L_5$ (with $w = 10$), generated the following rule: $!(\mathrm{e}(\leq \mathrm{i})\mathrm{yk} \mid \mathrm{kye}(\leq \mathrm{i})) \overset{1}{\Rightarrow} !((\leq \mathrm{i})(> \mathrm{i}))$. 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.

VI. SUMMARY AND CONCLUSIONS

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

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

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

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

Appendix
Rule Language Syntax

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

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

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

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

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

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

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

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

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

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

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Paper III

Multiobjective evolution of temporal rules

Multiobjective Evolution of Temporal Rules

Pål SÆTROM[1] and Magnus LIE HETLAND[2]
[1]*Interagon AS, Medisinsk-teknisk senter,*
NO–7489 Trondheim, Norway
paalsat@interagon.com
[2]*Norwegian University of Science and Technology,*
Dept. of Computer and Information Science,
Sem Sælands vei 9, NO–7491 Trondheim, Norway
magnus@hetland.org

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*. In this paper we propose an alternative to constructing an *ad hoc* aggregate fitness function: using well-established multiobjective evolutionary algorithms to evolve a Pareto optimal set of rules. We apply the method to several real-world data sets and demonstrate how the method is able to evolve a varied set of rules that explore different aspects of the time series in question.

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 [1, 2]. 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 [3], combined with the temporal rule mining approach described in [2].

2 Method

In [2] 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 [2] and [3] will be outlined. In Sect. 2.1 we give a brief description of our time series preprocessing. Following that, in Sect. 2.2 the rule format and internal rule representation used by the algorithm is described. Section 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 2.4 describes the objective functions used by the MOGP algorithm. Finally, Sect. 2.5 briefly outlines how the objective functions are evaluated.

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. 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 [2].

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 [4] to relatively complex pattern languages [2].

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 $r_T \bmod T + 1$.

The syntax of the sample rules in Sect. 3 is based on the IQL language [5] (see Sect. 3), with certain minor extensions for typographical convenience. The specifics of this syntax is not essential for this paper; details may be found in [2].

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* [6].

Assuming that the functions in F should be maximized, then a solution $\mathbf{x}$ is *Pareto optimal* if no other solution $\mathbf{x}'$ exists such that $f(\mathbf{x}') \geq f(\mathbf{x})$ for all $f \in F$ and $f(\mathbf{x}') > f(\mathbf{x})$ 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 $f(\mathbf{x}') \geq f(\mathbf{x})$ for all $f \in F$ and $f(\mathbf{x}') > f(\mathbf{x})$ 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 [6] for an introduction and [7] for a survey).

The algorithm used here is based on the SPEA2 [3], 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 [3] for further details.

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

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

2.4 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 [1]. 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.

2.4.1 Accuracy

Given a rule $R = R_a \overset{t}{\Rightarrow} R_c$ in the rule language $L_a \overset{T}{\Rightarrow} L_c$ (such that $t \leq T$ – see Sect. 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

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

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:

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

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:

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

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.

2.4.2 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 [8] for a survey). The J-measure ([9]), is one particular measure which has already been proven useful in mining time series. This is defined as

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

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).

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

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

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

2.5 Rule Evaluation

As in [2], a special purpose search chip [10, 11] 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. 2.2).

3 Experiments

The MOGP algorithm was tested on four data sets from the UCR Time Series Data Mining Archive [12]: ECG measurements from several subjects, concatenated; Earthquake-related seismic data; Monthly mean sunspot numbers from 1749 until 1990; and Network traffic as measured by packet round trip time delays. Figure 1 shows plots of sub-sequences of the different time series analyzed.

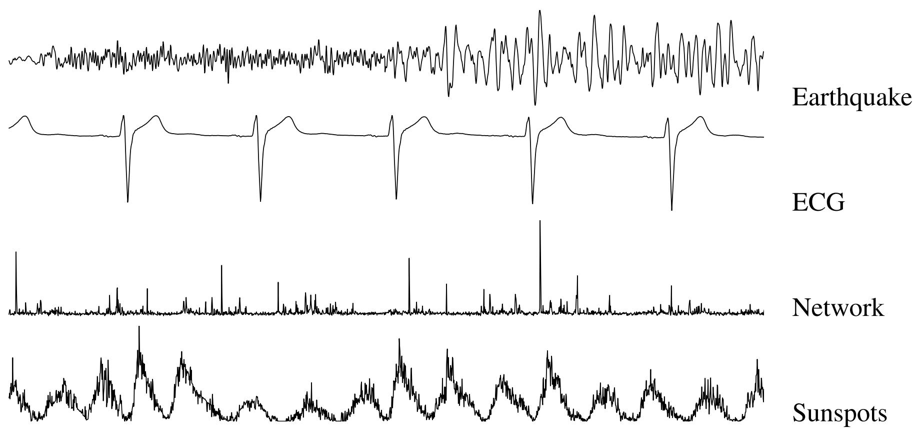

Figure 1: The time series analyzed.

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

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

We performed four sets of experiments. First, the four time series were mined using the four objective functions from section 2.4: support (2), confidence (3), J-measure (4), and rule complexity (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 (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 [5] as a template for generating rules. The antecedents were IQL expressions, while the consequents were single characters or concatenations thereof.

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

Table 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 2.

Table 1 and Figure 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 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 1 in Figure 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]

[1]Note, however, that the positions where the antecedent of the fifth rule in Table 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.

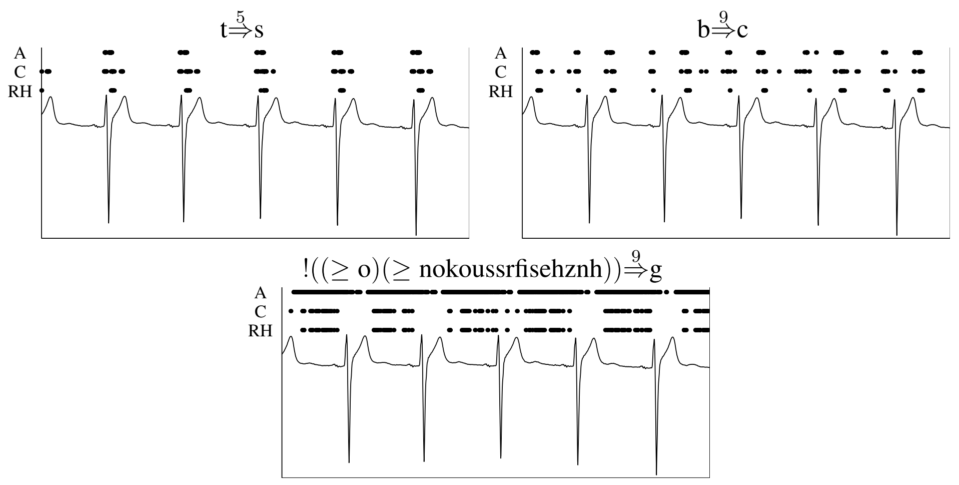

Figure 2: Hit locations in a subsequence of the ECG series of selected rules from Table 1. The dots following the *A* and *C* labels on the y-axis indicate the hit locations of the antecedent and consequent, respectively. The dots following the *RH* label indicate the positions where the rule hits, that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

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

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

To conclude this section, Figure 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.

3.2 Using Confidence, J-measure, and Rule Complexity

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

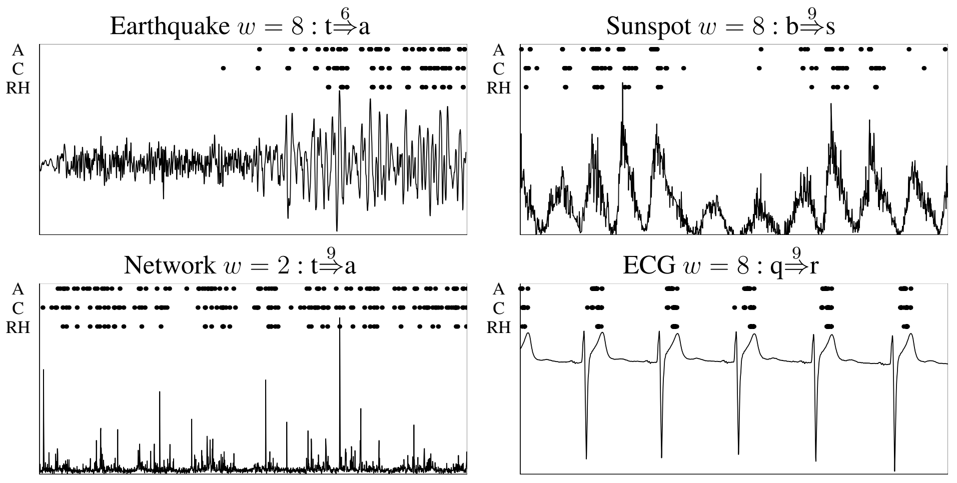

Figure 3: Hit locations of selected results on different sequences (see Figure 2 for the definitions of *A*, *C*, and *RH*).

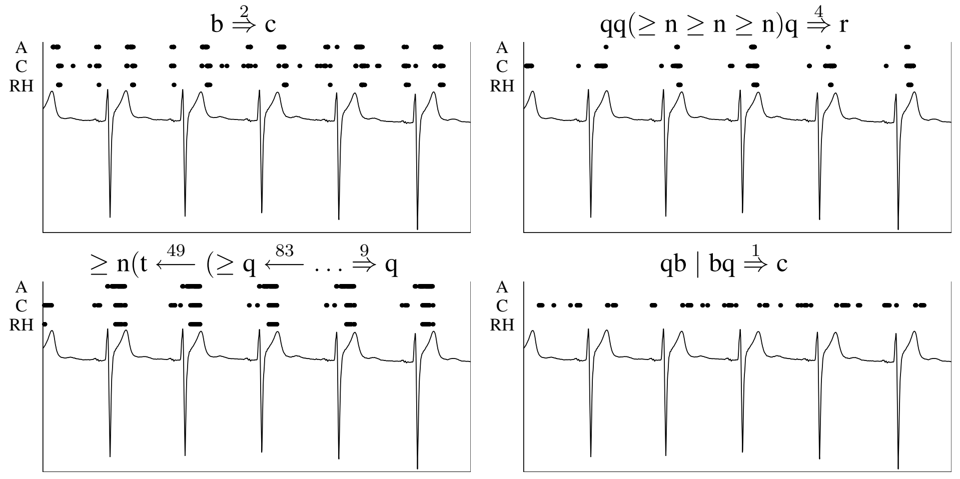

Figure 4: Hit locations in a subsequence of the ECG series of the rules from Table 2 (see Figure 2 for the definitions of *A*, *C*, and *RH*).

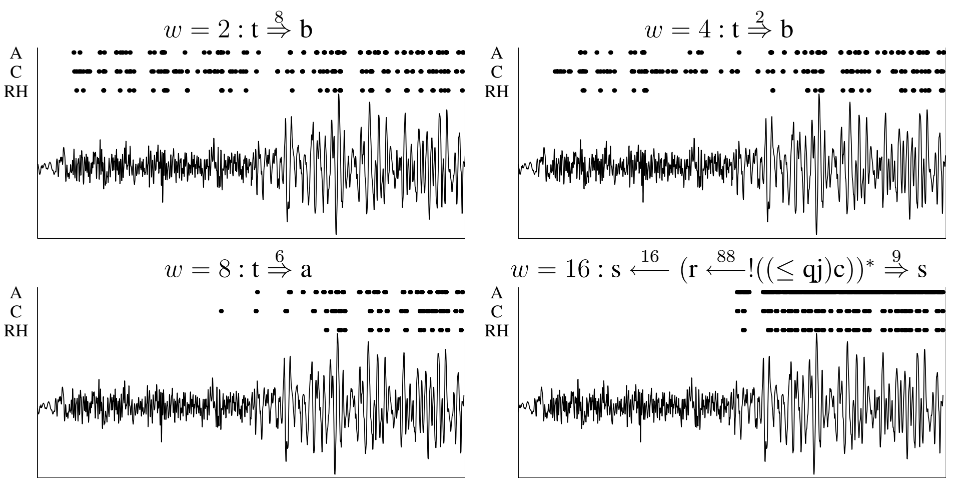

Figure 5: Rules generated from the earthquake series with increasing window sizes (see Figure 2 for the definitions of *A*, *C*, and *RH*).

Rules like these were, however, also generated when the support was optimized directly (data not shown).

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 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.

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 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 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 2.5). In addition, large window sizes represent more long term trends in the data (see section 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.

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 [2], 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 avoids diversity reduction, and leaves the task of evaluating the trade-offs between the different objectives to the human user.

References

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

[2] Pål Sætrom and Magnus Lie Hetland. Unsupervised temporal rule mining with genetic programming and specialized hardware. In *Proceedings of the International Conference on Machine Learning and Applications (ICMLA'03)*, pages 145–151, 2003.

[3] Eckart Zitzler, Marco Laumanns, and Lothar 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.

[4] G. Das, K. Lin, H. Mannila, G. Renganathan, and P. Smyth. Rule discovery from time series. In *Knowledge Discovery and Data Mining*, pages 16–22, 1998.

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

[6] Carlos A. Coello Coello. A short tutorial on evolutionary multiobjective optimization. In *First International Conference on Evolutionary Multi-Criterion Optimization*, pages 21–40. Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001.

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

[8] 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, October 1999.

[9] 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. MIT Press, Cambridge, MA, 1991.

[10] Fast Search & Transfer ASA. Digital processing device. PCT/NO99/00308, Apr 2000.

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

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

Paper VI

A comparison of siRNA efficacy predictors

Available online at www.sciencedirect.com

SCIENCE DIRECT®

Biochemical and Biophysical Research Communications 321 (2004) 247–253

BBRC

www.elsevier.com/locate/ybbrc

A comparison of siRNA efficacy predictors

Pål Sætrom, Ola Snøve Jr. *

Interagon AS, Medisinsk teknisk senter, NO-7489 Trondheim, Norway

Received 18 June 2004

Abstract

Short interfering RNA (siRNA) efficacy prediction algorithms aim to increase the probability of selecting target sites that are applicable for gene silencing by RNA interference. Many algorithms have been published recently, and they base their predictions on such different features as duplex stability, sequence characteristics, mRNA secondary structure, and target site uniqueness. We compare the performance of the algorithms on a collection of publicly available siRNAs. First, we show that our regularized genetic programming algorithm GPboost appears to have a higher and more stable performance than other algorithms on the collected datasets. Second, several algorithms gave close to random classification on unseen data, and only GPboost and three other algorithms have a reasonably high and stable performance on all parts of the dataset. Third, the results indicate that the siRNAs' sequence is sufficient input to siRNA efficacy algorithms, and that other features that have been suggested to be important may be indirectly captured by the sequence.
© 2004 Elsevier Inc. All rights reserved.

Keywords: siRNA; RNA interference; Efficacy prediction

RNA interference (RNAi) is a cellular process for sequence-specific depletion of mRNA [1]. Long double-stranded RNA duplexes or hairpin precursors are cleaved into short fragments by a ribonuclease III enzyme called Dicer. The resulting short interfering RNAs (siRNAs) are 21–23 nucleotides (nt) long and have characteristic 2 nt 3′ overhangs [2]. A ribonucleoprotein complex named RNA induced silencing complex (RISC) incorporates one of the siRNA strands, and cleaves mRNA with complementarity to the RNA component in an ATP-independent reaction [3]. Long RNA duplexes trigger the interferon response and yield non-specific degradation of mRNA when introduced into mammalian cells. The interferon response can, however, be circumvented by transfecting moderate concentrations of synthetic siRNAs into mammalian cells [4]. The knockdown effect is transient and diminishes after a few cell cycles [5]. A lasting knockdown effect can be obtained by endogenous transcription of hairpin precursors from vector [6] or virus-based [7] systems.

Several excellent reviews describe siRNA and RNAi [8–11].

The siRNAs must be optimized with respect to toxicity, specificity, and efficacy. First, both synthetic and endogenously transcribed siRNAs have been shown to induce the interferon response in a concentration-dependent manner [12–14]. Second, there is a risk that the siRNA may guide RISC to cleave mRNAs with sequence similarity to the target (shown indirectly in [15]) or that the siRNA may function as a microRNA and suppress protein translation [16]. Third, only a fraction of all siRNAs are effective at reducing the expression of their target genes, and two siRNAs that target mRNA sites that are separated by only a few nucleotides may have very different efficacies [5].

Genomewide specificity studies on the mRNA level have been published but the results are conflicting [14,17–19] and siRNAs' mismatch tolerance remains an open question. It seems clear, however, that central mismatches between the siRNA and the target mRNA

* Corresponding author. Fax: +47-23-01-12-35.
E-mail addresses: paal.saetrom@interagon.com (P. Sætrom), ola.snove@interagon.com (O. Snøve Jr.).

0006-291X/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.bbrc.2004.06.116

are more likely to abolish silencing than mismatches at the ends, and that the tolerance for mismatches is higher at the 5′ end than at the 3′ end of the siRNA [15,20]. Very specific target sites are available for most genes but many published siRNAs have a flawed design and therefore risk off-target effects [21].

Algorithms that predict siRNA efficacy increase the probability for obtaining an siRNA that induces effective silencing of the desired gene. The Tuschl rules [22] were the only criteria available until Reynolds et al. [23] published their algorithm for rational design of effective siRNAs. Several other algorithms have emerged since [24–30]. We recently used a hardware accelerated [31] regularized genetic programming algorithm to develop siRNA efficacy classifiers [32]. We aim to provide a comparison of the algorithms' performance on a large collection of publicly available functionally validated siRNAs.

Materials and methods

Sequence data

We collected a non-redundant database of functionally validated siRNAs from seven publications [20,23–25,27,33,34]. The database contains 581 siRNAs that target 40 genes. Detailed information about the siRNAs, target genes, and the assays that were used when the siRNAs were validated is in Supplementary Table ST1. Note that the database is biased in that the selection of target genes and siRNAs has not been random in the works in which they were published. For example, Hsieh et al. [27] select siRNAs that comply with the Tuschl rules in addition to other criteria. Note also that the database contains fewer siRNAs with intermediate efficacies than would be expected if the selection was random. Moreover, one has to expect that there is considerable noise in the data due to (i) a variety of assays for measurement of siRNA efficacy; (ii) very different concentrations of siRNAs; and (iii) sub-optimal time intervals between transfection and down-regulation measurement. We aimed to limit the heterogeneity of the siRNA database; therefore, we included only datasets of a certain size with respect to either targets or siRNAs.

Algorithms

Both strands of the siRNA can potentially be absorbed by RISC to guide mRNA cleavage. The findings of Schwarz et al. [35] and Khvorova et al. [34] that RISC prefers the uptake of one strand based on the thermodynamic stability of an siRNA duplex provided a new criterion for design of effective siRNAs: The siRNA's thermodynamic properties must be such that the RISC prefers the incorporation of the strand that is complementary to the intended target site.

For the most part, siRNA efficacy prediction algorithms have been constructed by investigating single-base frequencies in relatively small datasets containing effective and ineffective siRNAs. Any statistically significant single-base correlations with efficacy, either positive or negative, are used to construct scoring algorithms [23–25,27,30]. (Note that Ui-Tei et al. [25] and Hsieh et al. [27] do not explicitly construct scoring algorithms in their papers. The sequence criteria that they do suggest, however, can easily be used to construct such an algorithm.)

Many authors have hypothesized that the accessibility of the mRNA target site determines siRNA efficacy as is the case for antisense DNA technologies. There are conflicting reports on whether target accessibility is a determinant for siRNA efficacy [26,36]. The differing results may be due to unreliable in silico secondary structure predictions or small and biased datasets. Luo and Chang [26] recently proposed an algorithm that predicts siRNA efficacy based on the target site's secondary structure.

Pancoska et al. [28] speculate that a sequence segment's uniqueness compared with the rest of the targeted mRNA and the duplex melting temperature determines the efficacy of an siRNA targeting that particular site. Unfortunately, it was not possible to reproduce their algorithm from the original publication, and we therefore decided to omit the algorithm from our comparisons.

We recently used a regularized genetic programming approach to obtain patterns that discriminated between effective and ineffective siRNAs [32]. We hypothesized that complex sequence patterns can capture all the information necessary to predict the efficacy of siRNAs and constructed classifiers whose score is a weighted sum of many patterns (see [32] for details).

Table 1 shows an overview of the features that the design algorithms rely on to make an efficacy prediction. Note that the thermodynamic stability of an RNA duplex is calculated from its sequence composition [37]. Table 2 shows how various algorithms score an siRNA based on individual nucleotides. For example, Reynolds 1 + 2 adds one to the score if the second sense strand nucleotide is adenine, whereas they subtract one if the fifteenth nucleotide is guanine. Note that many of the algorithms that are based on sequence characteristics prefer certain bases at the ends of the siRNA, which is probably because it yields the right difference between the 5′ and 3′ thermodynamic duplex stability. Reynolds 1 + 2 also adds one to the score if the siRNA's GC-content is between 30% and 50%. In addition to the single-base scores in Table 2, Ui-Tei counts the number of AU- and GC-pairs in positions 13–19, and adds one, respectively, subtracts one from the score if there are five or more AU- or five or more GC-pairs. Moreover, stretches of nine or more GC-pairs are considered negative and one is subtracted from the score, whereas one is added to the score if no such stretches are present.

Implementation details

Reynolds 1. We use the mfold web server [38] instead of the Oligo 6.0 software to predict the siRNA antisense melting temperature. We use a cutoff of 57 °C, as this both best mirrors previous results [23] and gives the highest absolute correlation on the Reynolds training data ($r = -0.14$).

Reynolds 2. This is the algorithm of Reynolds et al. [23] without the hairpin melting temperature scoring.

Table 1
There are important differences between the siRNA design algorithms

Algorithm	Citation	Description
GPboost	[32]	Weighted sum of sequence motifs/patterns
Ui-Tei	[25]	Sequence features
Amarzguioui	[24]	Sequence features
Hsieh	[27]	Sequence features
Takasaki	[30]	Sequence features
Reynolds 1	[23]	Hairpin potential, sequence features
Reynolds 2	[23]	Sequence features
Schwarz	[35]	Difference between 3′ and 5′ stability
Khvorova	[34]	Duplex stability profile
Stockholm 1	[29]	Energy features
Stockholm 2	[29]	Energy features
Tree	[29]	Sequence features in decision tree
Luo	[26]	mRNA secondary structure features

See Implementation details for additional information on the different algorithms.

Table 2
Sequence characteristics used by different algorithms

Algorithm	siRNA sense strand position																																	
	1				2		3		6		7		8		9		10	11		13		15		16			17		18		19			
	A	C	G	U	A	U	A	U	A	C	G	U	A	G	G	U	U	C	G	A	G	A	U	A	G	U	A	U	A	U	A	C	G	U
Reynolds 1 and 2							1										1				−1	1	1	1		1	1	1	1	1	2	−1	−1	1
Ui-Tei	−1	1	1	−1																											1	−1	−1	1
Amarzguioui	−1	1	1	−2	−1	−1	−1	−1	1																		1	1	1	1	2		−1	2
Hsieh										−1								1	1	1					1								−1	1
Takasaki	−3.97		7.4	−3.75					2.33		2.4	−2.59	3.02	−2.35	−2.35	2.3							2.7										−2	

Schwarz. We compute the duplex stability [37] for the four first nucleotides in the antisense and sense strands and use the difference as the classification score.

Khvorova. This algorithm creates two average internal stability profiles from a set of training sequences—one for effective siRNAs and another for ineffective siRNAs. Then, a siRNA's score is the difference of the correlation between its internal stability profile and the average effective and average ineffective siRNA profiles. The internal stability profile is found by computing the duplex stability [34] for each pentamer in the sequence.

Stockholm 1. This is our implementation of the Stockholm rules as described in [29]. We use the mfold web server [38] to predict the total hairpin energy and the nearest neighbor parameters of Xia et al. [37] for duplex stability calculations.

Stockholm 2. This is the modified Stockholm rules from the web server of Chalk et al. [29] (http://sisearch.cgb.ki.se/). In our experiments, we ran the prediction server with as few restrictions as possible, but some of the siRNAs in our database were still not evaluated. The web server missed about the same percentage of effective and ineffective siRNAs.

Tree. This is the decision tree score from the web server of Chalk et al. [29], with the low, moderate, and high categories mapped to 0, 1, and 2.

Comparing algorithms

We use the correlation between classifier output and siRNA efficacy, and ROC analysis to measure the performance of the different classifiers (see [39] for a review). The correlation R measures the classifier's overall performance: R^2 represents the proportion of variation in the observed efficacy that can be explained by the classifier. A Student's t test gives the statistical significance of a given correlation.

ROC analysis requires that all siRNAs are classified as either effective or ineffective, typically by using a cutoff on the measured siRNA efficacy. Given such a classification, a prediction made by a classifier can be either a true positive, a false positive, a true negative, or a false negative. That is, an effective siRNA will either be a true positive or a false negative prediction depending on what cutoff the classifier uses to signal positive predictions.

A ROC-curve is constructed by varying the classifier's positive cutoff and plotting the relative number of true positives and false positives identified by the classifier at each cutoff. This shows the classifier's sensitivity Se for varying levels of specificity Sp, as the relative number of false positives is $1 - Sp$. The ROC-score is the area under the ROC-curve and can be used to characterize a classifier's performance. Perfect classifiers identify all true positives before returning the false positives and have a ROC-score of 1.0; random classifiers return relatively as many false as true positives at each cutoff and have a ROC-score of 0.5.

We use the ROCKIT software [40] for statistical ROC analysis.

Results

The GPboost classifier is significantly better than the energy-based classifiers

We trained the GPboost and Khvorova classifiers on the training sets used to train the Ui-Tei, Amarzguioui, Hsieh, and Reynolds algorithms. The training set also included the 14 SEAP siRNAs from Khvorova et al. [34], for a total of 453 unique siRNA sequences. We classified all siRNAs that gave a remaining mRNA level of ⩽20% as effective and the other siRNAs as ineffective. This gave 141 effective and 252 ineffective siRNAs.

We used 10-fold cross-validation to get an estimate of the algorithms' predictive accuracy, and measured the total ROC-score and correlation between algorithm output and siRNA efficacy in the 10 cross-validation test sets. This resulted in correlations −0.47, −0.39, and −0.23, and ROC-scores of 0.77, 0.69, and 0.63 for the GPboost, Schwarz, and Khvorova algorithms on the complete training set.

As the ROC-curves in Fig. 1 show, the GPboost classifier has higher sensitivity than the other two classifiers for all specificity levels. Indeed, the GPboost classifier's ROC-area is significantly greater than the ROC-areas of the other two classifiers ($p = 0.002$ and $p < 10^{-4}$ for the Schwarz and Khvorova classifiers). We also tested whether the GPboost classifier had a significantly higher sensitivity compared to the other two algorithms, in the important high specificity region (specificities 95%, 90%, 85%, and 80%). The GPboost classifier was better than that of Schwarz on 95% specificity ($p = 0.07$), and was significantly better (95% confidence level) than both classifiers on all other specificities.

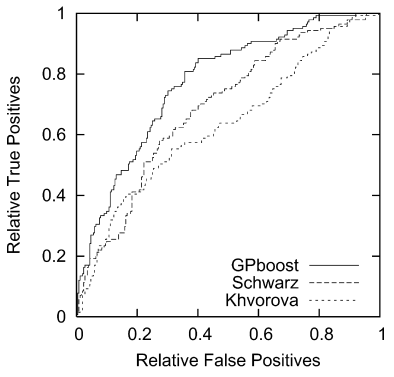

Fig. 1. ROC graphs for the GPboost, Schwarz, and Khvorova classifiers on the complete training set. The graphs are based on the test results from the 10-fold cross-validation procedure. The GPboost classifier has the highest sensitivity for all specificity levels.

The GPboost classifier has the best performance

It is often reasonable to expect that algorithms will be positively biased on their own training data as compared to independent test data. Indeed, when we tested the algorithms on their corresponding training data, the performance in terms of ROC-area and correlation was higher than the performance on the rest of the database (data not shown). The only exception was the Reynolds algorithms, which had a higher correlation on the rest of the database than on their training set. All the algorithms had a higher performance on their training sets than algorithms that were trained on other datasets (data not shown).

Table 3 shows the performance of the different classifiers when tested on the subsets of the database that did not include their corresponding training sets. Each classifier's performance is compared to the GPboost classifier's performance on the same data. Fig. 2 shows the Amarzguioui and Reynolds algorithms' ROC-curves compared to those of the GPboost classifiers. The ROC-curves for the other algorithms are in Supplementary figure SF1.

A closer inspection of the ROC-curves in Figs. 1 and 2 shows that the GPboost classifier generally has the best performance. It has the highest sensitivity for all specificity levels when compared to all the other algorithms. The ROC-curves and ROC-scores also show that some of the classifiers perform only slightly better than random. This is the case for the Luo classifier [26] and the modified Stockholm rules and decision tree of [29] from http://sisearch.cgb.ki.se/.

Statistical tests that compared the GPboost classifier to the other algorithms showed that the GPboost classifier

Table 3
Algorithm performance compared to that of the GPboost classifier

Algorithm	|siRNAs|		Algorithm		GPboost		
	|P|	|N|	ROC	R	ROC	R	p
Ui-Tei	112	229	0.65	−0.34	0.74	−0.42	0.008
Amarzguioui	107	206	0.72	−0.47	0.79	−0.48	0.05
Hsieh	140	145	0.67	−0.34	0.77	−0.50	0.02
Takasaki	137	242	0.62	−0.25	0.78	−0.48	$<10^{-4}$
Reynolds 1	53	161	0.64	−0.44	0.78	−0.46	0.0008
Reynolds 2	53	161	0.66	−0.46	0.78	−0.46	0.003
Stockholm 1	50	154	0.65	−0.31	0.78	−0.45	0.002
Stockholm 2	36	104	0.56	−0.21	0.78	−0.45	$<10^{-4}$
Tree	36	104	0.51	−0.24	0.78	−0.45	$<10^{-4}$
Luo	137	232	0.55	−0.14	0.78	−0.48	$<10^{-4}$

The algorithm performance is measured on the subset of the large training database that was not used to train the respective algorithm. |P| and |N| are the number of effective and ineffective siRNAs in the different sets; p is the p value for the test whether the GPboost classifier's ROC-score is significantly greater than that of the corresponding algorithm.

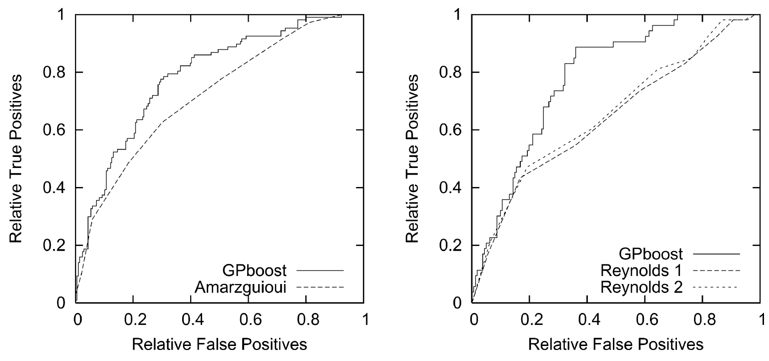

Fig. 2. The ROC graphs for the GPboost classifiers compared to those of the Amarzguioui and Reynolds classifiers; the ROC-curves for the other algorithms are in Supplementary Figure SF1. The GPboost classifier has the highest sensitivity for all specificity levels. The graphs were generated from different subsets of the large training database; see Table 3 and the main text for details.

had a significantly higher ROC-area than all the other algorithms (95% confidence level; *p* values in Table 3). Tests also showed that only the Amarzguioui and Reynolds algorithms have a performance that is comparable (95% confidence level) to that of the GPboost classifier in the high specificity region (the Amarzguioui and Reynolds 2 classifiers had *p* values 0.2, 0.1, 0.09, and 0.07, and 0.5, 0.3, 0.1, and 0.04 on specificities 95%, 90%, 85%, and 80%). Based on these results, one would expect that the GPboost classifier identifies more effective siRNAs.

Few classifiers have a stable and high performance

To further evaluate the classifiers' performance, we tested the different classifiers on three other datasets: the test set used by Reynolds et al. [23] to test their algorithm, the dataset of Harborth et al. [20], and the dataset of Vickers et al. [33]. To the best of our knowledge, none of these datasets were used to train any of the algorithms, except for the Vickers set, which was used to train the classifiers of Chalk et al. [29]. Since these sets are fairly large, come from three different sources, and have been generated using three different methods, they should give a fair estimate of the different classifiers' performance on unknown data.

Because the datasets were generated using different methods, and to get a representative number of effective and ineffective siRNAs in each set, we used different cutoffs for classifying the siRNAs as effective and ineffective. That is, we used 20%, 50%, and 10% for the Reynolds, Vickers, and Harborth data. This resulted in 17, 18, and 25 effective siRNAs, and 43, 58, and 19 ineffective siRNAs in the respective sets. Because of limitations in the web server of Chalk et al. [29], the Stockholm 2 and Tree classifiers were only tested on 13, 11, and 22 effective, and 32, 36, and 14 ineffective siRNAs.

Table 4 and Fig. 3 summarize the results on the three test sets (ROC-curves for the Vickers and Harborth data are in Supplementary figure SF2). The table and figure show that (i) the GPboost algorithm has the highest ROC-score on all datasets; (ii) only the GPboost, Amarzguioui, Ui-Tei, and Reynolds classifiers have a stable and high performance; and (iii) the performance of the remaining algorithms varies from random classification to intermediate performance. The Schwarz and Khvorova classifiers reach the performance of the best classifiers, but only on two of the three test sets.

Table 4
Results on the three independent test sets

Algorithm	Reynolds [23]		Vickers [33]		Harborth [20]	
	ROC	*R*	ROC	*R*	ROC	*R*
GPboost	0.84	−0.55	0.83	−0.35	0.82	−0.43
Ui-Tei	0.75	−0.47	0.77	−0.58	0.79	−0.31
Amarzguioui	0.75	−0.45	0.80	−0.47	0.76	−0.34
Hsieh	0.56	−0.03	0.51	−0.15	0.66	−0.17
Takasaki	0.49	−0.03	0.62	−0.25	0.51	0.01
Reynolds 1	0.70	−0.35	0.73	−0.47	0.79	−0.23
Reynolds 2	0.70	−0.37	0.71	−0.44	0.79	−0.23
Schwarz	0.71	−0.29	0.72	−0.35	0.51	0.01
Khvorova	0.68	−0.15	0.77	−0.19	0.60	−0.11
Stockholm 1	0.56	−0.05	0.58	−0.18	0.64	−0.28
Stockholm 2	0.63	0.00	0.56	−0.15	0.69	−0.41
Tree	0.50	−0.11	0.68	−0.43	0.54	0.06
Luo	0.50	−0.33	0.54	−0.27	0.71	−0.40

The GPboost algorithm has the highest ROC-score on all test sets and only a few algorithms (outlined in gray) have a stable, high performance on all sets.

Effective siRNAs are identified by sequence alone

The results for the Luo algorithm deserve some discussion. On most datasets, the algorithm has a ROC-score that is close to random classification, but at the same time the correlation between the algorithm's output and the siRNA efficacy can be well above random. Indeed, all the reported correlations for the Luo algorithm are

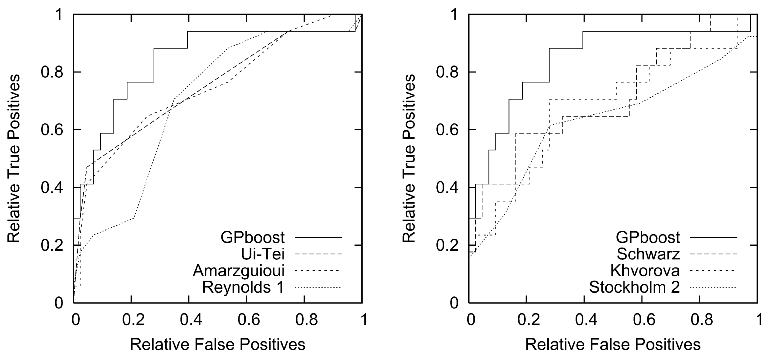

Fig. 3. ROC graphs for the seven highest scoring algorithms [23–25,29,32,34,35] on the Reynolds test sets. The GPboost classifier has the highest sensitivity for almost all specificity levels when compared to the other algorithms.

significant at the 95% confidence level. One possible explanation is that the mRNA secondary structure is important for siRNA efficacy, but that it is only a secondary effect compared to the siRNA sequence-based features, such as the duplex differential 5′/3′ free energy or sequence motifs. We tried to combine the Luo classifier with the GPboost classifier, which gave a small but insignificant improvement (the 10-fold cross-validation correlation and ROC-score were increased by approximately 0.02 and 0.005). Thus, it seems that on the data we examined here, highly effective siRNAs can be identified by the siRNA sequence alone, and that the secondary structure of the mRNA target sequence has limited influence on siRNA efficacy.

Discussion

We have shown that our regularized genetic programming approach (GPboost) [32] performs better than other published siRNA efficacy algorithms on a large collection of functionally validated siRNAs. We believe that the GPboost algorithm has a higher performance because (i) the algorithm was trained on a larger set of siRNAs than the other algorithms; (ii) the algorithm uses patterns that capture more complex characteristics of effective siRNAs than do the simpler motif algorithms; and (iii) the algorithm is very robust when it comes to noise in the training data, as, for instance, siRNAs that have been erroneously labeled as effective or ineffective.

Surprisingly, several algorithms gave close to random classification, and only the GPboost, Reynolds, Amarzguioui, and Ui-Tei algorithms have a high and stable performance on the whole dataset. This suggests that over-fitting is a problem with many algorithms, and that proper care needs to be taken when estimating the classification accuracy to avoid such effects.

The results suggest that it may not be critical to consider the target site's secondary structure, as the best algorithms only consider the sequence alone. Our analysis suggests that mRNA secondary structure has a minor influence on siRNA efficacy, but that highly effective siRNAs can be selected based on target sequence alone. This fact has not been proven, however, so secondary structure should still be investigated when analyzing new data.

We expect that the dataset we used is biased, as the siRNAs have not been randomly selected in the publications in which they appeared. Even so, we believe that the results of our comparison will generalize to other data as well, since all of the algorithms we investigated were trained on subsets of this dataset.

The RNAi field is maturing rapidly, and new siRNA efficacy prediction algorithms will emerge partly due to larger and better datasets. We expect that the need for a large publicly available set of randomly selected validated siRNAs will rise as more algorithms are published, since it is difficult to objectively compare their performance without an independent test set.

Acknowledgments

We thank A. Khvorova for providing details from [23], and H.E. Krokan, T. Holen, T.B. Grünfeld, and O.R. Birkeland for valuable comments on the manuscript.

Appendix A. Supplementary material

Supplementary data associated with this article can be found, in the online version, at doi:10.1016/j.bbrc.2004.06.116.

References

[1] A. Fire, S. Xu, M. Montgommery, S. Kostas, S. Driver, C. Mello, Potent and specific genetic interference by double-stranded RNA in *Caenorhabditis elegans*, Nature 391 (6593) (1998) 806–811.

[2] P. Zamore, T. Tuschl, P. Sharp, D. Bartel, RNAi: double-stranded RNA directs the ATP-dependent cleavage of mRNA at 21 to 23 nucleotide intervals, Cell 101 (1) (2000) 25–33.
[3] A. Nykanen, B. Haley, P. Zamore, ATP requirements and small interfering RNA structure in the RNA interference pathway, Cell 107 (3) (2001) 309–321.
[4] S. Elbashir, J. Harborth, W. Lendeckel, A. Yalcin, K. Weber, T. Tuschl, Duplexes of 21-nucleotide RNAs mediate RNA interference in cultured mammalian cells, Nature 411 (6836) (2001) 494–498.
[5] T. Holen, M. Amarzguioui, M.T. Wiiger, E. Babaie, H. Prydz, Positional effects of short interfering RNAs targeting the human coagulation trigger tissue factor, Nucleic Acids Res. 30 (8) (2002) 1757–1766.
[6] T. Brummelkamp, R. Bernards, R. Agami, A system for stable expression of short interfering RNAs in mammalian cells, Science 296 (5567) (2002) 550–553.
[7] D. Rubinson, C. Dillon, A. Kwiatkowski, C. Sievers, L. Yang, J. Kopinja, M. Zhang, M. McManus, F. Gertler, M. Scott, L. Parijs, A lentivirus-based system to functionally silence genes in primary mammalian cells, stem cells and transgenic mice by RNA interference, Nat. Genet. 33 (3) (2003) 401–406.
[8] D. Dykxhoorn, C. Novina, P. Sharp, Killing the messenger: short RNAs that silence gene expression, Nat. Rev. Mol. Cell Biol. 4 (6) (2003) 457–467.
[9] M. McManus, P. Sharp, Gene silencing in mammals by small interfering RNAs, Nat. Rev. Genet. 3 (10) (2002) 737–747.
[10] P. Zamore, RNA interference: listening to the sound of silence, Nat. Struct. Biol. 8 (9) (2001) 746–750.
[11] G. Hannon, RNA interference, Nature 418 (6894) (2002) 244–251.
[12] C. Sledz, M. Holko, M. de Veer, R. Silverman, B. Williams, Activation of the interferon system by short-interfering RNAs, Nat. Cell Biol. 5 (9) (2003) 834–839.
[13] A. Bridge, S. Pebernard, A. Ducraux, A.-L. Nicoulaz, R. Iggo, Induction of an interferon response by RNAi vectors in mammalian cells, Nat. Genet. 34 (3) (2003) 263–264.
[14] S. Persengiev, X. Zhu, M. Green, Nonspecific, concentration-dependent stimulation and repression of mammalian gene expression by small interfering RNAs, RNA 10 (1) (2004) 12–18.
[15] M. Amarzguioui, T. Holen, E. Babaie, H. Prydz, Tolerance for mutations and chemical modifications in a siRNA, Nucleic Acids Res. 31 (2) (2003) 589–595.
[16] J. Doench, C. Petersen, P. Sharp, SiRNAs can function as miRNAs, Genes Dev. 17 (4) (2003) 438–442.
[17] D. Semizarov, L. Frost, A. Sarthy, P. Kroeger, D. Halbert, S. Fesik, Specificity of short interfering RNA determined through gene expression signatures, Proc. Natl. Acad. Sci. USA 100 (11) (2003) 6347–6352.
[18] J.-T. Chi, H. Chang, N. Wang, D. Chang, N. Dunphy, P. Brown, Genomewide view of gene silencing by small interfering RNAs, Proc. Natl. Acad. Sci. USA 100 (11) (2003) 6343–6346.
[19] A. Jackson, S. Bartz, J. Schelter, S. Kobayashi, J. Burchard, M. Mao, B. Li, G. Cavet, P. Linsley, Expression profiling reveals off-target gene regulation by RNAi, Nat. Biotechnol. 21 (6) (2003) 635–637.
[20] J. Harborth, S.M. Elbashir, K. Vandenburgh, H. Manninga, S.A. Scaringe, K. Weber, T. Tuschl, Sequence, chemical, and structural variation of small interfering RNAs and short hairpin RNAs and the effect on mammalian gene silencing, Antisense Nucleic Acid Drug Dev. 13 (2003) 83–106.
[21] O. Snøve, T. Holen, Many commonly used siRNAs risk off-target activity, Biochem. Biophys. Res. Commun. 319 (1) (2004) 256–263.
[22] S. Elbashir, J. Harborth, K. Weber, T. Tuschl, Analysis of gene function in somatic mammalian cells using small interfering RNAs, Methods 26 (2) (2002) 199–213.
[23] A. Reynolds, D. Leake, Q. Boese, S. Scaringe, W.S. Marshall, A. Khvorova, Rational siRNA design for RNA interference, Nat. Biotechnol. 22 (3) (2004) 326–330.
[24] M. Amarzguioui, H. Prydz, An algorithm for selection of functional siRNA sequences, Biochem. Biophys. Res. Commun. 316 (4) (2004) 1050–1058.
[25] K. Ui-Tei, Y. Naito, F. Takahashi, T. Haraguchi, H. Ohki-Hamazaki, A. Juni, R. Ueda, K. Saigo, Guidelines for the selection of highly effective siRNA sequences for mammalian and chick RNA interference, Nucleic Acids Res. 32 (3) (2004) 936–948.
[26] K. Luo, D. Chang, The gene-silencing efficiency of siRNA is strongly dependent on the local structure of mRNA at the targeted region, Biochem. Biophys. Res. Commun. 318 (1) (2004) 303–310.
[27] A. Hsieh, R. Bo, J. Manola, F. Vazquez, O. Bare, A. Khvorova, S. Scaringe, W. Sellers, A library of siRNA duplexes targeting the phosphoinositide 3-kinase pathway: determinants of gene silencing for use in cell-based screens, Nucleic Acids Res. 32 (3) (2004) 893–901.
[28] P. Pancoska, Z. Moravek, U. Moll, Efficient RNA interference depends on global context of the target sequence: quantitative analysis of silencing efficiency using Eulerian graph representation of siRNA, Nucleic Acids Res. 32 (4) (2004) 1469–1479.
[29] A. Chalk, C. Wahlestedt, E. Sonnhammer, Improved and automated prediction of effective siRNA, Biochem. Biophys. Res. Commun. 319 (1) (2004) 264–274.
[30] S. Takasaki, S. Kotani, A. Konagaya, An effective method for selecting siRNA target sequences in mammalian cells, Cell Cycle, (2004) Epub ahead of print.
[31] A. Halaas, B. Svingen, M. Nedland, P. Sætrom, O. Snøve, O.R. Birkeland, A recursive MISD architecture for pattern matching, IEEE Trans. VLSI Syst. 12 (7) (2004) 727–734.
[32] P. Sætrom, Predicting the efficacy of short oligonucleotides in antisense and RNAi experiments with boosted genetic programming, Bioinformatics, (2004) Epub ahead of print.
[33] T.A. Vickers, S. Koo, C.F. Bennett, S.T. Crooke, N.M. Dean, B.F. Baker, Efficient reduction of target RNAs by small interfering RNA and RNase H-dependent antisense agents. A comparative analysis, J. Biol. Chem. 278 (9) (2003) 7108–7118.
[34] A. Khvorova, A. Reynolds, S.D. Jayasena, Functional siRNAs and miRNAs exhibit strand bias, Cell 115 (2003) 209–216.
[35] D.S. Schwarz, G. Hutvágner, T. Du, Z. Xu, N. Aronin, P.D. Zamore, Asymmetry in the assembly of the RNAi enzyme complex, Cell 115 (2003) 199–208.
[36] K. Yoshinari, M. Miyagishi, K. Taira, Effects on RNAi of the tight structure, sequence and position of the targeted region, Nucleic Acids Res. 32 (2) (2004) 691–699.
[37] T. Xia, J. SantaLucia Jr., M.E. Burkard, R. Kierzek, S.J. Schroeder, X. Jiao, C. Cox, D.H. Turner, Thermodynamic parameters for an expanded nearest-neighbor model for formation of RNA duplexes with Watson–Crick base pairs, Biochemistry 37 (1998) 14719–14735.
[38] M. Zuker, Mfold web server for nucleic acid folding and hybridization prediction, Nucleic Acids Res. 31 (13) (2003) 3406–3415.
[39] P. Baldi, S. Brunak, Y. Chauvin, C. Andersen, H. Nielsen, Assessing the accuracy of prediction algorithms for classification: an overview, Bioinformatics 16 (5) (2000) 412–424.
[40] C.E. Metz, B.A. Herman, C.A. Roe, Statistical comparison of two ROC-curve estimates obtained from partially-paired datasets, Med. Decis. Making 18 (1) (1998) 110–121.

Paper VII

Predicting non-coding RNA genes in *Escherichia coli* with boosted genetic programming

Published online June 7, 2005

Nucleic Acids Research, 2005, Vol. 33, No. 10 ***3263–3270***
doi:10.1093/nar/gki644

Predicting non-coding RNA genes in *Escherichia coli* with boosted genetic programming

Pål Sætrom*, Ragnhild Sneve[1], Knut I. Kristiansen[1], Ola Snøve Jr., Thomas Grünfeld, Torbjørn Rognes[1] and Erling Seeberg[1]

Interagon AS, Medisinsk teknisk senter, NO-7489 Trondheim, Norway and [1]Centre for Molecular Biology and Neuroscience, Institute of Medical Microbiology, Rikshospitalet University Hospital, NO-0027 Oslo, Norway

Received November 24, 2004; Revised February 22, 2005; Accepted May 20, 2005

ABSTRACT

Several methods exist for predicting non-coding RNA (ncRNA) genes in *Escherichia coli* (*E.coli*). In addition to about sixty known ncRNA genes excluding tRNAs and rRNAs, various methods have predicted more than thousand ncRNA genes, but only 95 of these candidates were confirmed by more than one study. Here, we introduce a new method that uses automatic discovery of sequence patterns to predict ncRNA genes. The method predicts 135 novel candidates. In addition, the method predicts 152 genes that overlap with predictions in the literature. We test sixteen predictions experimentally, and show that twelve of these are actual ncRNA transcripts. Six of the twelve verified candidates were novel predictions. The relatively high confirmation rate indicates that many of the untested novel predictions are also ncRNAs, and we therefore speculate that *E.coli* contains more ncRNA genes than previously estimated.

INTRODUCTION

Non-coding RNAs (ncRNA) are transcripts, whose function lies in the RNA sequence itself and not as information carriers for protein synthesis. Although long believed to be a minor gene class, recent discoveries have revealed that ncRNA genes are far more prevalent than previously believed and that they have other important roles beyond protein synthesis (rRNA and tRNA) (1–5).

In *Escherichia coli*, the number of experimentally verified small RNA (sRNA) genes (ncRNA genes excluding rRNA and tRNA) has increased rapidly. Only 10 sRNA genes were known in 1999 (6), whereas a recent survey listed 55 known sRNA genes (7). Subsequent RNA cloning experiments increased the number of known sRNA genes to 62 (8).

Most of these sRNA genes were identified in six studies describing systematic searches for new sRNA genes (9–14). All but one of these studies (14) used computational methods to predict sRNA genes. The computational methods ranged from analysis of sequence (9,10) and structure (11) conservation; to promoter and terminator identification (9,13); and machine learning based on sequence composition, known ncRNA motifs and RNA secondary structure stability (12). Together, these six studies have predicted ~1000 non-redundant sRNA candidates that are yet to be confirmed (7). Note, however, that only 95 candidates were predicted by more than one study.

We describe a method that uses automatic discovery of sequence patterns to predict ncRNA genes in *E.coli*'s intergenic regions. The main strengths of the method as compared to other methods are that (i) it uses the DNA sequence directly as input, which helps to reduce any potential bias from input feature selection and encoding (12); (ii) it works well with a much larger number of intergenic sequences (negative examples) than known ncRNA sequences (positive examples) (12); (iii) it is very robust when it comes to noise in the training data, as for instance intergenic regions that actually are ncRNAs; and (iv) it does not rely on sequence conservation to predict ncRNA genes.

The method predicts several hundred intergenic regions to contain ncRNA genes, and over half of these overlap with previous predictions. We test the 10 top-scoring candidates and verify 9 of these by northern analysis. In addition, we test six candidates of varying prediction confidence; three of these are confirmed by northern analysis. Only 6 of these 12 new ncRNA genes have been predicted by previous methods.

Our results indicate that the number of ncRNA genes in *E.coli* is larger than what has previously been estimated (15). This is because the estimates of Zhang and colleagues were partly based on the number of ncRNA genes predicted by more than one method, which, until now, was 95. We have extended

*To whom correspondence should be addressed. Tel: +47 9820 3874; Fax: +47 4559 4458; Email: paal.saetrom@interagon.com

The authors wish it to be known that, in their opinion, the first two authors should be regarded as joint First Authors this list by 44%, which is a significant increase. In addition, we have shown that our method detects ncRNA genes that have not been predicted by other methods.

MATERIALS AND METHODS

Sequence data

We downloaded the *E.coli* K-12 genome sequence (16) (U00096.1) and its annotations (release 73) from EMBL's FTP server (http://www.ebi.ac.uk/genomes/bacteria.html). Based on annotations and previous studies (9–11), we collected a set of 154 experimentally verified ncRNA sequences. These sequences consisted of 86 tRNAs, 22 rRNAs and 46 other sRNA genes. Note that one of these sRNAs was the strain-dependent uptR gene (17). The list of ncRNA sequences is given in the Supplementary Material.

Based on the positions of known ncRNA genes and protein coding sequences (CDS), we constructed a set of intergenic sequences (INT) by removing all parts of the genome containing ncRNAs and CDSs, along with 100 nt on each side. This resulted in 942 subsequences totaling 144 520 nt, which increased to 1884 sequences of 289 040 nt when we added the complement of each sequence.

Each ncRNA and INT sequence was then divided into 50 nt sequence windows with 25 nt overlap. If the final window in a sequence had <50 nt, we adjusted the overlap so that the final window also had 50 nt. For example, 90 nt sequences were divided into three 50 nt sequence windows consisting of nucleotides 1–50, 26–75 and 41–90. The 50 nt window size was chosen because the smallest ncRNA in our dataset was 53 nt (dicF). This procedure gave 1795 ncRNA sequence windows and 10 663 INT sequence windows; removing duplicates in the form of identical sequences reduced the number of ncRNA and INT sequence windows to 840 and 10 572. Of the 840 unique ncRNA sequence windows, 53% were from rRNAs, 30% from sRNAs and 17% from tRNAs.

Algorithms

We use a machine learning algorithm called GPboost$_{\mathrm{Reg}}$ to create classifiers that predict whether or not a sequence is an ncRNA gene. The algorithm has previously been used to predict the efficacy of short oligonucleotides in RNAi and antisense experiments (18,19). In the following, we will only give a basic description of the algorithm; interested readers should consult Sætrom (18) and the references therein for a complete description.

GPboost$_{\mathrm{Reg}}$ takes as input a set of positive and negative sequences and creates a classifier that predicts whether or not an unknown sequence belongs to the positive set. Here, the positive and negative sequences are the ncRNA and INT sequence windows described in the previous section. Thus, the classifier created by GPboost$_{\mathrm{Reg}}$ can predict whether or not a given sequence comes from an ncRNA.

To create the classifiers, GPboost$_{\mathrm{Reg}}$ combines genetic programming (GP) (20) and boosting algorithms (21). GP uses simulated evolution in a population of candidate solutions to solve problems, and here, each individual in the population is an expression in a formal query language (whitepaper available on request). GP evaluates how well each candidate solution separates between the positive and negative sequences and uses this fitness information to guide the simulated evolution. That is, our GP solution iteratively (i) selects candidate solutions based on fitness such that more fit solutions have a higher chance of being selected; (ii) introduces random changes in the selected solutions by exchanging subparts of two candidate solutions (crossover) or randomly changing a subpart of a candidate solution (mutation); and (iii) updates the solution population by replacing the old population with the randomly changed candidate solutions. We repeat this process a fixed number of iterations and choose, as the final solution of the GP run, the candidate solution that gave the best performance on the training set.

The classifiers created by our GP algorithm are sequence patterns that can only give binary answers. That is, given a sequence, each pattern answers either 'yes' (1) or 'no' (-1), as to whether the pattern matches parts of the sequence or not. To improve the confidence of our predictions, we combine the GP algorithm with a boosting algorithm. Boosting algorithms join several classifiers into a final weighted average of the individual classifiers such that the performance of the final classifier is increased compared to each of the single classifiers. To do this, the boosting algorithm guides each GP run's search for good solutions by adjusting the relative importance of each sequence in the training set. Then the boosting algorithm assigns a weight to the best expression from the GP run. This weight is based on the expression's performance in the corresponding training set and is assigned such that the output of the final classifier ranges from -1 to 1. As a result, the classifiers created by our algorithm are the weighted average of several different sequence patterns. We will occasionally refer to these classifiers as models. Note that GPboost$_{\mathrm{Reg}}$ uses regularized boosting (22) to handle noise in the training set.

To reduce the time needed to evaluate each individual expression in the GP population, we use a special purpose search processor designed to provide orders of magnitude higher performance than comparable regular expression matchers (23). The increased performance becomes important when the datasets are large, or when many expressions must be evaluated, for instance, in cross-validation experiments or when GP is used as the base learner in a boosting algorithm.

Quality measures

When a model is evaluated on a positive and negative set of sequences, four statistics (counts) can be defined: the number of true positives (*TP*), false positives (*FP*), true negatives (*TN*) and false negatives (*FN*). These represent the positive hits in the positive set, positive hits in the negative set, negative hits in the negative set and negative hits in the positive set, respectively. Several quality measures can be defined from these counts (24). This study uses the Matthews correlation *M* (Equation 1), false positive rate FP_p (Equation 2) and sensitivity *Se* (Equation 3):

$$M = \frac{FP \cdot TN + FP \cdot FN}{\sqrt{(TN+FN)\cdot(TN+FP)\cdot(TP+FN)\cdot(TP+FP)}} \qquad \mathbf{1}$$

$$FP_p = \frac{FP}{FP+TN} \qquad \mathbf{2}$$

$$Se = \frac{TP}{TP+FN} \qquad \mathbf{3}$$

Strain and growth conditions

Escherichia coli K-12 strain MG1655 cells (from overnight cultures were diluted 1/50 in Luria–Bertani (LB) medium and subsequently grown at 37°C) were grown in LB broth and used for inoculation of liquid cultures. Cells were grown in 100-ml batch cultures in 500-ml Erlenmeyer flasks at 37°C with aeration by rotary shaking (250 r.p.m.). The culture media used was LB as described elsewhere (25). Growth was monitored at 600 nm on a Shimadzu UV-1601 UV-visible spectrophotometer. Cells were harvested in four different growth phases: lag ($OD_{600} < 0.2$), log ($0.2 < OD_{600} < 1.0$), early stationary ($1.0 < OD_{600} < 2.0$) and late stationary phase ($OD_{600} > 2.0$).

RNA isolation

Total RNA was isolated from the cells using a procedure based on trizol reagent combined with RNeasy microcolumns (Qiagen). One milliliter of trizol was added per 10^6 cells and stored at room temperature for 5 min; 0.2 μl chloroform was added per ml of trizol and the sample was shaken for 15 s. The sample rested before centrifugation for 15 min at 12000 *g* and 4°C. The aqueous phase was slowly added 1:1 to 70% EtOH to avoid precipitation. The sample was further loaded to the RNeasy column and washed and DNase treated according to the RNeasy protocol (Qiagen). Isolated RNA was resuspended in RNase-free water and quantitated using Eppendorf BioPhotometer.

Oligonucleotides

The complete list of oligonucleotides used to generate probes for northern analysis and primer extension experiments is provided as Supplementary Material.

Northern analysis

RNA samples (~10 μg) were denatured for 10 min at 60°C in a buffer containing 95% formamide, separated on urea–polyacrylamide (8%) gels, and transferred to nylon membranes by electroblotting. Radiolabeled strand-specific RNA probes were synthesized using *in vitro* transcription according to MAXIscript™ (Ambion). Hybridization signals were visualized on Typhoon 9410 (Amersham).

Primer extension assay

Primer extension assay was carried out with AMV reverse transcriptase (Promega), on ~10 μg total RNA and 5′ end-labeled primers. The primers were end-labeled by using [γ^{32}-P]ATP and polynucleotide kinase. Products of the extension reactions were separated on 8% polyacrylamide sequencing gels alongside sequencing reactions performed on the corresponding PCR products from the intergenic regions. Sequencing reactions were carried out with a Thermo Sequenase Radiolabeled Terminator Cycle Sequencing Kit (USB, Amersham).

RESULTS

ncRNA gene predictions

We used a variant of 10-fold cross-validation to train and test our machine learning algorithm (26,27). More specifically, we randomly divided the sets of ncRNA and INT sequence windows into 10 non-overlapping subsets. Then, we iteratively trained classifiers on 8 of the subsets and tested the classifiers on the remaining 2 subsets. We used one of these test subsets to estimate the optimal value of the regularization parameter in the $\text{GPboost}_{\text{Reg}}$ algorithm and the other test subset as a completely independent test set. We ran this training and testing procedure for 10 iterations such that all the 10 subsets had been used as the independent test set.

To estimate the optimal regularization value, we tried several different values and used the one with the highest average correlation in the 10 ‘parameter estimation’ test subsets. These optimal models had an average correlation of 0.58 on the complete test set, and predicted on average 22 false positive sequence windows in the test subsets. This resulted in an average false positive rate of 2.1%. The models' average sensitivity was 54%. The following sections will examine the predictions in the original ncRNA set, the true positives and false negatives, and the potential new ncRNA genes, the false positives.

The algorithm identifies nearly 80% of the sRNAs in the database. As we used two subsets to test the classifiers, there was some overlap between each of the test sets (each unique sequence was present in two different test sets for two different models). The test set consisted of 840 unique sequences for a total of 1680 sequences: 913 of these were predicted as true positives and 767 were false negatives. When duplicates were removed from these sets, 564 of 840 were positive predictions and 491 of 840 were negative predictions. In other words, 215 sequences were predicted as being both positive and negative. This means that 42% of the sequences were strongly predicted by two models, and 26% were weakly predicted by a single model.

Two of 46 sRNA sequences were completely matched by the models and 10 were completely missed. The complete matches were the partially overlapping rydB and tpe7 found by Wassarman *et al.* (10) and Rivas *et al.* (11), and the misses were micF, oxyS, rybB, ryeE, ryhA, spf, sraB and sraE, and the overlapping ryhB and sraI found by Wassarman *et al.*(10) and Argaman *et al.* (9).

306 potential new ncRNA genes of which 152 confirm previous predictions. The models predicted a total of 438 false positive sequence windows; 57 of these were predicted by two models. Several of the predicted sequence windows overlapped or were located next to each other. When these were joined and treated as one continuous sequence, a total of 306 sequences remained.

A cross-reference of the 306 candidate ncRNA sequences with the list of predicted but unconfirmed ncRNA genes presented in (7) identified that 171 of the sequences overlapped with previous predictions; 152 of these were predicted to be on the same strand. Most of the predictions overlapped with the predictions of Carter and colleagues (12). This was expected, not only because their predictions were the most abundant in our INT set, but also because they base their predictions on the common sequence characteristics of ncRNAs, which is also the essence of our method.

Accounting for the number of predictions made by other methods that were significantly represented (>10 sequences) in our INT set, our predictions support 35, 51, 28 and 41% of the predictions of Rivas *et al.* (11), Carter *et al.* (12), Chen *et al.* (13) and Tjaden *et al.* (14). Thus, there is relatively good correspondence between our predictions and the predictions of these four methods.

Our results confirm several previous predictions that were not supported by other methods. In total, the intergenic regions in our dataset contained 288 sequences that have been predicted by only one previous method to be part of an ncRNA gene. Our predictions overlapped 123 of these 288 sequences. Excluding the predictions that were unique to the Carter algorithm, our predictions supported 42 of the remaining 166 sequences. Thus, although our predictions increased the list of candidates that are unique to a single study by 15%, we increased the list of candidates predicted by more than one study from 95 to 218 (7). Even when excluding the Carter specific sequences, we increased the list of candidates predicted by more than one study by 44% (7). This is a significant increase.

Table 1 shows the 10 highest scoring intergenic sequence windows (the complete list of predictions are available as Supplementary Material). The table is sorted according to the model output for the highest predicted window in the sequence.

After we started our experiments, several new ncRNA genes in *E.coli* have been identified. Table 2 lists the ncRNA genes that were not included as known ncRNAs in our training set, but that were included with at least 50 nt in our set of intergenic sequences. That is, they were falsely included as negative sequences in the training set. The genes were mainly collected from the *E.coli* genome project's (www.genome.wisc.edu) ASAP database (28) (*E.coli* K-12 Strain MG1655 version m54) and from Refs (7,8).

Although, as Table 2 shows, our method only predicts 2 of the 10 genes to be on the correct strand, the performance is not poorer than that of other methods. For instance, the method of Carter and colleagues (12), which is comparable to our method, predicts only one gene (SroG) correctly. Thus, these genes may be too different to be predictable without combining several of the available methods.

We also cross-referenced our predictions with the unconfirmed transcripts in the cDNA library of Vogel *et al.* (8). Table 3 lists the transcripts that were included with at least 50 nt in our set of intergenic sequences. As the table shows, we predict 5 of the 7 transcripts to be ncRNA genes with the correct orientation. Again, our predictions are comparable to or slightly better than other methods.

Finally, Kawano *et al.* (29) describes several new ncRNA genes. Not all these new ncRNAs were present in our dataset; of the three genes that were present, our predictions match one (RyfB). The other two genes (SokE and SokX), like rdlA, rdlB, rdlC and rdlD, may be involved in anti-sense regulation of hok and ldr (29–31). As these ncRNAs' function is closely linked to their targets' sequences, they may not share many sequence characteristics with other ncRNAs. This can explain why our method has problems predicting these hok/ldr-related ncRNAs.

ncRNA gene validations

To test our predictions, we selected 16 predictions for experimental validation. These included all the top 10 predictions from Table 1 and 6 additional predictions with varying prediction confidence (summarized in Table 4). We chose the 6

Table 1. Top ten predictions sorted by prediction confidence

ID	Position	Length	Strand	Score	Annotation
I001	271879	100	+	0.22	271880–272035 + Carter *et al.*
I002	4230937	150	−	0.22	4230927–4231086 − Carter *et al.*
I003	719883	75	+	0.21	719854–719973 + Carter *et al.*
I004	3766615	50	+	0.21	*Novel*
I005	303544	50	−	0.19	*Novel*
I006	262270	82	−	0.18	*Novel*
I007	4626216	75	+	0.17	*Novel*
I008	1702671	75	+	0.16	1702604–1702818 + Tjaden *et al.*
I009	1859481	125	+	0.16	1859567–1859646 + Carter *et al.*
I010	4527911	50	+	0.15	4527862–4527941 + Carter *et al.*

The given position is the 5′ end for predictions in the positive strand, and the 3′ end for predictions in the negative strand. The score is the classifier output for the highest scoring sequence window in a sequence.

Table 2. Known ncRNA genes included in the set of intergenic sequences

Gene	Overlap	Strand	Prediction	Previous predictions (7)
C0067 (12)	60 of 124	+	Not predicted	n/a
rdlA (30)	66 of 66	+	Predicted 50 nt (−)	?(11), − (12)
rdlB (30)	65 of 65	+	Not predicted	? (11), − (12)
rdlC (30)	67 of 67	+	Not predicted	? (11), − (12)
IS061 (13)	60 of 157	−	Not predicted	n/a
IS092 (13)	116 of 159	−	Not predicted	n/a
rygC (10)	76 of 150	+	Predicted 50 nt (+ and −)	+ (13), − (12)
SroG (8)	110 of 147	−	Predicted 89 nt (−)	− (12)
rdlD (30)	63 of 63	+	Not predicted	− (14), − (12)
SroH (8)	61 of 159	−	Not predicted	+ (13)

The overlap is the number of nucleotides from the ncRNA included as an intergenic sequence. The last column lists the strand and the reference to previous predictions overlapping the gene.

Table 3. Unconfirmed transcripts from (8) included in the set of intergenic sequences

Contig	Overlap	Strand	Prediction	Previous predictions (7)
Contig_440	68 of 105	+	Predicted 50 nt (+) and 50 nt (−)	+ (13), − (12)
Contig_68	76 of 157	+	Predicted 49 nt (+)	+ (14), − (13)
Contig_606	83 of 103	+	Predicted 63 nt (+) and 50 nt (−)	+ (14), − (12)
Contig_223	80 of 141	−	Predicted 50 nt (−)	− (12)
Contig_496	73 of 73	+	Predicted 61 nt (+) and 49 nt (−)	− (14), ± (12)
Contig_286	102 of 102	+	Predicted 50 nt (−)	+ (14)
Contig_181	43 of 43	−	Not predicted	? (11), + (13)

See Table 2 for header explanations.

Table 4. Six predictions with varying confidence experimentally tested in the lab

ID	Position	Length	Strand	Score	Annotation
I014	4373943	60	−	0.14	*Novel*
I016	1218274	50	−	0.14	*Novel*
I035	914278	100	+	0.1	914218–914571 ± Rivas *et al.* 914259–914378 + Carter *et al.*
I044	4366175	50	+	0.1	*Novel*
I209	4006562	50	+	0.025	4006513–4006565 − Carter *et al.*
I211	214141	50	−	0.025	*Novel*

See Table 1 for details on the prediction position.

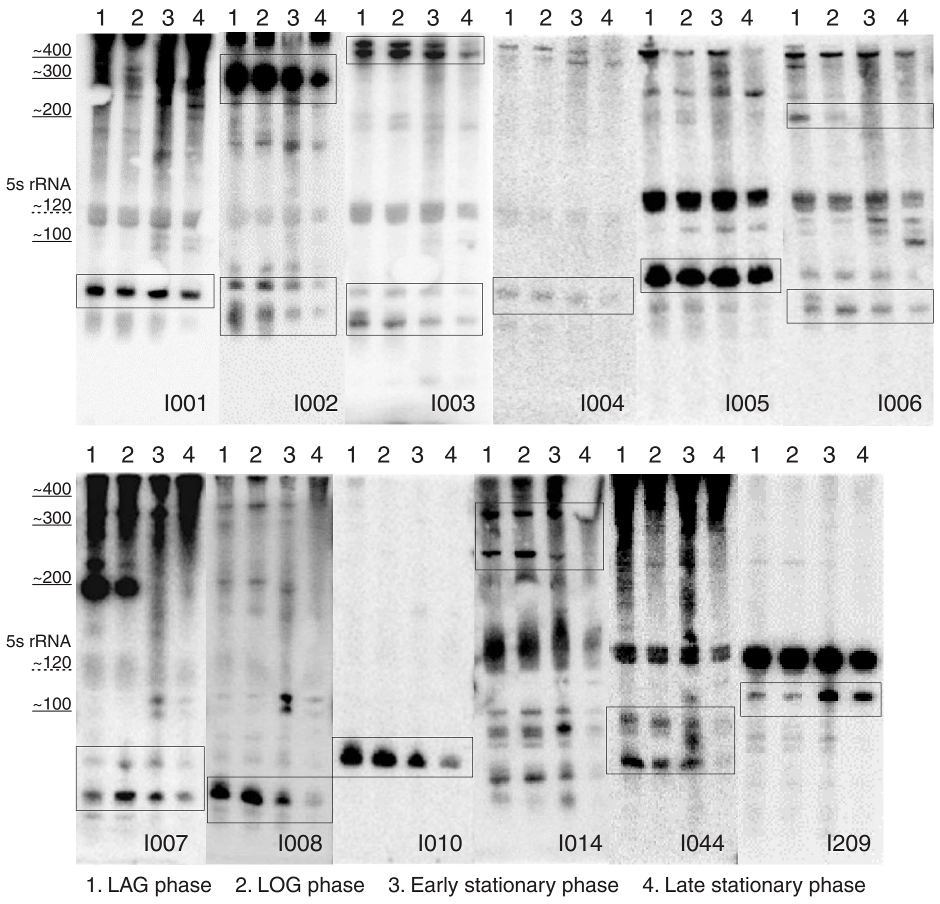

Figure 1. Northern hybridizations of selected predictions against total RNA from lag, log, and early and late stationary phases confirm 12 of 16 selected transcripts. The figure shows the complete northern blots after low stringency wash. The boxed bands indicate the bands that were still present after repeated washes of higher stringency, but the resulting blots are excluded because of poor resolution and picture quality. The indicated sizes are only approximate sizes because these are individual blots lined up together; see Supplementary Figure 2 for size estimates based on each individual blot. Note that most blots have a ∼120 nt band that corresponds to 5s rRNA.

additional predictions to have both high and low prediction confidence, and to be a mix of previously predicted and novel candidates. These 6 additions represented a more varying spectrum of predictions than did the top 10 predictions.

Figure 1 shows the results of northern hybridization with strand-specific probes from 12 of the 16 predictions against total RNA from the *E.coli* lag, log, and early and late stationary phases (see Materials and Methods). Most of the 12 confirmed transcripts were differentially expressed in the four phases, which is in agreement with previously known ncRNAs in *E.coli* (8–10). We did not detect transcripts from the four predictions not shown in Figure 1 (data not shown). The absence of detectable transcripts do, however, not imply that the predictions are wrong as some ncRNAs are only expressed under certain conditions [see for example (2,8,10)]. We also tried to map the 5′ start of 4 of the 12 verified transcripts (I001, I002, I004 and I014, chosen because these were a mix of high and low confidence, and previous and novel predictions). We identified potential 5′ start sites for all four transcripts (see Supplementary Material). Based on these results, we estimated the size of three of the transcripts; see Table 5 for additional information.

As Figure 1 shows, we detected more than one band for six of the predictions. These instances of multiple bands were either (i) a large sequence with one or two additional smaller sequences (I002, I003 and I006); (ii) two large sequences (I014); or (iii) two small sequences (I007 and I044). One possible explanation is that the multiple bands are processed or degraded forms of a single transcript. This may be the case for I002 and I014, as we saw only one 5′ start point for each region in the primer extension. These transcripts could be specifically processed by catalytically active enzymes,

Table 5. Transcripts detected by primer extension

Transcript	Strand	5′ start	Predicted distance	Size	5′ gene		3′ gene	
I001	+	271804	75	75	b0257	+	ykfC	+
I002	−	4231116	179	310	b4024 (‘lysC’)	−	b4025 (‘pgi’)	+
I004	+	3766359	256	n/a	o153 (‘yibG’)	+	yibH	−
I014	−	4374139	196	300	o188 (‘efp’)	+	o155 (‘sugE’)	+

The table lists the transcripts' 5′ ends; their orientation; the distance between the 5′ ends and the predicted transcripts; the transcripts' estimated size; and the name and orientation of 5′ and 3′ flanking genes (relative to the + strand). Note that the I004 5′ start point overlaps prediction HB_200 of Carter and colleagues (12), but we did not detect any northern signal that corresponded to this 5′ start (see Figure 1).

or unspecifically processed by ribonucleases. Several known ncRNAs in *E.coli* are specifically processed (32), and our results are similar to previously predicted and verified ncRNAs thought to be specifically processed (9).

It is possible that some of the larger transcripts detected could be processed 5′ or 3′ ends of neighboring mRNAs; e.g. I002 overlaps the 5′ CDS of lysC by 6 nt. The neighboring genes that the other large transcripts can and do overlap with (we did not establish the 5′ ends of I003 and I006, but I014 overlaps 4 nt in the 5′ CDS of efp) are on the opposite strand of the verified transcripts. Thus, it is possible that these transcripts can regulate their neighboring genes through an antisense mechanism.

Because the transcripts we have tested have not previously been detected, these transcripts may be unstable or of low abundance and therefore difficult to detect. Such instability may also explain some of the multiple bands. Another possible explanation could be that the strand-specific probes bind to other transcripts, but a Blast (33) search of the probes against the complete *E.coli* genome did not give any matches with *E*-values below 0.1, except for the intended target sites. Thus, it is unlikely that the multiple bands in the northern blots are caused by the probes hybridizing to other complementary transcripts.

Excluding tRNAs and rRNAs improves specificity

Our initial database of ncRNA genes was slightly biased towards rRNA and tRNA genes. As our main focus was to identify other small RNA genes, we did a separate analysis where we trained classifiers exclusively on the sRNA sequences. In this analysis, we used the query language and methodology from Saetrom (18), i.e. a classifier was the average of 10 GPboost runs instead of a single run as in our previous experiments.

Using this approach, we predicted 135 of 255 sRNA sequence windows, which included sequence windows from all but the micF and sraE genes. In addition, the approach identified 140 potential ncRNAs, 69 of which were novel.

A cross-reference of the potential ncRNAs identified by this method with the list of known genes (see Table 2) showed that it had correctly identified the rygC, SroG and rdlD genes. On the other hand, only Contig_496 of the sequences in Table 3 was correctly identified; two other predictions overlapped Contig_440 and Contig_286, but these were on the opposite strand.

As a comparison, we ran an experiment where we again used the approach of Saetrom (18), but also included the tRNAs and rRNAs. We now identified all the ncRNAs in the training set except spf, sraB, sraD and micF, and predicted 401 potential ncRNAs; 168 of these were novel. Although this approach identified slightly fewer of the sRNA genes in the training set compared to the classifiers that were trained only on the sRNA sequences, it identified all the tRNAs and rRNAs; the sRNA-based classifiers only identified 15 of 22 rRNAs and 21 of 86 tRNAs. Thus, as expected, when the rRNAs and tRNAs are excluded from the training set, the resulting classifiers become more specific. In accordance with this, the classifiers trained on the complete ncRNA set identified four of the known ncRNAs in our set of intergenic sequences (rdlA, rygC, SroG and rdlD), and seven of the nine contigs from Table 3 (Contig_440 and Contig_286 were identified on the wrong strand).

DISCUSSION

We have described a novel method for finding non-coding RNA genes and proved its applicability by analyzing *E.coli* intergenic regions, and testing and experimentally confirming 9 of the top 10 scoring predictions and 3 other predictions with lower score. Several groups have searched for new ncRNAs in *E.coli* (8–14), which have resulted in a list of about ∼1000 non-redundant and untested candidates (7). Our predictions mostly confirm the predictions of the other methods, but we also predict several new ncRNA genes, and, as our experimental verifications show, at least six of these new predictions are genuine ncRNAs: 12 of the 16 tested candidates, including 6 novel predictions, were verified. It would therefore be surprising if none of the other candidates are ncRNAs.

Northern analysis and primer extension showed that our method could not completely identify the true transcript of the verified predictions. That is, the algorithm either only predicted a portion of the transcript or misplaced its start and stop site. There are three main reasons for these errors. First, our data set consisted of 50 nt sequence windows with 25 nt overlap. Consequently, we could only predict the correct start and stop site if these regions aligned with any of the sequence windows in our data set. Here, we would expect that only 1 of 25 start sites would align by chance. Second, our algorithm did not recognize all the sequence windows of the known ncRNAs in the training set. We would therefore be surprised if it correctly predicted the complete sequence of any new transcripts. Third, our algorithm is biased in the sense that it will only detect regions that are similar to regions in the known ncRNAs. Thus, the algorithm would have trouble detecting the novel domains in the new transcripts.

Because of these three shortcomings, we did not expect the algorithm to correctly identify the complete sequence of any new transcripts. Rather, we developed the algorithm as a complementary tool to the existing ncRNA prediction algorithms, which use other features to predict ncRNAs. As an analogy to standard protein coding gene prediction, our algorithm can be considered a content analyzer (34). To get more reliable predictions of complete ncRNAs, we can for example combine our algorithm with algorithms that look for signals such as transcription initiation and termination (9,13). We are currently looking into this.

When comparing our predictions to those of other methods and to the known ncRNAs included in our set of intergenic sequences (see Table 2), we found that some of our predictions were on the opposite strand. In addition, 47 of our predictions overlapped predictions that our algorithm made on the opposite strand (see Supplementary Material). Thus, it appears that the algorithm has problems identifying the correct strand for some transcripts. These results are, however, related to the above discussion on the algorithm's bias: the algorithm will only detect domains that have a similar sequence to those in the known ncRNAs. An ncRNA's function often lies in its secondary structure, however, and in general, several different sequences can fold into the same secondary structure. In particular, for certain sequences both the original and reverse complementary sequence fold into similar secondary structures. Thus, if the reverse complementary of such sequences more closely resembles the known ncRNAs than does the original sequences, our algorithm will predict the reverse complementary sequence to be an ncRNA domain. This is for instance the case for rdlA in Table 2. Our algorithm incorrectly predicted the reverse complementary sequence of rdlA to be an ncRNA, but the secondary structures of the correct sequence mirrors that of the reverse complementary (data not shown).

A recent study uses the sequence conservation of known ncRNA genes and intergenic regions to estimate the number of sRNAs (ncRNAs other than tRNA and rRNA) in *E.coli* to be between 118 and 260 (15). The authors then argue that because the number of sRNA genes that either have been experimentally verified or predicted by at least two different studies in *E.coli* were 150 (at that time), their estimates may be an upper limit to the number of sRNA genes in *E.coli* (15). Following their logic, our results indicate that the number of sRNA genes in *E.coli* may be closer to their highest estimate than to their lowest. This is because we have significantly extended the list of ncRNAs predicted by more than one method, and because we have shown that our method predicts new ncRNAs that have remained undetected by other methods.

To summarize, we have shown that our approach for ncRNA prediction is both accurate and complementary to existing methods. That is, it identifies genuine ncRNA genes, some of which have not been predicted by any other methods.

SUPPLEMENTARY MATERIAL

Supplementary Material is available at NAR Online.

ACKNOWLEDGEMENTS

We thank K. Lagesen for providing the initial database of *E.coli* ncRNA genes, Y. Esbensen for providing the RNA isolation protocol, and H.E. Krokan and M. Bjørås for valuable comments on the manuscript. The work was supported by the Norwegian Research Council, grants 151899/150, 152020/310 and 152001/150, and the bioinformatics platform at the Norwegian University of Science and Technology, Trondheim, Norway. Funding to pay the Open Access publication charges for this article was provided by the Norwegian Research Council.

Conflict of interest statement. None declared.

REFERENCES

1. Eddy,S.R. (2001) Non-coding RNA genes and the modern RNA world. *Nature Rev. Genet.*, **2**, 919–929.
2. Wassarman,K.M. (2002) Small RNAs in bacteria: diverse regulators of gene expression in response to environmental changes. *Cell*, **109**, 141–144.
3. Storz,G. (2002) An expanding universe of noncoding RNAs. *Science*, **296**, 1260–1263.
4. Cawley,S., Bekiranov,S., Ng,H.H., Kapranov,P., Sekinger,E.A., Kampa,D., Piccolboni,A., Sementchenko,V., Cheng,J., Williams,A.J., Wheeler,R., Wong,B., Drenkow,J., Yamanaka,M., Patel,S., Brubaker,S., Tammana,H., Helt,G., Struhl,K. and Gingeras,T.R. (2004) Unbiased mapping of transcription factor binding sites along human chromosomes 21 and 22 points to widespread regulation of noncoding RNAs. *Cell*, **116**, 499–509.
5. Mattick,J.S. (2004) RNA regulation: a new genetics? *Nature Rev. Genet.*, **5**, 316–323.
6. Wassarman,K.M., Zhang,A. and Storz,G. (1999) Small RNAs in *Escherichia coli*. *Trends Microbiol.*, **7**, 37–45.
7. Hershberg,R., Altuvia,S. and Margalit,H. (2003) A survey of small RNA-encoding genes in *Escherichia coli*. *Nucleic Acids Res.*, **31**, 1813–1820.
8. Vogel,J., Bartels,V., Tang,T.H., Churakov,G., Slagter-Jager,J.G., Huttenhofer,A. and Wagner,E.G.H. (2003) RNomics in *Escherichia coli* detects new sRNA species and indicates parallel transcriptional output in bacteria. *Nucleic Acids Res.*, **31**, 6435–6443.
9. Argaman,L., Hershberg,R., Vogel,J., Bejerano,G., Wagner,E.G.H., Margalit,H. and Altuvia,S. (2001) Novel small RNA-encoding genes in the intergenic regions of *Escherichia coli*. *Curr. Biol.*, **11**, 941–950.
10. Wassarman,K.M., Repoila,F., Rosenow,C., Storz,G. and Gottesman,S. (2001) Identification of novel small RNAs using comparative genomics and microarrays. *Genes Dev.*, **15**, 1637–1651.
11. Rivas,E., Klein,R.J., Jones,T.A. and Eddy,S.R. (2001) Computational identification of noncoding RNAs in *E. coli* by comparative genomics. *Curr. Biol.*, **11**, 1369–1373.
12. Carter,R.J., Dubchak,I. and Holbrook,S.R. (2001) A computational approach to identify genes for functional RNAs in genomic sequences. *Nucleic Acids Res.*, **29**, 3928–3938.
13. Chen,S., Lesnik,E.A., Hall,T.A., Sampath,R., Griffey,R.H., Ecker,D.J. and Blyn,L.B. (2002) A bioinformatics based approach to discover small RNA genes in the *Escherichia coli* genome. *Biosystems*, **65**, 157–177.
14. Tjaden,B., Saxena,R.M., Stolyar,S., Haynor,D.R., Kolker,E. and Rosenow,C. (2002) Transcriptome analysis of *Escherichia coli* using high-density oligonucleotide probe arrays. *Nucleic Acids Res.*, **30**, 3732–3738.
15. Zhang,Y., Zhang,Z., Ling,L., Shi,B. and Chen,R. (2004) Conservation analysis of small RNA genes in *Escherichia coli*. *Bioinformatics*, **20**, 599–603.
16. Blattner,F.R., Plunkett,G.,III, Bloch,C.A., Perna,N.T., Burland,V., Riley,M., Collado-Vides,J., Glasner,J.D., Rode,C.K., Mayhew,G.F., Gregor,J., Davis,N.W., Kirkpatrick,H.A., Goeden,M.A., Rose,D.J., Mau,B. and Shao,Y.S. (1997) The complete genome sequence of *Escherichia coli* K-12. *Science*, **277**, 1453–1474.
17. Guigueno,A., Dassa,J., Belin,P. and Boquet,P.L. (2001) Oversynthesis of a new *Escherichia coli* small RNA suppresses export toxicity of DsbA'-PhoA unfoldable periplasmic proteins. *J. Bacteriol.*, **183**, 1147–1158.
18. Sætrom,P. (2004) Predicting the efficacy of short oligonucleotides in antisense and RNAi experiments with boosted genetic programming. *Bioinformatics*, **20**, 3055–3063.
19. Sætrom,P. and Snøve,Jr,O. (2004) A comparison of siRNA efficacy predictors. *Biochem. Biophys. Res. Commun.*, **321**, 247–253.
20. Koza,J.R. (1992) *Genetic Programming: On the Programming of Computers by Natural Selection*. MIT Press, Cambridge, MA.

21. Meir,R. and Rätsch,G. (2003) An introduction to boosting and leveraging. In Mendelson,S. and Smola,A. (eds), *Advanced Lectures on Machine Learning*. Springer-Verlag, Vol. 2600, pp. 118–183.
22. Rätsch,G., Onoda,T. and Müller,K.-R. (2001) Soft margins for AdaBoost. *Mach. Learn.*, **42**, 287–320.
23. Halaas,A., Svingen,B., Nedland,M., Sætrom,P., Snøve,Jr,O. and Birkeland,O.R. (2004) A recursive MISD architecture for pattern matching. *IEEE Trans. VLSI Syst.*, **12**, 727–734.
24. Baldi,P., Brunak,S., Chauvin,Y., Andersen,C.A. and Nielsen,H. (2000) Assessing the accuracy of prediction algorithms for classification: an overview. *Bioinformatics*, **16**, 412–424.
25. Sambrook,J., Fritsch,E.F. and Maniatis,T. (1989) *Molecular Cloning: A laboratory Manual*. 2nd edn. Cold Spring Harbor Laboratory Press, Cold Spring Harbor, NY.
26. Stone,M. (1974) Cross-validatory choice and assessment of statistical predictions. *J. R. Stat. Soc. [Ser. B] (Methodological)*, **36**, 111–147.
27. Kohavi,R. (1995) A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, San Mateo, CA, 1137–1143.
28. Glasner,J.D., Liss,P., Plunkett,G.,III, Darling,A., Prasad,T., Rusch,M., Byrnes,A., Gilson,M., Biehl,B., Blattner,F.R. and Perna,N.T. (2003) ASAP, a systematic annotation package for community analysis of genomes. *Nucleic Acids Res.*, **31**, 147–151.
29. Kawano,M., Reynolds,A.A., Miranda-Rios,J. and Storz,G. (2005) Detection of 5′- and 3′-UTR-derived small RNAs and *cis*-encoded antisense RNAs in *Escherichia coli*. *Nucleic Acids Res.*, **33**, 1040–1050.
30. Kawano,M., Oshima,T., Kasai,H. and Mori,H. (2002) Molecular characterization of long direct repeat (LDR) sequences expressing a stable mRNA encoding for a 35-amino-acid cell-killing peptide and a *cis*-encoded small antisense RNA in *Escherichia coli*. *Mol. Microbiol.*, **45**, 333.
31. Pedersen,K. and Gerdes,K. (1999) Multiple *hok* genes on the chromosome of *Escherichia coli*. *Mol. Microbiol.*, **32**, 1090–1102.
32. Li,Z., Pandit,S. and Deutscher,M.P. (1998) 3′ Exoribonucleolytic trimming is a common feature of the maturation of small, stable RNAs in *Escherichia coli*. *Proc. Natl Acad. Sci. USA*., **95**, 2856–2861.
33. Altschul,S.F., Gish,W., Miller,W., Myers,E.W. and Lipman,D.J. (1990) Basic local alignment search tool. *J. Mol. Biol.*, **215**, 403–410.
34. Mathé,C., Sagot,M.-F., Schiex,T. and Rouzé,P. (2002) Current methods of gene prediction, their strengths and weaknesses. *Nucleic Acids Res.*, **30**, 4103–4117.