DEPARTMENT OF COMPUTER SCIENCE
SERIES OF PUBLICATIONS A
REPORT A-2002-3

■

Pattern Discovery from Biosequences

■

Jaak Vilo

■

UNIVERSITY OF HELSINKI
FINLAND

DEPARTMENT OF COMPUTER SCIENCE
SERIES OF PUBLICATIONS A
REPORT A-2002-3

Pattern Discovery from Biosequences

Jaak Vilo

To be presented, with the permission of the Faculty of Science of the University of Helsinki, for public criticism in Auditorium, Department of Computer Science, on November 29th, 2002, at 12 o'clock noon.

UNIVERSITY OF HELSINKI
FINLAND

Contact information

Postal address:
Department of Computer Science
P.O. Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki
Finland

Email address: Jaak.Vilo@cs.Helsinki.FI, vilo@ut.ee

URL: http://www.cs.Helsinki.FI/

Telephone: +358 9 1911

Telefax: +358 9 191 44441

ISSN 1238-8645
ISBN 952-10-0792-3 (paperback)
ISBN 952-10-0819-9 (PDF)
Computing Reviews (1998) Classification: F.2.2, H.2.8, H.3.5, I.5.3, J.3
Helsinki 2002
Helsinki University Printing House

Pattern Discovery from Biosequences

Jaak Vilo

Department of Computer Science
P.O. Box 26, FIN-00014 University of Helsinki, Finland
Jaak.Vilo@cs.helsinki.fi, http://www.cs.helsinki.fi/u/vilo/

PhD Thesis, Series of Publications A, Report A-2002-3
Helsinki, November 2002, 149 pages
ISSN 1238-8645
ISBN 952-10-0792-3 (paperback)
ISBN 952-10-0819-9 (PDF)

Abstract

In this thesis we have developed novel methods for analyzing biological data, the primary sequences of the DNA and proteins, the microarray based gene expression data, and other functional genomics data. The main contribution is the development of the pattern discovery algorithm SPEXS, accompanied by several practical applications for analyzing real biological problems. For performing these biological studies that integrate different types of biological data we have developed a comprehensive web-based biological data analysis environment Expression Profiler (http://ep.ebi.ac.uk/).

Biosequences, i.e., the primary sequences of DNA, RNA, and protein molecules, represent the most basic type of biological information. Features of these sequences that are reused by nature help us to understand better the basic mechanisms of gene structure, function, and regulation. The SPEXS algorithm has been developed for the discovery of the biologically relevant features that can be represented in the form of sequence patterns. SPEXS is a fast exhaustive search algorithm for the class of generalized regular patterns. This class is essentially the same as used in the PROSITE pattern database, i.e. it allows patterns to consist of fixed character positions, group character positions (ambiguities), and wildcards of variable lengths. The biological relevance of the patterns can be estimated according to several different mathematical criteria, which have to be chosen according to the application.

We have used SPEXS for the analysis of real biological problems, where we have been able to find biologically meaningful patterns in a variety of different applications. For example, we have studied gene regulation mechanisms by a systematic prediction of transcription factor binding sites or other signals in the DNA. In order to find genes that potentially share common regulatory mechanisms, we have used microarray based gene expression data for extracting sets of coexpressed genes.

We have also demonstrated that it is possible to predict the type of interaction between the G-protein coupled receptors (GPCR) and its respective G-protein, the mechanism widely used by cells for signaling pathways. That prediction, although the GPCR's have been studied for decades, primarily for their immense value for the pharmaceutical industry, had been thought to be unlikely from the primary sequence of GPCR alone.

The tools developed for various practical analysis tasks have been integrated into a web-based data mining environment Expression Profiler hosted at the European Bioinformatics Institute EBI. With the tools in Expression Profiler it is possible to analyze a range of different types of data like sequences, numerical gene expression data, functional annotations, or protein-protein interaction data, as well as to combine these analyses.

Computing Reviews (1998) Categories and Subject Descriptors:

F.2.2	[Theory of Computation]: Analysis of Algorithms and Problem Complexity – Nonnumerical Algorithms and Problems
H.2.8	[Information Systems]:Database management – Database Applications
H.3.5	[Information Systems]:Information Storage and retrieval – Online Information Services
I.5.3	[Computing Methodologies]:Pattern Recognition – Clustering
J.3	[Computer Applications]:Life and Medical Sciences

General Terms:
Algorithms, Biology and Genetics, Bioinformatics

Additional Key Words and Phrases:
Pattern Discovery, Data Mining, Gene Expression Data Analysis, Functional Genomics, Scientific Visualization

Acknowledgements

I am most thankful to my educator and supervisor, Professor Esko Ukkonen. His great expertise and understanding of algorithms, interest in biological applications, and continued support has been the key for the outcome of current thesis. Esko also introduced me to Dr. Alvis Brazma, a colleague and friend with whom we have spent a lot of fruitful time working our way through the interdisciplinary field of bioinformatics. I have been privileged to work with many other excellent scientists, of whom in relation with current thesis I would like to mention especially Dr. Inge Jonassen, Dr. Mike Croning, Dr. Steffen Möller, Patrick Kemmeren, Misha Kapushesky, Dr. Ugis Sarkans, Thomas Schlitt, and Kimmo Palin. Finishing writing up the thesis is only the beginning of a career in science. I hope to see much more joint work with the people mentioned above as well as many other with whom I have shared many interesting moments in science.

I started my computer studies at the University of Tartu, spent a year as an exchange student at the University of Helsinki, and worked at the Institute of Cybernetics in Tallinn with Professor Jaan Penjam. Professor Martti Tienari, the head of the Department of Computer Science at the University of Helsinki at the time, invited me to do my Ph.D. in Helsinki. Since that the department became my ”second home” for years. I would like to thank professors Martti Tienari, Esko Ukkonen, Timo Alanko, and Jukka Paakki, the heads of the department, for the excellent working environment. I would also like to thank HeCSE and ComBI graduate schools, Professor Heikki Mannila and his ever encouraging role for my research, the great compute resources of the department supported by Dr. Petri Kutvonen and his team, the excellent library, and of course, the best coffee-room companions I have ever had.

In April 1999 I joined the European Bioinformatics Institute EMBL-EBI, a truely interdisciplinary and international environment, located in the UK at the Wellcome Trust Genome Campus, sharing the grounds with the Sanger Institute and HGMP. At the EBI I enjoyed the exposure to many bioinformatics as well as pure biological research problems. The time spent in the Industry Support and later Microarray Informatics teams lead by Dr. Alan Robinson and Dr. Alvis Brazma, has been most productive in many ways. That period has also contributed a lot to my research toward this thesis. During finishing the thesis, University of Tartu, Estonian Biocenter, and most importantly, EGeen, have to be acknowledged for providing me the necessary time and financial support.

Most of all, my gratitude goes to my family. First, to my parents Ago and Urve, and my sisters Elle and Kaja, who have been very supportive over the years that I have spent abroad doing my research. And finally, to Tiina, Miina, and Luisa, who have missed my presence while I have been working towards the results presented in this thesis. Miina and Luisa, this work is dedicated to you.

Contents

Chapter 1

Introduction

1.1 Motivation and background

The amount of the data collected and stored in databases worldwide is growing with increasing speed. While computers were initially designed and used mostly for numerical computations, a large proportion of the data is nowadays collected and processed in textual form. The sources of textual data can be very different, varying from documents in natural languages to the sequences of biological macromolecules. Efficient methods of analysis are required for understanding the underlying principles about the sequence data.

A natural way to represent the distinctive features of textual data is to use the linguistic properties of the data in the form of formal grammars and languages. Artificial languages like programming languages such as Pascal, C++, Java etc. or page layout and structure markup languages such as LaTeX, HTML, or XML, have a strict predefined grammar. Thus all the texts can be rigorously checked against these grammatical rules.

Unlike the formal languages, the natural languages have evolved over hundreds of thousands of years. Formalizing these languages using grammatical rules for written as well as spoken languages is necessary for many automated tasks. These grammars evolve as the language itself evolves. Some of the grammatical rules can be automatically learned from the data itself. For example, it is possible to learn hyphenation rules quite accurately from examples of correctly hyphenated words only (Liang 1983; Kivinen *et al.* 1994). However, formal grammars are usually not capable of capturing all the aspects of the natural languages, thus exceptions to the rules are common.

Perhaps one of the most fascinating languages being studied by the mankind is our own genetic code, the DNA, RNA, and protein molecules that are essential to all life on planet Earth. These macromolecules are built (usually in a linear manner) from a small amount of building blocks like nucleotides or amino-acids.

The genetic code, using perhaps an oversimplification, can thus be interpreted as sequences of these basic building blocks or letters. The function of each of the molecules is usually determined by their structure, and one of the key questions in modern molecular biology is to understand the relationship between the sequence, structure, and function of these molecules as well as the biological processes they are involved in.

The machinery that is able to read, interpret, replicate, and otherwise utilize the information stored in DNA, RNA and protein molecules is essential to life. This universal language of life has evolved over billions of years, producing many different life-forms from the simplest bacteria to human beings, being able to survive in extreme heat and pressure or under constant freezing conditions, or consuming completely different energy sources. Many of the properties of our genetic code and the ways to interpret that code remain yet to be discovered, described, and understood.

David Searls wrote (Searls 1992): "Linguistic metaphors have been a part of molecular biology ever since the structure of DNA was solved in 1953. Biologists speak of the *genetic code*, of gene *expression* and of *reading frames* in nucleic acids. DNA is *transcribed* into RNA, which is then *translated* into protein. Certain enzymes are even said to *edit* RNA. Despite all this linguistic terminology, however, there has been little effort to apply the tools of formal language theory to the problems of interpreting biological sequences."

Not all of the possible aspects of the biological sequence data, or texts written in natural languages, or sequences of signals arising from the output of various measurement devices (e.g. telecommunication network monitoring devices) can be described sufficiently by the formal grammars belonging to classes of the language hierarchy defined by Noam Chomsky (Chomsky 1956; 1959). This is due to ignoring the semantic aspects or physical properties of the underlying data sources. Nevertheless, we believe that many features of the data can be captured by restricted subclasses of these formal languages, and that these features can be further used for exploring the semantical aspects of the data.

In this thesis we study pattern discovery approaches for finding regularities in the sequence data. The main focus is on discovering patterns, the words or sentences according to some linguistic rules, that occur frequently in input sequences or are characteristic for certain subsets of the input data. Firstly, the frequently recurring patterns are often indicative of the underlying structure and function, as in biology, the conservation of certain features in the course of evolution usually indicates the importance of these features. Secondly, different subsets of input data may represent examples from meaningful concepts. Patterns common to these different subsets can help to distinguish between these sets, as well as to reveal features important for different classes of sequences.

These two cases of pattern discovery describe the two basic problems – the *family conservation problem* and the *family classification problem*, which both are discussed in the thesis.

In this thesis we focus on pattern discovery in biological sequences, as this is perhaps one of the most important application areas with implications to molecular biology and medicine. The aim of the current research is to develop methods for discovering patterns that can be used for advancing the biological knowledge about the structure and function of the genes and gene products.

1.2 Pattern Discovery

Sequence pattern discovery is a research area aiming at developing tools and methods for finding *a priori* unknown patterns in a given set of sequences, patterns that are frequent, unexpected, or interesting according to some formal criteria. Patterns are formal grammatical descriptions for certain *languages* representing subsets of all possible sequences over a finite alphabet. Patterns can be represented using different formalisms, for example, as regular expressions, or probabilistic weight matrices. The *interestingness* of patterns can be interpreted in relation to the pattern description itself or in relation to the sequences being analyzed. For example, if the pattern occurs significantly more frequently than expected by chance then the pattern may be considered interesting. In order to define the interestingness of a pattern a formal scoring mechanism is needed. And finally, for practical applications the algorithms and tools are needed that can be used for discovering interesting patterns. Overall, the pattern discovery problem can thus be divided into three subproblems (Brazma *et al.* 1998a):

1. choosing the appropriate language to describe patterns

2. choosing the scoring function for comparing patterns

3. designing an efficient algorithm for identifying the best-scoring patterns from the selected pattern class according to the chosen scoring function.

Appropriate language for describing patterns depends from the application area. For example, one can ask if the type of patterns one is looking to discover can in principle capture the biological phenomena one attempts to study. Thus, the pattern language has to be chosen appropriately from the biological point of view. Similarly, the scoring function has to be such that it has proven relevance also in the biological terms, not only in abstract mathematical or statistical sense. Unfortunately, not all pattern languages and scoring functions are such that efficient algorithms can be designed so that the pattern discovery could be performed in a reasonable time. From the practical point of view, one may have to balance between the desire to use the very complex pattern language and scoring functions which are biologically perhaps the most relevant, and the available compute resources that require to use computationally more feasible methods. For very large data sets, for example, one may have to use simpler pattern representation language that can offer improved speed in calculations. The balance has to be reasonable though, so that the chosen pattern language and scoring functions do not eliminate the biological relevance of the discovered patterns.

In current thesis we have developed methods and tools for the exhaustive search for the best patterns from a range of different pattern representation languages. In the practical applications we also demonstrate that the choice of simple pattern languages is often sufficient to capture rather complex biological information. This is the motivation throughout the thesis, to design practical algorithms that can be used for studying biologically relevant pattern classes in a variety of biological applications.

Before discussing the three aspects of pattern discovery in more detail we present a few practical applications of using the patterns to represent biologically meaningful concepts. These examples belong essentially to the same pattern representation languages which are later stydied in this thesis.

1.2.1 Applications of pattern discovery

Biological sequences, or *biosequences*, can be grouped in *families* based on their function, structure, cellular location, molecular processes, gene regulation, or other criteria. Here we present some applications, where the patterns common to these groups are able to capture very different biological features.

Many of the protein families and their characteristic patterns have been collected in the protein family database PROSITE (Bairoch 1992; Hoffmann *et al.* 1999). Finding characterizations of biosequence families is an important sequence analysis problem. If a feature common to all known sequences of a family is found, then it is likely that this particular feature is important for the biological role of the family. Algorithms for sequence pattern discovery have been widely used for characterizing protein families, *e.g.* (Jonassen 1997; Hart *et al.* 2000b; Rigoutsos & Floratos 1998b), for surveys see for example (Brazma *et al.* 1998a; Wang, Shapiro, & Shasha 1999).

Jonassen and colleagues (Jonassen, Eidhammer, & Taylor 1999) have studied patterns that incorporate structural information about the packing of *residues*, *i.e.* amino acids in the protein sequence, in three-dimensional space. They define a *packing motif* as a pattern that has multiple occurrences in a set of protein structures. Packing motifs describe clusters of residues that are spatially close together in the 3-D structure, but not necessarily in the primary sequence.

Patterns in protein sequences can represent potentially important features for their functional activities. We have applied pattern discovery combined with careful targeted input sequence selection for predicting the coupling specificity of specific transmembrane receptor proteins called G-protein coupled receptors (GPCR) and the G-proteins from G_s , $G_{i/o}$, or G_{q11} class (Möller, Vilo, & Croning 2001). The GPCR proteins represent the largest type of molecular targets of modern drugs, yet the prediction of the coupling specificity between GPCR proteins and G-proteins from a specific type has required expensive laboratory experiments. The patterns indicative of the most probable interaction type are also likely to reveal the specific residues (amino acids) that are important for these binding events. This study is presented in Section 6.2.

Gene regulation, *i.e.* the mechanism for regulating the levels of gene products in cells happens in a variety of ways. Usually, the first step is to transcribe the genes represented in the DNA into the RNA. It is believed that relatively short regions of DNA are recognized by the proteins involved in the transcription machinery as well as alternative splicing events, and that these features in the DNA determine a large extent of the gene regulation. The task of pattern discovery is to predict the potential regulatory signals, for example the transcription factor binding sites, from the DNA. This is considered in more detail in Section 7.1.

From the computer science viewpoint considering pattern discovery as pure string algorithms, the DNA and proteins differ only by the alphabet size (four and twenty, respectively). Yet, these sequences do represent different physical objects and hence the need for finding patterns may arise in different biological research domains. Often the respective research communities are separated, as well as the approaches developed. The biological features represented by patterns can vary in semantics depending on the biological application, and hence the language of representing the patterns and the criteria for evaluating their interestingness can be very different for different applications. Usually, the data sets involving DNA sequences are much larger than those for the proteins. Finally, the physical-chemical properties of the real atoms represented by letters of an alphabet, or other physical constraints of the molecules in the different application domains differ and may need to be taken into account in pattern discovery.

1.2.2 Pattern representation languages

According to the pattern language we can distinguish between discrete patterns like regular expression type motifs (Bairoch 1992; Jonassen 1997; Brazma *et al.* 1998b) and probabilistic patterns like probabilistic weight matrices (Hertz & Stormo 1999; Bailey & Elkan 1995; Roth *et al.* 1998; Neuwald, Liu, & Lawrence 1995), for example. In the current thesis we consider the deterministic regular patterns (defined in Chapter 2) and approximately matching patterns (see Chapter 4).

Although the probabilistic motif representation is more appropriate for describing certain physical features of the molecules, like a protein's binding efficiency to DNA, these motifs are more complex to discover by computational methods due to a much larger search space. In Section 5.2 we will outline one possible solution for combining the good sides from both the deterministic as well as probabilistic approaches.

One of the oldest and most prominent pattern databases, the PROSITE database (Hoffmann *et al.* 1999) stores information about protein families, their descriptions, and patterns that can be used to determine the membership of novel sequences to these families. Biologically significant patterns and profiles are formulated in such a way that with appropriate computational tools they can help to determine to which known family of proteins the new sequence may belong, or which known domain(s) it contains.

In this section we provide as an example the definition of the pattern language as used in the PROSITE database, as well as give two examples of the PROSITE entries showing how the patterns from this pattern language can capture biologically relevant features about real protein families. Later we show that the same pattern language can also capture other types of features. For example, many of the DNA binding sites can be expressed using similar pattern representation.

The patterns in PROSITE are defined in the Example 1.1. The patterns used in PROSITE actually correspond to the class of regular patterns (a subset of regular expressions) as defined in Chapter 2 and later studied throughout the thesis. The genetic code, including the amino acid alphabet, is also described in Chapter 2.

Example 1.1 *Pattern definitions from the PROSITE database (http://www.expasy.org/prosite/).*

The PA (PAttern) lines contain the definition of a PROSITE pattern. The patterns are described using the following conventions:

- The standard IUPAC one-letter codes for the amino acids are used.
- The symbol 'x' is used for a position where any amino acid is accepted.
- Ambiguities are indicated by listing the acceptable amino acids for a given position, between square parentheses '[ ]'. For example: [ALT] stands for Ala or Leu or Thr.
- Ambiguities are also indicated by listing between a pair of curly brackets '{ }' the amino acids that are not accepted at a given position. For example: {AM} stands for any amino acid except Ala and Met.
- Each element in a pattern is separated from its neighbor by a '-'.

- Repetition of an element of the pattern can be indicated by following that element with a numerical value or a numerical range between parenthesis. Examples: x(3) corresponds to x-x-x, x(2,4) corresponds to x-x or x-x-x or x-x-x-x.

- When a pattern is restricted to either the N- or C-terminal of a sequence, that pattern either starts with a '<' symbol or respectively ends with a '>' symbol.

- A period ends the pattern.

Examples:

$$\mathtt{PA:}\quad [\mathtt{AC}] - \mathtt{x} - \mathtt{V} - \mathtt{x}(4) - \{\mathtt{ED}\}.$$

This pattern is translated as: [Ala or Cys]-any-Val-any-any-any-any-{any but Glu or Asp}

$$\mathtt{PA:}\quad < \mathtt{A} - \mathtt{x} - [\mathtt{ST}](2) - \mathtt{x}(0,1) - \mathtt{V}.$$

This pattern, which must be in the N-terminal of the sequence ('<'), is translated as: Ala-any-[Ser or Thr]-[Ser or Thr]-(any or none)-Val.

□

Using this syntax for possible patterns in protein sequences, the sequence families can be described. The next example from PROSITE gives a shortened textual description of a particular protein family, called Zinc finger C2H2 family, and its characteristic consensus pattern.

Example 1.2 *The Zinc finger C2H2 family from the PROSITE database.*

Zinc finger domains are nucleic acid-binding protein structures, composed of 25 to 30 amino-acid residues including 2 conserved Cys and 2 conserved His residues in a C-2-C-12-H-3-H type motif. The 12 residues separating the second Cys and the first His are mainly polar and basic, implicating this region in particular in nucleic acid binding. The Zn binds to the conserved Cys and His residues. Fingers have been found to bind to about 5 base pairs of nucleic acid containing short runs of guanine residues. They have the ability to bind to both RNA and DNA, a versatility not demonstrated by the helix-turn-helix motif. The zinc finger may thus represent the original nucleic acid binding protein.

A schematic representation of a zinc finger domain (The two C's and two H's are zinc ligands):

```
           x  x
         x       x
   [LIVMFYWC]     x
        x        x
        x        x
        x        x
         C      H
       x   \  /   x
      x     Zn     x
       x  /    \  x
         C      H
x x x x x        x x x x x
```

```
Consensus pattern: C-x(2,4)-C-x(3)-[LIVMFYWC]-x(8)-H-x(3,5)-H
```

□

Usually the patterns in PROSITE are developed by first aligning the sequences by multiple sequence alignment tools and then manually developing the patterns that seem to be conserved in the right regions of the multiple alignments. Some of the pattern discovery effort can be automatized, however, as illustrated by the following example from the PROSITE database, where a pattern discovered by a computational method has been incorporated into the database.

Example 1.3 *Description of a PROSITE entry* `PS00272; SNAKE_TOXIN`
Snake toxins belong to a family of proteins which groups short and long neurotoxins, cytotoxins and short toxins, as well as other miscellaneous venom peptides. Most of these toxins act by binding to the nicotinic acetylcholine receptors in the postsynaptic membrane of skeletal muscles and prevent the binding of acetylcholine, thereby blocking the excitation of muscles.

Snake toxins are proteins that consist of sixty to seventy five amino acids. Among the invariant residues are eight cysteines all involved in disulfide bonds. A signature pattern[1] was developed (Jonassen, Collins, & Higgins 1995) which includes four of these cysteines as well as a conserved proline thought to be important for the maintenance of the tertiary structure. The second cysteine in the pattern is linked to the third one by a disulfide bond. The four C's are involved in disulfide bonds. The pattern itself is following:

$$\mathtt{G} - \mathtt{C} - \mathtt{x}(1,3) - \mathtt{C} - \mathtt{P} - \mathtt{x}(8,10) - \mathtt{C} - \mathtt{C} - \mathtt{x}(2) - [\mathtt{PDEN}]$$

□

[1] The signature pattern (or characteristic pattern) is a pattern that is common to all or nearly all of the members of the family. Sometimes they are also called a consensus pattern, especially when the pattern is common to all sequences.

Similar types of patterns can also be used for analyzing DNA sequences. The DNA-binding proteins are known to bind to specific parts of DNA, which can be described in terms of sequence motifs. For example, the pattern GGTGGCAA which has been shown to be a proteasome specific control element, discovered both by conventional wet-lab, as well as by *in silico* prediction methods (Mannhaupt *et al.* 1999; Jensen & Knudsen 2000; Vilo *et al.* 2000).

These DNA motifs are often shorter and more restricted than protein family signatures. However, as DNA is a very long molecule, the specificity of the motifs in DNA is usually much weaker than for protein family memberships.

For example, the so called TATA-box, that has a role in defining the transcription start point, is often considered a well-conserved fragment of DNA with consecutive basepairs TATAA. But sometimes the polymerase can also bind other sequence variants, like TATTA, which has one mutation as compared to TATAA. It is useful to also note that not all TATAA-substrings in DNA are the real binding sites for proteins.

We consider the discovery of putative transcription factor binding sites in more detail in Chapters 6 and 7.

1.2.3 Pattern rating functions

Given a family of related sequences, there may exist many patterns that are present in all or nearly all of the sequences. The more complex the pattern language, the more different patterns match at least some of the sequences. It is a challenging task to tell which of these patterns are relevant. For sorting the patterns according to their interestingness and relevance we need formal *fitness measures* that give to each pattern a score that can be used for comparing patterns.

These fitness measures can be based on the specificity and sensitivity of the patterns (see Section 2.7), the information content (Schneider *et al.* 1986; Jonassen, Collins, & Higgins 1995), the ratio (Brazma *et al.* 1998b), the probability statistics (van Helden, André, & Collado-Vides 1998), the minimum description length (MDL) principle (Brazma *et al.* 1997), and others.

Sometimes several simple quality indicators can be presented to users, as in the following example from PROSITE.

Example 1.4 *The quality indicators for the Zinc Finger motif from Example 1.2.*

SWISS-PROT release number: 40.7, total number of sequence entries in that release: 103373;

Total number of hits in SWISS-PROT: 3272 hits in 617 different sequences;

Number of hits on proteins that are known to belong to the set under consideration: 3222 hits in 568 different sequences;

Number of hits on proteins that could potentially belong to the set under consideration: 8 hits in 7 different sequences;

Number of false hits (on unrelated proteins): 42 hits in 42 different sequences;

Number of known missed hits: 6;

Number of partial sequences which belong to the set under consideration, but which are not hit by the pattern or profile because they are partial (fragment) sequences: 2;

Precision (true hits / (true hits + false positives)): 98.71%;

Recall (true hits / (true hits + false negatives)): 99.81%.

□

The aim of different pattern discovery methods is usually to find motifs that are overrepresented in the data set analyzed, or unexpected according to some other criteria. It is possible to count how many sequences contain the motif or how many occurrences of the motif there is in total (*i.e.* count numbers of occurrences within the same sequence). When counting several occurrences within each sequence the occurrences may be overlapping and not independent. Therefore, it is simpler to count just the number of sequences that contain the motif.

The ratio of pattern occurrences in two data sets tells how much more frequent the pattern is in one data set than in another. The problem with ratios is that if the frequencies are small then the ratios may be very high, even though the patterns do not represent meaningful concepts. These high ratios may be slightly compensated by assuming higher expected number of occurrences in the comparison set (Brazma *et al.* 1998b).

Given the background model for the expected number of occurrences, for example from the explicit counting of pattern occurrences in comparison data, one can estimate how many occurrences of each pattern to expect. This estimate can be used to calculate how probable the actual number of occurrences is (assuming the same background model) based on binomial or hypergeometric distribution, for example. Binomial distribution assumes independent random trials and allows to calculate the probability to observe each pattern at least a given number of times in the data. Binomial distribution has been used, for example, in (van Helden, André, & Collado-Vides 1998; Vilo *et al.* 2000). Hypergeometric distribution corresponds to selection without replacement, *i.e.* the probabilities depend on previous outcomes. For large data sizes and small numbers of trials binomial distribution approximates well the hypergeometric distribution. Hypergeometric distributions have been used for example in (Jensen & Knudsen 2000; Barash, Bejerano, & Friedman 2001; Palin *et al.* 2002).

A statistical measure for ranking patterns, Z-score (”normal deviate” or ”deviation in standard units”), which measures by how many standard deviations the number of occurrences of patterns in the sequences exceeds its expected number of occurrences, was studied by (Sinha & Tompa 2000). Their method for ranking patterns uses a Markov model generated from upstream sequences to establish the expected probabilities for patterns. The algorithm itself enumerates all possible patterns and tabulates their numbers of occurrences. Next, all patterns are ranked based on the Z-scores and the best patterns are output. This measure is however relatively time-consuming to calculate.

When calculating the total number of occurrences for patterns, *i.e.* possibly several occurrences per one sequence, one can in principle use the same statistical criteria. However, one has to be aware of the possibility that pattern occurrences may be overlapping and thus not independent. The cyclic patterns, *i.e.* patterns that can have an overlap with themselves, have a higher expected number of occurrences even under the assumption that all nucleotides have equal and independent probability of occurrence at each position.

Apostolico *et al.* (Apostolico *et al.* 2000) have developed methods to calculate different pattern rating scores, like mean and variance of pattern occurrences, and some derived significance measures in an optimal fashion using a suffix tree algorithm. In this way unusually frequent or infrequent words can be detected efficiently.

Measures like the sensitivity and specificity of the pattern (see Example 1.4) as well as the correlation coefficients (Brazma *et al.* 1998a) can also be readily applied to motif discovery, although for promoter analysis our knowledge about true positives (genes regulated depending on the presence of the motif) and true negatives (genes not regulated based on the motif) is largely missing.

A probabilistic model for segmenting strings into ”words” and concurrently building a ”dictionary” of these words was introduced (Bussemaker, Li, & Siggia 2000a; 2000b). The pattern rating tells which of these substring patterns to merge into bigger words, to be stored in the dictionary. In this statistical approach background probabilities (negative examples) are not used, yet the algorithm also reports words which occur infrequently in the data set analyzed.

1.2.4 Pattern discovery algorithms

Based on the algorithmic component, pattern discovery methods can be classified into 1) sequence driven, mostly alignment based approaches, and 2) pattern driven enumerative approaches, where the search algorithm evaluates the presence of each pattern by counting the numbers of their occurrences. Pattern driven approaches can be performed intelligently so that patterns that are not present in the data are not generated. For example, if a pattern π is not present frequently in the data, then no refinement that makes π more specific (hitting in even fewer places) can be frequent in the data either.

For probabilistic methods, some of the widely used motif finding methods are, for example, Gibbs Motif Sampling (Lawrence *et al.* 1993; Rocke & Tompa 1998), Expectation Maximization (Bailey & Elkan 1995), maximization of the information content (Wolfertstetter *et al.* 1996; Roth *et al.* 1998). Of these AlignACE (Roth *et al.* 1998) has been optimized for alignment of DNA sequences by automatic consideration of patterns possibly on both strands and for finding multiple motifs per sequence via an iterative masking procedure.

The methods developed at the IBM Bioinformatics Research Group for discovery of patterns in biological sequences (Rigoutsos & Floratos 1998b; Hart *et al.* 2000a), operate in two phases: scanning and convolution. During the scanning phase, elementary patterns with sufficient occurrence frequency are identified. These elementary patterns constitute the building blocks for the convolution phase. They are combined into progressively larger and larger patterns until all the existing, maximal patterns have been generated.

Some of the most efficient algorithms capable of discovering discrete patterns such as substrings of any length, are based on the suffix tree data structure (Weiner 1973; McCreight 1976; Ukkonen 1995). Suffix trees are used to index texts (sequences) in a way so that query times would not depend on the size of the indexed text. In the suffix tree all possible subwords can be read from the top of the tree-structured index regardless of original text size. There are many bioinformatics applications of suffix trees (Gusfield 1997; Bieganski *et al.* 1994; Gusfield, Landau, & Schieber 1992). The direct link to pattern discovery methods is given by the fact that all possible substrings (patterns) are presented in this tree structure. Suffix tree based approaches and extensions thereof have been used for promoter analysis by several groups, e.g. (Brazma *et al.* 1998b; Marsan & Sagot 2000; Jensen & Knudsen 2000; Lonardi 2001). For example a tool Verbumculus (Lonardi 2001; Apostolico, Bock, & Lonardi 2002) has been used for identifying substring patterns using suffix trees and visualizing the interesting patterns according to many different statistical criteria.

For frequent substring discovery and graphical analysis of word frequencies, an interactive tool Xlandscape (Levy *et al.* 1998) based on the suffix array construction algorithm (Manber & Myers 1990) has been used. The tool permits to study the sequence landscape – frequency of all words in the query sequence that can be found in database. A more traditional use of suffix arrays is to build compact full-text indexes. For example, in (Gonnet & Knecht 1996) the PAT indexes are built for searching the text databases.

In this thesis we demonstrate some ways how the methods motivated by the suffix trees can be applied for pattern discovery from (bio)sequences. For the discovery of the most frequent patterns we have modified the writeonly-topdown or simply *wotd*-algorithm for constructing the suffix trees (Giegerich & Kurtz 1995;

Giegerich, Kurtz, & Stoye 1999). Note that in the earlier work this method was called lazy suffix tree construction as it was originally designed for lazy functional programming language implementations (Giegerich & Kurtz 1995).

This approach is simple and easily modifiable, as different branches of the suffix tree can be constructed independently from each other. In the *wotd* implementation style only those branches of the suffix tree need to be constructed which are actually accessed by search procedures. Traditional linear-time algorithms (Weiner 1973; McCreight 1976; Ukkonen 1995) maintain complex data structures and they all construct the tree in a very specific order, thus making modifications into the search order hard or impossible.

For the clarity of presentation we have based our algorithms on a suffix trie structure that consumes $O(n^2)$ size in the worst case, where n is the total length of input sequences. While in the compact suffix tree representation each node is a branching node and each label on the edges can represent several characters, in the trie structure each label can represent only one character and nodes need not to be branching, *i.e.* they may have only a single child.

1.3 Contributions of this work

We have developed a new algorithm called SPEXS for discovering frequently occurring patterns from sets of sequences. SPEXS generates a pattern trie while maintaining information about the occurrences of each pattern. Patterns are constructed incrementally by expanding the prefixes of the frequent patterns. Only the patterns that do occur in input strings frequently enough, are generated and analyzed. Pattern classes that can be generated in this way include the substring patterns, substring patterns containing group characters (i.e. positions where alternative characters from a given list can be used), and patterns containing so-called wildcard positions. SPEXS is able to take as input multiple sets of sequences and to construct the patterns according to user-given pattern specification while tracking simultaneously the occurrences in each input set. Based on these occurrences different fitness functions can be used for evaluating the interestingness of discovered patterns.

A methodology for combining the discovery of the potentially coregulated sets of genes using clustering of gene expression profiles followed by pattern discovery from the upstream sequences to the genes in these clusters has been developed, and the respective experiments have been carried out in practice. The SPEXS algorithm has been used extensively for discovering putative transcription factor binding sites in gene upstream sequences.

A method for further analysis of the discovered patterns by large-scale co-visualization of gene expression, DNA sequence, and pattern data has been developed. This helps investigators in assessing the quality of the *in silico* predictions.

Additionally, we have developed a novel algorithm for discovering approximately matching patterns where the matches are not expected to be always perfect. We describe a new method for matching a pattern approximately against the suffix tree. In that approach we generalize the exact matching of the substring patterns against the suffix tree so that matching can be done with errors, i.e. substitutions, insertions, and deletions. Instead of computing the explicit dynamic programming table for all branches of the suffix tree, the approximate matching is simulated by treating the lists of nodes in the suffix tree as entries in the dynamic programming table.

Pattern discovery methods based on exhaustive search often output a large number of patterns, even if the most stringent quality criteria have been used for thresholding the interesting patterns. Clustering methods and other methods have been shown to be useful for facilitating further analysis of the set of patterns output by discovery algorithms. We provide a short introduction to these methods and show their applicability in real-world data analysis.

A novel pattern fitness measure based on the Minimum Description Length (MDL) principle has been developed that allows for constructing the union of patterns covering the set of input sequences. The experimental implementation of this pattern discovery method has been shown to produce biologically relevant clustering (resembling the evolutionary relationship between the sequences) of the input sequences.

For analyzing known or predicted transcription factor binding sites and their co-occurrences in the genome, a data mining approach that attempts to identify sets of co-occurring patterns has been developed. This method can also be used with the automatically predicted binding sites, to extract more meaningful knowledge about the regulatory mechanisms.

A well-studied, but previously unsolved problem of predicting G-protein coupling specificity to their respective G-protein coupled receptor (GPCR) proteins by the GPCR sequence alone has been shown to be solvable at least to a certain extent. Patterns present in the sequences corresponding to the internal loops of the transmembrane GPCR proteins, and showing the specificity toward the binding by different G-proteins from G_s , $G_{i/o}$, or G_{q11} classes, were discovered by the SPEXS algorithm.

A major result of this work is an extensive software package called Expression Profiler (http://ep.ebi.ac.uk/). This package integrates a collection of computational methods (including SPEXS) into a user-friendly web-based tool for the analysis of functional genomics data. It is briefly described in Chapter 7 and (Vilo *et al.* 2003; Vilo, Kapushesky, & Kemmeren 2002).

Many of the results of this thesis have appeared in a more complete form in the original articles related to this work (Brazma, Ukkonen, & Vilo 1996; Brazma *et al.* 1996; 1997; 1998b; 1998c; Brazma & Vilo 2000; Vilo *et al.* 2000; Vilo & Kivinen 2001; Möller, Vilo, & Croning 2001; Kemmeren *et al.* 2002; Vilo *et al.* 2003).

1.4 Structure of the thesis

The thesis is structured as follows. First we introduce definitions in Chapter 2. Next, the SPEXS algorithms are described in Chapter 3, and the discovery of approximately matching patterns in Chapter 4. The pattern analysis and post-processing is described in Chapter 5. Different applications of SPEXS for bioinformatics analysis are described in Chapter 6. The software tools are described in Chapter 7, and finally, the conclusions are presented in Chapter 8.

Chapter 2

Definitions

Pattern discovery, as we consider it in the current thesis, deals with methods for finding regularities in sequences. In this chapter we define the concepts of sequences, patterns, pattern classes and provide the basic framework used later for the design of algorithms for pattern discovery.

2.1 Strings

We use Σ to denote a finite set of characters, an *alphabet*. The *size* of the alphabet Σ is $|\Sigma|$. Any sequence $S = a_1 a_2 \ldots a_n$, such that $n \geq 0$ and each a_i is in Σ, is called a *string* (or *sequence*, or *word*) over the character set Σ. The *length* $|S|$ of the string S is n. The string of length 0, *i.e.* an empty string, is denoted by λ. The set of all possible strings over Σ is Σ^*.

We identify individual characters by their positions within the string. The character a_i at the position i can also be denoted by $S[i]$. Character positions of a non-empty string S are in the range $1 \leq i \leq |S|$, *i.e.* the first character of the string is at position 1, and the last character is at position $|S|$.

Consecutive characters $a_i \ldots a_j$ of S form a *substring* of S that starts from position i and ends at position j. We denote this substring by $S[i..j]$, where $1 \leq i \leq j \leq |S|$. An alternative definition which does not use character positions within the string states that x is a substring of S if $S = yxz$ for some strings y and z.

A substring $S[i..i]$ has length 1 and corresponds to the character a_i at position i. A substring $S[i..j]$ has length $j - i + 1$. We say that substring $S[i..j]$ *occurs* at the *position* or *location* j of the string S. We say that a substring x has multiple occurrences in S if $x = S[i..j] = S[i'..j']$, and $j \neq j'$.

2.2 Patterns

We follow closely the definitions of the generalized regular patterns (Brazma *et al.* 1998c).

Let $\Sigma = \{a_1, \ldots, a_m\}$ be the *basic alphabet* and let $L(a_i) = \{a_i\}$ be the *language* defined by a_i.

Let $g_1, \ldots, g_n$ be non-empty subsets of Σ such that each subset contains more than one element. For naming the subsets g_i we use another alphabet, $\Gamma = \{b_1, \ldots, b_n\}$, disjoint from Σ. We define the language $L(b_i) = g_i$, and call characters b_i the *group characters*.

Let $*$ be a symbol not in $\Sigma \cup \Gamma$. Define $L(*) = \Sigma^*$. We call $*$ the *wild-card of unrestricted length*.

Let $*(k, l)$ define the language $L(*(k, l)) = \cup_{i=k}^{l} \Sigma^i$, *i.e.* $L(*(k, l))$ is the set of all words over Σ with the length between k and l characters. We call $*(k, l)$ the *restricted length wild-card of the length between k and l*. Let X be the set $\{*(k, l) \mid 0 \leq k \leq l < \infty\}$ of restricted length wildcards.

We define the *generalized regular pattern*, or for short, *pattern* π as a string over the alphabet $\Sigma \cup \Gamma \cup \{*\} \cup X$. We define the language $L(\pi)$ of the pattern $\pi = c_1 \ldots c_r$, where $c_i \in \Sigma \cup \Gamma \cup \{*\} \cup X$, as

$$L(\pi) = \{\ \zeta_1 \ldots \zeta_r \ \Big|\ \zeta_1 \in L(c_1), \ldots, \zeta_r \in L(c_r)\ \}.$$

We say that pattern π *matches* the string α, if a substring β of α exists such that $\beta \in L(\pi)$. Locations of the occurrences of pattern π are defined by occurrences of the substrings β in α. Note that if $\beta \in L(\pi)$, then $\alpha \in L(*\pi*)$.

We define the *union of patterns* as an expression of the type $\pi_1 + \ldots + \pi_k$, where π_i ($1 \leq i \leq k$) is a pattern and $+ \notin \Sigma \cup \Gamma$. The language of the union is defined as

$$L(\pi_1 + \ldots + \pi_k) = L(\pi_1) \cup \ldots \cup L(\pi_k).$$

Note that the languages of the generalized regular patterns belong to the class of regular languages, the language class at the level 3 of the Chomsky language hierarchy.

We call π a *substring pattern* if $\pi = c_1 \ldots c_r$, and each $c_i \in \Sigma$. For example, given an alphabet $\Sigma = \{\mathtt{A}, \mathtt{T}, \mathtt{C}, \mathtt{G}\}$ of the DNA, where letters correspond to the nucleotides, the pattern AAGA is a substring pattern.

We call π a *substring pattern with group characters* if $\pi \in (\Sigma \cup \Gamma)^*$. Given the alphabet Σ as above, and a group $b \in \Gamma$, $L(b) = g = \{\mathtt{G}, \mathtt{T}\}$, the pattern $\mathtt{AA}b\mathtt{A}$ is an example of such substring pattern with group characters. We often represent the group character positions b in the pattern π by a bracketed list of all characters in $L(b) = g$. Hence the above pattern becomes $\mathtt{AA[GT]A}$. We use the symbols $b_i \in \Gamma$ as the names for character groups g_i. For notational convenience, we treat g_i as equal to b_i and say $g_i \in \Gamma$.

2.3 Pattern classes

2.3.1 Definition of pattern classes P1–P6

Our aim is to discover "interesting" patterns from the input data. The first task is to define the pattern language $\mathcal{P}$, the search space for the pattern discovery problem. We define the following pattern classes using the notations from the previous section.

P1 = $\{\pi \mid \pi \in \Sigma^*\}$ – substring patterns

P2 = $\{\pi \mid \pi \in (\Sigma \cup \Gamma)^*\}$ – substring patterns with group characters

P3 = $\{\pi \mid \pi \in \Sigma^+(*\Sigma^+)^*\}$ – patterns with wildcards of unrestricted length

P4 = $\{\pi \mid \pi \in \Sigma^+(*(k,l)\Sigma^+)^*$, where $*(k,l) \in X\}$ – patterns with wildcards of restricted length

P5 = $\{\pi \mid \pi \in (\Sigma \cup \Gamma)^+(*(\Sigma \cup \Gamma)^+)^*\}$ – patterns with group characters and wildcards of unrestricted length

P6 = $\{\pi \mid \pi \in (\Sigma \cup \Gamma)^+(*(k,l)(\Sigma \cup \Gamma)^+)^*\}$ – patterns with group characters and wildcards of restricted length

Example 2.1 DNA sequences can be considered as strings over the alphabet of four letters that represent the nucleotides, $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$. The character set $\Gamma = \{[\mathtt{AC}], [\mathtt{AG}], [\mathtt{AT}], [\mathtt{CG}], [\mathtt{CT}], [\mathtt{GT}], [\mathtt{ACG}], [\mathtt{ACT}], [\mathtt{AGT}], [\mathtt{CGT}], [\mathtt{ACGT}]\}$ contains all the possible groups of characters from Σ. The alphabet $\Sigma \cup \Gamma$ can be used for defining substring patterns with group characters.

□

2.3.2 Choice of group characters Γ

The number of all possible subsets of characters in Σ grows exponentially. The set of group characters Γ is called a *partitioning* of Σ if $\bigcup_{g_i \in \Gamma} g_i = \Sigma$, and $g_i \cap g_j = \emptyset$ for all $i \neq j$. In other words, if no group intersects with others and every character from Σ belongs to exactly one of the groups in Γ.

In practice, the set of useful character groups depends on the application domain, hence it is reasonable to assume that users provide the set of meaningful groups Γ. In biosequences, i.e. the DNA and amino acid sequences, the character groups in Γ are usually chosen based on the physico-chemical properties of nucleotides or amino acids respectively (see Table 2.2).

Alphabet indexing is a mapping from Σ to a smaller alphabet. It replaces symbols in Σ by symbols from Γ that is a partitioning of Σ. It has been shown that in some cases this technique does not lose any essential information as compared to original strings over Σ (Shimozono *et al.* 1993). This mapping can be done as a preprocessing step.

In the current thesis we assume that the groups Γ are given by users, and that the groups can be overlapping, i.e. they do not have to be a partitioning of the alphabet.

Example 2.2 Protein sequences are strings over the alphabet of amino-acids $\Sigma = \{\mathtt{A}, \mathtt{R}, \mathtt{N}, \mathtt{D}, \mathtt{C}, \mathtt{Q}, \mathtt{E}, \mathtt{G}, \mathtt{H}, \mathtt{I}, \mathtt{L}, \mathtt{K}, \mathtt{M}, \mathtt{F}, \mathtt{P}, \mathtt{S}, \mathtt{T}, \mathtt{W}, \mathtt{Y}, \mathtt{V}\}$ (see Table 2.1). Amino acids can be grouped by their properties to small, large, hydrophobic, hydrophilic, charged, or neutral etc. (see Table 2.2). According to this classification, $\Gamma = \{[\mathtt{AGILPV}], [\mathtt{HFWY}], [\mathtt{DE}], [\mathtt{RHK}], [\mathtt{ST}], [\mathtt{CM}], [\mathtt{NQ}], [\mathtt{ILV}]\}$ defines one set of possible biologically substantiated groupings. □

Name	Three-letter code	1-letter code
Alanine	Ala	A
Arginine	Arg	R
Asparagine	Asn	N
Aspartic acid	Asp	D
Cysteine	Cys	C
Glutamine	Gln	Q
Glutamic acid, Glutamate	Glu	E
Glycine	Gly	G
Histidine	His	H
Isoleucine	Ile	I
Leucine	Leu	L
Lysine	Lys	K
Methionine	Met	M
Phenylalanine	Phe	F
Proline	Pro	P
Serine	Ser	S
Threonine	Thr	T
Tryptophan	Trp	W
Tyrosine	Tyr	Y
Valine	Val	V

Table 2.1: Twenty amino acids, their three- and IUPAC single-letter codes.

2.3.3 Alternative pattern representations

The language to represent patterns that we have outlined is essentially the same as that used in the PROSITE database, described in Example 1.3. A slightly different syntactic representation for the same pattern language may be used by different

Amino acid properties	Amino acids in each group	1-letter codes
Aliphatic	Ala, Gly, Ile, Leu, Pro, Val	A G I L P V
Aromatic	His, Phe, Trp, Tyr	H F W Y
Acidic	Asp, Glu	D E
Basic	Arg, His, Lys	R H K
Hydroxylic	Ser, Thr	S T
Sulphur-containing	Cys, Meth	C M
Amidic (containing amide group)	Asn, Gln	N Q
Polar	Arg, Asn, Asp, Cys, Gln, Glu, His, Lys, Ser, Thr, Tyr	R N D C Q E H K S T Y
Hydrophobic	Ala, Cys, Gly, Ile, Leu, Met, Phe, Pro, Trp, Tyr, Val	A C G I L M F P W Y V
Hydrophilic	Arg, Asn, Asp, Gln, Glu, His, Lys, Ser, Thr	R N D Q E H K S T
Charged	Arg, Asp, Glu, His, Lys	R D E H K
Positive charge	Arg, His, Lys	R H K
Negative charge	Asp, Glu	D E
Neutral	Ala, Asn, Cys, Gln, Gly, Ile, Leu, Met, Phe, Pro, Ser, Thr, Trp, Tyr, Val	A N C Q G I L M F P S T W Y V

Table 2.2: The amino acids grouped according to their physico-chemical properties.

authors or by different tools and databases. In practical applications we prefer the syntax of regular expressions as used in UNIX, perl, egrep, and less, for example.

Example 2.3 Consider the Snake Toxin motif from the PROSITE database as described in Example 1.3:

$$\mathtt{G-C-x(1,3)-C-P-x(8,10)-C-C-x(2)-[PDEN]}.$$

In our pattern representation language, if we use a single period to represent a group $g = \Sigma, g \in \Gamma$ consisting of all possible amino acids, the above pattern would read

$$\mathtt{GC*(1,3)CP*(8,10)CC..[PDEN]}$$

For many software tools that have built-in regular expression matching capabilities, the above pattern is represented as

$$\mathtt{GC.\{1,3\}CP.\{8,10\}CC..[PDEN]}$$

□

2.3.4 Characterizing the language of patterns and language described by each pattern

To understand the complexity of the the pattern discovery problem we need to understand the size of the possible search spaces involved. Given a language $\mathcal{P}$ for defining the patterns π, we can ask the following questions. How many different patterns of length r can one construct? Given a pattern $\pi \in \mathcal{P}$, how many strings α belong to the language $L(\pi)$? Given a string α, how many different patterns $\pi \in \mathcal{P}$ are there such that $\alpha \in L(\pi)$?

Given a string S and two different-looking patterns π_1 and π_2, these two patterns may match at exactly the same locations of S despite the fact that in general $L(\pi_1) \neq L(\pi_2)$. Moreover, an individual pattern π may match the same string S in different ways, for example different overlapping substrings of S may belong to $L(\pi)$, or the correspondence between the character positions in the sequence S and pattern π can be established in different ways. The ambiguity of pattern matching is considered later in Chapter 3.

Given a character set Σ and a set of group characters Γ, there are $|\Sigma|^r$ substring patterns, $|\Sigma \cup \Gamma|^r$ substring patterns with group characters, $|\Sigma|^2(|\Sigma|+1)^{r-2}$ substring patterns with unrestricted length wildcards (assuming that the first and last positions of the pattern are not wildcards). For wildcards of restricted length there are many possible wildcard representations depending on k and l, the minimum and maximum wildcard lengths, respectively.

Given a pattern $\pi = c_1 \ldots c_r \in \mathcal{P}$, the size of $L(\pi)$, where $L(\pi) = \{\ \zeta_1 \ldots \zeta_r \ \big|\ \zeta_1 \in L(c_1), \ldots, \zeta_r \in L(c_r)\ \}$, can be as much as $|L(c_1)| \cdot |L(c_2)| \cdot \ldots \cdot |L(c_r)|$. The number of different strings $\alpha \in L(\pi)$ grows exponentially in the number of group character positions in π. The size of $L(c_i)$ where c_i is one of the wild-card positions, grows exponentially in the length of the wild-card.

As the size of the pattern class as well as the number of sequences belonging to the language represented by a single pattern both grow exponentially for complex pattern classes, the simple enumerative methods for discovering such patterns are not feasible. However, if we are asked to find patterns that are frequent in a given set of sequences, we have the following question. Given a string $\alpha \in \Sigma^*$, how many different patterns $\pi \in \mathcal{P}$ there are such that π has at least K occurrences in α. The answer to this question is a basis for algorithm complexity estimations in the following chapters, and is considered for each pattern class and problem type separately.

2.4 Pattern discovery problem

We define V to be a set of possible labelings for representing concepts. We consider strings over an alphabet Σ that belong to classes represented by $\sigma \in V$. Typically V consists of two elements, which we often represent as $+$ and $-$, or in other words, $\sigma \in \{+,-\} = V$.

Assume that a family of related sequences F_+ exists. We denote the rest of the sequences by F_-. I.e., $F_- = \Sigma^* - F_+$ and $\Sigma^* = F_+ \cup F_-$. Let f be a function $f : \Sigma^* \to \{\text{FALSE}, \text{TRUE}\}$ assigning Boolean values to strings, we call such a function a *string function*. The string function f is a *characteristic function* for the family F_+ if

$$f(s) = \begin{cases} \text{TRUE} & \text{if } s \in F_+ \\ \text{FALSE} & \text{if } s \in F_- \end{cases}$$

We would like to find automatically the string functions that are characteristic for a given family, say F_+. Unfortunately, we do not even know the full set of sequences in F_+, but only a *sample* $S \subset F_+$ (*i.e.* some members of the family F_+). We denote the sample as a *set of examples* or simply *examples*. When there are two classes in V, we say that sample S consists of positive (S_+) and negative (S_-) examples, $S = S_+ \cup S_-$. This is illustrated in Figure 2.1.

The string function $f(s)$ that is characteristic for a specific family may not be computable in principle, if the family membership is determined by non-computable decisions, or it may be uncomputable based on our limited knowledge about the domain. For example, string functions that are able to determine a set of proteins (represented by their primary sequences) that have certain 3-D structure features and interact with another set of proteins, may be well beyond our current scientific capabilities.

The string functions we are interested in in this thesis are expressed in terms of patterns π matching or not matching the sequences s respectively. We consider string functions $f_\pi(s)$, alternatively represented as $f(\pi, s)$,

$$f_\pi(s) = f(\pi, s) = \begin{cases} \text{TRUE} & \text{if } \pi \text{ matches the string } s \\ \text{FALSE} & otherwise \end{cases}$$

The basic problem in pattern discovery is, given a sample from family F_+ and, possibly, a sample from F_-, to automatically find the string functions f_π that approximate the characteristic function f for the family F_+ and so finding a representation π for set F_+.

The motivation for pattern discovery arises from the need to discover common (syntactic) features to the sets of semantically related sequences. For example, patterns conserved across many sequences in a protein family may reveal functionally or structurally relevant features. Motifs conserved in regulatory regions of genes sharing similar expression profiles potentially carry some information for gene regulation machinery, for example, binding sites for DNA-binding transcription factors, structural elements of DNA, or other signals.

We can separate two main types of the pattern discovery problem.

Family classification problem. Assume that we have positive and negative examples, $S_+ \subseteq F_+$ and $S_- \subseteq F_-$. The goal is to find a string function approximating the characteristic function for the family F_+. That goal has two parts. We want to find compact “explanations” of known sequences, and we want to predict the family membership of yet unknown sequences.

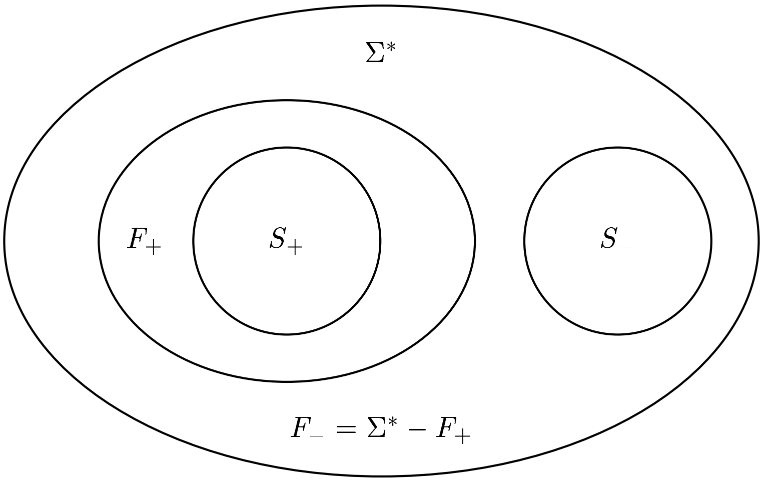


Figure 2.1: The family F_+ of related sequences, and samples S_+ and S_- graphically.

Family conservation problem. Assume that we know only positive examples $S_+ \subseteq F_+$ from the family. The goal is to find features characteristic to the sequences in S_+. The most interesting string functions in this case are the ones that have low probability of returning TRUE for random[1] sequences and high probability to return TRUE for the sequences in F_+. This is similar to the family classification problem with the sample S_+, S_-, where the sample S_- consists of randomly chosen strings from $\Sigma^* - F_+$. If S_- contains sequences that are very similar to sequences in F_+, the family classification problem should be solved instead of the family conservation problem.

Note that the patterns discovered from the family conservation problem are often used for family classification. For example, for a new sequence one can ask, which of the motifs that are conserved for different families are present in the new sequence. Based on these motifs the new sequence can be suggested to belong to the same families. Perhaps the main reason to use the family conservation problem for a given set of sequences is, that often we do not know the representative enough set of negative examples, that could be used to guide the search in the way as for the family classification problem.

[1] We assume some model for generating random sequences. Most commonly we assume that symbols in the sequences are independently distributed. The frequencies of individual symbols can be either equal to each other or they can be estimated from the sample or some database of sequences.

2.5 Solving the pattern discovery problem

The pattern discovery problem can be divided into three separate subproblems according to the following paradigm (Brazma *et al.* 1998a):

1. **Choose the class $\mathcal{P}$ of patterns** that will be used for family classification or conservation studies (i.e., the hypotheses space).

2. **Design a *fitness measure* $\mathcal{F}$** that, given (a) a set of sequences, samples for classification or conservation problem, and (b) a pattern, returns a rating of this pattern in respect to the examples.

3. **Develop an algorithm $\mathcal{A}$** that, given a set of examples, returns patterns $\pi \in \mathcal{P}$ with high ratings with respect to $\mathcal{F}$.

In the current thesis we have developed methods for discovering patterns from the language of generalized regular expressions. The features of biological significance may well be more complex than simple regular patterns, however there are many examples that these pattern classes are often sufficient to carry at least some biologically significant information.

Once we have chosen the pattern search space $\mathcal{P}$, it is natural to try out different fitness measures $\mathcal{F}$. Different data sets may require fitness measures tailored according to the specific needs of the particular problem. For example, in some cases researcher may be interested in long conserved patterns stretching along the full length of sequences, and in other cases it is interesting to find the minimal yet significant differences between two sets of sequences.

We would like to avoid the need to design a different algorithm for each different fitness measure. Instead, we want to use efficient generic search strategies that would allow to incorporate alternative fitness measures. In order to have a generic framework, we prefer exhaustive search strategies to be sure that the "best" patterns are always found. Alternatively, for some fitness measures individually tailored search heuristics could be used that may work better for the particular pattern language and fitness measure.

To be able to perform the exhaustive search in the reasonable time we have chosen the approach where the pattern language can be specified on an individual case to case basis depending on the problem statement and data size. We want to have enough flexibility in the pattern language itself, yet also to be able to restrict the search space for larger data sizes. And finally, although several different pattern fitness measures could be used, the real relevance of the discovered patterns has to be justified in the biological terms.

2.6 Problem types considered in the current study

In this thesis we will study the following types of pattern discovery problems for different pattern classes.

Problem type A: Given a string $S \in \Sigma^*$ and an integer K, construct all patterns π (of type P1–P6) such that π has at least K occurrences in S.

Problem type B: Given a set of n strings $S_1, S_2, \ldots, S_n$, $S_i \in \Sigma^*$, and an integer K, construct all patterns π (of type P1–P6) such that π has at least one occurrence in at least K sequences of $S_1, S_2, \ldots, S_n$.

Problem types A and B are useful in restricting the search space to most frequently occurring patterns. Given an exhaustive search strategy that is able to generate all possible patterns according to problem types A and B, the potential list of patterns produced may be huge. On their own these problem statements do not tell us which of the frequently occurring patterns are the most "interesting". Thus, we need to define an order or ranking in which to present the patterns, as well as a strategy to establish which cut-off values produce all the relevant patterns, and how to summarize them. We order the patterns based on the fitness measure. The pattern that has the best fitness value is most interesting.

Problem type C: Given a string $S \in \Sigma^*$, find all patterns π (of type P1–P6) in the decreasing order of fitness $\mathcal{F}$.

Problem type D: Given a set of n strings $S_1, S_2, \ldots, S_n$, $S_i \in \Sigma^*$, find all patterns π (of type P1–P6) in the decreasing order of fitness $\mathcal{F}$.

Problem type E: Given two sets of strings S_+ and S_-, find all patterns π (of type P1–P6) such that π has significantly more occurrences in S_+ than in S_-.

Our aim is to provide a generic framework for algorithms that can be used for discovering patterns according to problem type specifications A–E. We want the algorithms to be exhaustive in the sense that given the precise definition of the pattern language, the search is guaranteed to find all patterns satisfying the given criteria. Pattern fitness scores can be used for ranking the patterns and the aim is to find the patterns with the best fitness measure.

2.7 Rating functions for family classification problem

The question of which patterns are the most interesting can depend on the application area. One simple criteria is of course that the most frequent patterns are more interesting, as they probably tell us more about the general features of the strings. On the other hand, frequent patterns tend to be too general, and not specific enough. Thus, we are interested in the unexpectedly frequent patterns. The less probable the occurrences are, the more interesting they are.

For the problem type E, we are given sets of sequences from different input classes, usually a sample of positive and negative examples. The task of the pattern discovery algorithm is to find patterns that are most predictive for the class prediction.

Given a pattern π, let $TP = |L(\pi) \cap F_+|$ be the number of *true positives*, $TN = |F_- - L(\pi)|$ the number of *true negatives*, $FP = |L(\pi) \cap F_-|$ the number of *false positives*, and $FN = |F_+ - L(\pi)|$ the number of *false negatives*. In practice all the members of the families F_+ and F_- are not known and one has to estimate these numbers from the currently available data, often the S_+ and S_-, respectively. Some of the commonly used fitness measures are:

- Sensitivity $TP/(TP + FN)$
- Specificity $TN/(TN + FP)$
- Positive Predictive Value PPV $TP/(TP + FP)$
- 2x2 correlation coefficient

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

In practice, the choice of the negative sample S_- can be complicated. These sequences can be chosen based on the biological question one wants to study, they may be generated according to some probabilistic generative model, or they can be chosen from the real data representing the “real” background.

We discuss other fitness measures later in Chapters 6 and 7, in relation to practical applications.

2.8 Suffix trees and substrings of the string

In the current thesis we study two approaches for solving the pattern discovery problems outlined earlier. The first is based on the writeonly-topdown algorithm (*wotd*-algorithm) (Giegerich & Kurtz 1995; Giegerich, Kurtz, & Stoye 1999) that we have generalized for constructing all the patterns from the defined pattern language. The second pattern discovery approach introduced in current thesis utilizes the previously built suffix tree for the pattern discovery process.

Both methods are representatives of the exhaustive search algorithms in the sense that all patterns from the predefined pattern language are discovered and reported. Efficient pruning of the search space guarantees that only these patterns that are frequently present in input data, are constructed and evaluated.

Every string S of length n has exactly $n + (n-1) + \ldots + 1 = \frac{n^2+n}{2} = O(n^2)$ substrings $S[i..j]$, where $1 \leq i \leq j \leq |S|$. Depending on S, some of these substrings can be equal to each other. Surprisingly, it is possible to enumerate and index all of them in $O(n)$ time by constructing a data structure called suffix tree. Suffix tree is a variant of PATRICIA trees (Morrison 1968). There are several different linear-time algorithms for constructing suffix trees (Weiner 1973; McCreight 1976; Ukkonen 1995); the textbook (Gusfield 1997) gives a clear exposure of these methods.

Definition 2.4 (Gusfield 1997) A suffix tree T for an n-character string S is a rooted directed tree with exactly n leaves numbered 1 to n. Each internal node, other than the root, has at least two children and each edge is labeled with a nonempty substring of S. No two edges out of a node can have edge-labels beginning with the same character. The key feature of the suffix tree is that for any leaf i, the concatenation of the edge-labels on the path from the root to leaf i exactly spells out the suffix of S that starts at the position i. That is, it spells out $S[i..n]$. □

Figure 2.2 gives an example of a suffix tree. The size of the suffix tree for S is linear in $|S|$. This can be seen by the following argumentation. By introducing a special end-of-string character \$, all suffixes of $S\$$ end in different leaves. As there are exactly n leaves and every internal node is branching, there are at most $2n$ nodes in the tree. Each node and edge can be implemented with a size $O(1)$ by representing each edge label with two pointers to the corresponding substring of S. Hence the size of the compact suffix tree is $O(n)$.

The ability to construct suffix trees in linear time and space makes it a very attractive starting point for further algorithmic developments. On the other hand, the full suffix tree construction may not be necessary when we want the patterns to occur very frequently in sequences. The *wotd*-algorithm was first developed for lazy functional programming languages that evaluate the code only when it is really needed, *i.e.* the suffix tree branches are calculated when they are needed during the tree traversal. This approach requires ideally that different branches of the tree can be generated independently from each other. The *wotd* approach to suffix tree generation can be more attractive in this case, as it does not make any “unnecessary” work. Although the *wotd*-algorithm is a $O(n^2)$ time algorithm for the text of length n, it has been shown sometimes to outperform the linear-time algorithms (Giegerich, Kurtz, & Stoye 1999).

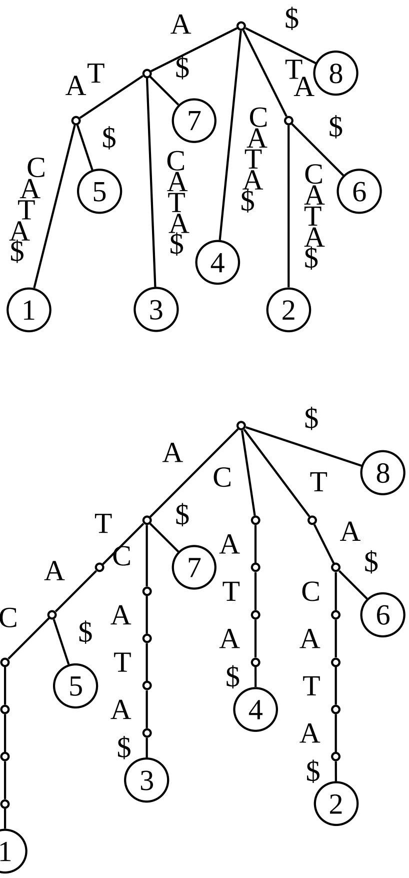
The *string-length* of an edge label in the suffix tree is the length of the string labeling the edge (even though the label is compactly represented in constant space). The *length of a suffix tree* is the sum of the string-lengths of all the edges of the suffix tree.

We will use the simplified, but quadratic-size version of the suffix tree called *suffix trie*. A suffix trie for S is a standard trie structure (Morrison 1968) that represents all the suffixes of S. It differs from the suffix tree in that every edge-label is a symbol in Σ, *i.e.* it has length exactly one. At the same time we loose the condition that every node has to have at least two children. Therefore, the size of the suffix trie can be quadratic in $|S|$ (see Figure 2.2).

The suffix trie lists all the suffixes on the paths starting from the root and ending in the leaves. Each leaf contains the starting position of the particular suffix. Each substring is a prefix of some suffix. Hence, all substrings of S are enlisted on the paths starting from the root. In a suffix trie for S, every substring of S is identified by exactly one node and conversely, every node in the suffix trie identifies exactly one substring. The substring identified by a node N can be read from the character labels of the arcs along the path from the root to N. The string-length of the suffix trie is the same as the number of the edges in the suffix trie. Let us denote by $N(s)$ the node where the labels of the nodes from the root to $N(s)$ spell out the string s. All starting positions of every substring s can be read from the leaves of the subtrie under the node $N(s)$ in time proportional to the number of occurrences of the substring s.

A *suffix array* (Manber & Myers 1990) of S is an array of positions $1, 2, \ldots, |S|$ in S, sorted according to the lexicographic order of the suffixes starting at these positions. A suffix array can be constructed directly or it can be created from the suffix tree by reading the labels (start positions of suffixes) from the leaves of the suffix tree in the inorder from left to right. The suffix array needs only one third of the space of the corresponding tree (Baeza-Yates & Gonnet 1990). Every pointer of the suffix tree can be simulated on the suffix array by performing two indirect $O(\log |S|)$ binary searches. Thus, the tree can be used as a conceptual model for many problems while suffix arrays can be used in implementations. There are also some other approaches to reduce the space requirements of suffix trees using modifications of suffix arrays while maintaining the efficiency of the tree representation (Kärkkäinen 1995).

1	2	3	4	5	6	7	8
A	T	A	C	A	T	A	$



1	5	3	7	4	2	6	8

Figure 2.2: Suffix tree, suffix trie, and suffix array for the string $S = \mathsf{ATACATA\$}$. The last character, $\mathsf{\$}$ is added to make all suffixes of S different from each other. The linear size of suffix trees is achieved by representing arbitrary-length edge labels with two pointers. For example, `CATA`$ can be represented by two integers $(4, 8)$, *i.e.* `CATA`$ = S[4..8]$.

Chapter 3

Discovery of frequently occurring patterns

In this chapter we present algorithms for the pattern discovery problems A–E for pattern types P1–P6 outlined in Chapter 2. The proposed algorithms are generalizations to the *wotd* suffix trie construction algorithm so that the trie nodes represent the patterns from the specified pattern language. Patterns are generated systematically starting from the simplest (and thus most frequent) patterns, and gradually specializing each pattern within the limits of the chosen pattern class. The search is exhaustive, *i.e.* all variants of patterns within the chosen pattern class are potentially generated. The search space is narrowed down by eliminating patterns that can not match enough of the input sequences. The generation of patterns is speeded up by keeping pointers to all the occurrences of each pattern and updating this information for each possible pattern extension. Based on the numbers and exact locations of pattern occurrences, different fitness scores can be calculated.

The algorithms described in this chapter have been implemented in a software tool SPEXS, described later in Section 7.1.1. The SPEXS tool is integrated into a larger analysis package called Expression Profiler (Section 7.1), where gene expression, protein interaction, functional annotation, genomic sequence, and other functional genomics data can be analyzed.

First, we describe the approach in detail for generating the substring patterns (class P1) for problem types A–E. Later, pattern classes P2-P6 are discussed.

3.1 Discovery of substring patterns (P1)

Substring patterns are the simplest type of regular patterns. Although a very restricted pattern class, they have been shown to be able to capture many interesting features of DNA or protein sequences in practice (see, for example, (van Helden, André, & Collado-Vides 1998; Mannhaupt *et al.* 1999; Vilo *et al.* 2000)).

3.1.1 Frequent substrings of a string S (P1:A)

Given a string S, for example the full-length chromosome, a natural question is to ask which elements occur repeatedly in that sequence. More formally, we consider the following problem.

Problem P1:A Given a string $S \in \Sigma^*$ and an integer K, construct all substring patterns $\pi \in \Sigma^*$ (of type P1) such that π has at least K occurrences in S.

3.1.1.1 Solution based on traditional suffix tree algorithm

A linear time and space suffix tree index of a string S efficiently enumerates all possible substrings of S. The number of leaves under each node in the suffix tree is equal to the number of different occurrences of substrings represented by that node. The substrings having at least K occurrences can then be output.

Analysis of this method is straightforward. Construction of the suffix tree takes $O(n)$ time, and the corresponding tree has $O(n)$ nodes. Depth-first traversal for counting the number of leaves under each node can be done in $O(n)$ time.

The full answer, *i.e.* the list of all frequent substrings, may be quadratic $O(n^2)$ according to the total tree-length. Hence, the algorithm is called *output sensitive*, *i.e.* its running time is linear in the size of the produced output.

This solution seems optimal, as it should take time $O(n)$ to scan through the data and the time proportional to the size of the answer to output it. From the practical point of view the space required for storing the full suffix tree can be large. In an efficient implementation the size of the tree is on the average at least 10-15 times the size of S.

3.1.1.2 Basis for an alternative approach

We are interested in patterns that occur at least K times in S. By constructing the full-size suffix tree, unnecessary effort is spent for constructing the subtrees with less than K leaves. Ideally, we do not need the full suffix tree, but only the part that corresponds to the most frequent substrings of S.

Traditional linear time suffix tree construction algorithms are unable to "predict" which subtrees will represent frequent substrings and which do not. If the length l of the longest substring occurring at least K times in S could be estimated somehow, the modification of the algorithm for building the tree up to depth l only, could be used. Unfortunately, there is no tight upper bound for the maximum depth l that could guarantee the inclusion of all frequent substrings. On the average, the depth $l > \log_{|\Sigma|}(n)$ should be sufficient for random strings where each position is independently randomly chosen from the letters of the alphabet.

In the worst case, however, even very long substrings can occur frequently in S. Moreover, the biological sequences are known not to be random sequences, for example the genomic sequences contain longer repeats than expected.

We aim at a solution that does not assume the randomness of S, is faster for larger values of K, keeps the space requirement relatively low, and at the same time is simple to understand and implement. The solution is motivated by the *wotd*-algorithm for suffix tree construction (Giegerich & Kurtz 1995; Giegerich, Kurtz, & Stoye 1999). We represent the algorithm for constructing the $O(n^2)$ time and space, suffix trie instead of the compact suffix tree. The trie variant is easier to describe and implement, as well as it allows us to generalize this algorithm for discovering patterns from more complex pattern classes (P2–P6).

3.1.1.3 Notations

For identification of an individual node in the trie we use the string α over the trie label alphabet (for substring patterns this alphabet is Σ). The node $N(\alpha)$ in the trie uniquely defines a path from the root such that the node labels along that path spell out the string α. For example, $N(\texttt{ABC})$ is the node identified by substring ABC. The node $N(\alpha\texttt{C})$ is the child of $N(\alpha)$ with character label C. We use the dot-notation to represent additional information about the node N, e.g. $N.label$, $N.parent$, and $N.child$. In that notation, $N(\alpha X).label = X$, and $N(\alpha X).parent = N(\alpha)$, where X belongs to the label alphabet. Given a node N, we denote its children by $N.child(c)$ meaning the child P of node N such that $P.label = c$. The pattern associated to node N can be spelled out by $N.pattern()$. The occurrences of the pattern π are denoted by $N(\pi).pos$. We use a shorthand $N.sibling(c)$ for identifying the siblings $N.parent.child(c)$ of node N. Note that $N.sibling(c)$ is N if $N.label = c$.

3.1.1.4 Algorithm

Now we can present Algorithm 3.1 for solving the Problem **P1:A**. Algorithm 3.1 builds the suffix trie for the input string S in a systematic order, e.g. in the breadth-first order, level by level. For each node $N(\alpha)$ we create the list of positions $N(\alpha).pos$ to each location of S where α occurs. To represent the occurrence that ends at position j of S we use a *pointer* to position $j + 1$; this is just for technical convenience. To create the children of node $N(\alpha)$, we find characters $\texttt{a} \in \Sigma$ for which the substring $\alpha\texttt{a}$ occurs in at least K different locations of S. This is achieved by one traversal of the position list $N(\alpha).pos$ and creating the position lists for every character occurring at these positions in S. Only these nodes $N(\alpha\texttt{a})$ are inserted into the trie, for which the character a occurs at least K times at positions $N(\alpha).pos$.

Algorithm 3.1 (P1:A) Frequent substrings of a string S
Input: *String S, integer K*
Output: *Substring patterns that occur at least K times in string S*
Method:

```
$Root$ ← new node
$Root.label \leftarrow \lambda$
$Root.pos \leftarrow (1, 2, \ldots, |S|)$
$enqueue(Q, Root)$
while $N \leftarrow dequeue(Q)$
    Output $N.pattern()$ and its occurrences from $N.pos$
    foreach $c \in \Sigma$
        $Set(c) \leftarrow \emptyset$
    foreach $p \in N.pos$
        add $p + 1$ to $Set(S[p])$ unless $p = |S|$
    foreach $c \in \Sigma$ where $|Set(c)| \geq K$
        $P$ ← new node
        $P.label \leftarrow c$
        $P.pos \leftarrow Set(c)$
        $N.child(c) \leftarrow P$
        $enqueue(Q, P)$
    delete $N.pos$
end
```

The trie is constructed by first generating the root node and then systematically adding children to each of the leaves in the resulting tree. Each node in the trie represents a unique substring of S. The position lists associated with each node provide the information where all the occurrences of the substring corresponding to that node are. Note that position lists are only needed for the leaves during the tree construction, hence they can be deleted for internal nodes.

An advantage in constructing the tree in this way is that all children of a node are inserted in one step. There is no need for multiple visits to nodes in different parts of the trie and the physical implementation of tree nodes can be optimized by knowing exactly how many children the node will have. Example of such a trie construction is in Figure 3.1.

Construction of the trie explicitly as done in Algorithm 3.1 is not necessary as the relevant information can be stored in the nodes inserted into queue Q only. By maintaining the trie the actual patterns can be read from it.

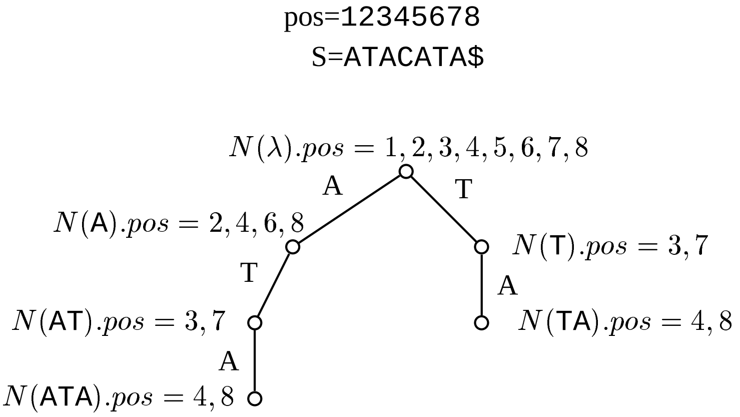

pos=12345678
S=ATACATA$
$N(\lambda).pos = 1,2,3,4,5,6,7,8$
A
T
$N(\mathsf{A}).pos = 2,4,6,8$
$N(\mathsf{T}).pos = 3,7$
T
A
$N(\mathsf{AT}).pos = 3,7$
$N(\mathsf{TA}).pos = 4,8$
A
$N(\mathsf{ATA}).pos = 4,8$

Figure 3.1: Discovering the substrings of string $S =$ ATACATA$ having at least 2 occurrences in S. The frequent patterns are λ,A,T,AT,TA, and ATA.

3.1.1.5 Analysis of the algorithm

The correctness of the algorithm. All starting positions of any possible pattern $(1, \ldots, |S|)$ are inserted to the tree root $N(\lambda).pos$. For each pattern all possible extensions are generated unless their number of occurrences drops below the threshold K. The generated patterns are inserted to the queue for further extensions, making the search exhaustive. Once the prefix of a pattern occurs less than K times, no extension of that pattern can occur more frequently and the construction of respective subtree is not necessary. Therefore, Algorithm 3.1 is correct.

Algorithm complexity. Let us analyze the time and space complexity of the Algorithm 3.1 for discovering the most frequent substrings. First we prove two lemmas.

Lemma 3.2 *For any node N in the suffix trie, $|N.pos| \geq \Sigma_{c\in\Sigma}|N.child(c).pos|$.*
Proof Follows from the fact that only $|N.pos|$ positions are considered, and that the position lists of children of a node are disjoint as the respective substrings end by different characters. □

Lemma 3.3 *The total size of the position lists of the leaves at any time during the execution of Algorithm 3.1 is at most $|S|$.*
Proof Follows from Lemma 3.2 and the fact that the root node in the tree has $|S|$ positions and the step 17 in Algorithm 3.1. □

Note that the size of the trie structure is optimal in the sense that only the nodes $N(\pi)$ corresponding to substrings π that occur at least K times in S, are inserted. Extra work has to be done for creating and maintaining the position lists.

Theorem 3.4 *Given a string S, $|S| = n$, the total time used by Algorithm 3.1 is linear in the total number of occurrences of all frequent substrings π in S* i.e. $O(\Sigma_\pi |N(\pi).pos|)$=$O(n^2)$. *Assume that there are p frequently occurring patterns π in S. The working space used by Algorithm 3.1 is $O(p + n)$.*

Proof Algorithm 3.1 visits each node in the trie twice. First, when it is constructed and put into the queue, and second, when it is retrieved from the queue and extended by all possible one-character extensions. The time used for construction of each node $N(\pi)$ corresponding to a unique frequent pattern $\pi \in \Sigma^*$ is proportional to the size of its position list, *i.e.* the number of all occurrences of that pattern. It remains to analyze how much effort is spent for constructing patterns that are not included into the trie, *i.e.* those, whose numbers of occurrences do not exceed the minimum frequency threshold K. This effort can in fact be accounted for the node representing the longest prefix of a pattern that is still frequent enough. The verification that a possible extension of a node N is not frequent enough is achieved at the same time as frequent extensions are calculated, with a single traversal of the position list $N.pos$. In total, the work is proportional to $\Sigma_\pi |N(\pi).pos|$ over all patterns π that occur at least K times in S.

There are $n - l + 1$ possible locations for all possible substrings of length l. At every depth l of the trie the work is proportional to the total size of the position lists of all nodes at that depth, *i.e.* $O(n)$. As the trie of the frequent patterns π has depth $O(n)$, it follows that $\Sigma_\pi |N(\pi).pos| = O(n^2)$.

If $K = 1$, Algorithm 3.1 constructs the full suffix trie, the size of which is $O(n^2)$.

The working space needed for the construction of the trie consists of the space for the trie and the position lists of all current leaves. The size of the trie is $O(1)$ per each node in the trie (that is, per each frequently occurring pattern π), $O(p)$ in total. When extending a particular node, the occurrences associated to that node are stored until all children have been calculated. As the total size of all position lists of the leaves (at any time during the trie construction) is at most n (Lemma 3.3), and the largest position list (*e.g.* for empty pattern λ), has at most n members, in total at most $2n$ positions are present in all the position lists jointly at any given time.

□

The worst-case time requirement of Algorithm 3.1 depends on the size of the trie. The work that is done at any depth l in the trie is $O(n)$. Hence, the time complexity is $O(nd)$, where d is the depth of the tree, *i.e.* the length of the longest substring π that occurs at least K times in S. In the worst case, the running time of Algorithm 3.1 can be quadratic in $|S|$ even for large K. One example of such a worst case input is the string $S = aaa \dots a$.

It may seem a bad idea to use this potentially quadratic time algorithm when $O(n)$ time algorithms for suffix tree construction exist. Interestingly, the experiments have shown that the *wotd* suffix tree construction algorithm, resembling the one described above, can compete quite well with the theoretically faster linear-time algorithms (Giegerich, Kurtz, & Stoye 1999). The reasons are mostly due to the non-locality properties of linear-time suffix tree generation algorithms which may cause slow-downs due to memory paging in current computer architectures. Therefore, this quadratic time suffix tree (and suffix trie) construction algorithm is interesting as such.

Theorem 3.5 *The average running time of Algorithm 3.1 for constructing all the substrings that occur at least $K > 1$ times in a random string S where each character is equally probable at each position is $O(|S| \log_{|\Sigma|} \frac{|S|}{K})$.*

Proof Given a random string S with uniformly distributed characters over alphabet Σ, all patterns of the same length can be assumed to occur equally probably in S. By adding one character c to the pattern π, the number of occurrences of πc is on the average $1/|\Sigma|$ times the number of occurrences of π. Hence, for $l > log_{|\Sigma|} \frac{|S|}{K}$ the sizes of individual position lists of the nodes at depth l contain typically less than K elements. Therefore, we can conclude the theorem. □

3.1.1.6 Discussion

Note that Algorithm 3.1 does not fix the order in which the leaves are considered during the tree construction. The order of the tree construction is determined by the implementation of the queue Q. If it is a standard FIFO queue, the pattern search is performed in breadth-first order, level by level. This allows to output the results in a systematic order from shorter to longer patterns, all the substrings of the same length ordered alphabetically. The construction and/or output order can also be different. For example, if queue Q acted like a stack (LIFO queue), the tree would be constructed in the depth-first order.

If the queue Q was implemented as a priority queue using the size $|N.pos|$ for ordering its entries, the topmost node in the queue would always represent the most frequent substring. In this way the search would be effectively performed from the most frequent to less frequent order. The search could also be stopped at any given moment as all the more frequent patterns would already be output.

Next we show how to modify Algorithm 3.1 for solving problem types (B-E).

3.1.2 Substrings common to a set of input sequences S^n (P1:B)

Problem type B deals with a typical pattern discovery situation, identifying patterns common to a set of sequences. Typically, these sequences may represent proteins from a single protein family, or DNA sequences assumed to share common regulatory motifs, for example.

Problem P1:B Given a set of strings $S^n = \{S_1, S_2, \ldots, S_n\}$, $S_i \in \Sigma^*$, and an integer K, construct all patterns π of type P1 such that π has at least one occurrence in at least K sequences of $S_1, S_2, \ldots, S_n$.

We solve this problem by first catenating all individual sequences $S_1, \ldots, S_n$ using a character $\# \notin \Sigma$ as a separator, to construct a single sequence $S = S_1\#S_2\#\ldots\#S_n$. This catenated sequence S is used for pattern discovery almost in the same manner as for the problem P1:A, only few modifications to Algorithm 3.1 are made.

Algorithm 3.6 *(P1:B) Frequent substrings of set of strings*
Input: *Strings* $S^n = \{S_1, \ldots, S_n\}$*, integer* K
Output: *Substring patterns that occur in at least* K *strings of* S^n
Method:

```
$S \leftarrow S_1\#\ldots\#S_n$ , s.t. $\# \notin \Sigma$
Generate a mapping $\{1,2,\ldots,|S|\} \rightarrow \{1,2,\ldots,n\}$ for countseq($Set$)
$Root \leftarrow$ new node
$Root.label \leftarrow \lambda$
$Root.pos \leftarrow (1,2,\ldots,|S|)$
$enqueue(Q, Root)$
while $N \leftarrow dequeue(Q)$
    Output N.pattern()
    foreach $c \in \Sigma$
        $Set(c) \leftarrow \emptyset$
    foreach $p \in N.pos$
        add $p+1$ to $Set(S[p])$ unless $p = |S|$ or $S[p] = \#$
    foreach $c \in \Sigma$ where countseq($Set(c)) \geq K$
        $P \leftarrow$ new node
        $P.label \leftarrow c$
        $P.pos \leftarrow Set(c)$
        $N.child(c) \leftarrow P$
        $enqueue(Q, P)$
    delete $N.pos$
end
```

First, we avoid patterns that could span across the string boundaries by disregarding any patterns that contain the separator character '$\#$' (line 12 in Algorithm 3.6).

Second, we count the number of sequences S_i that have at least one pattern occurrence. For this we generate a mapping (*e.g.* based on lookup-table) from each position in the catenated sequence S to index i based on the sequence S_i (line 2). The number of different sequences S_i can be counted in linear time in the length of the position list $N(\pi).pos$ by simply traversing the list and counting each sequence index i once (function *countseq*$(N(\pi).pos)$, line 13).

Algorithm 3.6 is correct based on the same justification as Algorithm 3.1. Every possible pattern is generated as long as it occurs in at least K input sequences.

Algorithm 3.6 runs in the same time and space as Algorithm 3.1 for the catenated string S. The length of the longest possible frequent pattern is bound by $max(\{|S_i| \bigm| S_i \in S^n\})$. This can improve the worst-case performance, especially if S^n consists of short sequences only.

Counting the number of sequences by function *countseq*$(N(\pi).pos)$ does not add more than one extra traversal through each position list, hence the asymptotic running time and space remains the same as for Algorithm 3.1.

3.1.3 The most "interesting" substrings of sequence S (P1:C)

The most frequently occurring patterns are obviously the empty pattern λ (occurs at each position) and patterns of length one. The occurrences of single-character patterns correspond to the occurrences of each letter in the input sequence. These rather trivial "patterns" are not necessarily what users would like to see reported. Instead, they want the patterns to be output according to their fitness $\mathcal{F}$. This gives us the following problem statement.

Problem P1:C Given a string $S \in \Sigma^*$, and an integer K, find all patterns π of type P1 that occur at least K times in S and report them in the decreasing order of their fitness $\mathcal{F}$.

Note that we have introduced the requirement for the minimum number of occurrences which was not mentioned in the problem statement in Section 2.6. We assume that users can require discovered patterns to occur at least K times (or in K sequences, where appropriate). This, besides reducing the search space, usually has a good justification from the analysis domain. If the minimum frequency requirement is not given, the pattern discovery procedure could be forced to study through all possible patterns, even these that are unique within the sequence S.

We modify Algorithm 3.1, so that pattern fitness measures will be calculated and patterns can be presented to users in the order based on that fitness. We assume that different functions $\mathcal{F} : \mathrm{P1} \times \Sigma^* \rightarrow \mathbb{R}$ for evaluating the fitness can be used. This gives us Algorithm 3.7.

Algorithm 3.7 (P1:C) Frequent and interesting substrings of a string S
Input: *String S, integer K, fitness function $\mathcal{F} : \mathrm{P1} \times \Sigma^* \rightarrow \mathrm{I\!R}$*
Output: *Substring patterns π with best fitness $\mathcal{F}(\pi, S)$ that occur at least K times in S*
Method:

```
1.  $Root \leftarrow$ new node
2.  $Root.label \leftarrow \lambda$
3.  $Root.pos \leftarrow (1, 2, \ldots, |S|)$
4.  $enqueue(Q, Root)$
5.  while $N \leftarrow dequeue(Q)$
6.      foreach $c \in \Sigma$
7.          $Set(c) \leftarrow \emptyset$
8.      foreach $p \in N.pos$
9.          add $p + 1$ to $Set(S[p])$ unless $p = |S|$
10.     foreach $c \in \Sigma$ such that $|Set(c)| \geq K$
11.         $P \leftarrow$ new node
12.         $P.label \leftarrow c$
13.         $P.pos \leftarrow Set(c)$
14.         $N.child(c) \leftarrow P$
15.         $enqueue(Q, P)$
16.         $enqueue(B, P, \mathcal{F}(P.pattern, S))$    // Store the patterns and their fitnesses
17.     delete $N.pos$
18. // Output the "best" patterns stored in priority queue $B$
19. while $(N, f) \leftarrow dequeue(B)$
20.     Output $N.pattern$ and $f$
21. end
```

The command $enqueue(B, P, \mathcal{F}(P.pattern, S))$ on line 16 inserts the node P and its fitness $\mathcal{F}(P.pattern, S)$ to a priority queue B (line 16), from where they can later be retrieved (line 19) in the order of their fitness. Note that usually the function $\mathcal{F}(P.pattern, S)$ does not require the full input sequence S, but only the locations of the matches of π on S. These matches are stored in $P.pos$ and can be made available to calculate the fitness $\mathcal{F}$. In that case, one can assume the function $\mathcal{F}(P.pattern, P.pos)$ instead of $\mathcal{F}(P.pattern, S)$.

Algorithm 3.7 is correct, it exhaustively enumerates all patterns π that occur at least K times in input S, while storing the patterns into the priority queue based on the fitness $\mathcal{F}(P.pattern, S)$.

The exhaustive enumeration runs in the same time and space as Algorithm 3.1. For each frequent pattern, extra work is needed for calculating the fitness function

$\mathcal{F}(P.pattern, S)$. If the fitness function computation takes linear time in the number of pattern occurrences $O(|P.pos|)$ or the pattern length $O(|P.pattern|)$, the total time complexity can be cubic $O(n^3)$ in the worst case. If the pattern fitness function can be computed in a constant time based on the fitness of the pattern of the parent node only, the dependency on the pattern length could be avoided. Some of such techniques are described by Apostolico *et al.* (Apostolico *et al.* 2000).

The node identifiers are stored in the priority queue B based on the fitness. If there are p frequent patterns ($p = O(n^2)$ in the worst case), the additional time $O(p \log p)$ may be spent for storing and retrieving patterns from B. Thus the overall worst case time complexity is $O(n^2 \log n)$.

Not all frequent patterns are interesting in practice, but only the q top ranking patterns with respect to the fitness function $\mathcal{F}$. In that case the priority queue B of best-scoring patterns can be modified to hold a maximum of q items. Each newly generated pattern is compared to the worst performing pattern generated so far and if the new pattern has a better score, the previous q^{th} worst pattern will be removed and the new pattern inserted. The size of B remains constant as well as the time spent for each priority queue operation.

3.1.4 The most "interesting" substrings for a set of input sequences S^n (P1:D)

Problem P1:D Given a set of strings $S^n = \{S_1, S_2, \ldots, S_n\}$, $S_i \in \Sigma^*$, find all patterns π of type P1 that occur at least in K sequences of $S_1, S_2, \ldots, S_n$, and report them in the decreasing order of the fitness $\mathcal{F}$.

Previously we showed that Problem P1:C can be solved by modifying Algorithm 3.1 for solving Problem P1:A. Problem P1:D is a similar generalization to Problem P1:B. Algorithm 3.8 for solving Problem P1:D is a straightforward modification of Algorithm 3.6 in similar manner as Algorithm 3.7 was modified from Algorithm 3.1. The same applies to the analysis of the time and space complexity, these remain essentially the same as for Algorithm 3.6 and we omit the analysis.

Algorithm 3.8 *(P1:D) Most "interesting" substrings of a set of strings*
Input: *Strings $S^n = \{S_1, \ldots, S_n\}$, integer K , fitness function $\mathcal{F} : \mathrm{P1} \times S^n \rightarrow \mathrm{I\!R}$*
Output: *Substring patterns that occur in at least K strings of S^n in the order of fitness $\mathcal{F}$*
Method:

```
$S \leftarrow S_1\# \ldots \#S_n$ , s.t. $\# \notin \Sigma$
Generate a mapping $\{1, 2, \ldots, |S|\} \rightarrow \{1, 2, \ldots, n\}$ for countseq$(Set)$
$Root \leftarrow$ new node
$Root.label \leftarrow \lambda$
$Root.pos \leftarrow (1, 2, \ldots, |S|)$
$enqueue(Q, Root)$
while $N \leftarrow dequeue(Q)$
    foreach $c \in \Sigma$
        $Set(c) \leftarrow \emptyset$
    foreach $p \in N.pos$
        add $p + 1$ to $Set(S[p])$ unless $p = |S|$ or $S[p] = \#$
    foreach $c \in \Sigma$ where countseq$(Set(c)) \geq K$
        $P \leftarrow$ new node
        $P.label \leftarrow c$
        $P.pos \leftarrow Set(c)$
        $N.child(c) \leftarrow P$
        $enqueue(Q, P)$
        $enqueue(B, P, \mathcal{F}(P.pattern, S^n))$
    delete $N.pos$
while $(N, f) \leftarrow dequeue(B)$
    Output $N.pattern$ and $f$
end
```

Note that we assume that the fitness function $\mathcal{F}$ can actually have access to the mapping from sequence positions to sequence numbers, to the $countseq(Set)$ function, as well as the pattern itself. We do not want to restrict this presentation to any specific function.

3.1.5 Substring patterns overrepresented in S_+ vs S_- (P1:E)

As discussed in Section 2.4, it is often important to be able to discriminate between two sets of sequences. The most obvious candidates for such patterns are the ones that do occur in one set but not in the other. In practical tasks, this is usually not the case. Here we are interested in the patterns that are significantly more abundant in one set of the input sequences than in the other.

Problem P1:E Given two sets of strings S_+ and S_-, find all patterns π of type P1 such that π occurs in at least K sequences out of $S_+ \cup S_-$ and is "significantly" more frequent in S_+ than in S_-.

This problem can be viewed as a special case of the problem P1:D. The main difference is that the input sequences represent two sets of sequences instead of one, and the aim of the fitness function is to rank the patterns so that the best differentiation between these two sets is achieved. The significance of the over-representation is calculated by a special case of fitness measure $\mathcal{F}$ that, given a pattern π and two sets of sequences S_+ and S_-, estimates whether the number of matches of π is significantly higher in S_+ than in S_-.

Algorithm 3.9 *(P1:E) Substrings more frequent in S_+ than S_-*
Input: *Strings $S_+ = \{S_1^+, \ldots, S_{n_+}^+\}$, $S_- = \{S_1^-, \ldots, S_{n_-}^-\}$, integer K, fitness measure $\mathcal{F}$*
Output: *Substring patterns that occur at least K times in $S_+ \cup S_-$, and significantly more frequently in S_+ than in S_- according to $\mathcal{F}$*
Method:

```
$S \leftarrow S_1^+ \# \ldots \# S_{n_+}^+ \# S_1^- \# \ldots \# S_{n_-}^-$ , s.t. $\# \notin \Sigma$
Generate a mapping $\{1, 2, \ldots, |S|\} \to \{1, 2, \ldots, (n_+ + n_-)\}$ for countseq$(Set)$
$Root$ ← new node
$Root.label \leftarrow \lambda$
$Root.pos \leftarrow (1, 2, \ldots, |S|)$
$enqueue(Q, Root)$
while $N \leftarrow dequeue(Q)$
    foreach $c \in \Sigma$
        $Set(c) \leftarrow \emptyset$
    foreach $p \in N.pos$
        add $p + 1$ to $Set(S[p])$ unless $p = |S|$ or $S[p] = \#$
    foreach $c \in \Sigma$ where countseq$(Set(c)) \geq K$
        $P$ ← new node
        $P.label \leftarrow c$
        $P.pos \leftarrow Set(c)$
        $N.child(c) \leftarrow P$
        $enqueue(Q, P)$
        $enqueue(B, P, \mathcal{F}(\pi, S_+, S_-))$
    delete $N.pos$
while $(N, f) \leftarrow dequeue(B)$
    Output $N.pattern$ and $f$
return $Root$
end
```

The most important change in Algorithm 3.9 as compared to Algorithm 3.8, is the way how the fitness measure is calculated. In order to know how frequent the pattern π is in each input set, we need to count numbers of occurrences in S_+ and S_- separately. This can easily be achieved by a straightforward modification to the $countseq(N(\pi).pos)$ function. Based on these numbers different fitness measures can be used.

In practice we may be interested in patterns that occur at least a given number of times in a particular input set, For example, in patterns that occur at least K_+ times in set S_+. This is easy to accommodate by slight modifications to the mapping from sequence positions to sequence numbers, and to the $countseq()$ function. This feature has been used in some of the applications discussed in Chapter 6.

Note that it is straightforward to generalize Algorithm 3.9 for more than two sets of input sequences. In that case, however, the relevant fitness measures need to be chosen so that they are suitable for particular analysis of the multi-class input.

3.2 Substring patterns with group characters (P2)

In this section we present algorithms for discovering substring patterns with group positions, *i.e.* patterns of type $\pi \in (\Sigma \cup \Gamma)^*$ (class P2) from Section 2.1.

As we have showed in Section 3.1, the different pattern discovery problems A–E can be solved in a rather similar way for the class of substring patterns. The same applies for most other pattern classes. To save space and to keep readers from being bored, we show only how the pattern discovery problem B can be solved for the pattern class P2.

3.2.1 Frequent substrings with group characters common to a set of strings S^n (P2:B)

Problem P2:B Given a set of strings $S^n = \{S_1, S_2, \ldots, S_n\}$, $S_i \in \Sigma^*$, and an integer K, construct all patterns π of type P2 ($\pi \in (\Sigma \cup \Gamma)^*$) such that π has at least one occurrence in at least K sequences of $S_1, S_2, \ldots, S_n$.

We modify Algorithm 3.6 in order to permit symbols from the extended character set $\Sigma \cup \Gamma$ as node labels. For each group symbol $g \in \Gamma$ following the node $N(\alpha)$, the node $N(\alpha).child(g) = N(\alpha g)$ is created.

We do not need to store explicitly the positions $N(\alpha g).pos$. These can be calculated from the position lists of individual characters $c \in g$, as $N(\alpha g).pos = \bigcup_{c \in g} N(\alpha c).pos$. However, it is necessary that nodes $N(\alpha c)$, $c \in g$, are inserted into the trie with corresponding $N(\alpha c).pos$ even if $|N(\alpha c).pos| < K$ and $N(\alpha c)$ is not extended further.

Algorithm 3.10 *(P2:B) Generation of frequent patterns with character group positions*
Input: *Strings $S^n = \{S_1, \ldots, S_n\}$, integer K, character groups Γ*
Output: *Patterns from $(\Sigma \cup \Gamma)^*$ that occur in at least K strings of S^n*
Method:

```
$S \leftarrow S_1\# \ldots \#S_n$, $\# \notin \Sigma$
Generate a mapping $\{1, 2, \ldots, |S|\} \rightarrow \{1, 2, \ldots, n\}$ for countseq$(Set)$
$Root$ ← new node; $Root.label \leftarrow \lambda$
$Root.pos \leftarrow (1, 2, \ldots, |S|)$
$enqueue(Q, Root)$
while $N \leftarrow dequeue(Q)$
    Output N.pattern()
    // Construct the position list for pattern defined by node $N$
    if $N.label \in \Sigma$ then $Pos \leftarrow N.pos$
    else $Pos \leftarrow \bigcup_{c \in N.label} N.sibling(c).pos$
    // Group the positions according to characters in string $S$
    foreach $c \in \Sigma$
        $Set(c) \leftarrow \emptyset$
    foreach $p \in Pos$
        add $p + 1$ to $Set(S[p])$ unless $p = |S|$ or $S[p] =' \#'$
    foreach $c \in \Sigma$
        if $countseq(Set(c)) \geq K$ then
            $N.child(c)$ ← new node with label $c$
            $N.child(c).pos \leftarrow Set(S[c])$
            $enqueue(Q, N.child(c))$
    foreach $g \in \Gamma$
        if $\exists f \in (\Gamma \cup \Sigma), f \subset g, \sum_{c \in f} |Set(c)| = \sum_{c \in g} |Set(c)|$ then next $g$
        if $countseq(\cup_{c \in g} Set(c)) \geq K$ then
            $N.child(g)$ ← new node with label $g$
            foreach $c \in g$ s.t. $\neg \exists N.child(c)$
                $N.child(c)$ ← new node with label $c$ and positions $Set(S[c])$
            $enqueue(Q, N.child(g))$
    if all nodes $N.sibling(c)$, $c \in \Sigma \cup \Gamma$ have been expanded
        delete all lists $N.sibling(c).pos$, where $c \in \Sigma$
end
```

We say that the pattern $\pi \in \mathrm{P2}$ is most *specific* with respect to S if by replacing any of the group character $g \in \Gamma$ in pattern π by a character $c \in g$ or by a group $g' \in \Gamma$, $g' \subset g$, none of the modified, more specific patterns π', has exactly the same occurrences in S as the original pattern π has. In other words, no position in π can be made more specific without losing hits in S.

When adding group character extensions to the tree, we perform an extra test to avoid inserting such group characters g for which there exists a sibling f that corresponds to a subset of characters from g yet occurs in all the same places (line 22). This test eliminates some replication of work.

Algorithm 3.10 does not guarantee that the discovered patterns would be most specific. For example, given two sequences $\mathtt{GAS}, \mathtt{HAT}$, all the possible character groups Γ and $K = 1$, the pattern discovery procedure outlined here would construct the patterns $[\mathtt{GH}], [\mathtt{GH}]\mathtt{A}, [\mathtt{GH}]\mathtt{AS}, [\mathtt{GH}]\mathtt{AT}$. The last two would not be desired in this context, as the first group character position would be redundant. By reversing the algorithm and working with position lists toward the top of the trie, it would be obvious that the first group should be replaced by the more strict character $\mathtt{G}$, *i.e.* the patterns would be $\mathtt{GAS}, \mathtt{HAT}$. The time and space requirements for this reverse calculation are proportional to the length of the pattern times the number of the occurrences of the pattern. This reverse "specificity" check can be performed for patterns that are most interesting.

In practice, it is easy to modify Algorithm 3.10 so that on each path from the root at most x group characters are allowed. By adding a counter of group characters that have been used on the path from root to each node, if the limit is exceeded, then only the single character extensions ($c \in \Sigma$) are allowed. This allows to be more specific about the pattern language and to speed up the algorithm significantly (at the expense of the pattern language of course).

For all nodes N and symbols $c \in \Sigma \cup \Gamma$, $|N.pos| \geq |N.child(c).pos|$, *i.e.* the sizes of the position lists are non-increasing along any path from the root down to leaves.

Theorem 3.11 *Assume that x is the maximum number of group characters $g \in \Gamma$ allowed in a pattern, and p is the maximum number of group characters $g \in \Gamma$ that all contain a particular $c \in \Sigma$ (each c belongs to at most p different groups g). The maximum number of patterns of length at most l over $\Sigma \cup \Gamma$ that may have at least one occurrence in string S of length n is $O(xnp^x l^{x+1})$. The worst case time complexity of Algorithm 3.10 is $O(xn^2 p^x l^{x+1})$.*

Proof There are $n - l + 1$ possible substrings of length l of string S, $|S| = n$. By allowing character group positions we essentially allow that some positions of substrings can be represented by a group of characters instead of a single character from Σ. Assume that each character $c \in \Sigma$ belongs to (at most) p different character groups. If at most x positions of a substring of length l may be replaced by one of these p character groups, there are at most $1 + pl + p^2 l(l-1) + \ldots + p^x(l(l-1)\cdots(l-x+1)) = O(xp^x l^x)$ different patterns that match this substring. String S can contain $O(ln)$ substrings of length less than l, hence the total number of patterns that can match substrings of length at most l in S is $O(xnp^x l^{x+1})$.

The patterns are constructed incrementally by expanding shorter patterns one position at a time. The work used for creating each pattern is proportional to the number of occurrences of each pattern. In the worst case it is $O(n)$. Hence, the worst case time complexity of Algorithm 3.10 is $O(xn^2 p^x l^{x+1})$. In reality only the patterns that occur frequently in S, *i.e.* the patterns that match at least K different substrings of S, are constructed. □

In Algorithm 3.6 for discovering the frequent substrings, all the position lists of the nodes at the same level in the trie were disjoint. Now, allowing group characters, different patterns can match exactly the same substrings of S. It means that the catenation of the position lists at the same level of the trie could grow exponentially in the number of group positions in the patterns.

In the breadth-first search these overlapping positions would all have to be stored. However, if the search for patterns is performed in the depth-first order, only the position lists of the nodes (and the immediate siblings of these nodes) along a single path from the root would be needed. The position lists of siblings do not overlap, hence at most n positions are stored in them. This means that during the construction of the trie of depth l, only $O(ln)$ positions are stored in the nodes along the path from the root to the leaf at depth l. Hence, the space requirement of Algorithm 3.10 is $O(T + ln)$, where T is the size of the trie of frequently occurring patterns and ln is the amount of memory used for storing the position lists along any single path from the root to leaf at depth l (the length of the longest pattern).

3.3 Patterns with unrestricted length wildcards (P3)

Given a string S, patterns with unrestricted length wildcards matching S are commonly called *subsequences*[1] of S. Hence, the patterns from class P3 can also be referred to as subsequences. In this section we consider the discovery of patterns of form $\pi \in \Sigma^+(*\Sigma^+)^*$, *i.e.* the patterns whose first and last symbol are in Σ.

In order to identify frequently occurring subsequences we need to be able to count the number of occurrences of each subsequence. As previously, we will

[1] Note that subsequence and substring are not the same concept although the string and sequence in general are used interchangeably. See also (Gusfield 1997).

identify the occurrence of a pattern by the location of the last position of the pattern in the sequence S. This definition is not free of problems though.

Example 3.12 Given a string $S =$ABAACA some of the subsequences of S are, for example, AB*C, A*A*C, and A*A. According to our definition, A*A has three occurrences in S and A*A*C and AB*A*C only one. The two different subsequences A*C and B*C are different patterns even though they "occur" at exactly the same positions. Note that pattern B occurs once in S, while its refinement B*A has three occurrences. □

Example 3.13 Let S be a string $S \in \Sigma^*$ of length n. Let us consider the string $S' = Sca^n$, such that c occurs only once in S'. Every pattern $\alpha * c * a$, where α is a substring of S occurs n times in S' even though $\alpha * c$ occurs only once. □

It is clear from these examples that a prefix of the subsequence pattern π (see $\alpha{*}c$ above) may have fewer occurrences than the pattern π itself ($\alpha{*}c{*}a$). Hence, the definition of the frequency of occurrences is perhaps not so intuitive any more. In practice our pattern discovery algorithm requires all prefixes of a pattern to be frequent.

For multi-sequence inputs we can use a different criteria for counting occurrences. Namely, we can count the number of sequences that contain a match (one or more occurrences) by the pattern π. Then the ambiguity of counting pattern match locations is avoided. First, however, we consider a single-sequence input.

3.3.1 Frequent subsequences of a string S (P3:A)

Problem P3:A Given a string $S \in \Sigma^*$ and an integer K, construct all patterns π with unrestricted length wildcards (patterns of type P3) such that π has at least K occurrences in S.

We will extend our pattern trie construction algorithms so that symbols of the type $*c$, $c \in \Sigma$, can also be used as labels for nodes. When the wildcard is used to extend the pattern, it is immediately followed by a character from Σ. For example, if $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$, the pattern trie is built over the set of node labels $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}, *\mathtt{A}, *\mathtt{C}, *\mathtt{G}, *\mathtt{T}\}$.

Algorithm 3.14 *(P3:A)* **Frequent subsequence generation**
Input: *Input string S, threshold K*
Output: *All frequently occurring subsequences (patterns over $\Sigma \cup \{*c|c \in \Sigma\}$)*
Method:

```
1.  $Root \leftarrow$ new node
2.  $Root.label \leftarrow \lambda$
3.  $Root.pos \leftarrow (1, 2, \ldots, |S|)$
4.  $enqueue(Q, Root)$
5.  while $N \leftarrow dequeue(Q)$
6.      foreach $p \in N.pos$
7.          add $p + 1$ to $Set(S[p])$
8.      if $N \neq Root$ then                // Create position lists for ’*c’ extensions
9.          $p \leftarrow$ smallest value from $N.pos$
10.         $w \leftarrow 0$
11.         while $p + w < |S|$
12.             add $p + w + 1$ to $Set('*S[p + w]')$
13.             $w \leftarrow$w+1
14.     foreach $c \in (\Sigma \cup' *\Sigma')$ where $|Set(c)| \geq K$
15.         $P \leftarrow$ new node
16.         $P.char \leftarrow c$
17.         $N.child(c) \leftarrow P$
18.         $P.pos \leftarrow Set(c)$
19.         $enqueue(Q, P, front)$          // Enqueue to the front of queue
20.     foreach $c \in (\Sigma \cup' *\Sigma') deleteSet(c)$
21.     delete $N.pos$
22. end
```

Every position of S is considered as a possible first character of pattern π in Algorithm 3.14. During the process of extending patterns, first all single character extensions are considered (see line 7). Next, all extensions with a wildcard followed by a character from the alphabet of S are added. Note that the test on line 8 does not allow to start a pattern with a wildcard $*$.

When we add nodes corresponding to wildcards followed by a fixed character, only the first occurrence of a pattern prefix needs to be considered (lines 9 – 13). This is enough for creating the maximal set of occurrences.

There is an exponential number of subsequences of S (in the length of S). This is easy to see by noting that each of the positions of S can be either represented in pattern π by the character itself or a wildcard. Additionally, for each pattern potentially all its occurrences need to be enumerated. Hence, the worst case time complexity of Algorithm 3.14 is $O(n2^n)$.

The problem statement P3:A required to enumerate only the frequently occurring subsequences of S. Note that we require each prefix of a subsequence pattern to be a frequent subsequence pattern as well, as we think it is more intuitive in this case.

The space complexity. The maximum number of occurrences of any pattern π is $O(n)$. The sum of the total number of occurrences of all the extensions to a pattern α of type $\alpha * c$ is $O(n)$, as the occurrences must be mutually exclusive. Hence, the total number of occurrences of all the extensions αX is $O(n)$. Note that the search is performed in a depth-first order, and hence only the positions along a single search path need to be stored at any given moment. Therefore the maximum number of positions needed to store during the search is $O(ln)$ where l is the length of the longest path in the tree corresponding to the longest commonly occurring pattern. The pattern trie size itself accounts for the rest of the space requirement. If the traversed paths are deleted immediately this space can be reduced. Otherwise the trie size is proportional to the total number of frequent subsequences.

3.3.2 Subsequences common to multiple input strings (P3:B)

The problem statement P3:A at the first glance seems impractical for both the practical relevance as well as the time complexity analysis point of view. Note that the main complexity of the algorithm is due to the length of the input sequence S. If the input data consists of a set of sequences S_i, we get the following problem statement.

Problem P3:B Given a set of strings $S^n = \{S_1, S_2, \ldots, S_n\}$, $S_i \in \Sigma^*$, and an integer K, construct all patterns π of type P3 such that π has at least one occurrence in at least K of the sequences $S_1, S_2, \ldots, S_n$.

Note that the longest common subsequence problem is directly related to the sequence alignment problem (Gusfield 1997). Hence, Problem P3:B can be viewed as a multiple sequence alignment problem where all local alignments common to at least K input sequences are searched for.

Algorithm 3.15 *(P3:B)* **Frequent subsequences common to a set of sequences**
Input: *Input strings* $S^n = \{S_1, \ldots, S_n\}$ *, threshold* K
Output: *Patterns over* $\Sigma \cup \{c * | c \in \Sigma\}$ *that occur in at least* K *different strings from* S^n
Method:

```
$S \leftarrow S_1\# \ldots \#S_n\#$
Generate a mapping $\{1, 2, \ldots, |S|\} \rightarrow \{1, 2, \ldots, n\}$ for countseq$(Set)$
$Root$ ← new node
$Root.label \leftarrow \lambda$
$Root.pos \leftarrow (1, 2, \ldots, |S|)$
$enqueue(Q, Root)$
while $N \leftarrow dequeue(Q)$
    $s \leftarrow 0$ // for keeping track of sequence number $S_s$
    foreach $p \in N.pos$
        add $p + 1$ to $Set(S[p])$
        if $N \neq Root$ and $p \in S_i$ and $i > s$    // if p is within a new sequence $S_i$
            $s \leftarrow i$        // Remember which sequence $S_i$ is being studied
            $w \leftarrow 0$
            while $S[p + w] \neq' \#'$
                add $p + w + 1$ to $Set(*S[p + w])$
                $w$ ←w+1
    foreach character $c \in (\Sigma \cup \{*c|c \in \Sigma\})$ where $countseq(Set(c)) \geq K$
        $P$ ← new node
        $P.char \leftarrow c$
        $N.child(c) \leftarrow P$
        $P.pos \leftarrow Set(c)$
        $enqueue(Q, P, front)$            // Enqueue to the front of queue
    foreach $c \in (\Sigma \cup \{*c|c \in \Sigma\})$ delete $Set(c)$
    delete $N.pos$
end
```

Algorithm 3.15 does not generate subsequences that would cross delimiters between original strings S_i. Otherwise it is only a small modification from Algorithm 3.14, and constructs all possible subsequences common to at least K input sequences S_i.

Note that no extension to a subsequence π can be more frequent than π itself, as we only count the number of sequences π matches, not the positions (how many occurrences in total). The subsequences that are not common to enough many original sequences S_i are eliminated.

The space required to store position lists at any time during the algorithm execution is potentially the same as for Algorithm 3.14. However, the number of all potential subsequences is $O(n2^{\max(|S_i|)})$.

The time complexity of the Algorithm 3.15 is $O(n^2 2^{\max(|S_i|)})$.

3.4 Patterns with restricted-length wildcards $*(k, l)$ (P4)

As shown for pattern language P3, the wildcards of unrestricted length may be considered impractical, especially for analyzing longer sequences. By restricting the length of a wild-card to a "reasonable" range, a more practical pattern language can be defined. As for unrestricted wildcards in the previous section, we consider here the problem types A and B only.

Problem P4:A Given a string $S \in \Sigma^*$ and an integer K, construct all substring patterns π with restricted-length wildcards $*(k, l)$ (patterns of type P4) such that π has at least K occurrences in S.

Problem P4:B Given a set of strings $S^n = \{S_1, S_2, \ldots, S_n\}$, $S_i \in \Sigma^*$, and an integer K, construct all patterns π of type P4 such that π has at least one occurrence in at least K sequences of $S_1, S_2, \ldots, S_n$.

These two problems are directly related to the problems P3:A and P3:B, only we need to handle the wildcards of restricted length between k and l positions instead of unrestricted wildcards. Other problem types C, D, and E can be solved by straightforward generalizations of these algorithms.

The modifications required are necessary to the loops that add wildcard positions, as now only the positions within a range from k to l have to be considered. In the algorithms for solving problems P3:A and P3:B we considered only the first occurrence of π, as the wildcard could span to the end of the sequence. Now we can restrict ourselves to gaps of length from k to l characters. Note that these gaps can still overlap for different occurrences of pattern prefixes. To avoid repeated work on these gap areas, we maintain a pointer to the rightmost position of the previously inserted gaps, and based on that increase the start of a leftmost gap position k for the next pattern occurrence, if necessary (see lines 11 and 15). We illustrate this in the Example 3.17, where the calculation of the second block of occurrences is redundant and can be avoided.

Algorithm 3.16 *(P4:A)* **Discovery of patterns with wildcards of restricted length**
Input: *Input string S, threshold K, integers k and l*
Output: *All patterns over $\Sigma \cup \{*(k,l)c | c \in \Sigma\}$ that occur in at least K positions*
Method:

```
Root ← new node
$Root.label \leftarrow \lambda$
$Root.pos \leftarrow (1, 2, \ldots, |S|)$
$enqueue(Q, Root)$
while $N \leftarrow dequeue(Q)$
    foreach $p \in N.pos$
        add $p + 1$ to $Set(S[p])$
    if $N \neq Root$ then                    // Create position lists for '*c' extensions
        $lastpos \leftarrow -1$
        foreach $p \in N.pos$
            $w \leftarrow$ if $lastpos > p + k$ then $lastpos - p$ else $k$
            while $p + w < |S|$ and $w \leq l$
                add $p + w + 1$ to $Set('*S[p + w]')$
                $w \leftarrow w + 1$
            $lastpos \leftarrow w - 1$
    foreach $c \in (\Sigma \cup \{*(k,l)d | d \in \Sigma\})$ where $|Set(c)| \geq K$
        $P \leftarrow$ new node
        $P.char \leftarrow c$
        $N.child(c) \leftarrow P$
        $P.pos \leftarrow Set(c)$
        $enqueue(Q, P, front)$      // Enqueue to the front of queue
    foreach $c \in (\Sigma \cup \{*(k,l)d | d \in \Sigma\})$ delete $Set(c)$
    delete $N.pos$
end
```

Example 3.17 Consider a sequence AATACAATACAAAAC and occurrences of a pattern AT and its extensions AT.$\{$1,8$\}$x where x is one of the alphabet characters.

```
AATACAATACAAAAC
     || | ||||     All occurrences of AT.{1,8}A
    |    |    |    All occurrences of AT.{1,8}C
       |           Single occurrence of AT.{1,8}T

 AT.C
 AT..A
 AT...A
 AT....T
 AT.....A
 AT......C
 AT.......A
 AT........A

      AT.C
      AT..A
      AT...A

      AT....A
      AT.....A
      AT......C
```

The first occurrence of AT, when expanded, produces the first block of 8 matches of patterns of type AT.$\{$1,8$\}$x. The second occurrence of AT produces additional 6 matches, of which the first three are already covered in the first block. We can avoid repeating that work, by considering for the extensions of the second occurrence of AT effectively only the patterns of type AT.$\{$4,8$\}$x. □

In practice, the discovery of patterns of type P3 can be simulated by patterns of type P4 by choosing appropriate k and l for $*(k,l)$ wildcards, *e.g.* $k = 0$ and $l > |S|$.

3.4.1 Minimizing the flexibilities in restricted length gaps

The aim of the pattern discovery is to find previously unknown patterns. Hence, it is probable that users do not know much about the possible flexibilities $X(k,l)$ in the patterns. We would like to know what are the real lengths of each of the gaps so that their exact boundaries would make them as specific as possible. The question is, given a pattern with restricted length wildcards, what is the most conserved form of that pattern? For that we need to have a mechanism for minimizing the k and l values for each $X(k, l)$ position individually so that the discovered patterns would be the most conserved ones.

Intuitively, this is a question about the maximality of the patterns in the sense of the maximum information that the patterns can contain (being most specific) without losing any significant hits in the sequences.

More formally, assume that we have a pattern of type $\pi = a_1 * \{k, l\} a_2 * \{k, l\} \ldots * \{k, l\} a_n$ (or $\pi = a_1 * a_2 * \ldots * a_n$). We want to modify this pattern into the form of $\pi' = a_1.\{k_1, l_1\}a_2.\{k_2, l_2\} \ldots .\{k_{n-1}, l_{n-1}\}a_n$ in the way that the pattern π' is the most restricted representation of π. We want the total flexibility of the pattern π' to be as small as possible. For this we need to minimize the $\Sigma_{i=1..n-1}(l_i - k_i)$. At the same time the modified pattern π' should still match all the same sequences of the input as the π does.

As the pattern π can match a sequence in many different locations, a more restrictive form of the problem is to minimize the flexibilities such that the pattern π' will match all the same locations within the input sequences, where pattern π matched.

In general, the solution to this problem can be achieved for the $X(k, l)$ representation of restricted length wildcards by counting the actual lengths of the wildcard-spanning regions in the text and replacing each wildcard symbol by a more restricted wildcard symbol $X(k_i, l_i)$.

Consider the zinc finger motif from the Introduction `C-x(2,4)-C-x(3)-[LIVMFYWC]-x(8)-H-x(3,5)-H`. By removing the middle character group it becomes slightly more general `C-x(2,4)-C-x(12)-H-x(3,5)-H`. As we can see, the gap-lengths between 2 and 12 would be required to discover the pattern `C-x(2,12)-C-x(2,12)-H-x(2,12)-H`, which is not as specific as `C-x(2,4)-C-x(12)-H-x(3,5)-H`, which we would like to discover.

Before discussing some of the possible approaches for solving the gap-minimization problem, let us consider a few more examples that illustrate the non-triviality of this problem.

Let the pattern be `A*A*A` and assume that we know that in the construction phase the gap `*` was actually a gap of length between 0 and 5 positions, `.{0,5}`. Then we could use these lengths 0 and 5 to come up with a pattern `A*{0,5}A*{0,5}A`.

If the two sequences in the input matched by that pattern were `ATTATTTA` and `ATTTATTA`, then clearly the most specific pattern π should be unambiguously `A.{2,3}A.{2,3}A` .

If the two sequences were `ATTATTTATTA` and `ATTTATTATTA`, then there are more choices to match pattern A.{0,5}A.{0,5}A. If the requirement was that one match per sequence is enough, the restricted length gaps can be replaced by fixed length gaps: A...A..A that matches once in both sequences. On the other hand, if all the original match positions should be preserved, the modified pattern should be A.{2,3}A.{2,3}A .

Ambiguity also comes from the question of what is meant by a "more restricted" version of the pattern. Let us consider two sequences CATTTATTTTA and CTTTATTTA, and a pattern C*A*A. There are two options to match this pattern: C.{0,3}A...A and C.{3,4}A.{3,4}A . Which pattern of the two is better? Note that the sum of the lengths for flexibilities is the same for both of the variants.

These examples show that in order to give any algorithms we first have to define the criteria for comparing the patterns. First, do we require the pattern in its limited form to match all the original input sequences, or do we require that the pattern should match all the locations where the original pattern matched? Second, what is the optimization function, how do we tell which presentation of the pattern is better than other? And third, how do we find the pattern with the highest score that occurs in all the required locations of the sequences?

These questions are not trivial as the answers may depend on the input data. In some cases it is sufficient to have only one motif per sequence, sometimes the motif should be repeated many times in sequences. Our task is to restrict the flexibilities of each pattern so that all the (biologically relevant) occurrences of the patterns can still be found. This gives us the formulation of the two problems:

1. Find a pattern with minimal total number of flexibilities that covers all the sequences it occurs in.

2. Find a pattern with minimal total number of flexibilities that covers all the matches within the sequences it occurs in.

The second formulation can be useful when dealing with sequences with multiple occurrences of the same motif. E.g. many of the Zinc finger proteins have up to ten repetitions of the same Zinc finger motif. However, is it biologically relevant to require all of these matches, or only the majority of them (assuming that other "hits" are just due to random features of the sequences).

As known from the studies in proteins and DNA, it is often the case in biology that the gap penalty should not be linear in the length of the gap but logarithmic. This is due to the fact that once there is an insertion or deletion, then it is more probable to extend an existing gap than to introduce a new one in a different place. Instead of the sum above we might want to minimize some other function, as for example $\Sigma_{i=1..n-1} \log(l_i - k_i)$. The different cost function models for penalizing different gaps should be developed together with the algorithms for finding them.

We do not propose at this stage any definite solutions to the problem of minimizing the gaps in patterns (as matched against the set of sequences). We propose this as an open problem that can have alternative interpretations and solutions.

3.5 Patterns with groups and unrestricted wildcards (P5)

We have moved to the pattern classes that combine different features. First, we consider a class of patterns that combines the group positions and unrestricted length wildcard positions.

Problem P5:B Given a set of strings $S^n = \{S_1, S_2, \ldots, S_n\}$, $S_i \in \Sigma^*$, and an integer K, construct all patterns π of type P5 ($\pi \in (\Sigma \cup \Gamma)^+ (*(\Sigma \cup \Gamma)^+)^*$) such that π has at least one occurrence in at least K sequences of $S_1, S_2, \ldots, S_n$.

The Algorithm 3.18 that implements the discovery of patterns for problem type P5:B, essentially combines the implementation ideas from the pattern classes P2 (how to handle group positions) and P3 (how to handle unrestricted wildcards).

The correctness follows from the correctness of the original algorithms, and the complexity analysis follows the ideas from the previous analysis.

Algorithm 3.18 Frequent subsequences with group positions from a set of sequences (P5:B)

Input: *Strings* $S^n = \{S_1, \ldots, S_n\}$ *, threshold* K*, character groups* Γ

Output: *Pattern trie with nodes defining patterns over* $(\Sigma \cup \Gamma)((\Sigma \cup \Gamma) \cup \{*(\Sigma \cup \Gamma))^*\}$ *that occur in at least* K *different strings from* S^n

Method:

```
$S \leftarrow S_1\# \ldots \#S_n\#$
Generate a mapping $\{1, 2, \ldots, |S|\} \rightarrow \{1, 2, \ldots, n\}$ for countseq($Set$)
$Root$ ← new node
$Root.char \leftarrow \lambda$
$Root.pos \leftarrow (1, 2, \ldots, |S|)$
$enqueue(Q, Root)$
while $N \leftarrow dequeue(Q)$
    $s \leftarrow 0$
    foreach $p \in N.pos$
        add $p + 1$ to $Set(S[p])$
        foreach $\gamma \in \Gamma$, s.t. $S[p] \in \gamma$ add $p + 1$ to $Set(\gamma)$
        if $N \neq Root$ and $p \in S_i$ where $i > s$   // Only if $p$ is within a new sequence $S_i$
            $s \leftarrow i$        // Remember which sequence $S_i$ is being studied
            $w \leftarrow 0$
            while $S[p + w] \neq' \#'$
                add $p + w + 1$ to $Set('*S[p + w]')$
                foreach $\gamma \in \Gamma$, s.t. $S[p + w] \in \gamma$ add $p + w + 1$ to $Set(\gamma)$
                $w$ ←w+1
    foreach character $c \in (\Sigma \cup \Gamma \cup \{*d|d \in \Sigma\} \cup \{*g|g \in \Gamma\})$ where $countseq(Set(c)) \geq K$
        $P$ ← new node
        $P.char \leftarrow c$
        $N.child(c) \leftarrow P$
        $P.pos \leftarrow Set(c)$
        $enqueue(Q, P, front)$          // Enqueue to the front of queue
    foreach character $c \in (\Sigma \cup \Gamma \cup \{*d|d \in \Sigma\} \cup \{*g|g \in \Gamma\})$ delete $Set(c)$
    delete $N.pos$
end
```

3.6 Patterns with groups and restricted-length wildcards (P6)

Finally, consider the pattern discovery problem where group character positions and wildcards of restricted length are allowed in the pattern language.

Problem P6:B Given a set of strings $S^n = \{S_1, S_2, \ldots, S_n\}$, $S_i \in \Sigma^*$, and an integer K, construct all substring patterns π with restricted-length wildcards $*(k,l)$ and group character positions (patterns of type P6, $\pi \in (\Sigma \cup \Gamma)^+(*(k,l)(\Sigma \cup \Gamma) \cup (\Sigma \cup \Gamma))^*$), such that π has at least one occurrence in at least K sequences of $S_1, S_2, \ldots, S_n$.

As with previous pattern language types, we can combine algorithms for creating more complex pattern tries. The computational complexity grows proportionally to the number of potential patterns, hence the time and space complexities of the combined algorithms also increase. In practice, however, these pattern classes are interesting and useful. For carefully selected input data sets the limited versions of these pattern classes can be used for practical pattern discovery purposes.

The complexity of the algorithms increases both in the number of group positions allowed in the pattern, as well as the number and lengths of the wildcards. As the search is exhaustive, the worst case time complexity can grow exponentially.

3.7 General representation of the SPEXS algorithms

To summarize the algorithms of this Chapter, we present a more general version of the pattern discovery problem.

General pattern discovery problem Given a set (or possibly several sets) of input strings, a pattern representation language $\mathcal{P}$, threshold K for the frequency of the patterns, a pattern "fitness" function $\mathcal{F}$, possible thresholds according to the fitness, frequency or other criteria, discover and output the patterns that have the highest fitness, are frequent in input and satisfy all the other criteria specified by users.

The general solution is to generate a pattern trie according to the pattern language while efficiently tracking the occurrences of patterns in the input sequences for speeding up the pattern extension steps.

Algorithm 3.19 *The SPEXS algorithm*
Input: *String S, pattern class $\mathcal{P}$, output criteria, search order, and fitness measure $\mathcal{F}$*
Output: *Patterns π from pattern class $\mathcal{P}$ fulfilling all the criteria, and output in the order of fitness $\mathcal{F}$*
Method:

```
Convert input sequences into a single sequence, initiate the data structures
$Root \leftarrow$ new node
$Root.label = \lambda$
$Root.pos \leftarrow (1, 2, \ldots, n)$ // Assume empty pattern to match everywhere
$enqueue(Q, Root, order)$
while $N \leftarrow dequeue(Q)$
    Create all possible extensions $P \in \mathcal{P}$ of $N$ using $N.pos$ and $S$
    foreach extension $P$ of $N$
        if pattern $P$ and position list $P.pos$ fulfill the criteria
        then
            $N.child \leftarrow P$
            calculate $\mathcal{F}(P, S)$
            $enqueue(Q, P, order)$ // Insert to $Q$ for further extensions
            if $P$ fulfills the output criteria store $P$ into output queue $O$
end
Report the list of top-ranking patterns from output queue $O$
```

Only the patterns that fulfill the output criteria are output in the order of fitness $\mathcal{F}$.

Note that the pattern generation (search) order is usually different from the output order. Alternative search orders can be used, like

1. Breadth first (from shorter to longer patterns)
2. Depth first (e.g. alphabetic order)
3. Most frequent motifs first
4. Greedy search heuristics based on fitness measure (expand the most promising patterns first).

For example, the breadth-first order is achieved by a FIFO queue, the depth-first order is achieved by a LIFO queue (stack). These can be easily implemented by priority search queues.

Different search orders may change the size complexity of the algorithm, depending for example on how much of the unnecessary information can be deleted during the search.

In practical applications we have found it useful to assume that the most interesting patterns are those that are common to many sequences. Using the SPEXS algorithm it is easy to generate patterns that are required to occur in all the input sequences, all except one, all except two, etc. As the most frequent patterns are the short ones that are not interesting as such, the SPEXS algorithm can be asked not to report those.

For multiple input sequence sets the frequency can also be based on any individual subset of input sequences. Then the search can proceed so that first all patterns that are common to all sequences in one set of the input sequences, in one minus two, etc., regardless how frequent or infrequent they are in other sets. Then the most discriminative patterns can be found relatively quickly as they should usually be frequent in one set and not frequent in other sets of input sequences.

3.8 Practicalities and discussion

In this study we have not discussed the maximality of the patterns. For better coverage on the subject see (Rigoutsos & Floratos 1998a). In our algorithms we expand patterns always to the right, and in that way the maximality would be harder to achieve. This is not a restriction however, as each “interesting” pattern, before it is output, can be expanded to its maximal variant in both directions so that no occurrence would be lost and each group position would contain only the minimal number of characters. This can be done essentially in a time proportional to the number of matches of the pattern and the length of the pattern span. For gap minimization the problem is harder however, as noted in previous sections.

With our pattern discovery approach it is possible that two different pattern prefixes (branches in the pattern trie giving alternative pattern representations) have exactly the same set of occurrences. As each branch would be expanded independently, the subtrees below these branches would be identical. In general, it is not possible to discover such patterns without comparing the sets of occurrences. In our algorithm we maintain the lists of locations of each pattern, and the problem can be solved by keeping track of which sets of positions have been studied earlier. This can be done by maintaining the set of position lists in the data structure, from where the lookups (whether exactly the same set of positions has occurred elsewhere in the pattern trie) can be made. We have experimented with this feature and found that for some data sets with conserved complex motifs we can get a considerable speedup to the pattern discovery process. This area of research will need to be investigated further.

Chapter 4

Pattern discovery from suffix trees

In this chapter we consider a slightly different pattern language. First we present a biological motivation.

Consider a DNA-binding protein that is able to recognize specific stretches of DNA. Assume that this stretch is represented by a substring pattern. The protein, however, is also able to recognize variants of the motif where one or a few positions have changed. We could define the binding site in the DNA as the most favorable for the protein. At the same time also all other reasonably similar stretches of the DNA would be potential binding sites. It may not be so significant in which positions exactly the bases of the DNA are different, as long as there are not too many of them. These changes can not be covered by group character positions within the motif.

To facilitate finding these patterns, we first define the approximate occurrences of a pattern allowing some characters to be changed, deleted, or added. These "near-perfect" matches cannot be captured by the regular patterns considered in Chapters 2 and 3.

We define two more pattern classes:

- Substring patterns with mismatch distance at most D
- Substring patterns with edit distance at most D

By restricting ourselves to substrings of S, we are not able to find patterns that occur only approximately and never as exact substrings of S. This harder problem is described and studied in (Pevzner & Sze 2000), who call it the Challenge Problem:

Challenge Problem. (Pevzner & Sze 2000) Find a signal in a sample of sequences, each 600 nucleotides long and each containing an unknown signal (pattern) of length 15 with 4 mismatches.

A quote from (Pevzner & Sze 2000): *Why is finding a rather strong (15; 4)-signal so difficult? The problem is that any two instances of the mutated (l;*

d)-signal may differ in as many as $2d$ positions. As a result, any two instances of the signal in the Challenge Problem may differ by as many as 8 mutations, a rather large number. The numerous spurious similarities with 8 mutations in 15 positions disguise the real signal and lead to difficulties in signal finding.

The Challenge Problem can be solved by first generating all the possible strings and then matching them approximately against S. The number of such patterns is $l^{|\Sigma|}$, where l is the length of patterns. There is over 1 billion such 15-mers over four-letter DNA alphabet.

We make a key simplification that the signal is present in at least one of the sequences. Other matches are then identifiable by an approximate matching of that exact motif.

A simple solution to the discovery of the above pattern classes would be to extract all substrings of S and match them approximately against the whole S.

We introduce a new all against all approximate matching procedure that is essentially a pattern discovery approach for approximately matching substrings. At the end of the chapter we describe how patterns with group characters (P2) defined in Section 2.1 can be discovered from a preconstructed suffix trie of S.

First, we formalize the notion of approximately matching (sub)strings.

4.1 Edit and mismatch distances and approximately matching patterns

The *edit distance* $d(S,T)$ between two strings S and T is defined as the minimum number of edit operations needed to transform a string S into T. The allowed edit operations are *insertion*, *deletion*, and *change*. An insertion operation inserts a character into the string, *i.e.* it replaces the empty string by a character from Σ. A deletion removes one character from the string. And a change replaces a character by a different one. The edit distance $d(S,T)$ can be calculated by the following recursive formula:

$$d(S,\lambda) = d(\lambda,S) = |S|$$

$$d(S,T) = d(S'c,T'b) = min(d(S',T'b)+1, d(S'c,T')+1, d(S',T')+t(c,b)),$$

where $S = S'c$ and $T = T'b$, and

$$t(c,b) = \begin{cases} 0 & \text{if } c = b \\ 1 & \text{otherwise} \end{cases}$$

The *mismatch distance*, also called the *Hamming distance*, is a special case of the edit distance with change as the only edit operation allowed. Mismatch distance $d(S, T)$ between two strings S and T of equal length can be calculated by the recursion

$$d(\lambda, \lambda) = 0$$

$$d(S, T) = d(S'c, T'b) = d(S', T') + t(c, b),$$

where S, S', T, T', and $t(c, b)$ are defined as for the edit distance.

Given a substring π of S and a distance D, the pattern π *matches the string* α *approximately* with distance at most D if there exists a substring β of α such that $d(\beta, \pi) \leq D$. We define the language of approximate pattern occurrences as

$$L(\pi, D) = \{ \beta \mid \beta \in \Sigma^*, and\, d(\pi, \beta) \leq D \}.$$

4.2 Approximate matching in suffix trees

The suffix trie data structure enumerates all the possible substrings of the sequence S. First we provide a framework for matching a single string approximately against the suffix trie, and then we generalize it for the all against all matching of all the substrings of S against all other substrings of S. Note that the algorithm works in a similar way also for the compact suffix tree version, where more care is needed in the implementation.

Consider a substring pattern P and a problem of finding its exact occurrences in S. This matching is achieved by simply following a unique path in the trie defined by P.

The approximate matching of pattern P under edit distance measure takes $O(|P| \cdot |S|)$ time by straightforward dynamic programming approach. By building a suffix tree for S the search can be improved by avoiding possible repeat regions in S (Ukkonen 1993). The approximate search of the pattern P against suffix tree T is usually performed by the depth-first dynamic programming approach against all the branches of the tree (Ukkonen 1993; Hunt, Atkinson, & Irving 2001). As the string-length of the suffix tree can be quadratic in $|S|$, it is not obvious that this matching can always outperform the trivial $O(|P| \cdot |S|)$ algorithm.

We propose an alternative algorithm for matching P approximately against suffix tree T. We perform a single search over T that follows the approximate matches of P to all the branches of the tree T, where the distances from the prefixes of P stay below the threshold D. Dynamic programming is simulated by maintaining lists of nodes organized according to the distance from the pattern P. We call this approach the *generalized traversal* of the tree.

We use the node identifier N also to represent the string associated with that node, $N.pattern()$. In this notation, $d(P[1..i], N)$ is the edit distance between the prefix of P and a substring $N.pattern()$ of S. For a string P we denote nodes N in suffix trie T such that $d(P, N) \leq D$ as follows:

$$Q(P) = \{N \mid N \in T, d(P, N) \leq D\}.$$

We divide $Q(P)$ further into subsets

$$Q^i(P) = \{N \mid N \in T, d(P, N) = i\}.$$

Obviously,

$$Q(P) = \bigcup_{i=0}^{D} Q^i(P).$$

The generalized traversal is given in Algorithm 4.1. We assume that $Q(P[1..i])$ can be calculated efficiently from $Q(P[1..i-1])$ and the new character $P[i]$. We denote this by function $newdistances(Q(P[1..i-1]), P[i])$ on line 3 of the algorithm.

Algorithm 4.1 *Find approximate occurrences of string P in S*
Input: *Suffix trie T for string S, pattern P, distance D*
Output: *All substrings s of S such that $d(P, s) \leq D$.*
Method:

```
1.  $Q(\lambda) \leftarrow \{N \mid N \in T, d(\lambda, N) \leq D\}$
2.  foreach $i \in \{1..|P|\}$
3.      $Q(P[1..i]) \leftarrow newdistances(Q(P[1..i-1]), P[i])$
4.      if $Q(P[1..i]) = \emptyset$ then return "No match"
5.  foreach $N \in Q(P)$
6.      print $N.pattern()$ // A substring represented by node $N$
```

Using this generalized traversal, we can introduce the all-against-all approximate matching algorithm that finds for each substring p of S all other substrings s of S such that $d(p, s) \leq D$. Algorithm 4.2 traverses the tree T in a depth-first manner and computes for each node in the tree the lists of nodes that are within the distance D.

Algorithm 4.2 *General approximate all against all matching*
Input: *Suffix trie T for string S, distance threshold D*
Output: *All substrings p of S, with all substrings s of S such that $d(p, s) \leq D$.*
Method:

```
procedure Match$(P, Q)$
    Output the pattern $P$ and its approximate occurrences $N$, $N \in Q$
    foreach $P.child(c)$
        $Q' \leftarrow newdistances(Q, c)$
        $Match$(P.child(c), Q') unless $Q' = \emptyset$
end
begin
    $Q \leftarrow \{N \mid N \in T \textbf{ and } d(\lambda, N) \leq D \}$
    Match( $T.root, Q$ )
end
```

Next we show how to implement Algorithms 4.1 and 4.2 efficiently for the mismatch distance and the edit distance.

4.3 Substring patterns with D mismatches

Problem P1:A:mismatch Given a string $S \in \Sigma^*$, and two integers K, D, construct all substring patterns π (patterns of type P1) such that π has at least K approximate matches in S with mismatch distance at most D.

The proposed Algorithm 4.3 traverses the trie T in depth-first manner. For each node $P \in T$, a list of nodes $Q(N)$ is generated such that $d(P, N) \leq D$. The lists $Q(P)$ are organized into sublists $Q^i(P)$ based on distance i from P, as defined in the previous subsection.

The extension of the pattern P by one character c at a time is given on lines 4–10, which essentially implement the function $newdistances(Q(P), P.child(c))$ of Algorithm 4.2. The temporary lists of nodes R^i are used to collect the results during the extension by c.

Algorithm 4.3 *All-against-all matching under mismatch distance*
Input: *Suffix trie T for string S, and integer K and distance D*
Output: *Substrings of S that have at least K approximate matches in S with mismatch distance at most D.*
Method:

```
1.  procedure Match($P, Q$)
2.      Output pattern $P$ and all its matches $N$ , $N \in Q = \cup_{i=0}^{D} Q^i$
3.      foreach $P.child(c)$
4.          foreach $i \in \{0..D\}$
5.              foreach $M \in Q^i$
6.                  foreach $M.child(e)$
7.                      if $e = c$
8.                      then enqueue( $R^i, M.child(e)$ )
9.                      else enqueue( $R^{i+1}, M.child(e)$ ) unless $i + 1 > D$
10.         $R \leftarrow \bigcup_{i=0}^{D} R^i$
11.         if $\sum_{r \in R} |r.pos| \geq K$ then Match($P.child(c), R$)
12. end
13. begin
14.     $R^0 \leftarrow T.root$
15.     Match($T.root, R^0$)
16. end
```

The mismatch distance is defined only for sequences of the same length. The occurrences of different substrings of equal length are disjoint. Hence, the number of approximate occurrences of a pattern is the sum of the number of occurrences of each approximately matching substring and the test on line 11 correctly calculates the number of approximate matches by pattern $N.child(c).pattern$.

Under mismatch distance there can be at most D possible character changes from the pattern. For a string P of length l there is a total of $\sum_{i=1}^{D} \binom{l}{i} \cdot (|\Sigma| - 1)^i$ strings α of length l with $1 \leq d(P, \alpha) \leq D$: if there is i changes, they can be in any i positions out of l possible ones. Each mismatch can replace a character in P by any of the $|\Sigma| - 1$ remaining characters. Each list Q can contain only the nodes at the same depth in the tree, thus the worst case estimate for the number of nodes in all different lists during the depth-first trie traversal is the size of the trie. This gives us the worst-case space requirement for the Algorithm 4.3.

Algorithm 4.3 reports all the substrings p of S, and for each p a list of all other substrings s of S with distance $d(p, s) \leq D$. For this the algorithm visits each node P in the trie T and outputs the substrings from the list Q containing nodes with edit distance at most D from P. In order to output a substring corresponding to a child $P.child(c)$, the new list R needs to be constructed from Q first. Potentially every child of every node in Q needs to be considered for inclusion into R. There are at most $min(|Q| \cdot |\Sigma|, w)$ such nodes, where w is the width of the trie at that level. For constructing the new list R the algorithm visits the nodes in the list Q and each of their children. As the alphabet size $|\Sigma|$ can be considered constant, the running time of the Algorithm 4.3 is linear in the size of output.

Algorithm 4.3 avoids visiting subtrees where respective substrings are known to occur less frequently than K times in S (see line 11). If the substring α, represented by a node $N(\alpha)$, occurs in K different locations with mismatch distance of at most D, then any extension αc can occur in no more than K locations with distance D. If the number of approximate occurrences of α drops below K, we can skip the study of the subtrees of $N(\alpha)$.

The number of all possible strings with mismatch distance at most D grows exponentially with the length of the string and number of mismatches allowed. However, in the suffix trie the width of the trie determines the possible size of lists Q.

4.4 Substring patterns with edit distance D

In this section we generalize approximate patterns from using the mismatch distance to the more general edit distance, allowing also insertions and deletions.

Problem P1:match:edit Given a pattern $P \in \Sigma^*$, a string $S \in \Sigma^*$, and distance threshold D, find all the approximate matches of P in S with edit distance at most D from P.

For each prefix p of P we generate a list of nodes in the trie that have distance at most D to the prefix p, *i.e.* $Q(p) = (Q^0(p), Q^1(p), \ldots, Q^D(p))$ and $R \in Q^i(p)$ if $d(p, R) = i$. The sublists $Q^i(p), Q^j(p)$ are disjoint when $i \neq j$.

Given a pattern P and the suffix trie T of S we want to find all the nodes corresponding to substrings s of S with edit distance $d(P, s) \leq D$. We use the generalized tree walk approach for matching the pattern P against all the branches of the trie simultaneously. The algorithm proceeds by identifying all the nodes from T that correspond to substrings with edit distance of at most D to prefixes $P[1..1], P[1..2], \ldots, P[1..|P|]$ of P. The nodes corresponding to substrings with an edit distance i are kept in lists Q^i.

Edit distance has been defined as the minimum number of edit operations needed to transform one string to another. There can be many other ways as well, which are not necessarily optimal. For example, the string `AG` can be converted to `ATG` by first substituting `G` to `T` and then inserting `G`. In a more effective way, it is enough to insert `T` between `A` and `G`. To make sure that the minimal distance is always considered, we perform the minimization step (lines 11 to 13). Essentially, the node N is removed from a list Q^j if it already belongs to a list Q^i where $i < j$.

Algorithm 4.4 *Algorithm for finding approximate occurrences of string P in S*
Input: *Suffix trie T for string S, pattern P, edit distance threshold D*
Output: *Substrings s of S such that $d(s, P) \leq D$.*
Method:

```
foreach $(i \in \{0..d\})$ $Q^i \leftarrow \{N \mid d(\lambda, N) = i\}$
$Q \leftarrow \bigcup_{i=0}^{D} Q^i$
foreach $j \in \{1..|P|\}$
    $c \leftarrow P[j]$
    foreach $i \in \{0..D\}$
        $Q^i(P[1..j]) \leftarrow \{N|N.parent \in Q^i(P[1..j-1]), N.label = c\}$// Identity
                $\bigcup\{N \mid N.parent \in Q^{i-1}(P[1..j-1]), N.label \neq c\}$ // Change
                $\bigcup\{N \mid N.parent \in Q^{i-1}(P[1..j])\}$ // Insertion
                $\bigcup\{N \mid N \in Q^{i-1}(P[1..j-1])\}$ // Deletion
    $Q(P[1..j]) \leftarrow \bigcup_{i=0}^{D} Q^i(P[1..j])$
    foreach $N \in Q(P[1..j])$
        if $\exists m < i\ such that\ N \in Q^m(P[1..j])$
        then remove $N$ from $Q^i(P[1..j])$
    if $Q(P[1..j]) = \emptyset$ then return "No match"
Report all approximate occurrences of pattern $P$ from $Q(P)$
```

Problem P1:A:edit Given a string $S \in \Sigma^*$, and two integers K, D, construct all substring patterns π (patterns of type P1) such that π has at least K approximate matches in S with edit distance at most D.

All-against-all matching of substrings of S can be done by traversing the suffix tree T of S and computing for each node N the set of nodes in T that are within distance d from N. We present a procedural implementation for finding the approximately matching substring patterns in Algorithm 4.5 .

Algorithm 4.5 *Approximate all-against-all matching under edit distance measure*
Input: *Suffix trie T for string S, integers K and edit distance threshold D*
Output: *Substrings of S with at least K approximate matches in S with distance at most D*
Method:

```
procedure Match$(P, Q(P))$
    Output the pattern $P$ and its approximate occurrences $N$, $N \in Q(P)$
    foreach $P' \leftarrow P.child(c)$
        foreach $i \in \{0..D\}$
            foreach $M \in Q^i(P)$
                enqueue$(Q^{i+1}(P'), M)$  unless  $i + 1 > D$ // Deletion of $c$
                foreach $M.child(e)$
                    if  $e = c$  then  enqueue( $Q^i(P'), M.child(e))$  // Identity
                    else  enqueue$(Q^{i+1}(P'), M.child(e))$  unless  $i + 1 > D$ // Change
            foreach $M \in Q^i(P')$
                foreach $M.child(e)$
                    enqueue$(Q^{i+1}(P'), M.child(e))$  unless  $i + 1 > D$ // Insertion of $e$
        $Q(P') \leftarrow \cup_{i=0}^{d} Q^i(P')$
        foreach $M \in Q(P')$
            remove $M$ from $Q^i(P')$  if  $\exists j < i, M \in Q^j(P')$
        Match $(P', Q(P'))$  if  number of matches of $M \in Q(P') \geq K$
end
begin
    foreach $i \in \{0..D\}$
        $Q^i \leftarrow \{N \mid d(\lambda, N) = D\}$
    $Q \leftarrow \cup_{i=0}^{D} Q^i$
    Match( T.root , Q )
end
```

The complexity of Algorithm 4.5 for string S of length n can be estimated as follows. First, the full suffix tree is traversed. At each node P the work is proportional to the time used to calculate new lists for its children P', in other words, to calculate $Q(P')$ from $Q(P)$, such that $d(P', N) \leq D$ for all $N \in Q(P')$. This is done by minimizing over 4 possible edit operations (identity, change, insertion, and deletion) from lists of nodes $Q(P)$ and $Q(P')$.

The list of nodes $Q(P)$ contains only those nodes N, for which $d(P, N) \leq D$, which is also the size of the output for substring $N.pattern()$. Thus, the work is proportional to the size of the output.

The lengths of the strings that have edit distance at most D to a string s of length l, cannot differ by more than D from l. Suffix trie T for the string S of length n can have at most $n-l+1$ nodes representing different substrings of length l. Thus for any substring of S there can be at most $(2D+1)(n-l) = O(Dn)$ substrings of S within the edit distance D. The size of the trie can be quadratic $O(n^2)$ hence the algorithm has worst case time complexity $O(Dn^3)$.

An all-against-all sequence matching algorithm that aims at finding pairwise distances for all possible substring pairs has been proposed before (Baeza-Yates & Gonnet 1990; 1999). If a database has N sequences, each with an average of m characters (a total of Nm characters), a naive approach would require $O(N^2m^4)$ operations. The solution in (Baeza-Yates & Gonnet 1999) provides an $O((Nm)^{\alpha}m\log(Nm))$ average time solution, where α is a real number between 0 and 2, depending on the similarity matrix used. The same problem can be studied for two strings P and S of lengths m and n respectively. The all-against-all approximate matching problem can call for a total output of size n^2m^2. A trivial algorithm that does not use suffix trees can be used to achieve the calculation time of $O(n^2m^2)$ by simply matching each of the substrings of shorter string P against longer sequence S. The suffix-tree based solution of (Baeza-Yates & Gonnet 1999) has a worst-case time complexity $O(C+R)$ where C is the time used for computations and R is the size of the output (Gusfield 1997). The computation time C is the suffix-tree length of the suffix tree for string S times the length of the suffix tree for string T.

Our approach has a similar time complexity to that of (Baeza-Yates & Gonnet 1999), but as the method is presented in quite a different form, it may be easier to implement with all the required modifications necessary for efficient frequent pattern discovery. The evaluation of the two methods would need to be measured in practical applications.

4.5 Patterns with group characters

In Section 3 we studied the systematic construction of patterns from a predefined pattern class by creating a pattern trie. Here we study the same problem of pattern discovery based on the suffix trie T of S.

Generalized regular patterns can be searched from T by comparing them against all possible branches of T. For example, occurrences of [AB]ABA are found by looking for nodes $N(A)$ and $N(B)$ and then following the paths ABA from both of these. The result contains two substrings AABA and BABA if they are both present in the data. Instead of building a pattern trie as in Chapter 3 it is possible to use an implicit search over the pattern language to perform the generalized tree-walk in the precomputed suffix trie.

For the generalized tree-walk we use the list of nodes as a generalization of a single node. For example to follow the path labeled by character 'A' from this generalized node, we follow the paths labeled 'A' from all nodes in the corresponding list. To insert a group character $[\mathtt{AB}]$ in the pattern α we need to follow all potential branches, *i.e.* the labels A and B from all nodes in the list.

In our example, given a list of nodes (generalized node, which we represent by pattern prefixes) $N([\mathtt{AB}])$, the tree-traversal algorithm traverses at first nodes $N([\mathtt{AB}]\mathtt{A})$, then $N([\mathtt{AB}]\mathtt{B})$ etc. After traversing paths with single-character labels, also the group characters can be followed. Thus, the generalized node $N([\mathtt{AB}][\mathtt{AB}])$ corresponds to a list of four nodes: $N(\mathtt{AA})$, $N(\mathtt{AB})$, $N(\mathtt{BA})$, and $N(\mathtt{BB})$.

Algorithm 4.6 *Generate patterns with group characters*
Input: *Suffix trie T for string S, integer K, character groups $\Gamma = \{g_1, \ldots, g_{|\Gamma|}\}$*
Output: *All substring patterns with group characters that occur at least K times in S*
Method:

```
procedure tree-walk $(P, L)$
    // Pattern $P$ matches all substrings of $S$ defined by nodes in list $L$
    Output $P$ and combined information about it from nodes in $L$
    foreach $c \in \Sigma$
        $Q \leftarrow \{N \mid N.parent \in L, N.label = c\}$
        tree-walk$(Pc, Q)$ if $\sum_{N \in Q} |N.pos| \geq K$
    foreach $g \in \Gamma$
        $Q \leftarrow \{N \mid N.parent \in L, N.label \in g\}$
        tree-walk$(Pg, Q)$ if $\sum_{N \in Q} |N.pos| \geq K$
end

begin
    $Q \leftarrow \{T.root\}$
    tree-walk$(\lambda, Q)$
end
```

Algorithm 4.6 is similar to the algorithms presented in Section 3.2. Here the explicit construction of the pattern trie is replaced by an implicit search over the pattern space while using a suffix trie for evaluation of the frequency of the occurrences of each pattern. When pattern occurrence frequency drops below K, the search is stopped and no other extensions of the pattern are attempted.

Similarly to the Algorithm 4.6 it is possible to modify all algorithms from Chapter 3 to use the search over the pattern space combined with efficient pruning of that space by using the preconstructed suffix trees. The time complexity of the algorithms will not change, although depending on the frequency thresholds K the search can consume less memory (if threshold K is very low). On the other hand, for large K the construction of the suffix tree can be time-consuming and could be avoided.

Chapter 5

Post-processing discovered patterns

In the previous chapters we studied several pattern discovery problems and algorithms for solving them. In practical applications for novel data sets, users need to make decisions (educated guesses) about the ways they approach the pattern discovery problem. The obvious questions are the choice of pattern language, the fitness functions, and significance cut-off values.

When analyzing a novel data set it is not known in advance whether the data contains any patterns that would be described as interesting by the domain experts. Some of the findings may be trivial in the sense that they are well-known, or they are just some artifacts of the data like simple repeats, for example. Some of the interesting features in the real data may occur in the *twilight zone*, where their significance is obscured by occurrences of many irrelevant patterns.

It is quite common that pattern discovery tools report hundreds or thousands of patterns, which need to be summarized in order to make them useful for the researchers. For example, it is common that pattern discovery is performed to many sets of sequences independently, and the results need to be collected and summarized across these analyses. Some of the discovered patterns may be variants of each other. They may differ only in one position or add a few extra positions to the left or right of the pattern.

In this chapter we discuss some techniques that help to deal with the problem of managing the size of the output by postprocessing the discovered patterns. We do not attempt to be complete or even describe all the approaches in detail. Rather we want to present and discuss some of the practical approaches which we have found useful ourselves while analyzing real data sets.

5.1 Clustering of patterns by their mutual similarity

In a recent study (Vilo *et al.* 2000) (see also Chapter 6.1), we applied pattern discovery for many sets of sequences, partly overlapping with each other. The patterns were discovered by independent analysis tasks, often revealing either exactly the same or only slightly different representations of the same motifs. When we combined the results from these independent pattern discovery tasks we had an excess of interesting patterns, too large for human inspection.

We used a clustering technique to group together patterns that were likely to represent the same biological motifs. Most of the common clustering procedures use the pairwise similarity between the objects to be clustered, and perform the clustering based on these distances. The following distance measures can be used for clustering patterns.

5.1.1 Pattern similarity measures

5.1.1.1 Similarity based on edit and mismatch distances

The two simplest measures are based on the mismatch distance and edit distance defined in Chapter 4. These measures however are not directly usable, as the distance is not normalized by sequence length. For example, the following two pairs of sequences both have the same edit distance between them $d(\mathtt{AT}, \mathtt{CG}) = 2$, and $d(\mathtt{ATCGATCGATCG}, \mathtt{ATCGAAGGATCG}) = 2$, although the first pair clearly does not represent similar motifs. By normalizing the edit distance by the length of the shorter sequence, a distance measure that is also a metric, can be defined:

$$d_d(\alpha, \beta) = \frac{d(\alpha, \beta)}{\min(|\alpha|, |\beta|)}.$$

This distance can be used for comparing substring patterns. It is possible to generalize edit and mismatch distances for patterns with group character positions.

5.1.1.2 Pattern similarity based on mutual information content

Consider patterns with character group positions *i.e.* the patterns of the form $\pi = a_1 a_2 \ldots a_p$, where each a_i is a non-empty set of nucleotide letters, *i.e.* , $a_i \subseteq \{A, C, T, G\}$. For this case the similarity can be measured using a definition of information content $I(\pi)$ of a pattern π (Jonassen, Collins, & Higgins 1995). Let $\pi_{1,2}$ be a pattern from the predefined pattern class that is a generalization both of π_1 and of π_2 maximizing the information content $I(\pi_{1,2})$. The similarity between pattern π_1 and π_2 is then defined as

$$d_I(\pi_1, \pi_2) = \frac{I(\pi_{1,2})}{\min(I(\pi_1), I(\pi_2))}.$$

In our experiments with yeast (Vilo *et al.* 2000) (see also Section 6.1.3) we clustered substring patterns using substring patterns also as the class of generalizations of the two substring patterns π_1 and π_2. The similarity of two substring patterns is then the ratio of the length of the maximum overlap between π_1 and π_2 divided by the length of the shorter pattern.

This distance can not be directly used for patterns with wildcards as the pattern generalization $\pi_{1,2}$ is not well defined.

5.1.1.3 Distance measures based on the match content

For complex pattern classes the similarity of the patterns may be hard to determine from the patterns themselves. Wildcards, for example, can hide important regions that would allow to determine the similarity. In order to compare patterns it is sometimes useful to measure the similarity between the original sequence stretches matched by the patterns, not the patterns themselves. Note that each pattern may define a large set of matched substrings in input sequences, hence the similarity measure should be based on the similarity between two sets of sequences. This is a rather generic approach, as it can be used even when original patterns are allowed to match approximately.

5.1.2 Clustering of patterns

Given a distance measure for comparing pairwise similarities between the patterns, any clustering procedure that relies on pairwise distances can be used for clustering the patterns. A standard hierarchical clustering that merges smaller clusters (initially consisting of single objects) into larger ones. This tree hierarchy can be explored for example by cutting the tree at a certain height, or used interactively by looking at smaller and more restrictive clusters. The hierarchical clustering of substring patterns has been implemented in the EPCLUST package of Expression Profiler (see Section 7.1).

Partitioning based clustering methods are an alternative to hierarchical clustering. Many of these standard methods (K-means, Self Organizing Maps, etc) use the Euclidean properties of the object space, for example for defining the cluster centers. This is not appropriate for sequence patterns. Alternatively, graph theory based methods could be used, for example.

The K-medoids clustering could also be readily used for pattern clustering. K-medoids is similar to K-means, only it uses as cluster centers the original data objects. Instead of moving cluster centers to the center of gravity of a particular cluster, the object which appears to be most central to the cluster is chosen.

Once the patterns are clustered, a report about each cluster is shown to users. The report may simply be a list of all original patterns in each cluster, an alignment of patterns, a consensus sequence representing all patterns in a cluster, a list of sequences matched by patterns, an alignment of the sequences matched by patterns, a position weight matrix generated from that alignment, or its graphical representation by sequence logos. These features are implemented in Expression Profiler with a combination of EPCLUST, URLMAP, and SEQLOGO packages. The position weight matrices are described in the next subsection and a strategy for identifying them is outlined.

5.2 Approximate patterns and probabilistic profiles

In biological applications the discovered patterns identify certain features and characters on sequences corresponding to regions and atoms in real physical molecules. Often the real physical properties are not as deterministic on-off properties as the discrete patterns that either match or do not match the sequence. For example, the proteins that recognize certain features of DNA may recognize ”fairly similar” sequences of DNA. The concept of similarity is usually more complex than a simple edit or mismatch distance used in Chapter 4. Usually the cost of the mismatch is context-dependent, *i.e.* in some positions the change is more permissible than in others. An elegant way to capture this is based on position weight matrices, which assign a different weight or cost to every position of a motif. For the history of DNA binding site representation and discovery see (Stormo 2000).

The most commonly used weighting scheme for position weight matrices is to use the information content of each position normalized based on the base frequency of each site.

$$I(i) = \sum_{b \in \{A,C,G,T\}} f_{b,i} \log_2 \frac{f_{b,i}}{p_b}$$

In the above formula $f_{b,i}$ is the observed frequency of character b in position i of the set of aligned sequences. The base probability p_b corresponds to the probability of occurrence of a character in the entire genome. When matching the position weight matrix against a sequence, the weights $f_{b,i} \log_2 \frac{f_{b,i}}{p_b}$ are summed over the length of the sequence for each position i based on the base b occurring at that position. Depending on the cut-off threshold one can then determine whether the matrix matches the sequence or not.

The most common methods for identifying these position weight matrices are based on Expectation Maximization which is computationally intensive. Here we propose an alternative algorithm based on the previously identified discrete pattern.

Assume that we are given a set of positive sequences S_+ and a set of negative sequences S_- and a pattern π that is more frequent in S_+ than in S_- based on the ratio or probability as defined in Sections 6.1.1 and 6.1.3.2 respectively. The task is to modify π into a position weight matrix representation while increasing the sensitivity of the pattern and not losing the specificity.

Algorithm 5.1 (Pattern2Motif) Position weight matrix generation
Input: *Two sets of strings* S_+, S_-, *fitness measure* $\mathcal{F}$, *pattern* π
Output: *Position weight matrix* $PWM_{\pi,d}$
Method:

```
1.  for $d = 1, 2, \ldots D$ // $D$ is usually small, 2-5
2.      Match $\pi$ against sequences of $S_+$ using approximate matching
        allowing for $d$ mismatches
3.      Create a position weight matrix $PWM_{\pi,d}$ based on the matched regions
4.      Estimate the fitness $\mathcal{F}(PWM_{\pi,d}, S_+, S_-)$
5.  Output the $PWM_{\pi,d}$ that gave the best $\mathcal{F}(PWM_{\pi,d}, S_+, S_-)$
```

We show in Section 7.2 how this simple procedure, applied in a manual manner, has allowed us to improve the quality of the prediction of the DNA binding site. The full capability of such an approach needs to be investigated further.

This approximate matching based approach uses discrete patterns for speeding up the search for the most interesting probabilistic weight matrices. Usually the techniques for identifying these probabilistic profiles are time-consuming. The discovered matrices can be used as such or they can be further refined by seeding them as an input to Expectation Maximization based methods.

Position weight matrices however are not able to handle insertions and deletions (indels), hence that representation may not be appropriate for patterns with flexible wild-card positions or regions matched by patterns approximately, if the approximate matching allowed for indels.

5.3 Secondary pattern discovery - patterns from patterns

A strong recurring feature that is common to many of the interesting patterns may be the core of the discovered patterns. Thus, it may be useful to repeat the pattern discovery on a set of interesting patterns. A pattern that covers a large set of interesting motifs, and the respective alignment of these motifs based on the common pattern can be shown to end-users.

Given a list of interesting patterns we are faced with a family characterization problem. In a separate study (Brazma *et al.* 1996; Brazma, Ukkonen, & Vilo 1996) we developed a general pattern fitness criteria for the pattern conservation problem based on the Minimum Description Length (MDL) and demonstrated how this measure can be used directly both for finding the patterns as well as for producing the grouping of the sequences based on these patterns (see Section 6.3).

This approach would be suitable for pattern discovery from interesting patterns providing a conceptual clustering type of approach where each cluster is described by a suitable concept, the consensus pattern.

5.4 Visualization techniques

The list of patterns, even if summarized by the methods described above, does not tell the context in which these patterns occur. For users, it is often more useful to see where along the sequences the patterns occur. Visualization allows to determine the relationships between the important regions in the sequences. An effective visualization technique is to display sequences graphically and visualize the regions that are matched by each pattern by color-coding.

We have developed methods and tools that allow to visualize the sequences and occurrences of motifs on these sequences alongside with other information about these sequences. Most notably, we have combined the visualization of the gene expression profiles and gene regulatory sequences with potentially important regulatory signals along these upstream sequences (see Section 7.1, and Figures 7.2, 7.3). Through such a combined visualization a new quality in understanding the significance of discovered patterns can be achieved.

Another example of the usefulness and applicability of these visualization methods is discussed in Section 6.2. There, the visualization shows the protein topology (all the start positions of intracellular loops are aligned) and occurrences of patterns on the different regions (intracellular loops) along these sequences.

Chapter 6

Applications and experimental results

6.1 Discovery of the putative transcription factor binding sites

The 12 million character long complete genome of the yeast *S.cerevisiae* has been publicly available for a while (Goffeau *et al.* 1996), thus making it one of the first major model organisms for data analysis methods development. It has been estimated that yeast contains about 6000 genes, i.e., fragments of the DNA that encode the proteins. The putative genes (so-called open reading frames - ORFs) have been annotated in several yeast databases (MIPS, SGD, YPD).

A major role in gene regulation in eukaryotic organisms is played by specific proteins, called *transcription factors*. By binding to sequence-specific sites in the DNA, called *transcription factor binding sites*, they influence the transcription of a particular gene. The transcription factor binding sites are located in *promoter regions*. In yeast these regions are predominantly (but not exclusively) in the immediate vicinity of the gene (typically less than a thousand basepairs upstream of the translation start site). These sites are specific DNA sequences of length from about 5 to 25 nucleic acids, and in yeast they are usually located within a few hundreds of nucleotides upstream from the gene. For the schematic overview of information flow inside the cells, see Figure 6.1.

Recently many surveys about transcription factor binding site representation and discovery methods have been published. The history of these methods has been discussed in (Stormo 2000), mathematical and algorithmic models for promoter sequence analysis with the emphasis on prokaryotic promoter sequences in (Vanet, Marsan, & Sagot 1999), and combining the gene expression data and promoter analysis in (Zhang 1999; Vilo & Kivinen 2001; Ohler & Niemann 2001; Werner 2001).

We describe here computational experiments on yeast *Saccharomyces cere-*

visiae that are aimed at discovering the binding sites. The first method is based on direct comparison of upstream sequences (putative promoter regions) to genomic DNA; and the second one utilizes the gene expression microarray data for first grouping together genes that are coexpressed and then identifying the motifs in the upstream sequences of the coexpressed genes.

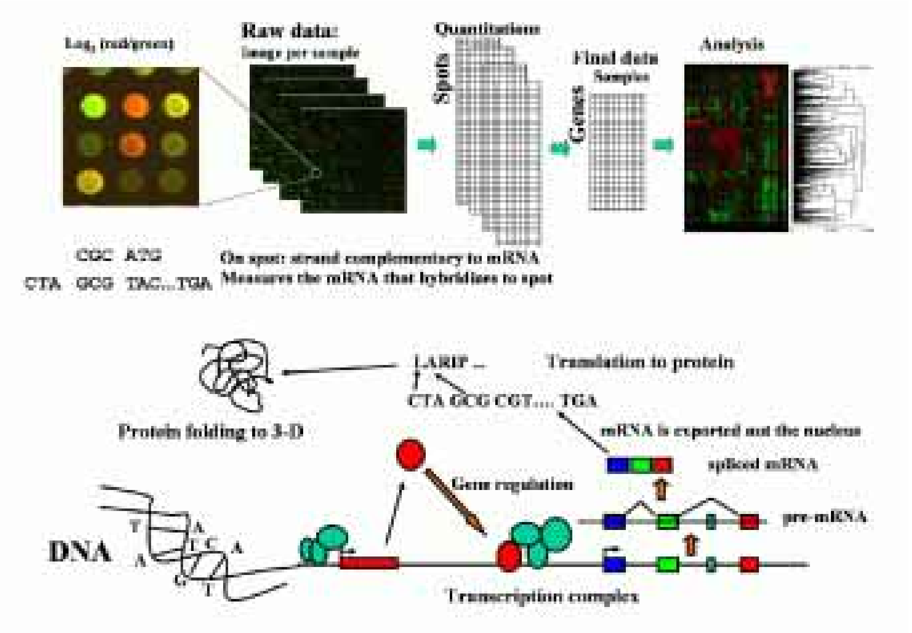


Figure 6.1: Schematic view of the information flow in the cell and the microarray technology measuring the mRNA expression levels. Genes in the DNA encode the proteins, which are produced by first transcribing the DNA into mRNA taking into account the intron-exon structure of the gene, and then translating the mRNA into amino-acid sequences of proteins. Microarray holds the fingerprints of genes, the single-stranded DNA complementary to the mRNA molecules. The mRNA from the two samples are then extracted, labeled by different fluorescence labels (e.g. red and green respectively), mixed together and hybridized competitively on the microarrays. The relative abundances of the mRNA in two samples are quantified by comparing red and green channel intensities on the two channels. Spot measurements and possible replicate experiments are then normalized and transformed into gene expression data matrix where gene expression levels (rows) in different samples (columns) are represented.

6.1.1 Comparison of upstream sequences to random genomic regions on a full genomic scale

One of the first questions when analyzing full genomes is whether there are any particular features in the putative promoter regions that would make them different from the rest of the genome. If there are, then we could identify putative transcription factor binding sites by purely comparing the upstream regions to other genomic regions. We have performed this analysis in a systematic manner for the yeast *Saccharomyces cerevisiae* (Brazma *et al.* 1998b).

The motivation is to find the patterns that are more often observed in the upstream regions than in the genome in general. To evaluate the patterns we use as the fitness function $\mathcal{F}$ the *ratio* of the number of positive examples matching the pattern divided by the number of negative examples matching the pattern. Intuitively, this ratio tells how much more frequent are the occurrences in the upstream sequences as compared to the genome in general. Additionally, we require that the pattern matches some minimal number t of upstream sequences. We say that the upstream sequences form the positive examples and random genomic regions form the negative examples, and define the ratio as follows.

$$R_t(\pi, S_+, S_-) = \begin{cases} \frac{|S_+ \cap L(\pi)|}{|S_- \cap L(\pi)|}, & \text{if } |S_+ \cap L(\pi)| \geq t \text{ and } |S_- \cap L(\pi)| > 0 \\ \infty, & \text{if } |S_+ \cap L(\pi)| \geq t \text{ and } |S_- \cap L(\pi)| = 0 \\ 0, & \text{otherwise} \end{cases} \tag{6.1}$$

A normalized version of this ratio takes into account the possible differences in the sizes of sets S_+ and S_-. Additionally, by adding two small constants c_1 and c_2 to the numbers of occurrences in each input set, we can eliminate the “division by zero” problem and the problem of having possibly very high ratios when numbers of pattern occurrences are very small.

$$R'(\pi, S_+, S_-) = \frac{(|S_+ \cap L(\pi)| + c_1)(|S_-|)}{(|S_- \cap L(\pi)| + c_2)(|S_+|)} \tag{6.2}$$

We selected all sequences of length 100, 300 and 600 nucleotides upstream to every annotated gene. For each upstream region we also extracted a random region of the same length from the genome. We searched these sets of sequences for all the patterns from the classes $P1$ and $P2$ (allowing only for a limited number of single-position wildcard characters, *i.e.* allowing only for one character group $g = \Sigma$ for no more than in a few positions in π) that occur in at least some given number (in practice 10) of upstream sequences, and rated these patterns according to the fitness function $R_t(\pi, S_+, S_-)$ given in equation (6.1), treating the random genomic regions as the negative examples. We compared the distribution of pattern occurrences in upstream sequences to the distribution in the random regions.

The results from the control experiments comparing two sets of random regions show a clear difference between upstream and random sequences, suggesting that there are specific sequence patterns distinguishing the upstream regions from random ones, and that this cannot be explained as a statistical coincidence. See Figure 6.2 for an overview of the results.

The inspection of the patterns that occur substantially more frequently in upstream regions than in random (i.e., having a high rating $R(\pi, S_+, S_-)$) showed, that many of these can be regarded as "simple" (i.e., easily compressible) sequences (e.g., AAAAAAATA). This is not unexpected, as upstream regions have higher A-T content, while genes rarely contain long simple sequences. Among other patterns, one of the top scoring was a pattern AAAGCGAAA. Matching this pattern against the transcription factor database TRANSFAC (Wingender *et al.* 1996) we found that it was similar to the binding site for the yeast transcription factor URS1 (Turi & Loper 1992). The pattern given for URS1 in TRANSFAC is AAACGAAACGAAACGAAACTAA. This pattern has only one single match in the entire yeast genome, which means that it cannot be a generic actual binding site. The pattern AAAGCGAAA, on the other hand, has 119 matches in the upstream regions of length 600, and a total of 222 matches in the full yeast genome of 12Mbp. For more detailed analysis and patterns see (Brazma *et al.* 1998b).

The study shows that, given a new genome with computationally predicted genes, some putative transcription factor binding site descriptions can be generated automatically, without any other background knowledge about the real transcription factors or their binding sites for the given organism. We have made similar observations also for another yeast *S. pombe* and the worm *C. elegans* (unpublished results).

6.1.2 Gene expression data analysis and putative transcription factor binding site prediction

Microarray technologies for measuring mRNA abundances in cells allow monitoring of gene expression levels for tens of thousands of genes in parallel. By measuring expression responses across hundreds of different conditions or timepoints a relatively detailed gene expression map starts to emerge.

A collection of gene expression level measurements taken under various experimental conditions by microarray or any other technology, define expression profiles of the respective genes. There are many surveys of the technology and analysis in general (The Chipping Forecast 1999; Brazma & Vilo 2000; Celis *et al.* 2000; Hegde *et al.* 2000; Dopazo *et al.* 2001; Quackenbush 2001). The simple query "review microarray analysis" from PubMed revealed 72 articles from Medline in April 2001 and 117 in August 2001.

It seems reasonable to hypothesize that genes with similar expression profiles,

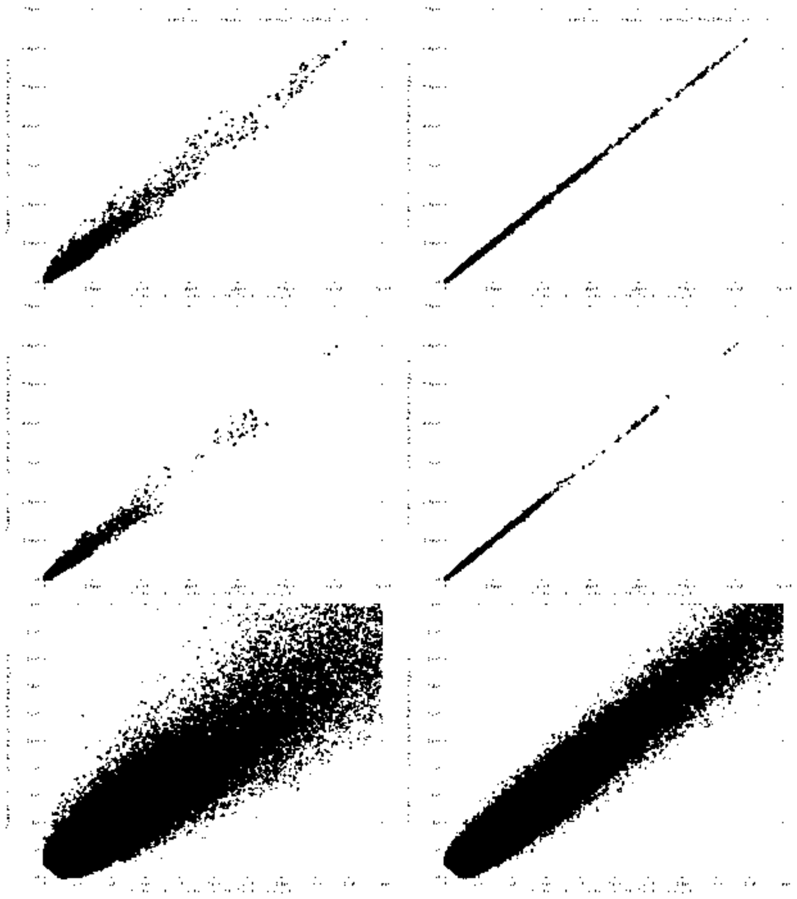


Figure 6.2: The distribution of all patterns (of unrestricted length) with at most one wildcard symbol in the regions $-250..-150$ (upstream from the ORFs) and randomly chosen genomic regions of length 100 bp. Dots in the left column correspond to patterns that occur in x sequences from the random regions (along horizontal axis) and y sequences from the upstream regions (vertical axis). In the right column the upstream regions are replaced by another set of random regions, therefore these plots show the expected statistics if the regions are chosen at random. Top row – all patterns with at least 10 occurrences. Second row – the subset of the patterns in the top row containing at least two characters C or G and not containing any of the substrings AAAA, TTTT, ATAT, or TATA. Bottom row – the same plot as in the second row, but only including patterns with at most 200 occurrences in upstream or random regions (i.e., zoomed to the lower left corner).

i.e., genes that are *coexpressed*, may share something common in their regulatory mechanisms, i.e. may be *coregulated*. Therefore, by clustering together genes with similar expression profiles we find groups of potentially coregulated genes allowing one to search for putative regulatory signals.

The first whole-genome microarray gene expression data set published was a diauxic shift experiment performed on yeast *Saccharomyces cerevisiae*, where expression levels for all genes during a metabolic shift from fermentation to respiration due to glucose starvation were measured in two-hour intervals (DeRisi, Iyer, & Brown 1997). The authors identified several distinct clusters in the gene expression profiles, and were able to show the presence of several previously characterized transcription factor binding sites (for example the stress responsive element CCCCT) located upstream to many of the genes in those clusters. Fascinated by these results, many researchers asked the following quite natural question - ”can we identify novel putative binding sites automatically by combining gene expression data clustering and sequence pattern discovery methods?”

The same data set of diauxic shift was soon analyzed by several other groups (van Helden, André, & Collado-Vides 1998; Brazma *et al.* 1998b). Van Helden *et al.* (van Helden, André, & Collado-Vides 1998), searched for oligonucleotides overrepresented upstream to potentially coregulated genes (clusters from the paper of (DeRisi, Iyer, & Brown 1997)) and showed that potential new transcription factor binding sites can be found in this way.

We used the information about the gene expression profiles to extract smaller sets of genes that potentially share similar regulation mechanisms and maybe also transcription factor binding sites (Brazma *et al.* 1998b). We used the data from the yeast gene expression studies reported in (DeRisi, Iyer, & Brown 1997)[1] and clustered the genes by similarities in their expression profiles in several alternative ways. These clusters were used for the discovery of patterns “characteristic” to the upstream regions of these clusters, i.e., patterns with high rating $R_t(\pi, S_+, S_-)$, where S_+ are the sequences from the cluster, and S_- are the other upstream sequences. Some of the patterns with a high rating $R_t(\pi, S_+, S_-)$ were already known transcription factor binding sites. Some examples of the discovered patterns are `CCCCT` matching 64% (35 out of 55) of sequences in the respective cluster and 21% (1280 out of 5921) of remaining upstream regions (and thus getting a score of 2.95), `C..CCC.T` (score 2.88), `T.C..CCC` (score 2.85), and `T.AGGG` (score 2.27). Moreover, it was shown that many of these binding sites could not be discovered from the global comparison of all upstream sequences against random genomic regions.

Later, more expression studies have been carried out under various condi-

[1] J. F. DeRisi *et al.* studied the relative expression rate changes of all (over 6000) genes of yeast during the diauxic shift from anaerobic (fermentation) to aerobic (respiration) metabolism.

tions (e.g. sporulation (Chu *et al.* 1998) and cell cycle studies (Cho *et al.* 1998; Spellman *et al.* 1998) and the amount of the expression data is increasing rapidly. Simultaneously, pattern discovery methods have been developed and applied. Surveys of these studies have appeared (Zhang 1999; Vilo & Kivinen 2001; Ohler & Niemann 2001; Werner 2001).

As noted by several authors, the task of identifying promoter sequences can be very difficult (Vanet, Marsan, & Sagot 1999; Sinha & Tompa 2000; Vilo *et al.* 2000). The reasons include the uncertainty in promoter region prediction, the noise level in microarray expression measurements, the question about what the appropriate motif description language is, and the algorithmic problems of identifying subtle signals from sets of sequences that do not even need to share the same motifs. Algorithms used for transcription factor binding site prediction may have to detect only marginally overrepresented patterns in sets of hundreds or thousands of sequences of length of thousands of nucleotides (Vilo *et al.* 2000).

The transcription factor binding sites do not usually act alone. It is assumed that genome-wide control is achieved by a combinatorial use of multiple sequence elements (Werner 1999).

The aim of our studies is to mine automatically for new, statistically significant patterns in putative regulatory regions of genes. Such data mining experiments are not a substitute for "conventional single-gene dissections" (Zhang 1999). Their aim is instead to explore simultaneously thousands of genes *in silico* (which cannot be done by conventional methods) to generate targets for conventional studies *in vitro*.

6.1.3 A systematic analysis of yeast *Saccharomyces cerevisiae*

Here we discuss the results for automatic transcription factor binding site prediction, the method published in (Vilo *et al.* 2000). First we present a short outline of the study and then discuss the details.

We clustered systematically all yeast genes based on their expression responses to 80 experimental conditions (Eisen *et al.* 1998) by K-means clustering, evaluating simultaneously the "goodness" of each cluster by average silhouette value. By choosing different K values and varying the initial partitioning, we obtained over 52 thousand different clusters (many of these overlapping). For each of the clusters we retrieved the 600 bp DNA sequences upstream of the respective gene, and exhaustively searched for all the sequence patterns of unrestricted length that are overrepresented in the sequences of the cluster. Patterns were rated for each cluster according to a binomial distribution with expected probability calculated from occurrence frequency in all upstream sequences. Pattern discovery was repeated for randomized clusters to assess the significance threshold for such patterns. Of the over 6000 significant patterns we excluded the ones discovered only from the clusters containing highly homologous upstream sequences. In this way we could list 1498 of the most interesting patterns for further studies. We clustered these patterns into 62 groups. For all of these groups an approximate alignment and consensus pattern were generated. To assess the quality of the patterns we matched all 1498 patterns against the experimentally verified yeast binding sites as given in the SCPD database (Zhu & Zhang 1999). Of the 62 groups 48 had patterns matching some sites in SCPD database.

6.1.3.1 Clustering the gene expression profiles

The result of the expression profile clustering is sensitive to the choice of the distance measure in the expression profile space, as well as on the clustering algorithm itself. Apparently there is no single "right way" of clustering the expression profiles, since various elements in each profile may be influenced by some particular regulation aspects and regulation is usually not based on a simple on/off switching. It has been generally acknowledged that currently we do not know what the most appropriate distance measure or clustering method is.

An alternative to selecting a particular clustering method is to study a number of different clusterings in parallel. To avoid manual intervention or setting the arbitrary thresholds required to get clusters from hierarchical clustering methods we used K-means clustering algorithm. By repeating clustering for many different K values as well as varying the initial cluster center choices, we were able to create many clusters together with the formal "goodness" measure for each.

The "goodness" of a cluster depends on how close its elements are to each other, and how far they are from the next closest cluster. One such measure has been proposed by Rousseeuw (Rousseeuw 1987) based on the notion of a *silhouette plot* and an average silhouette value of a cluster (for a detailed definition see (Rousseeuw 1987)) defined as follows.

For each two objects i and j, we denote by $d(i, j)$ the distance between i and j. For a set A, we denote by $|A|$ the number of elements in A. For each object i we denote by A the cluster to which it belongs and define a value

$$a(i) = \frac{1}{|A| - 1} \sum_{j \in A, j \neq i} d(i, j),$$

the average distance to elements within A. For any cluster C different from A we define

$$d(i, C) = \frac{1}{|C|} \sum_{j \in C} d(i, j),$$

an average distance of i to objects in C, and

$$b(i) = \min_{C \neq A} \{d(i, C)\},$$

the average distance to the members of the closest cluster. The *silhouette value* $s(i)$ of the object i is defined as

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}.$$

The silhouette value $s(i)$ for each object i lies between -1 and 1. If $s(i) = 1$, the object is well classified, if $s(i) < 0$, the object is badly classified, in fact it is on average closer to members of some other cluster. The average silhouette value for a cluster can be used as a measure of the "goodness" of that cluster. The silhouette value characterizes not only the "tightness" of the given cluster, but also how far each element of the cluster is from the next closest cluster.

6.1.3.2 Rating of patterns based on the probability of occurrences

Given a set S of N sequences, a subset $C \subseteq S$ of size n, and a pattern π that occurs in k sequences from C, we can calculate the probability of such an event from the binomial distribution. Note that in this application we count as an occurrence only the fact that pattern matches a sequence, regardless in how many places it matches. Thus the number of occurrences in a set of sequences is the number of sequences from that set that contain at least one match by pattern. We estimate the *background probability* p that pattern π matches an individual sequence of C from the observed total number of sequences K that have an occurrence of that pattern in the set of all N sequences, $p = K/N$. Given k occurrences in the set C, we ask how probable this is given the background distribution p? According to the binomial distribution, the probability of a pattern occurring in exactly k sequences "by chance" is

$$P(k, n, p) = \binom{n}{k} p^k (1 - p)^{n-k}.$$

The probability of a pattern π occurring k or more times in a set of C sequences is

$$P(\pi, C) = \sum_{i=k}^{n} P(i, n, p).$$

The probability of the pattern occurrences as such however does not tell about the significance of the findings. Due to the fact that there are many possible ways to select a subset C and a large number of patterns that match sequences in each subset, there must be patterns which have small probabilities even if sets C are chosen randomly.

To tell which probabilities p_s are "significant", we can apply randomization techniques in order to determine how often we can observe patterns with score p_s when the set C is chosen randomly.

6.1.3.3 Grouping patterns by similarity

Having defined the similarity measure between patterns as in Section 5.1.1.2, we used an average linkage hierarchical clustering algorithm to group them. For generating the alignments of patterns within each cluster we used the pattern discovery algorithm SPEXS to find the consensus pattern common to a high percentage of the patterns in each group. This pattern was used as an anchor for guiding the alignment of the group (see Table 6.3).

6.1.3.4 A computational experiment

We performed an experiment analyzing yeast expression and sequence data. We used the public data set combining various yeast expression experiments from Stanford University (Eisen *et al.* 1998). The data set consists of gene expression levels for 6221 yeast genes with a total of 80 experimental conditions. These 80 measurements are related to time course analysis of yeast cell cultures during the cell cycle, sporulation, and diauxic shift experiments. The data has been downloaded from P. Brown's laboratory (`http://rana.stanford.edu/`). We implemented the following computational experiment. Note that the steps described below have been performed in a highly automated way and formal selection criteria were applied in each step.

1. **Clustering the expression data.** We clustered 6221 genes based on their expression profiles by the K-means clustering algorithm using Euclidean distance in 80-dimensional space. We varied the value of K (the number of clusters) between 2 and 1000 and repeated the clustering for each selected K ten times with different random sets of initial cluster centers. In total we did over 900 separate clusterings. For each cluster we computed the average silhouette (Section 6.1.3.1) value. We selected the clusters of size between 20 to 100 genes and obtained in this way over 52,100 different clusters.

2. **Sequence pattern discovery.** For each cluster we took the set of gene upstream sequences of length 600 bp and enumerated all patterns occurring in at least 10 of these. We scored all patterns according to the probability of their occurrences in the cluster using a binomial distribution and background probability estimation as described in Section 6.1.3.2.

3. **Finding the significance threshold by control experiment.** To determine the statistical significance threshold for the patterns, we repeated step 2 on randomized data by replacing the cluster contents by upstream sequences from random sets of genes. We plotted the average silhouette value and the score of the best pattern of each real cluster on a two-dimensional plot

in Figure 6.3 (top left), and for the randomized data similarly Figure 6.3 (top right). The threshold `10e-8` was chosen and all patterns less probable (from step 2) were reported.

4. **Pattern selection.** There were in total over 6000 significant patterns (see Table 6.1 for the 30 most significant patterns). The distribution of the number of patterns discovered in each cluster suggested that the clusters can be divided into two groups: ones producing more than 600 patterns, and others producing less than 600 patterns. The reason for cutoff at 600 was set based on the "jump" in the number of good patterns found in one cluster. Up to 218 patterns this number was almost continuous, then the next values were 336 and 746 "good" patterns from one cluster. The 508 clusters producing more than 600 patterns each contained only 169 different ORFs (see Figure 6.3 bottom).

   A study by ClustalW (Thompson, Higgins, & Gibson 1994) showed that the upstream sequences of these 169 ORFs were highly homologous, thus distorting the pattern statistics. The homology of sequences in the clusters that produce a smaller number of patterns is low, therefore the significant patterns in these clusters (containing together 3727 ORFs) are candidates for regulatory signals. There were 1498 such patterns, which is still too many for human study one by one.

5. **Grouping the patterns.** We clustered these 1498 patterns by an average linkage hierarchical clustering algorithm using a similarity measure based on common information content (Section 5.1.1.2). This produced 62 clusters of similar patterns.

6. **Aligning and summarizing each pattern group.** For each cluster we generated an approximate alignment and a consensus pattern (see Table 6.2 and 6.5). For generating the alignments of patterns within each cluster we used the pattern discovery algorithm SPEXS to find the consensus pattern common to a high percentage of the patterns in each group. This pattern was used as an anchor for guiding the alignment of the group (see Table 6.3).

7. **Comparing the discovered patterns to known transcription factor binding sites.** We matched all 1498 interesting patterns against experimentally verified DNA binding sites of yeast as given in SCPD (Zhu & Zhang 1999).

   We say that a pattern matches a site if the pattern is a substring of a mapped site. The opposite, matching of sites against patterns, is also possible, but some of the sites in SCPD are rather short and can have matches by chance (in fact there are many sites consisting of a single nucleotide, and these should be excluded before such matching). We say that a cluster of patterns matches the site, if at least one of the patterns in the cluster matches it. We found that 48 of the 62 pattern clusters had matches in SCPD. The results are summarized in Table 6.2.

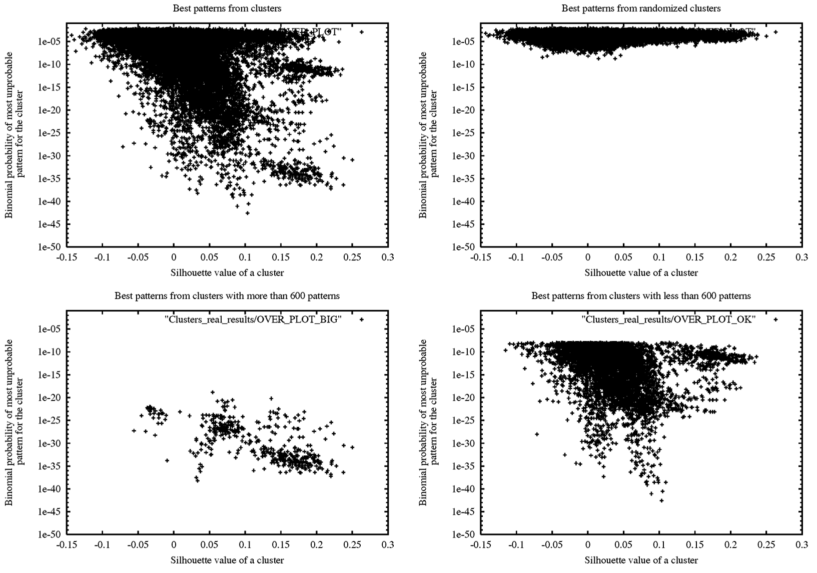


Figure 6.3: Top left: the silhouette values of the 52,000 clusters (horizontal axis) plotted against the probabilities (vertical axis) of only the best substring pattern discovered for each of the respective clusters. Top right: the same for randomized clusters, i.e., for sets of upstream sequences of the same number of randomly chosen genes. Consider the plots in top right and left: if we consider only the patterns that are less probable than the ones discovered in the randomized sets, the plot in the top diagram is stretched along the diagonal, suggesting a correlation between the pattern and the cluster scores. This has to be viewed with caution however, as the clusters are not mutually independent. Bottom left: the subplot with patterns obtained from clusters that produce more than 600 significant patterns each (these are from clusters of highly homologous sequences). Bottom right: the interesting patterns above the significance threshold and not due to high sequence homology between upstream regions. These patterns are the best candidates for regulatory signals and are the subject of further studies.

We studied more closely some of the clusters and the respective discovered patterns. For instance, the expression profile for the cluster that produced the 15^{th} highest scoring pattern GGTGGCAA is shown in Table 6.4 (we refer to this cluster as GGTGGCAA-cluster). This pattern occurs in 20 out of 40 upstream sequences of the cluster, and only in 96 out of all 6221 upstreams. Note that most of the genes in the cluster are related to the functioning of the proteasome. It was not known to us at the time of publishing, but just under a year earlier it had been identified as a proteasome regulatory element (Mannhaupt *et al.* 1999).

Pattern	Probability	Cluster size	Occurrences in cluster	Total occurrences	K
AAAATTTT	2.59075e-43	96	72	830	60
ACGCG	6.41023e-39	96	75	1088	50
ACGCGT	5.23109e-38	94	52	387	40
CCTCGACTAA	5.42764e-38	27	18	23	220
GACGCG	7.88674e-31	86	40	284	38
TTTCGAAACTTACAAAAAT	2.08201e-29	26	14	18	450
TTCTTGTCAAAAAGC	2.08201e-29	26	14	18	325
ACATACTATTGTTAAT	3.80588e-28	22	13	18	280
GATGAGATG	5.59927e-28	68	24	83	84
TGTTTATATTGATGGA	1.8998e-27	24	13	18	220
GATGGATTTCTTGTCAAAA	5.04076e-27	18	12	18	500
TATAAATAGAGC	1.51458e-26	27	13	18	300
GATTTCTTGTCAAA	3.40261e-26	20	12	18	700
GATGGATTTCTTG	3.40261e-26	20	12	18	875
GGTGGCAA	4.17788e-26	40	20	96	180
TTCTTGTCAAAAAGCA	5.09734e-26	29	13	18	250
CGAAACTTACAAA	5.09734e-26	29	13	18	290
GAAACTTACAAAAATAAA	7.9186e-26	21	12	18	650
TTTGTTTATATTG	1.73752e-25	22	12	18	600
ATCAACATACTATTGT	3.62348e-25	23	12	18	375
ATCAACATACTATTGTTA	3.62348e-25	23	12	18	625
GAACGCGCG	4.47204e-25	20	11	13	260
GTTAATTTCGAAAC	7.22797e-25	24	12	18	400
GGTGGCAAAA	3.37381e-24	33	14	31	475
ATCTTTTGTTTATATTGA	7.18849e-24	19	11	18	675
TTTGTTTATATTGATGGA	7.18849e-24	19	11	18	475
GTGGCAAA	1.13567e-23	28	18	137	725
CGAACTGCCAT	1.74392e-23	20	10	10	92
CGAACTGCCATCTC	1.74392e-23	20	10	10	190
CCTCGAACTGCCATCT	1.74392e-23	20	10	10	170

Table 6.1: The 30 highest scoring patterns discovered in the genome regions upstream from genes of the clusters. Note that the smallest probability of a pattern discovered in the randomized data is 1.74434e-09. The last column shows the number of clusters in the respective clustering by K-means.

Tables 6.2 and 6.5 show consensus patterns (used here for naming of the pattern clusters only) that have been calculated from pattern alignments. The nucleotide groups have been introduced when the frequency of the less frequent nucleotide in respective column is over 25% of the frequency of the more frequent nucleotide. Inside the groups nucleotides are ordered based on their frequency. The lower case letters are used when the majority of the patterns does not have any nucleotide in that position i.e., when the most frequent nucleotide in the respective column is a dash.

Nr	Consensus pattern	Factors that have matching binding sites
1	tctcaTCTCA[TC][CT][tag]catc	ABF1 ABF1,BAF1 UASPHR
3	cctcGAA[CG]TGCCATCtca	BAS1 BAS1,PHO2 CCBF,SCB,SWI6 HSE,HSTF HSE,HTSF SCB UASH UASPHR XBP1
4	a[ta][CG]CCTA[AT]Aat	MCM1
6	acc[ac]CCCC[CT][CGT][ag]a	MIG1 RAP1 RAP1,EBF1
7	gT[TA][CA]TCCT[CG]g	BAS1 BAS1,PHO2 UASPHR
9	a[ct][at]GTGACA[GTC][cta]t	ADR1 MATalpha1 MATalpha2 MCM1 UASH
10	tt[tc]ACAGT[GT][AT][tc]g	ABF1 ABF1,BAF1 ADR1 BAS1 BAS1,PHO2 GAL4 GCN4 GCN4,GCRE GCRE,GCN4 PHO2 RAP1 RAP1,EBF1
11	[at][ATC]TACACAt	MATalpha2
12	tttGTCACA[GAT]gg	ABF1 ABF1,BAF1 PAE UASH
13	t[gc]ACATT[GC][CT]tg	HSE,HSTF HSE,HTSF PAE RAP1 RAP1,EBF1
14	ata[TC]TGGTTCt	ROX1 URSSGA
15	acaTCCGTAC[acg]tt	HSE,HSTF HSE,HTSF RAP1 RAP1,EBF1
16	a[gca][atc]TAAG[CG][TAG][tga]a	ABF1 ABF1,BAF1 GATA GLN3 MCM1 URS1ERG11
18	t[ct][at][AG]AAGT[AT][TA]c	PRP1 URSPHR
19	gtT[AG]TTA[CT][TG][AG]ca	GRF2 MATalpha2 MCM1 REB1 UASH
22	t[ACT]CGCTTA[AT]	UASGATA
23	gaa[ca][gat][acg][AG]CGCG[cta][gat][ca]gc	ABF1 ABF1,BAF1 CCBF,SCB,SWI6 DAL82 GAL4 HAP1 HAP2 HAP2;HAP3;HAP4 HAP3 HAP4 LEU3 MAL63 MCB PDR1 PDR3 PHO4 RAP1 RAP1,EBF1 REB1 SCB SWI4 SWI6 UASGABA repressor_of_CAR1
24	[ac][at][GT]ACGCcaa	ABF1 ABF1,BAF1
25	GGTCG[CT]Ac	UASPHR URS1ERG11
27	tgtTAACGAATCGTTtaa	GFI,TAF MCM1 TAF
28	ga[at][TC]CGTTTA[ag]g	ABF1 ABF1,BAF1 MAL63 MCM1
30	aA[CAG][AT]GAATCttc	ADR1
31	t[ac][tc][at]CGACT[CA][ca][cg]aa	BAS1 BAS1,PHO2 GAL4 GCN4 GCN4,GCRE GCRE,GCN4 GFI,TAF PHO2 TAF URSSGA
32	tcCACGAA[gc][ta]g	ABF1 ABF1,BAF1 BAS1 BAS1,PHO2 CCBF,SCB,SWI6 GA-BF GFI,TAF HSE,HSTF HSE,HTSF PDR1 PDR3 PHO4 SCB SWI4 SWI6 TAF URS1ERG11
33	c[ga][ctg][ACG]TACG[AT][atc]tat	ABF1 ABF1,BAF1 PHO4 URS1HO
34	aC[CA]CATAC[AT]t	MCM1 RAP1 RAP1,EBF1
35	atat[CT][AG]GCAC[TC][ac]a	GAL4 MCM1 PHO4 RAP1 RAP1,EBF1 URSSGA
36	taGCGCA[GT][ga]cc	ABF1 ABF1,BAF1 ARC CUP2 SWI5 UASPHR repressor_of_CAR1
37	cgGTGGCAA[AC][ag]	ABF1 ABF1,BAF1 HAP2 HAP2;HAP3;HAP4 HAP3 HAP4 RAP1 RAP1,EBF1 UASCAR UASPHR repressor_of_CAR1
38	t[ca][ga][GA]CGGC[TG][GTA][cta]tttt	ABF1 ABF1,BAF1 GAL4 HAP1 LEU3 MCM1 PHO4 QBP RP-A SWI5 UASGABA URS1H URSF URSINO repressor_of_CAR1
39	a[cat][AGC]AGGG[GT][ctg][ac]a	13nt_repeat BUF GAL4 HAP1 IRE MCM1 MIG1 PHO4 RAP1 RAP1,EBF1 RC2;RC1 UAS1ERG11 UAST52,ORE URS1ERG11 URSSGA
40	gcg[ag][at][ga][ac]GATGAG[AC]t[ag][at]g	BUF HAP1 HSE,HSTF HSE,HTSF PQBOX REB1 SWI5 UASH UASPHR
41	aTGGATGCc	MOT3
44	gc[TAG]TATAT[ATC][gat][ag][tg]gg	TATA,TBP
47	gtaTAAATAGAGCtgct	QBP TATA,TBP UIS URS1H URS1HSC82
48	[at]a[ag][TG][AT]GCC[CG][ac][ac]ga	BUF GAL4 GCFAR QBP UME6 URS1H URS1HSC82 repressor_of_CAR1
49	aC[CT]CAAT[AT][tg]t	MATalpha1 MCM1
51	aaacaAAACAAA[AT][ca][ac]aata	GCR1 GCR1,CTBOX MCM1 MSE ROX1 UASPHR
52	tgtGTAAA[TC]ATtt	SFF UAS2CHA URS1ERG11
53	ataaaa[gt][CA][GT]AAAA[GA][cg][gac]aaaag	BAS1,PHO2 CCBF,SCB,SWI6 MAL63 MCM1 MIG1 PHO2 SCB SWI4 SWI5 SWI6 TATA,TBP UASPHR
54	t[gt][TC]GAAAG[AG]Tt	XBP1
55	[at][tac]t[gta][ag]AAAATTTT[tg][tc][at]tt	ABF1 ABF1,BAF1 CSRE DAL82 MAL63 MATalpha2 MCM1 NBF UASH UASINO UIS
56	ga[at][acg][CA]GGAA[AG]T[gt]gaa	GAL4 MCM1 UAS2CHA UASH
57	t[tc][cat][AT][TC]TTC[GA][ACT][ga]t	GAL4 GCR1 GCR1,CTBOX REB1
58	cgg[ct][ctg][gct][ctg]CTTTTT[CTG][TC][atc][tg]cc	ACE1 CUP2 DAL82 GAL4 HSE,HSTF HSE,HTSF LEU3 RAP1 RAP1,EBF1 UASCAR URSSGA
60	t[ta][gta][gtc][TG]TCTA[TG][GTC]a[at][ct]	HSE,HSTF HSE,HTSF ROX1
61	taaat[AT]TTTGTG[ta]ca	MATalpha1 MATalpha2 MCM1 MIG1 UASH
62	t[acg]CTGTG[CT]a[ac]	UASH

Table 6.2: Results of the automatic matching of the discovered patterns against SCPD.

We also studied how our discovered patterns compare to experimentally proven binding sites in yeast by comparing them to SCPD database. For instance, two of the pattern clusters (cluster nr. 15 and 34 in the numeration produced by the clustering algorithm) have matches in the “RAP1,EBF1” binding

```
-----------------ACCCAGACATCGGGCTTCCAC-
---------------ACACCCAGACATC-----------
---------------ACACCCAGACATC-----------
---------------GAACCCATACACT-----------
---------------ACACCCAGACCGCG----------
---------------GCACCCACACATTT----------
------------GCTAAACCCATGCACAGTGACT-----
-----------------ACCCAGACACGCTCGA------
-------------CTTCACCCTCATAC------------
---------------ACACCCCTTTTCT-----------
---------------GCACCCAGTCTT------------
---------------GCACCCAAACACCTGCATATTTGG
---------------GCACCCAATCACC-----------
---------------ACACCCAGACCTC-----------
---------------AAACCCACACAT------------
--------------TGCACCCATACCTT-----------
--------------AACACCCAAGCACAG----------
-ATCTCTCGCAACG-------------------------
ACCTCCGTACATTC-------------------------
ACACCTGGACACC--------------------------
ACATCCGTACAACGAGAACCCATACATTA----------

---TCCGTAC---    ACCCATAC--
-CATCCGTAC---    ACCCATACA-
--ATCCGTA----    ACCCATACAT
--ATCCGTACA--    -CCCATAC--
----CCGTAC---    -CCCATACA-
----CCGTACA--    --CCATACAT
---TCCGTACAT-    -CCCATACAT
--ATCCGTACAT-    --CCATACA-
---TCCGTACA--    -AACATAC--
----CCGTACAT-    --ACATACT-
---TCCGTA----    ---GATACT-
--ATCCGTAC---    --AGATACT-
----CCGTACC--
---ACCGTACC--
---ACCGTAC---
--CACCGTAC---
----CCGTACATT
----GCGTAG---
----GCGTAGG--
-----CGTAGG--
-CATCCGTA----
ACATCCGT-----
-CATCCGT-----
```

Table 6.3: The upper part of the table shows the alignment of experimentally proved RAP1,EBF1 binding site taken from SCPD database. We excluded the sites ATGCCCGTGCAC and GTCACTAACGACGTGCACCA, which did not give a good alignment. The alignments below are produced automatically by our pattern grouping algorithm. Left is from cluster 15 and right is from cluster 34. Patterns GTACATT, AACATCCG, TACATCC, ACATCC, ACATCCG and ACCCA, ACCCAT, ACCCATA were left out from these clusters respectively as the alignment was done by simple heuristics based on one conserved block.

sites. The first consists of 29 patterns, 20 of which match "RAP1,EBF1" sites. The second one consists of 15 patterns and has 11 matches. Both alignments match different parts of the "RAP1,EBF1" site as illustrated in Table 6.3. The site names have been automatically downloaded and analyzed from the SCPD database http://cgsigma.cshl.org/jian/.

Potentially the most interesting patterns however are the ones that do not have matches in the known binding sites, and they can be targets for further research (see Table 6.5).

ORF	Cytoplasmic degradation	Gene	Length	Disruption	Description from MIPS
YBL041W	⋆	PRE7	241	lethal	20S proteasome subunit(beta6)
YBR170C		NPL4	580	lethal	nuclear protein localization factor and ER translocation component
YDL126C	⋆	CDC48	835	lethal	microsomal protein of CDC48/PAS1/SEC18 family of ATPases
YDL100C			354		similarity to E.coli arsenical pump-driving ATPase
YDL097C	⋆	RPN6	434	lethal	subunit of the regulatory particle of the proteasome
YDR313C		PIB1	286		phosphatidylinositol(3)-phosphate binding protein
YDR330W			500		similarity to hypothetical S. pombe protein
YDR394W	⋆	RPT3	428	lethal	26S proteasome regulatory subunit
YDR427W	⋆	RPN9	393	viable	subunit of the regulatory particle of the proteasome
YDR510W		SMT3	101	lethal	ubiquitin-like protein
YER012W	⋆	PRE1	198	lethal	20S proteasome subunit C11(beta4)
YFR004W	⋆	RPN11	306	lethal	26S proteasome regulatory subunit
YFR033C		QCR6	147	viable	ubiquinol–cytochrome-c reductase 17K protein
YFR050C	⋆	PRE4	266	lethal	20S proteasome subunit(beta7)
YFR052W	⋆	RPN12	274	lethal	26S proteasome regulatory subunit
YGL048C	⋆	RPT6	405	lethal	26S proteasome regulatory subunit
YGL036W		MTC2	909	viable	Mtf1 Two hybrid Clone 2
YGL011C	⋆	SCL1	252	lethal	20S proteasome subunit YC7ALPHA/Y8 (alpha1)
YGR048W	⋆	UFD1	361	lethal	ubiquitin fusion degradation protein
YGR135W	⋆	PRE9	258	viable	20S proteasome subunit Y13 (alpha3)
YGR253C	⋆	PUP2	260	lethal	20S proteasome subunit(alpha5)
YIL075C	⋆	RPN2	945	lethal	26S proteasome regulatory subunit
YJL102W		MEF2	819		translation elongation factor, mitochondrial
YJL053W		PEP8	379	viable	vacuolar protein sorting/targeting protein
YJL036W			423		weak similarity to Mvp1p
YJL001W	⋆	PRE3	215	lethal	20S proteasome subunit (beta1)
YJR117W		STE24	453	viable	zinc metallo-protease
YKL145W	⋆	RPT1	467	lethal	26S proteasome regulatory subunit
YKL117W		SBA1	216	viable	Hsp90 (Ninety) Associated Co-chaperone
YLR387C			432		similarity to YBR267w
YMR314W	⋆	PRE5	234	lethal	20S proteasome subunit(alpha6)
YOL038W	⋆	PRE6	254		20S proteasome subunit (alpha4)
YOR117W	⋆	RPT5	434	lethal	26S proteasome regulatory subunit
YOR157C	⋆	PUP1	261	lethal	20S proteasome subunit (beta2)
YOR176W		HEM15	393	viable	ferrochelatase precursor
YOR259C	⋆	RPT4	437	lethal	26S proteasome regulatory subunit
YOR317W		FAA1	700	viable	long-chain-fatty-acid–CoA ligase
YOR362C	⋆	PRE10	288	lethal	20S proteasome subunit C1 (alpha7)
YPR103W	⋆	PRE2	287	lethal	20S proteasome subunit (beta5)
YPR108W	⋆	RPN7	429		subunit of the regulatory particle of the proteasome

Table 6.4: The 40 genes from the `GGTGGCAA`-cluster. The annotations are taken from the MIPS database. Note that many of the genes are related to proteasome. The 25 ORFs marked with ⋆ belong to the functional class of "cytoplasmic degradation" containing 93 ORFs in total according to the Functional Catalogue of *Saccharomyces cerevisiae* (MIPS, `http://www.mips.biochem.mpg.de/proj/yeast/catalogues/funcat/`.)

6.1.4 Discussion on gene regulatory motif discovery

Our observation that the pattern and cluster scores correlate is consistent with the observation of Tavazoie *et al.* (Tavazoie *et al.* 1999). We have performed a more systematic experiment for precisely defined pattern and cluster "goodness" measures for considerably more clusters and reported the numeric evidence. Although the observation is not surprising, it suggests that the fast and simple K-means clustering algorithm can be used in the large scale analysis of all genes of an organism. It enables the finding of coexpressed genes based on the expression profiles allowing the consecutive search for coregulated genes.

Cluster Nr.	Consensus pattern
2.	aaTCTTCATGt
5.	cgTACCTCTa
8.	gACAGCTAc
17.	tAT[TAC]GTTAAgc
20.	ACTTTATTT
21.	[ag]TAACTT[AT]Ca
26.	TATCGAG *(singleton)*
29.	t[ta]CGAATA[AG]aaaa
42.	[ta]TGCATGAAc
43.	a[TG][GC]GTATAc
45.	g[ag][ga][ag][AG][TAG]AT[GA]TG[agt][ga][ag]
46.	tag[AG]TAGA[TA]A[ga]aaaa
50.	ATCCAAGAg
59.	tTTTTCTG[CT][TA]c

Table 6.5: Consensi of the pattern clusters that do not have matches in SCPD database. See text for explanations and Table 6.2 for other consensi.

Promoter analysis using gene expression experiments is a difficult problem due to limited knowledge about gene regulation in eukaryotic organisms and the many steps involved in the analysis while no step is straightforward and error-free.

First, the expression data itself is hard to analyze due to the amount of data combined with the inherent fuzziness and low accuracy of the measurements. It is unlikely that a single best method for expression data clustering exists, and for each clustering method the strict cluster boundary detection remains a challenge. Thus for a systematic analysis of gene expression data one should be able to combine different expression profile clustering methods and perform pattern discovery for large numbers of clusters. Although in a couple of studies it has been shown that the clustering of the gene expression data is not necessary for pattern discovery, simple sorting may be sufficient together with an application of some test, like for example the Kolmogorov Smirnov rank test for detecting the "cluster" boundaries (Jensen & Knudsen 2000).

Second, the challenge to pattern discovery approaches is the right choice of pattern representation languages combined with the statistical and computational problems of detecting subtle signals from many clusters of sequences.

Third, the analysis and interpretation of all discovered significant patterns can still be a big problem, as there can be too many of them for human inspection. This is a typical second order data mining problem, where the vast amount of simple patterns or rules needs to be summarized further. Using a restrictive pattern class for which there exist fast pattern discovery methods allows the analysis of large sets of sequences. For these simple pattern classes, however, good postprocessing is needed to obtain patterns of a more general class. Large numbers of simple patterns can be clustered based on either their mutual similarity or on the locations of their occurrences. From these groups it is possible to build more general sequence profiles and consensi to produce "executive summaries" of the discovered patterns. These pattern groups as well as areas of high density of occurrences of improbable patterns can be analyzed further by computationally more demanding methods, for example by a multiple local alignment program such as MEME (Bailey & Elkan 1995) or Consensus (Hertz & Stormo 1994).

Individual regulatory sites in isolation do not provide enough information about gene regulation, as most of the motifs representing these sites occur almost randomly over all chromosomes (Werner 1999). Approaches where combinations of individual patterns are analyzed (Brazma *et al.* 1997; Wagner 1999) or methods that allow to discover combinations of sites during the pattern discovery phase, may become very useful in solving the need for higher-level organization of individual sites. The data mining techniques to discover frequent combinations (Mannila, Toivonen, & Verkamo 1994), association rules (Mannila, Toivonen, & Verkamo 1994) and episode rules (adds order to association rules) (Mannila, Toivonen, & Verkamo 1997) could be directly applied to binding site combination studies (Brazma *et al.* 1997).

Combination of binding sites can be evaluated for example based on the following parameters (Brazma *et al.* 1997):

1. **Coverage:** The number of its occurrences in upstream regions

2. **Goodness:** The ratio of the number of its occurrences in the upstream regions vs the number of occurrences in random regions (of the same length and number)

3. **Unexpectedness:** The ratio of its occurrences vs the expected number of occurrences based on the individual sites.

One has to be careful in using these criteria though. Most importantly because we lack good models for representing binding sites, thus they may not be independent occurrences. Also, the precision used for describing "known" binding sites can vary and hence the frequencies of their occurrences on the full genome can be very different.

Expression analysis with DNA microarrays is unable to distinguish direct regulatory effects from indirect effects and thus our ability to identify genes that are controlled by specific regulatory factors is limited. Genome-wide location analysis that combines chromatin immunoprecipitation procedure with DNA microarray analysis, provides information on the binding sites at which proteins reside through the genome under various conditions *in vivo*. This novel microarray analysis method will allow to distinguish binding sites that are active in vivo from others, thus enabling more direct dissection of regulatory networks (Ren *et al.* 2000; Iyer *et al.* 2001).

Identifying regulatory sequences in the human genome presents new challenges as compared to yeast. First, the sheer size and complexity of the human genome makes predictions more difficult. Gene regulatory elements in humans are frequently found much farther away than with yeast, and can easily be hidden in the bulk of non-coding sequences. There are some strategies which could be used (Zhang 1998), and intensive research continues on the subject. A review on identification of mammalian regulatory sequences was published recently (Rubin & Rubin 2001). Annotation systems for large-scale annotation of human promoters have also started to emerge (Scherf *et al.* 2001).

One way to succesfully identify regulatory elements in the human genome would be to compare human gene upstream sequences with upstream sequences from other species, e.g. other mammals or birds. So far, most studies have compared partly sequenced human and mouse genomes, but it has become clear that no single species can be completely informative, mainly due to different mutation rates of genes and genomes. Thus, it has been suggested that several organisms should be used to aid prediction of regulatory elements in the human genome.

Many researchers have been successful in discovering known as well as novel putative binding sites. The maturation of the field will be proven only by carrying out systematic follow up studies in wetlabs. To assist this goal, promoter databases should be developed further so that verifications of in silico predictions will become easier. The information stored in these databases will help us build up the knowledge base of known and hypothesised gene regulatory network models, and they can be invaluable in assisting biologists in experiment design.

6.2 Prediction of the GPCR and G-protein coupling specificity

6.2.1 G-protein coupled receptors

G protein coupled receptors (GPCR) are the biggest single class of receptors in biology, playing key roles in a remarkably wide range of physiological and pathophysiological conditions. The actions of a large and structurally diverse range of hormones, neurotransmitters, tastants, odourants, photons, and peptidases, are initiated by their binding to GPCRs located on the cell surface (Bockaert & Pin 1999). Such binding activates the receptor, causing helical rearrangements within the receptor, which (by way of unmasking binding sites) transmits the activation signal to a guanine nucleotide-binding protein (G protein) located on the cytoplasmic surface of the membrane, closely apposed to the receptor (Schoneberg, Schultz, & Gudermann 1999; Gether 2000).

Activation of the heterotrimeric G protein (consisting of α, β, and γ subunits) promotes exchange of the guanosine diphosphate (GDP), bound to the α subunit, for guanosine triphosphate (GTP). This allows the dissociation of the α subunit (with GTP bound) from both the receptor and $\beta\gamma$ complex. The separate moieties can then modulate several cell signalling pathways, and the activities of certain ion channels. Termination of the response occurs as a result of the intrinsic catalytic activity of the α subunit, which hydrolyses the bound GTP to GDP. Subsequently the α-GDP then re-associates with the $\beta\gamma$ complex to form the inactive heterotrimer. Amongst the biochemical responses that have been observed following receptor activation (LeVine 1999) are both stimulation and inhibition of adenylate cyclase activity. The G_s class, and the $G_{i/o}$ class of the G proteins, respectively, mediate these opposing effects. The G_{q11} family activate phospholipase C enzymes, resulting in phosphatidylinositol hydrolysis. Together these three families constitute the major functional classes of G proteins, and studies have revealed this specificity is determined by the particular subtype of the α subunit, making up the G protein (Simon, Strathmann, & Gautam 1991; Bourne 1997).

Characteristically each GPCR subtype appears to couple only to a subset of the G proteins that may be found in a particular cell. Elucidation of the mechanism(s) underlying this coupling specificity has been a central theme in GPCR research over the last 15 years. Together the large number of studies have revealed that the selectivity of G protein recognition (and hence coupling) is determined by multiple intracellular receptor regions. The most important regions appear to be the second intracellular loop, and the start and end of the third intracellular loop, which are close to the cytoplasmic surface of the membrane (Wess 1999).

6.2.2 Predicting the coupling specificity

We selected receptors that had apparently non-promiscuous coupling properties, i.e. typically couple to only a single type of G-proteins, and grouped them into the three functional classes of $G_{i/o}$, G_s and G_{q11} . These classes are known to inhibit or stimulate adenylate cyclase, and stimulate phosphatidylinositol hydrolysis, respectively. The membrane topologies of the receptors were computationally predicted, allowing the extraction of just the putative intracellular loops and C-termini from the receptor sequences.

Our strategy to predict the coupling specificity of GPCRs for their G proteins was to attempt to find patterns of amino acid residues in their sequences that appeared to be specific to a particular class of the G proteins. In order to do this

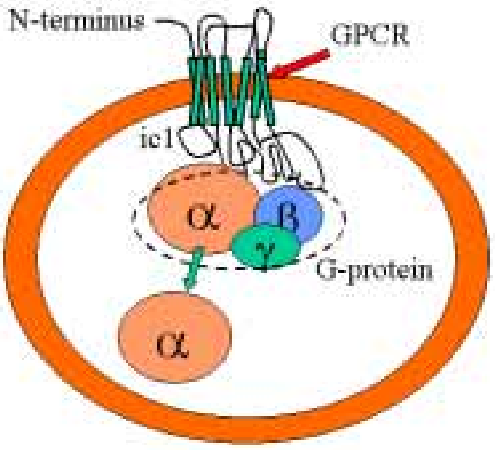


Figure 6.4: The schematic mechanism of GPCR G-protein coupling. The signal is transferred from extracellular domain to intracellular domain. The α-subunit of the G-protein dissociates and forwards the signal to other processes.

we retrieved 103 diverse receptor sequences from SWISS-PROT and TrEMBL for which an apparently non-promiscuous coupling had been determined and was summarized in the TIPS Nomenclature Supplement (TiPS 2000). These were grouped into the three functional classes $G_{i/o}$, G_s , and G_{q11} .

To constrain the search for patterns to the putative intracellular domains of the sequences, we required an accurate membrane topology prediction. In a previous study it has been shown that TMHMM is the best currently available topology prediction method for determining the membrane spanning regions (MSRs) of GPCR sequences (Möller, Croning, & Apweiler 2001). Here, we used a modified version of this program, called 7TMHMM, which was designed specifically to predict the MSRs of GPCRs. The model employed assumes exactly 7 MSRs, with extracellular and intracellular, N- and C-termini, respectively (Möller, Croning, & *et al.* 2001).

We used SPEXS for discovering patterns specific to each of the input data sets. For group characters we used amino acid grouping by properties as described in (Barton & Livingstone 1993). We used groups Γ =

$\{\Sigma, \{\mathtt{RK}\}, \{\mathtt{ED}\}, \{\mathtt{AGS}\}, \{\mathtt{ILV}\}, \{\mathtt{FWY}\}, \{\mathtt{CM}\}\}$, where Σ consists of all amino acids and the respective group, denoted by a single dot . in the pattern, represents a single position wildcard. For restricted length wildcards we used SPEXS repeatedly with different settings and then combined the answers. This produced a large number of patterns ($>$4000) which we then evaluated for their usefulness. The most discriminative patterns are those that occur in large number of sequences of one receptor-G protein coupling group and infrequently in the others. The specificity of all the patterns occurring in each group of receptor sequences were determined using the binomial probability as described in Section 6.1.3.2. Background probabilities were identified by combining sequences from two other groups respectively.

A visual inspection of the patterns revealed the significance of the patterns. The tool PATMATCH (see Chapter 7) allowed us to visualise pattern matches upon the sequences, having grouped the latter by their G protein coupling specificity (see Figure 6.5). Most of the patterns were seen to match in just one of the three groups of receptor sequences, with few matches to the other two groups, demonstrating their specificity. Additionally all of the GPCR sequences were matched by at least a few patterns. PATMATCH also allowed us to determine where the patterns matched onto the intracellular domains of the receptor sequences, and from this to deduce whether match positions are conserved both within a particular receptor-G protein coupling group, and between the three groups. Moreover, by visualizing patterns individually, we could establish immediately that some of the patterns have very strict preferentiality to occur only in certain intracellular loops, even though that information was not used during the pattern discovery phase. This is one more implication that the patterns discovered have real biological significance and are not just a statistical coincidence.

We hypothesised that we might improve the classification of the receptor-G protein coupling groups (and thus subsequent prediction of the coupling for a novel sequence) if we considered the specificity of combinations of patterns, rather than just single patterns. This is similar to the concept of collections of motifs (called fingerprints) that are found in the secondary protein database PRINTS (Attwood *et al.* 2000), or the analysis of the regulatory regions in DNA (Scherf, Klingenhoff, & Werner 2000). Derived pairs and triplets of patterns that are specific for the binding to G proteins are used in conjunction to act as a classifier.

If all the patterns making up a particular combination were found in a sequence, the combination was said to match as a whole. For each sequence presented to the classifier we report the total number of combinations found, and if 30% or more of the matches happened to belong to a specific receptor-G protein coupling group, then this coupling was assumed to be a putative prediction. This potentially allows one to predict promiscuous receptor-G protein coupling.

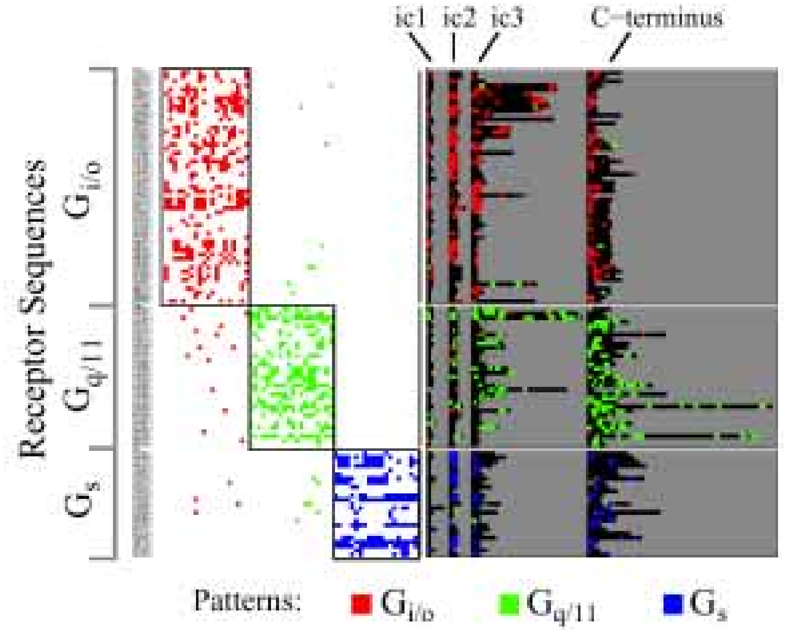


Figure 6.5: Visualisation of the intracellular positions of pattern matches. On the right side of figure 1 the sequences of all intracellular loop regions are displayed as black horizontal bars, proportional their sequence length, ordered in their coupling specificity. The left side shows all patterns that match a specific sequence, identified by their position, again ordered and coloured according to their coupling specificity. Ic1, ic2, ic3 and C-terminus stand for the respective intracellular domains. The formation of blocks on the left side is evidence for the quality of the patterns.

The classifier was applied to the sequences of 10 receptor subtypes not present in the training set, as shown in Table 6.6. Pairwise alignments revealed that these test sequences were in general 30-40% identical to their most similar paralogue in the training set. All 10 predictions appeared to be correct when we consulted the primary literature. We were surprised that the predictions for both P41180 and P25105 were correct given that they resulted from a rather low number of matches. Further we have applied the prediction to other GPCR sequences and observed the predictions to fit the biochemical responses observed upon their activation (unpublished results).

The tables 6.7, 6.8, and 6.9 summarize the most distinguishing motifs for three classes of G-protein coupling mechanisms.

Accession	Class	Hits (class/total)	Protein description
P49190	G_s	123 / 124	PARATHYROID HORMONE RECEPTOR
Q03431	G_s	123 / 132	PARATHYROID HORMONE/PARATHYROID HORMONE-RELATED PEPTIDE RECEPTOR
Q02643	G_s	123 / 124	GROWTH HORMONE-RELEASING HORMONE RECEPTOR
O95838	G_s	181 / 192	GLUCAGON-LIKE PEPTIDE 2 RECEPTOR
P41180	G_{q11}	11 / 14	EXTRACELLULAR CALCIUM-SENSING RECEPTOR
P47872	G_s	123 / 124	SECRETIN RECEPTOR
P43220	G_s	125 / 142	GLUCAGON-LIKE PEPTIDE 1 RECEPTOR
P48546	G_s	124 / 140	GASTRIC INHIBITORY POLYPEPTIDE RECEPTOR
P25105	G_{q11}	4 / 7	PLATELET ACTIVATING FACTOR RECEPTOR
O43613	G_{q11}	52 / 56	OREXIN RECEPTOR TYPE 1

Table 6.6: The classification of unseen proteins using discovered patterns shows that the method is able to predict correct coupling spoecificity.

6.2.3 Discussion

In order to determine whether pattern discovery was strictly necessary for correct prediction, we also took the simpler approach of building a dendrogram from a multiple sequence alignment of the inner domains of the training set sequences. We did observe some propensity for receptors with the same coupling preference to be near each other in the tree, however, the delineation between the three groups of receptors was far from distinct.

The dependency of the classification on a prior determination of the topology of analysed protein sequences (whether GPCR or not, and whether the extracellular loops, transmembrane spanning regins, or intracellular loops) implies a context-dependence for the usage of the patterns. This *a priori* knowledge can be derived from protein domain databases like PRINTS or PFAM, from the results of similarity searches, or can be read directly from the manual annotations present in databases such as SWISS-PROT. With the increasing modularity of large-scale annotation efforts (Fleischmann *et al.* 1999; Moller *et al.* 1999; Rust, Mongin, & Birney 2002) such contextual information can now be technically incorporated into genome annotation. The present study thus represents an early example of a new breed of context-dependent protein domain annotation.

Clearly many aspects of the interaction between a receptor and its G protein(s) remain to be investigated. Our method of modelling whether a receptor is likely to be promiscuous in G protein coupling is straightforward. It would be worthwhile to determine whether unique interaction motifs exist for promiscuous coupling in receptors that have been demonstrated to lack selectivity in their G protein interactions. We did not try to construct patterns for the exclusion of certain G protein couplings, *i.e.* a pattern to represent an exception to a rule. Our approach

Pattern	$G_{i/o}$ (55)	G_{q11} (33)	G_s (25)	Sensitivity	Specificity	Best
[ILV]...SG.{0,10}R	15	0	0	0.273	1	$G_{i/o}$
N..R.{1,4}R	15	0	0	0.273	1	$G_{i/o}$
Y.A.{1,8}A[ILV]	15	0	0	0.273	1	$G_{i/o}$
A[ILV].{2,5}RT	15	0	0	0.273	1	$G_{i/o}$
N..[RK]..R	17	1	0	0.309	0.9444	$G_{i/o}$
K.[RK].{0,10}K.[ILV]	17	1	0	0.309	0.9444	$G_{i/o}$
V...[RK]....R	17	1	0	0.309	0.9444	$G_{i/o}$
[RK]...[CM][RK]	23	1	2	0.418	0.8846	$G_{i/o}$
V[RK].{1,10}SG	16	1	0	0.291	0.9412	$G_{i/o}$
K.[RK].{1,4}L[RK]	16	1	0	0.291	0.9412	$G_{i/o}$
[FWY][ILV]..V.{2,10}R	15	1	0	0.273	0.9375	$G_{i/o}$
Y.[RK].[RK].{0,9}T	15	1	0	0.273	0.9375	$G_{i/o}$
[ILV].A[AGS].{1,4}R	15	1	0	0.273	0.9375	$G_{i/o}$
FR....[RK].{0,3}L	15	1	0	0.273	0.9375	$G_{i/o}$
DRY.[AGS].{3,6}A	15	1	0	0.273	0.9375	$G_{i/o}$
F[RK]....K.{1,7}C	15	0	1	0.273	0.9375	$G_{i/o}$
A....[ILV].{1,8}RT	15	1	0	0.273	0.9375	$G_{i/o}$
[RK]....R.{0,9}EK	15	0	1	0.273	0.9375	$G_{i/o}$
[RK]R.{0,3}TR	15	1	0	0.273	0.9375	$G_{i/o}$
KA.{3,6}T	15	1	0	0.273	0.9375	$G_{i/o}$
DR.{4,11}H...[AGS]	15	1	0	0.273	0.9375	$G_{i/o}$
R....K.{0,8}T[AGS]	15	1	0	0.273	0.9375	$G_{i/o}$
[RK][FWY][ILV].{2,5}V	18	1	1	0.327	0.9000	$G_{i/o}$
N.{2,5}R.[FWY]	18	1	1	0.327	0.9000	$G_{i/o}$
Y.[AGS].{1,8}A[ILV]	18	2	0	0.327	0.9000	$G_{i/o}$
N..[RK].{1,4}R	23	3	1	0.418	0.8519	$G_{i/o}$
[ED].{0,3}N..[RK]	23	2	2	0.418	0.8519	$G_{i/o}$
Y.{2,5}I..[AGS]	23	0	4	0.418	0.8519	$G_{i/o}$
N..[RK].{1,11}R	30	6	2	0.545	0.7895	$G_{i/o}$
[RK].R.{2,12}K[RK]	20	4	0	0.364	0.8333	$G_{i/o}$
[ILV]...SG	20	1	2	0.364	0.8696	$G_{i/o}$
[AGS][RK]..[ED].{0,10}R	17	1	1	0.309	0.8947	$G_{i/o}$
[FWY].A.{1,9}A[ILV]	17	2	0	0.309	0.8947	$G_{i/o}$
R[FWY].[AGS][ILV].{0,7}A[ILV]	17	2	0	0.309	0.8947	$G_{i/o}$
[ILV].R....V	17	0	2	0.309	0.8947	$G_{i/o}$
[RK]Y.[AGS].{3,5}A	17	0	2	0.309	0.8947	$G_{i/o}$
[ILV]...SG.{0,8}E	17	0	2	0.309	0.8947	$G_{i/o}$
[FWY].[AGS][ILV]..A	17	1	1	0.309	0.8947	$G_{i/o}$
[RK]..[RK].{0,3}R[ILV]	32	8	2	0.582	0.7619	$G_{i/o}$
[ED]A.{0,3}E	19	3	0	0.345	0.8636	$G_{i/o}$

Table 6.7: The 40 best patterns found for each receptor-G protein coupling group, together with the number of times they match in each of the three training set groups, and their calculated sensitivity and specificity.

could eventually be improved by ignoring any pattern combination that does not span at least two inner loops, since from prior biochemical investigations it is unlikely this would be sufficient to provide an effective and selective G protein interaction (Wess 1998).

Pattern	$G_{i/o}$ (55)	G_{q11} (33)	G_s (25)	Sensitivity	Specificity	Best
T..[RK].{0,10}S..T	0	11	0	0.333	1	G_{q11}
A.{3,6}V[ILV][RK]	0	11	0	0.333	1	G_{q11}
P..[AGS]T.{0,10}S	0	10	0	0.303	1	G_{q11}
[AGS][ILV][ILV][RK].{2,10}S	0	10	0	0.303	1	G_{q11}
S[FWY].{1,11}Q[ILV]	0	10	0	0.303	1	G_{q11}
[AGS].{0,3}S..T[ILV]	0	10	0	0.303	1	G_{q11}
S...L.{2,9}TL	0	10	0	0.303	1	G_{q11}
[RK]F....K	0	10	0	0.303	1	G_{q11}
[AGS].[ILV].{0,10}K.F	0	10	0	0.303	1	G_{q11}
[AGS].S.[RK].{0,10}F	1	13	0	0.394	0.9286	G_{q11}
S...L.{1,10}T[ILV]	1	12	0	0.364	0.9231	G_{q11}
[RK].T.{0,10}Q[AGS]	0	12	1	0.364	0.9231	G_{q11}
[AGS]...L.{1,10}TL	1	12	0	0.364	0.9231	G_{q11}
[AGS][ILV][ILV][RK]	0	12	1	0.364	0.9231	G_{q11}
A.{0,10}V[ILV][RK]	1	14	1	0.424	0.8750	G_{q11}
[AGS].{0,3}V[ILV][RK]	1	14	1	0.424	0.8750	G_{q11}
F.{0,10}Y...[RK]	0	14	2	0.424	0.8750	G_{q11}
[CM].[FWY].{3,12}P	1	11	0	0.333	0.9167	G_{q11}
S.[AGS].{3,13}TL	1	11	0	0.333	0.9167	G_{q11}
V[AGS].{0,10}S.[AGS].[ILV]	1	11	0	0.333	0.9167	G_{q11}
Y....[RK]P.{2,10}A	0	11	0	0.333	1	G_{q11}
[ILV]......A.T	1	11	0	0.333	0.9167	G_{q11}
S...L.{1,11}Y	1	11	0	0.333	0.9167	G_{q11}
A.{3,12}V[ILV][RK]	0	11	1	0.333	0.9167	G_{q11}
[AGS].{2,5}V[ILV][RK]	1	11	0	0.333	0.9167	G_{q11}
[FWY].{4,7}KP	1	11	0	0.333	0.9167	G_{q11}
R.[RK].{0,10}K[AGS][AGS]	1	11	0	0.333	0.9167	G_{q11}
[ILV]A.{2,4}S.[ILV]	1	11	0	0.333	0.9167	G_{q11}
[AGS].[ILV].{2,10}L.[FWY]	0	11	1	0.333	0.9167	G_{q11}
[AGS][FWY]..[FWY]	1	11	0	0.333	0.9167	G_{q11}
S.S.{1,11}L.S	0	11	1	0.333	0.9167	G_{q11}
[ILV].L.{6,11}A.T	1	11	0	0.333	0.9167	G_{q11}
K.{0,3}N.P	1	11	0	0.333	0.9167	G_{q11}
[ILV].L.{6,10}A.T	0	11	0	0.333	1	G_{q11}
[RK][FWY]....K	2	13	0	0.394	0.8667	G_{q11}
[AGS].S.[RK].{2,10}F	1	13	0	0.394	0.9286	G_{q11}
[ILV].{3,6}S.Q	3	18	3	0.545	0.7500	G_{q11}
C.[FWY].{2,11}K	0	10	1	0.303	0.9091	G_{q11}
C.[FWY].{2,12}K	0	10	1	0.303	0.9091	G_{q11}
S....[RK]A.{3,10}S	1	10	0	0.303	0.9091	G_{q11}

Table 6.8: The 40 best patterns found for each receptor-G protein coupling group, together with the number of times they match in each of the three training set groups, and their calculated sensitivity and specificity.

Receptor-G protein recognition is known to be regulated by both post-transcriptional and post-translational modifications, likely of both the GPCR and the G-protein heterotrimer (Wess 1998). Analysing just the translated receptor coding sequence does not allow us to model such events. In spite of these issues, the discovered patterns are sensitive and selective, enabling our construction of a

Pattern	$G_{i/o}$	G_{q11}	G_s	Sensitivity	Specificity	Best
	(55)	(33)	(25)			
A[ILV].{1,5}Y..[ILV].T	0	0	10	0.400	1	G_s
A.{1,5}RY....T	0	0	10	0.400	1	G_s
I....RY.{1,10}R	0	0	9	0.360	1	G_s
I....RY.{4,6}T	0	0	9	0.360	1	G_s
LR.{1,9}T...[ILV]	0	0	9	0.360	1	G_s
RS.{3,13}C[AGS]	0	0	9	0.360	1	G_s
[ILV].[FWY]H.{1,3}I	0	0	9	0.360	1	G_s
F.{1,4}Y....T	0	0	9	0.360	1	G_s
I....RY.{4,4}T	0	0	9	0.360	1	G_s
I....R[FWY]	0	0	9	0.360	1	G_s
I....RY....T	0	0	9	0.360	1	G_s
I....RY	0	0	9	0.360	1	G_s
[FWY].A.{2,6}Y..[ILV]	0	0	9	0.360	1	G_s
I.[AGS].{1,10}S...R	0	0	8	0.320	1	G_s
[ILV].[FWY]H.{3,12}T	0	0	8	0.320	1	G_s
L..H.[ILV]	0	0	8	0.320	1	G_s
[ILV].[FWY]H.[ILV]	0	0	8	0.320	1	G_s
[ILV].[FWY]H.I	0	0	8	0.320	1	G_s
A....[RK][RK]I	0	0	8	0.320	1	G_s
[AGS].{0,10}L..H.[ILV]	0	0	8	0.320	1	G_s
[ILV].[FWY]H.{3,10}T	0	0	8	0.320	1	G_s
[FWY]H.I.{0,3}T	0	0	8	0.320	1	G_s
S.{5,12}S.L.[RK]	0	0	8	0.320	1	G_s
S.{5,9}S.L.[RK]	0	0	8	0.320	1	G_s
Q.{0,9}S.L.[RK]	0	0	8	0.320	1	G_s
A.{1,5}RY..[ILV].T	0	0	8	0.320	1	G_s
F.{1,10}A...H	0	0	8	0.320	1	G_s
[ILV]..H.[ILV].{1,3}T	0	0	8	0.320	1	G_s
[FWY]H.I.{0,10}V	0	0	8	0.320	1	G_s
A..[FWY].{0,3}H	0	1	10	0.400	0.9091	G_s
I....[RK]Y.{4,6}T	0	0	10	0.400	1	G_s
A.{1,5}R[FWY]....T	1	0	10	0.400	0.9091	G_s
A.{2,6}Y..[ILV].T	0	1	10	0.400	0.9091	G_s
A..[FWY].{0,8}H	1	1	11	0.440	0.8462	G_s
[AGS].{1,5}RY....T	0	2	11	0.440	0.8462	G_s
I....[RK]Y.{1,10}R	0	1	9	0.360	0.9000	G_s
R[FWY]H.{5,14}R	0	1	9	0.360	0.9000	G_s
[RK]S.{3,13}C[AGS]	1	0	9	0.360	0.9000	G_s
[RK].[ILV].C.R	1	0	9	0.360	0.9000	G_s
[RK].[ILV].C.[RK]	1	0	9	0.360	0.9000	G_s

Table 6.9: The 40 best patterns found for each receptor-G protein coupling group, together with the number of times they match in each of the three training set groups, and their calculated sensitivity and specificity.

useful predictor, allowing us to address a problem that has repeatedly been stated to be a rather difficult one (Wess 1998; Sautel and Milligan 2000; Horn, Vriend et al. 2001).

6.3 Simultaneous discovery of patterns and subfamilies

In Chapters 1 and 2 and previous subsections of Chapter 6 we used some relatively straightforward fitness measures for identifying the interestingness of patterns. These measures were based on the occurrences in one set versus another. For example, the specificity and sensitivity of the patterns, the ratio of occurrences in two different subsets of sequences, or the probability of observing the given number of patterns in the set of input sequences.

If the pattern discovery task is to find interesting patterns from a single set of input sequences, the question of what is "interesting" becomes trickier. We can say that a pattern is interesting if it occurs more often than expected. It is possible to estimate the number of expected occurrences by assuming some generative model and calculating the expected number of occurrences (and the standard deviation) either analytically or estimating it from data. Usually, the more complex the pattern is, the less occurrences it should have in a random text, and hence the more interesting the pattern is if it does occur frequently in the data. Hence, usually the aim is to find the patterns that occur in as many of the input strings as possible (Bairoch 1992; Jonassen, Collins, & Higgins 1995).

Here we show how a novel fitness measure based on the Minimum Description Length (MDL) principle (Li & Vitanyi 1993; Rissanen 1978) can be used for discovering biologically meaningful patterns (Brazma *et al.* 1996; Brazma, Ukkonen, & Vilo 1996; Brazma *et al.* 1998c).

The MDL principle means in the pattern discovery context that the best pattern is the one which minimizes the sum of

- the length (in bits) of the pattern; and
- the length (in bits) of the data when encoded with the help of the pattern.

Let us describe the application of the MDL principle in more detail for unions of substring patterns.

Let $B_1, \dots, B_k$ be a partition of a set of sequences $S = \{\alpha_1, \dots, \alpha_n\}$ (*i.e.* , $B_1, \dots, B_k$ are disjoint and their union is S) and let $\Pi = \{\pi_1, \dots, \pi_k\}$ be a set of patterns such that the pattern π_j matches all the sequences of the set B_j. We call $\Omega = \{(\pi_1, B_1), \dots, (\pi_k, B_k)\}$ a *cover* of S. We call Π the *pattern set* of Ω. We define $\|S\| = \sum_{j=1}^{n} |\alpha_j|$, and $|S| = n$. Let us assume that we want to transmit the set of sequences S over some channel. A trivial way would be to transmit $\alpha_1, \dots, \alpha_n$ one after another. If we abstract from the fact that some delimiters between sequences also have to be transmitted, the message length (in characters) would be $\|S\|$. Suppose that a substring pattern $*\varepsilon*$ is present in all the sequences, *i.e.* , $\alpha_i = \delta_i \varepsilon \zeta_i$, for some $\delta_i, \zeta_i \in \Sigma^*$. Then we can compress the message by first transmitting ε, and then $\delta_1, \zeta_1, \dots, \delta_n, \zeta_n$. Similarly, if $\Omega = \{(*\varepsilon_j*, B_j) | j =$

$1, \dots, k\}$ is a cover of S (*i.e.* , $B_j = \{\delta_i^j \varepsilon_j \zeta_i^j | j = 1, \dots, k\})$, then S can be transmitted using Ω, as:

```
for $j = 1$ to $k$ do
    send $\varepsilon_j$
    for $i = 1$ to $|B_j|$ do
        send $\delta_i^j$, send $\zeta_i^j$
```

The message length obtained this way is $M(\Omega) = \|S\| - \sum_{j=1}^{k} (|B_j| - 1)|\varepsilon_j|$. The MDL principle suggests that the best pattern set Π is the pattern set of a cover Ω that minimizes the message length $M(\Omega)$. The second term

$$C(\Omega) = \sum_{j=1}^{k} (|B_j| - 1)|\varepsilon_j|$$

in the expression can be considered the *compression* in comparison to $\|S\|$. Minimizing $M(\Omega)$ equals maximizing $C(\Omega)$ and thus C defines a rating function for pattern set Π in respect to S.

For generalized regular patterns, when we take into account the frequencies of different alphabet characters and the delimiter problem, these MDL considerations lead to the rating function of the type

$$R(\Omega) = \sum_{j=1}^{k} (u_j |B_j| - w_j), \tag{6.3}$$

where u_j and w_j are parameters (positive numbers) that depend on the pattern π_j and assumptions about various probability distributions. We have shown that under a certain Bayesian assumption this rating function maximizes the probability that the particular union of patterns is the "source" of the examples (Brazma *et al.* 1996; Brazma, Ukkonen, & Vilo 1996).

We have developed a program called MDL-Pratt by modifying the tool Pratt (Jonassen 1997), which allows us to discover and rate patterns using the developed MDL measure.

To test the MDL based measure in practice, we collected a set of 31 protein sequence segments, each of which is believed to contain a chromo domain (Paro & Hogness 1991). Aasland and Stewart identified two subfamilies (subsets); (1) the classical chromo domains linked to chromo shadow domains and (2) the chromo shadow domains (Aasland & Stewart 1995). Our sample set contained 8 members of each subfamily.

MDL-Pratt, when run on this set of 31 sequences, produced three patterns given in Figure 6.6, covering subsets of sizes 7, 6, and 8 respectively (the remaining 10 sequences were included in the union individually). The first two sets produced by MDL-Pratt correspond closely to the two subfamilies given in (Aasland

& Stewart 1995). Compared to these subfamilies, one segment is missing from the first set, and two segments are missing from the second (see Figure 6.7).

The result indicates that MDL-Pratt can be used to discover family- and subfamily-relationships in a set of sequences and filtering out the "noise".

```
1.  E-x(0,1)-E-E-[FY]-x-V-E-K-[IV]-[IL]-D-[KR]-R-x(3,4)-G-x-V-x-Y-x-L-K-
    W-K-G-[FY]-x-[ED]-x-[HED]-N-T-W-E-P-x(2)-N-x-[ED]-C-x-[ED]-L-[IL]

    common to: DmHP1_A DvHP1_A HuHP1_A MoMOD1_A MoMOD2_A PcHET1_A PcHET2_A

2.  L-x(2,3)-E-[KR]-I-[IL]-G-A-[TS]-D-[TSN]-x-G-[EDR]-L-x-F-L-x(2)-[FW]-
    [KE]-x(2)-D-x-A-[ED]-x-V-x-[AS]-x(2)-A-x(2)-K-x-P-x(2)-[IV]-I-x-F-Y-E

    common to: DmHP1_B DvHP1_B HuHP1_B MoMOD1_B MoMOD2_B PcHET1_B

3.  Y-x(0,2)-L-[IV]-K-W-x(6)-[HE]-x-[TS]-W-E-x(4)-[IL]

    common to: DmPc MoMOD3 HuMG44 CfTENV FoSKPY MoCHD1_A MoCHD1_B ScYEZ4_B
```

Figure 6.6: The patterns obtained when running MDL-Pratt on the 31 chromo domain sequence segments. The patterns are given in PROSITE notation, and are followed by the names of the set of sequences (among the initial set of 31 sequences) matching this pattern. The remaining 10 sequences were represented by singleton sets in the cover.

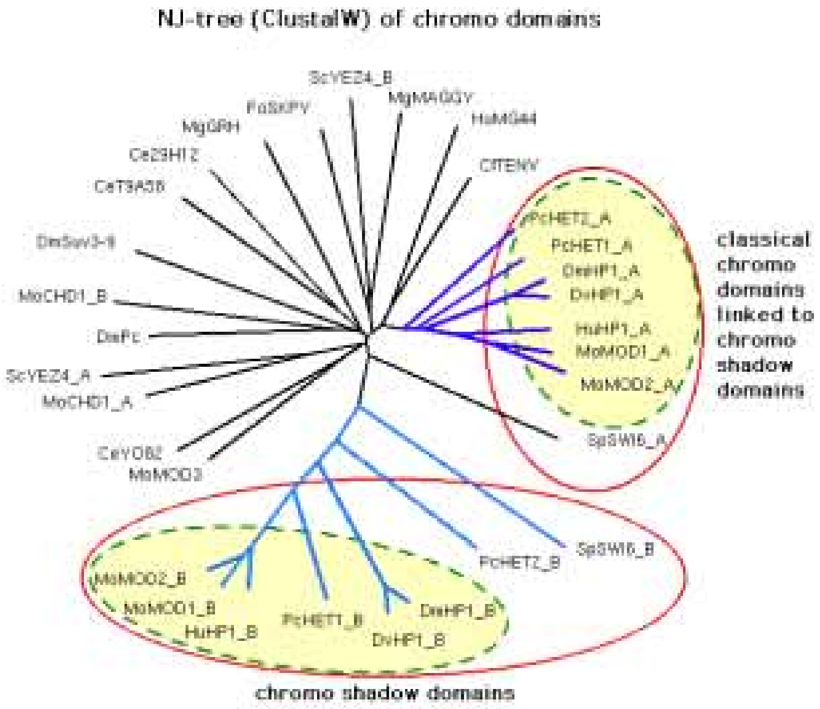


Figure 6.7: Estimate of the phylogenetic tree for the chromo domain segments included in the analysis. The tree was produced using Clustal W (Thompson, Higgins, & Gibson 1994) which uses the neighborhood joining method (NJ) for estimating phylogenetic trees (Saitou & Nei 1987), and was taken from the 'official' World-Wide-Web pages for (Aasland & Stewart 1995). The dashed ellipses show the two most important subsets identified by MDL-Pratt.

Chapter 7

Software tools

Bioinformatics research is often application-driven requiring novel practical tools. This Chapter describes briefly a set of tools that have been implemented based on the methods developed in this thesis and that have been used to achieve most of the practical results described.

7.1 Expression Profiler

Expression Profiler (EP, http://ep.ebi.ac.uk/) is a set of tools for the analysis and interpretation of gene expression and other functional genomics data. For example, the tools enable the integration of expression data with protein interaction data and functional annotations, like Gene Ontology (The Gene Ontology Consortium 2000), or the analysis of promoter sequences for predicting transcription factor binding sites. The developed tools are:

EPCLUST for gene expression data clustering, analysis, and visualization,

SPEXS for sequence pattern discovery,

URLMAP for integrating tools and databases over the Internet,

EP:GO for using Gene Ontology ontologies,

EP:PPI for integrated analysis of protein protein interaction and gene expression data sets,

GENOMES for storage and retrieval of genomic DNA (e.g. upstream sequences) and gene annotations,

PATMATCH for matching and visualization of sequence patterns with the added possibility to combine gene expression and sequence data analysis results,

SEQLOGO for visualization of sequence alignments and position weight matrices using sequence logos.

Several clustering analysis methods of EPCLUST and sequence pattern discovery methods of SPEXS, PATMATCH and SEQLOGO provide a rich data mining environment for various types of biological data. All the tools are web-based with minimal browser requirements. Analysis results are cross-linked to other databases and tools available on the Internet. This enables further integration of the tools and databases, for instance with such public microarray gene expression databases as ArrayExpress (Brazma *et al.* 2000; 2002). Expression Profiler is described in more detail in (Vilo *et al.* 2003) and in the on-line documentation of Expression Profiler (Vilo, Kapushesky, & Kemmeren 2002).

We started the development of Expression Profiler originally for identifying sets of co-expressed genes and predicting their potential co-regulation mechanisms (Vilo *et al.* 2000). The main design principle of Expression Profiler has been rapid prototyping of new analysis methods and integration of different data types and tools for analyzing these data. More recently we have focused on providing a reliable service by offering a web-based access to different analysis methods. Access to the tools over the web allows users to share their data and analysis results with other people and to create their own collaborative research environments.

7.1.1 SPEXS - Sequence Pattern EXhaustive Search

SPEXS is a software tool that implements the main algorithmic ideas from Chapter 3. The command-line parameters determine the pattern language and elementary requirements that pattern should satisfy, as well as the choice of the search order and method for pattern discovery. Some of the pattern language definition parameters are the maximum motif length, the number of allowed group character positions, number and length of allowed wild-card characters. Configuration files are used to represent the character set Σ, character groups Γ, and some extra information like stop-characters.

The fitness measures used for outputing the discovered patterns correspond to elementary goodness criteria based on the ratio and probability based statistics discussed in Sections 6.1.1 and 6.1.3.2. Users can set the thresholds for the fitness measure, according to which all patterns that are at least as good are output.

Patterns can be output together with the counts of occurrences in each input set, as well as all the positions of matches, if needed. Using this output users can further analyse the discovered patterns and apply their own fitness criteria to further select the most interesting patterns.

When SPEXS searches for patterns, it follows strictly the pattern language specified by a collection of parameters describing the acceptable patterns. The different search orders (depth-first, breadth-first, frequent patterns first, *etc.*) allow for flexible search strategies.

7.1.1.1 WWW-interface

We have developed a WWW-interface that allows to use different parameter settings in a more user-friendly manner. It has been developed in conjunction with the Expression Profiler that is described in the next section. The development of Expression Profiler is an ongoing project and will also affect the SPEXS interfaces. They share the same code for data and folder handling, as well as the tools need to be integrated into joint analysis flows. The SPEXS interface can be roughly divided into three larger parts:

- Data upload and folder handling
- Interface to the pattern discovery algorithm parameters
- Sorting and analysis of the discovered patterns, links to other tools

The data sets can be either uploaded from the end-user machine or extracted from other tools (GENOMES) of Expression Profiler. Data files are stored in data folders, *i.e.* containers of related data sets. Within the folder the users can perform the analysis, results are stored in the same folder.

Parameters for describing the pattern language and fitness measures can be chosen from pulldown menus or inserted in text fields. The parameters are presented on a single page, which can be modified until the user is sure that the pattern discovery process is ready to be launched.

The results, *i.e.* the reported discovered patterns, can be sorted according to different sort criteria. Patterns are directly linked to PATMATCH tool that allows to visualize the locations of each pattern in the input (or other) data.

7.1.1.2 Performance measurements

We have measured the time to run SPEXS using different data sets and parameter settings.

First, the construction of the most frequent substring patterns as discussed in Section 3.1.2. We used the file `Yeast_-600_+2_W_all.fa`[1] containing all upstream sequences (6423 sequences) of length 600bp (plus the start codon marked `_ATG_`) as the input data set. We varied the threshold K and the pattern search order. The input sequences when appended one after another form a sequence of 3.89 million characters. Measurements are given in Table 7.1.

[1] We have used this file commonly for yeast *Saccharomyces cerevisiae* upstream sequence analysis, it can be extracted from the GENOMES, and is available from `http://ep.ebi.ac.uk/EP/PATMATCH/SEQUENCES/Yeast_-600_+2_W_all.fa`

K	breadth-first	depth-first	frequent first
6000	7.1	7.1	7.1
4000	7.9	7.7	7.9
1000	9.9	9.1	9.9
100	13.9	11.4	13.8
10	19.5	14.6	20.3

Table 7.1: Speed measurements for problem type P1:B, *i.e.* substring patterns common to at least K sequences (of length 605 characters) out of 6423. The times are real wallclock times in seconds, measured on Compaq Alpha ES40 server running a True64 Operating system. The different columns show the times for different search orders.

Next, we measured the time for finding the most overrepresented patterns (problem type E) for different pattern classes. We used as the first set of sequences the set of 98 upstream sequences belonging to a cluster of co-expressed genes (see Section 7.2) and compared it to all upstream sequences in the file `Yeast_-600_+2_W_all.fa`. We used a binomial probability based measure (Section 6.1.3.2) to estimate the proability of a pattern to be overrepresented. We asked SPEXS to report all patterns that have probability less than 10^{-5} to be present in at least that number of times in the cluster of 98 sequences.

Pattern language	K=90	K=50	K=20
Substrings	9.0	11.0	12.6
Subsrings with 1 single-character wildcard	38.0	51.4	66.4
Subsrings with 2 single-character wildcards	97.1	150.6	262.1
Subsrings with 3 single-character wildcards	200.2	343.9	553.0
Substrings with 2 of any 2-character groups	489.3	757.0	914.7
Substrings with one restricted wildcard $*(0,15)$	200.8	312.4	469.0

Table 7.2: Speed measurements for finding patterns overrepresented in a cluster of 98 sequences vs. in all upstream sequences of yeast. Different columns show the effect of changing K, the number of sequences in a cluster, where the pattern should be present. The times are real wallclock times in seconds, measured on Compaq Alpha ES40 running True64 Operating system.

Finally, for protein sequences, we measured the time for the pattern discovery task described in Section 6.2, *i.e.* the pattern discovery for finding patterns more frequent in one set of internal loop containing sequences vs. two other sets. The running times varied between 17 and 25 seconds for patterns containing up to 5 group character positions $\Gamma = \{\Sigma, \{\mathtt{RK}\}, \{\mathtt{ED}\}, \{\mathtt{AGS}\}, \{\mathtt{ILV}\}, \{\mathtt{FWY}\}, \{\mathtt{CM}\}\}$ and one restricted wildcard of length between 0 and 3.

The similar task, when allowing for up to three occurrences of restricted wildcards of length between 0 and 7 positions, took about 18 minutes. The number of patterns generated was over 7 million and of those with a probability less than 10^{-5} about 11000. Note that this pattern language is more complex than that used in Section 6.2. For predicting the $G_{i/o}$ membership, for example, it revealed a pattern `R[FWY].[AGS].{0,7}A...{0,7}R.{0,7}R` that occurs in 19 sequences from class $G_{i/o}$ and not a single time in other sets. Of the 19 occurrences in $G_{i/o}$ all except one are in the short second intracellular loop (one occurrence is in the third loop).

For smaller sets of sequences, as is the case for predicting the coupling specificity for GPCR proteins, we have experimented with eliminating the construction of multiple equivalent pattern subtries. This is achieved by keeping track where exactly are all the occurrences for each pattern. If another pattern during the construction has exactly the same set of occurrences, then that particular subtree is not constructed.

7.1.1.3 Discussion

The challenge for postprocessing the SPEXS results remains an open research area. Given the large number of potentially reported patterns there is a need for mechanisms that are able to reliably identify the most important patterns that cover all the ”interesting” cases. Some approaches have already been discussed in Chapter 5. More are needed. For example, for showing the patterns in different levels of detail and allowing users to dig in deeper into more interesting pattern types.

7.1.2 EPCLUST

EPCLUST (Expression Profile data CLUSTering and analysis) is the module for gene expression data matrix analysis, including data selection and filtering, clustering, visualization, and similarity searches. Additionally, it supports missing data imputation, data randomization, and data rescaling.

EPCLUST provides a folder-based analysis environment to operate with data sets in server-side folders. The folders serve as secure data authoring, analysis and publishing environments with multi-user access capability over the web, allowing distributed collaboration.

Raw data may be uploaded to EPCLUST in a number of different formats, either from the users’ own computers or from gene expression databases available on the web. These raw data then should be filtered to create an analysis data set.

Upon the transfer of a large data set to an EPCLUST server, the user will often select only part of the data for a particular purpose. For example, after uploading the data set of the expression of an entire genome in a time-course experiment with numerous measurements, the user may desire to focus on specific time-points or experiments, and only on those genes that have shown significant differential expression under certain specified conditions.

The rows and columns of data have tags by which they can be identified. The row annotations are read from a file that provides a description for every row ID. These descriptions may contain HTML markup, to link to corresponding entries in a database, or to highlight the key biological roles known for the genes. The annotations will be shown together with the clustering results, facilitating data interpretation, if the annotations were created carefully and with detail.

The prepared data files are stored in data folders, where various analysis can be performed. The intermediate (e.g. the all against all distance matrices) as well as the final analysis results (clustering results and visualizations) are also stored in the same folders, where they can be viewed later without having to redo the same analysis steps.

Distance measures

Cluster analysis is a type of classification algorithm that is intended to organize a collection of objects into meaningful structures. This meaning is given by the relative proximity of objects within one cluster in comparison to objects in other clusters. There exist many different distance measures for the comparison of expression profiles. These measures are grouped in EPCLUST on the basis of their general properties.

The first group consists of the standard metric distances: the Euclidean distance, the Manhattan distance, and the average distance (Euclidean distance that has been normalized by the vector length).

The second group consists of distances that essentially measure the correlation (or angle) between vectors. Note that these distances capture the idea that the direction and relative intensity of the values are what is important, and not the absolute values and magnitudes of the change. This corresponds well with the tasks of expression analysis where the absolute change can be meaningless, while the direction and relative scale of the change are crucial.

The third group is based on measures that work on ranked expression vectors instead of on the original values (rank correlation), or measures that are applicable only to discrete (*e.g.* binary vectors) data.

Clustering methods

We implemented fast versions (see Table 7.3) of standard agglomerative (bottom up) hierarchical clustering methods and partitioning-based K-means methods, and incorporated them into the web interface with extensive visualization options.

The hierarchical clustering method requires that all pairwise distances are calculated first. For longer vectors, *e.g.* for many hybridizations analyzed simultaneously, this can be the most time-consuming step of the analysis. The distance

Vector length Number	10 clustering	10 distances	100 clustering	100 distances	1000 clustering	1000 distances
10	0.00	0.00	0.00	0.00	0.00	0.00
100	0.00	0.00	0.00	0.00	0.00	0.03
500	0.05	0.02	0.05	0.09	0.05	0.78
1000	0.19	0.08	0.20	0.35	0.21	3.12
2000	0.82	0.31	0.87	1.40	0.91	12.57
3000	2.41	0.71	2.60	3.17	2.67	28.46
4000	4.77	1.26	5.15	5.65	5.34	50.61
5000	8.02	1.95	8.70	8.81	9.04	79.58
6000	12.28	2.79	13.23	13.42	13.85	114.64
7000	17.08	3.80	18.45	17.94	19.43	156.53
8000	22.78	5.31	24.76	23.75	26.29	204.28
9000	31.52	6.53	34.87	31.26	34.97	270.30
10000	40.03	8.16	43.02	35.80	43.74	319.52
15000	96.46	27.03	101.33	82.23	108.99	719.68
20000	195.09	51.92	194.81	148.97	201.31	1279.05

Table 7.3: Times of all-against-all distance comparisons and hierarchical clustering for up to 20,000 vectors of length 10, 100, and 1000 numeric attributes. Times are in seconds, as measured on an Alpha ES40 ev64 server on a single processor.

matrices are reused by different clustering methods. The hierarchical clustering methods include:

complete linkage using maximum distance between members of two clusters for defining the distance between clusters;

average linkage using weighted and unweighted group-wise average methods WPGMA and UPGMA;

single linkage using minimum distance;

The K-means clustering method requires users to provide the number of clusters as well as a method for choosing the initial cluster centers. Starting from potential cluster centers the K-means procedure assigns all genes to their respective clusters (the cluster to whose center they are closest) and refines cluster centers to be the geometric centers of gravity for each defined cluster. This process is repeated until the clustering stabilizes or the number of allowed cycles is reached.

Other clustering methods are currently being added to EPCLUST. These include the linear-time top-down divisive hierarchical clustering method SOTA (Self Organizing Trees (Herrero, Valencia, & Dopazo 2001)), and some variants of the standard Self Organising Map (SOM) algorithms (Kohonen 1997). For extensive coverage of clustering methods see, for example, (Legendre & Legendre 1998; Everitt, Landau, & Leese 2001).

Similarity searches

Similarity searches provide users with alternative straightforward ways to study gene expression without requiring to cluster the whole data set first.

The user can select genes of interest by typing in their IDs or by querying from data set annotations and then use these genes to query the entire data set for similar ones.

Genes whose expression profiles are similar under a chosen distance measure (a variety are available) are reported. To reduce the size of the meaningful answer, users can set cut-off thresholds for the number of genes to be displayed or for the maximal distance that is relevant to their query (*e.g.* the estimate of the significance threshold for a particular data set (Kruglyak & Tang 2001)).

There exists also an "opposite" search, in the sense that only the farthest distance genes are displayed in the result set. This way the user can study the genes that are anti-correlated with the gene of interest.

The similarity search can be performed either on one gene at a time, or on a set of genes simultaneously. If many genes are used in the search together, the results can be displayed either for each one individually, or as a combined result merging all the individual results. This way, for example, starting from a tight cluster of co-expressed genes, one can search for other genes that have similar profiles to any of the original ones. This can be seen as generating a "supercluster".

Visualization methods

The results of the cluster analysis and similarity searches are presented visually in the heat-map format, made popular in the microarray community by Mike Eisen (Eisen *et al.* 1998). The results are presented in PNG (Portable Network Graphics) or GIF formats, with clickable regions for drilling into the expression data - *e.g.* exploring subtrees in a hierarchical clustering.

7.1.3 URLMAP: integrating the web applications

URLMAP is a tool for integrating different web based tools and databases by providing a mechanism for cross-linking data or results from one application to others. Links are provided in a hierarchical structure managed by the tool administrator, thus enabling for example the thematic categories for improved usability. URLMAP takes as an input the data (for example the gene id's) from the source application and creates as an output the ready-made HTML form based links that are tailored automatically for the target application. Link descriptions in URLMAP contain information about the necessary fields for the target application and possible rules (expressed as perl code) for modifications to the data content, enabling for example the replacing of gene id's by their synonyms used in the target application.

From the very early days of development of EPCLUST it was obvious that expression data analysis, however important, can not be viewed in isolation from other available data and information about the genes and experiments. The question is how to integrate expression data analysis results (*e.g.* the list of genes in a particular cluster) to genomic databases, metabolic pathways, available annotations, putative promoter sequences and their analysis, *i.e.* to all the resources available on the Internet, scattered between many different sites and locations. Rather than trying to build into the EPCLUST clustering tool all the possible cross-links, we wanted to concentrate the cross-linking functionality into one central tool.

The main problem is that all these resources have different web interfaces and users often need to copy-paste the gene id's. Moreover, each of these interfaces have different parameters and choices that need to be filled in for the queries to be meaningful. Finally, different databases often use different naming conventions for the same genes, so semi-automatic translations between these id's are necessary. URLMAP handles query generation automatically; users are presented with pre-configured hyper-links and/or buttons, making it easy to try out different database queries and tools.

URLMAP is configurable on the server side with simple text file formats. End-users, however, can assist in developing and maintaining these lists. For example, one can design and test custom-made mappings and suggest them to server administrators for inclusion in URLMAP.

Currently the URLMAP tool provides mainly links for major yeast *Saccharomyces cerevisiae, Saccharomyces pombe* related tools and databases, and links for integrating different modules of Expression Profiler.

Most databases do not allow queries with multiple IDs simultaneously, yet this is the typical case when studying gene clusters. For these cases URLMAP creates multiple individual queries and users can click on each one separately.

A more serious problem is if the databases use different names for the same objects. Sometimes there exists a one-to-one mapping between these different naming conventions. For example, the yeast community has been standardizing gene identifiers: the yeast ORF names provide the most commonly used references for each gene (although even the ORF names may contain synonyms or homonyms). In SwissProt, for example, the proteins corresponding to yeast genes also have SwissProt accession numbers. The Stanford Saccharomyces Genome Database (SGD) uses its own unique IDs, which are also used as primary identifiers in the *S. cerevisiae* GeneOntology association files.

URLMAP provides ways to map between different identifiers. If the site administrator knows which of these naming conventions is most appropriate for the given database, (s)he may provide an automatic mapping to that naming convention before creating the query. Currently, for example, there exist mappings between yeast ORF names, gene names, SwissProt IDs, and SGD identifiers in URLMAP.

7.1.4 EP:GO Gene Ontology browser

EP:GO is a tool for browsing and using the ontologies developed by the Gene Ontology consortium (The Gene Ontology Consortium 2000). EP:GO allows to study the ontology by searching for specific entries and by traversing the ontology directly. It provides cross-links to other databases, mapping the IDs when necessary. For example EP:GO maps GO entries to enzyme databases by the EC numbers.

EP:GO incorporates information about gene associations, *i.e.* the lists of genes corresponding to each ontology entry for each annotated organism. Thus it becomes easy to extract expression information for all genes in any particular GO category. To get all the genes in any particular category, all the subcategories have to be queried and combined. These extracted gene lists can then be analyzed by other methods; for example, their expression profiles can be looked up with the EPCLUST tool, or their respective upstream sequences can be analyzed for identifying GO category-specific transcription factor binding site motifs. Links between these analysis tools are provided by URLMAP.

Gene associations are also useful in another respect, namely, in trying to understand the biological relevance of a cluster of co-expressed genes. Given a list of genes it is natural to ask in which processes these genes are involved, what are the possible common functions of the genes, and whether they might be expressed in the same subcellular locations. EP:GO takes as input a list of genes and ranks the GO categories based on how well they correlate with the genes from the cluster.

7.1.5 EP:PPI: comparison of protein pairs and expression

EP:PPI is a tool for analyzing protein-protein interaction data in the context of gene expression data.

High-throughput functional genomics are generating a wealth of information. Inherent is the artificial nature of many of these assays and the heterogeneity in data quality. Therefore, additional verification is necessary to reduce the number of false positives and to provide a more accurate functional annotation.

Using EP:PPI, protein-protein interactions (PPI) can be compared with mRNA expression data. This comparison enables the prioritization of protection-protein interactions. A more reliable functional annotation can then be obtained for uncharacterized proteins, using a significance threshold, based upon previously known protein interactions (Kemmeren *et al.* 2002).

EP:PPI uses existing PPI data sets and compares these with mRNA expression data sets in EPCLUST. The protein pairs are ranked according to their expression distance and plotted on a graph. Other genes that show a similar pattern in mRNA expression levels can be obtained using the similarity-based search (a direct hyperlink to EPCLUST similarity search). This allows to observe whether genes closely related according to mRNA expression also belong to the same functional class.

Another advantage of using mRNA expression data to prioritize the PPI is that it gives insights into particular conditions under which these proteins are co-expressed. In the future EP:PPI will be expanded to include more different types of functional genomics data, so that these can be used to pinpoint false positives *i.e.* the protein pairs that are not likely to interact although present in the data, as well as to speed up the functional annotation efforts. The method has been described in more detail in (Kemmeren *et al.* 2002), where the predictions are also supported by biological experiments.

7.1.6 SPEXS, PATMATCH, and SEQLOGO: Pattern discovery, pattern matching and visualization tools

The numerical data analysis methods for gene expression data are complemented within Expression Profiler with the sequence analysis tools. In the context of gene expression data analysis these tools facilitate primarily the discovery, analysis, and visualization of putative transcription factor binding site motifs. This type of analysis is illustrated in the example in the next section. Here we provide a brief description of the available tools.

SPEXS (Sequence Pattern EXhaustive Search) is a tool for discovery of novel patterns from sets of unaligned sequences. The SPEXS algorithm has been described in Chapter 3 and the tool has been discussed in Section 7.1.1.

PATMATCH is a tool for pattern matching and visualization, used for matching regular expression patterns in sequences and for visualizing the results. PATMATCH also facilitates approximate matching of patterns for certain pattern classes. Visualization of PATMATCH shows the sequences and the matches within them. Sequence and pattern visualization can be combined with gene expression clustering and heat-map visualizations, thus allowing to see the correlations between sequences, sequence motifs, and respective gene expression profiles.

SEQLOGO is a pattern visualization tool for DNA and protein motifs (position weight matrices) that can be used in combination with PATMATCH and SPEXS. It takes as input a set of sequences or a position count matrix and outputs a visual representation of the respective position weight matrix in the form of a sequence logo (Schneider & Stephens 1990).

7.2 Data analysis and visualization example with Expression Profiler

Here we present an example that illustrates how the web-based tools in Expression Profiler can be used for analyzing the various aspects of gene expression.

Consider a yeast gene, YGR128C; it was recently given a reserved name UTP8. This gene has been classified as of unknown function, deletion of this gene, however, results in a lethal phenotype (Winzeler *et al.* 1999), suggesting that this gene plays an important role .

Searching for profiles by similarity

As has been suggested before (Eisen *et al.* 1998), proteins with related functions often show coexpression on the mRNA level. In this case we want to identify genes that are possibly functionally related, *i.e.* have similar expression profiles to YGR128C. For that we use the data set from (Eisen *et al.* 1998), available in EPCLUST in the folder `All_PB`. In EPCLUST, we first go to the folder named `All_PB`, then, after choosing the data set called `All_genes`, we select the action "Search profiles by their similarity".

Upon entering the gene name YGR128C, and searching for the 100 most similar genes using correlation distance (non-centered), we receive a list of 101 genes: YGR128C and 100 other genes whose expression profiles are most similar to it. These can be viewed in the results, under the expression heat map and the profile graph, where there will be a listing of 101 genes, with YGR128C at the top. It should be noted that the majority of these have been annotated as of unknown function. The heat map of the expression of these 101 genes is presented in Figure 7.1.

Analyzing annotations

To analyze this set further, we click the "Submit to URLMAP" button below the brief annotations. We choose the category "Bioinformatics for yeast ORFnames" from URLMAP and press "Redirect" to get the links to different databases for these 101 genes.

We select "Annotate a cluster of yeast ORFnames by GO". This analysis shows that the genes in the cluster of 101 that have annotated functions are mostly related to ribosomal RNA, transcription, the Pol I promoter; rRNA processing; ribosome biogenesis; RNA binding; RNA processing; RNA metabolism and cytoplasm organization.

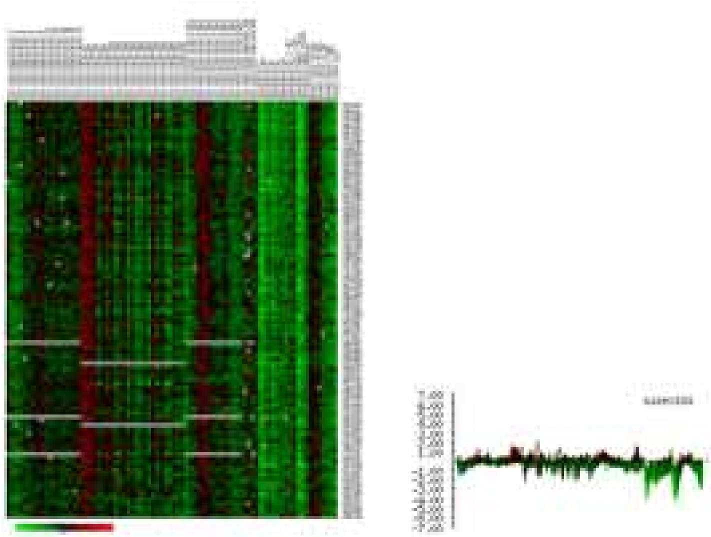


Figure 7.1: Expression profiles of the 100 genes most similar to YGR128C as heat-map and line plot.

Promoter analysis

We demonstrate how to analyze the promoter region of the gene YGR128C and provide evidence for two independent binding sites that possibly regulate the expression of this particular gene.

Sequence extraction

We choose “Genome tools: Yeast, full table” to get to the GENOMES tool where we can study the MIPS annotations for these genes, and extract suitable length upstream sequences (start- and end-positions of the sequences can be determined relative to the ORF start position). These extracted sequences can be analyzed with SPEXS for identifying the patterns that are overrepresented in the sequence set.

Pattern discovery

We choose “SPEXS pattern discovery” to start the pattern discovery process. SPEXS can see the sequences extracted by the GENOMES tool; we recommend using the 600bp-long sequences, upstream relative to the ORF start, *i.e.* the data set `Yeast_-600_+2_W_all.fa`. In our example only 98 genes were found in the genome instead of 101. This may be due to lack of knowledge or simply inconsistencies at the time of producing an array. These 98 sequences will be stored in a new folder of SPEXS.

SPEXS is able to study two data sets simultaneously and we want to discover motifs that occur more frequently in the set of 98 upstream sequences of co-expressed genes than in the set of *all* upstream sequences in the yeast genome. In fact, a random sample of, say, 1000 upstream sequences would be sufficient for this analysis, so for the background data set users may want to use 600bp upstream sequences of 1000 randomly chosen genes, all mapped to the same sense strand (data set `Yeast_-600_+2_W_random_1000_all.fa`).

For the current data set we require SPEXS to search for unrestricted length patterns that occur in at least 20 sequences within the cluster, allowing up to 2 wild card characters within a motif, and to report the patterns that are significantly overrepresented. For overrepresentation we require it to be at least twice as frequent within the cluster as in the random set of sequences, with the respective binomial probability less than 1e-08.

Here is the list of the top ten most significant patterns, as provided by the SPEXS pattern report:

```
    Pattern      Cluster    Background   Ratio        Binomial Prob.
1.  G.GATGAG.T   1:39/49    2:23/26      R:17.3026    BP:1.12008e-37
2.  G.GATGAG     1:45/60    2:44/50      R:10.436     BP:1.61764e-34
3.  GATGAG.T     1:52/70    2:72/78      R:7.36961    BP:2.79148e-33
4.  TG.AAA.TTT   1:53/61    2:79/84      R:6.84578    BP:1.83509e-32
5.  AAAATTTT     1:63/77    2:137/154    R:4.69239    BP:1.19109e-30
6.  TGAAAA.TTT   1:45/53    2:59/61      R:7.78277    BP:3.86086e-29
7.  AAA.TTTT     1:79/145   2:264/392    R:3.05349    BP:5.66833e-29
8.  G.AAA.TTTT   1:51/62    2:84/94      R:6.19534    BP:5.69933e-29
9.  TG.GATGAG    1:30/35    2:19/22      R:16.1117    BP:9.35765e-28
10. TG.AAA.TTTT  1:40/43    2:46/48      R:8.87311    BP:1.11240e-27
```

This output means that, for instance, the top-ranking pattern `G.GATGAG.T` occurs in 39 sequences in the first data set of 98 sequences with 49 matches in total. *i.e.* some sequences have multiple occurrences of the motif. It also occurs in 23 out of 1000 sequences in the second, background, dataset, hence the relative ratio being $39/98 \cdot 1000/23 = 17.3$. The probability 1.12008e-37 is the binomial probability to observe 39 sequences out of 98 to contain the pattern given the background probability that on average only 23/1000 (*i.e.* 2.3%) of the randomly chosen sequences would contain that very motif.

From this list the first and the fourth patterns represent the most significant distinct motifs (the second and third are variations of the first). These two motifs have been identified previously as the PAC and RRPE motifs respectively and thought to be involved in rRNA transcription and processing (Pilpel, Sudarsanam, & Church 2001).

Matching discovered patterns to sequences

To analyze further the discovered motifs we use the pattern matching and visualization tool PATMATCH. The significance of the PAC and RRPE motifs is well supported, as PATMATCH shows that both of them are in fact well-conserved and occur on average between 50 and 250bp upstream from the ORF start (see Figure 7.2).

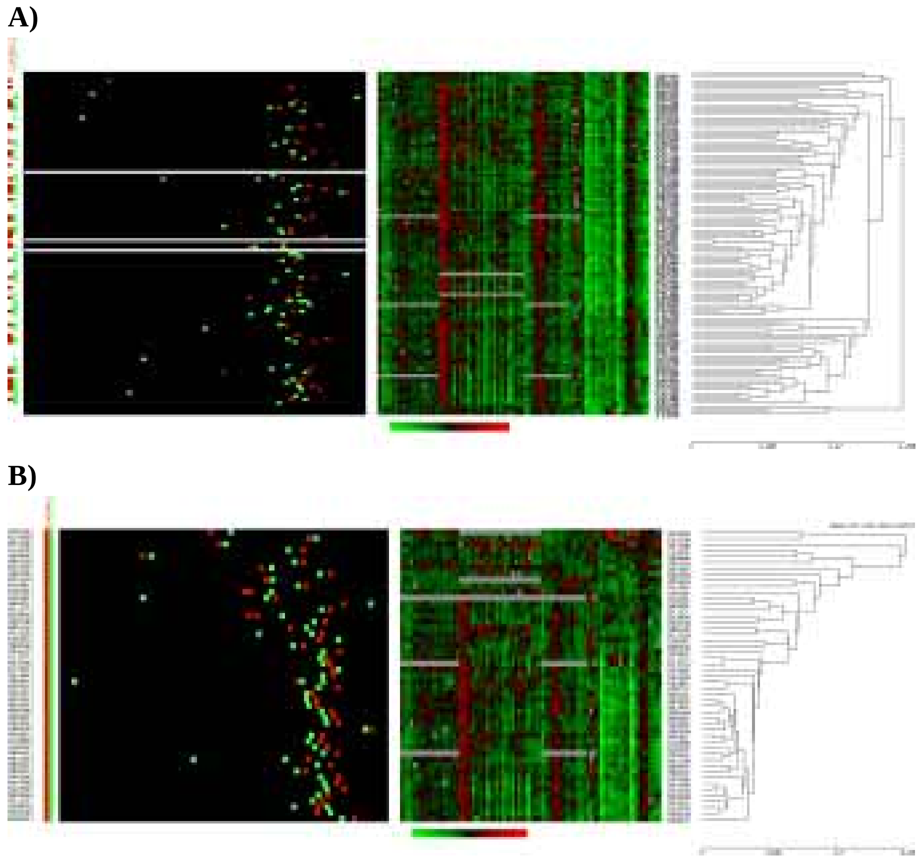


Figure 7.2: A) A cluster of 98 genes based on the expression data (middle) and the two SPEXS-found motifs `G.GATGAG.T` and `TG.AAA.TTT` specific to the cluster (clustering on the right), matched to promoter regions of respective upstream sequences (left). B) Hierarchical clustering (right) of all the yeast genes where the same two motifs occur within 40bp on their 600bp upstream sequences (left). It shows that all the 52 genes where these two motifs are in close vicinity to each other have an almost unique expression response (middle).

In PATMATCH the users can search the sequence data for specific regular expression type patterns and see the graphic visualizations of the occurrences of these patterns on the sequences. Several patterns can be matched simultaneously and the occurrences of each pattern visualized by a different color. For simple patterns approximate matching is implemented: executing a match for -1:TGAAAA.TTT is equivalent to matching the pattern TGAAAA.TTT and allowing one mismatch.

The sequence visualizations by PATMATCH can be combined with gene expression data visualization from the EPCLUST. This is illustrated in Figures 7.2 and 7.3.

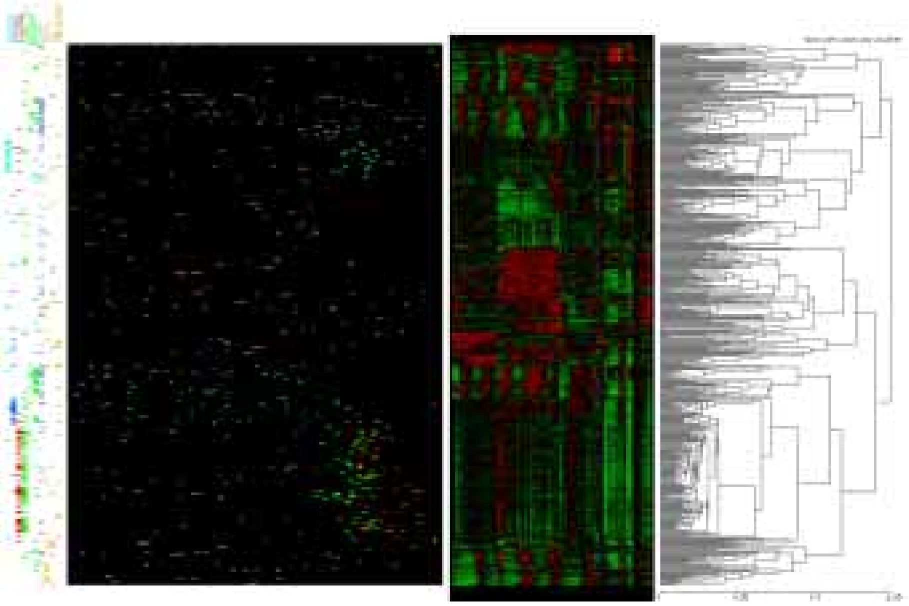


Figure 7.3: The combined visualization of microarray gene expression data clustering (right), the respective upstream sequences (middle), and several different patterns and their occurrences on these upstream sequences (left, middle). This illustrates that PATMATCH, in combination with EPCLUST and SPEXS, can uncover motifs that are specific to certain gene expression clusters and are often also conserved relative to the ORF start position. Note the grouping of motifs near the bottom right corner of the left images, corresponding to the highly co-expressed genes in the middle. Hierarchical clustering verifies these findings. The significant sequence patterns visualised each by a different colour code are from left to right: GGTGGCAA, CCGTACA, G.GATGAG, TGAAA..TTT, CGCGAAAA, ACGCG, ACCAGC, CGG...........CCG, TGA[CG]TCA.

Sequence Logos

The motif-matching regions extracted from PATMATCH can be submitted to EP:SEQLOGO, a tool that creates position weight matrices and respective sequence logos from the aligned or unaligned sequences. Alignments are made by looking for conserved motifs, if necessary. The position matrix is then formed by taking counts of occurrence of each nucleotide in each of the motif's positions within the matching sequence regions.

To test the goodness of weight matrices derived in such a way, we used the tool ScanACE (Hughes *et al.* 2000) with default options to match the generated position weight matrix against the sequences in the cluster as well as against all the upstream sequences, to obtain comparable statistics for top three motifs above. The results are summarized in Figure 7.4.

Assessing the quality of the motifs

By visual inspection it seems that most sequences have occurrences of both patterns. The query `G.GATGAG.T W/40 TG.AAA.TTT` (pattern `G.GATGAG.T` within at most 40bp from `TG.AAA.TTT`) performed against the upstream sequences of all yeast genes shows that there are only 52 yeast genes that have both of these motifs within 40bp from each other in their upstream sequences. See Figure 7.2 for the query results, showing also the gene expression profiles for these 52 genes. We can hypothesize that the presence of both motifs together determines the majority of the gene expression responses for this set of genes, including the YGR128C (Figure 7.2). This has also been suggested previously by Pilpel *et al.* (Pilpel, Sudarsanam, & Church 2001), who report strong correlation between these two motifs during the cell cycle, sporulation, heat shock, and DNA-damage experiments.

7.3 Integration of Expression Profiler to public microarray databases

ArrayExpress is a public repository for microarray gene expression data housed in the EBI (see `http://www.ebi.ac.uk/arrayexpress/`). It can accommodate microarray design descriptions, experiment annotations and experiment results, satisfying the requirements posed by MIAME (Minimal Information About Microarray Experiments, (Brazma *et al.* 2001)). An interface has been implemented that allows Expression Profiler to import experiment results from ArrayExpress for analysis.

ArrayExpress is based on the MAGE object model, which is a standard model for microarray gene expression experiment domain; See `http://www.mged.org/` and (Brazma *et al.* 2002) for the details. In MAGE the experimental data are represented as 3-dimensional matrices, with every measurement having a corresponding:

- microarray design element (feature or group of features);
- bioassay, *i.e.* , hybridization (for data coming out of feature extraction software) or data transformation (for derived data)
- quantitation type, *i.e.* , intensity, ratio, present/absent call etc.

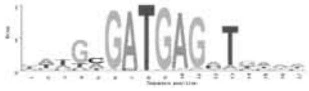
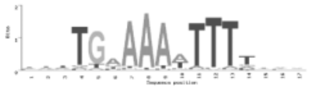
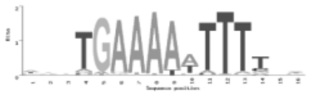
Pattern	In cluster	Total nr	Ratio	Probability
G.GATGAG.T	39	193	13.24	2.490e-33
TG.AAA.TTT	53	538	6.46	3.248e-31
TGAAAA.TTT	45	333	8.86	1.699e-31
-1:G.GATGAG.T	61	1295	3.09	1.441e-19
-1:TG.AAA.TTT	89	3836	1.52	6.126e-12
-1:TGAAAA.TTT	76	2190	2.27	1.654e-18
	62	395	10.29	6.909e-50
	83	1227	4.43	1.703e-44
	69	593	7.63	1.585e-48

Figure 7.4: Putative yeast transcription factor binding site motifs and statistics of their occurrences in the 600bp ORF upstream regions. The second column shows the number of sequences in the cluster that match the pattern. The third column shows the total number of upstream sequences matched by the motif (including the cluster). The last two columns show the relative frequency of matches in the cluster vs. the genome, and the probability of such events, respectively. The results are grouped by matching patterns exactly as found by SPEXS (the top three), matching the same patterns with one mismatch (the middle three), and finally, by matching of position weight matrices derived using approximate matching within the cluster (represented by the sequence logos). Note that with one mismatch, the number of motif occurrences increases dramatically, creating also many false matches. When approximate matching within the sequences of one cluster is used to create position weight matrices, false positive matches are reduced and the probability scores are actually improved over those of the exact patterns, discovered by SPEXS.

Expression Profiler operates with data as two-dimensional matrices. The design element dimension is transferred one-to-one from ArrayExpress to Expression Profiler. In order to specify the other dimension, the user can select which bioassays (s)he wants to analyze and which quantitation types should be used. The simplest case is to select all bioassays in the experiment and just one quantitation type, typically log-ratio (see Figure 7.5). However, it is possible to select more than one quantitation type and not all bioassays.

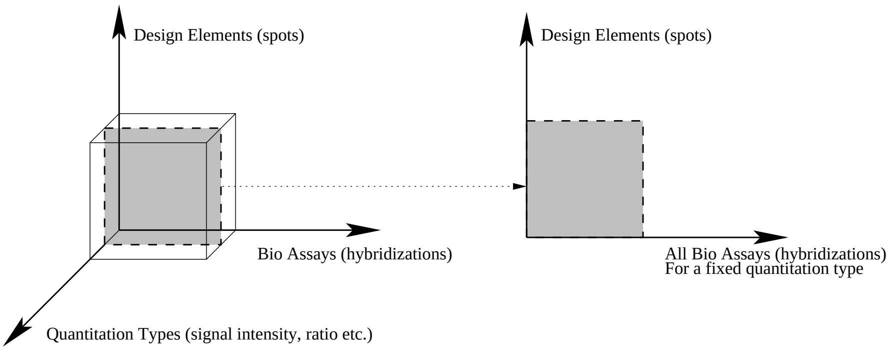

Design Elements (spots)
Bio Assays (hybridizations)
Quantitation Types (signal intensity, ratio etc.)
Design Elements (spots)
All Bio Assays (hybridizations)
For a fixed quantitation type

Figure 7.5: The schematic view of selecting a subset of data from the ArrayExpress into Expression Profiler for the analysis.

7.4 Development challenges for Expression Profiler

There are many standalone programs that do some combinations of what Expression Profile offers. Expression Profiler stands out from these by providing a simple web-based interface to a sophisticated, integrated collection of modules that span the needs of expression data analysis from initial data retrieval and filtering, to data manipulation, and various means of analysis and cross-comparison.

An important distinguishing feature of this collection of tools is their flexibility: the tools are implemented as server-side components, which means that they can easily be scaled up to suit heavier computational means, they can be used concurrently by multiple users in a collaborative fashion, and, perhaps most significantly, every tool in Expression Profiler was designed with extensibility in mind. When new algorithms and databases become available, they can be added to Expression Profiler seamlessly, fitting neatly into the existing framework.

Furthermore, as we are designing Expression Profiler for future use, it is becoming more and more apparent that there is a need for the support of a canonical, open interface for the exchange of experimental expression data between software tools. We tackle this by providing a number of simple ways for data import/export and also by working closely with the ArrayExpress database team on implementing the support for the MAGE-ML data format and linking directly to this standard repository of expression experiment data.

It is already evident that expression data analysis is not a simple matter of following a cookbook recipe in a step-by-step fashion. There are no algorithms that lead to a full, unambiguous, analytical description of the experiment - instead the user should be able to choose from an overwhelmingly large number of methods and specialized algorithms and, importantly, often by way of experiment, arrive at conclusions, and then validate them through further analysis yet. We hope to continue to extend Expression Profiler in such a way that the numbers of available approaches will be accessible to users at every level of bioinformatical sophistication and that it will guide them through the data analysis maze. Future releases of Expression Profiler will not only concentrate on improving and adding analytical algorithms but also on improving the user interface, developing a collaborative environment and offering ways to integrate disparate but complementary data for analysis.

Chapter 8

Conclusions

In this thesis we have considered application-driven algorithm design where we are often first presented with the challenge to analyze real-world data without the tools for doing so. The challenge may come from the biologists or from the bioinformatics researchers themselves when thinking about what one could do in principle with the available data. To solve the challenge, the computational method needs to be designed and implemented, the analysis of the data needs to be performed, and the appropriateness of the developed methods verified. The developed algoriths need also to be analyzed from the computational point of view (correctness and complexity) and made as efficient as possible (with the reasonable effort).

We have demonstrated in this thesis how theoretical and practical results of algorithm design can be applied in the fields of molecular biology and bioinformatics. The software tools SPEXS and Expression Profiler have been developed and used for solving practical data analysis tasks in order to solve real biological problems.

The future challenges include the better integration of the tools into graphical user interfaces for average biologists' use, as well as into analysis pipelines. These pipelines should allow to combine different analysis methods so that the results of one analysis are directly submitted as an input for another. This way large sets of data can be analyzed without human intervention at every single step as would be required in most graphical user interfaces. The analysis steps can also be repeated when more data is available or some methods have been improved.

The results of the analysis of biological data need to be disseminated to the biolocal research community in the form of new information and knowledge. This requires substantial effort on tool, database, and interface development, not to mention the publications in the relevant publications aimed for people with different research background.

For promoter analysis and prediction of putative gene regulatory elements, one of the main applications described in this thesis, we see the real challenges in applying the developed methods to higher organisms, including humans. The problem is at first perhaps not even so much algorithmic (due the larger data sets) but instead biological. For example, how to collect biologically meaningful sets of sequences for the promoter analysis. Current gene prediction methods relying either on ab initio methods or mapping of EST or protein sequences back to genomic DNA are not guaranteed to identify even the first exons or 5' untranslated regions (UTR's) that would help to identify the actual transcription start sites and putative promoter regions. Also, the regulatory signals are not present only in upsream sequences but also within the genes (exons and introns), and in downstream sequences. Also, it is known that some of the signals can be quite far from the actual genes. This area of research, however, remains to be a hot topic in studies aiming at interpreting the human DNA sequences.

References

Aasland, R., and Stewart, F. A. 1995. The chromo shadow domain, a second chromo domain in heterchromatin-binding protein 1, HP1. *Nucleic Acids Research* 23:3168–3173.

Apostolico, A.; Bock, M. E.; Lonardi, S.; and Xu, X. 2000. Efficient detection of unusual words. *Journal of Computational Biology* 7(1/2):71–94.

Apostolico, A.; Bock, M. E.; and Lonardi, S. 2002. Monotony of surprise and large-scale quest for unusual words (extended abstract). In E.W.Myers; Hannenhalli, S.; Istrail, S.; Pevzner, P.; and Waterman, M., eds., *International Conference on Research in Computational Molecular Biology (RECOMB)*, 22–31. Washington, DC: ACM.

Attwood, T. K.; Croning, M. D.; Flower, D. R.; Lewis, A. P.; Mabey, J. E.; Scordis, P.; Selley, J. N.; and Wright, W. 2000. PRINTS-S: the database formerly known as PRINTS. *Nucleic Acids Res* 28(1):225–227.

Baeza-Yates, R. A., and Gonnet, G. H. 1990. All-against-all sequence matching. Report, Department of Computer Science, Universidad de Chile.

Baeza-Yates, R. A., and Gonnet, G. H. 1999. A fast algorithm on average for all-against-all sequence matching. In *6th Internat. Symp. on String Processing and Information Retrieval (SPIRE/CRIWG)*, 16–23. Los Alamitos, California: IEEE Computer Society.

Bailey, T. L., and Elkan, C. 1995. Unsupervised learning of multiple motifs in biopolymers using expectation maximization. *Machine Learning* 21:51–83.

Bairoch, A. 1992. PROSITE: a dictionary of sites and patterns in proteins. *Nucleic Acids Research* 20:2013–2018.

Barash, Y.; Bejerano, G.; and Friedman, N. 2001. A simple hyper-geometric approach for discovering putative transcription factor binding sites. In *Algorithms in Bioinformatics*, volume 2149, 17pp.

Barton, G. J., and Livingstone, C. D. 1993. Protein sequence alignments: a strategy for the hierarchical analysis of residue conservation. *Comput Appl Biosci* 9(6):745–756.

Bieganski, P.; Riedi, J.; Carlis, J. V.; and Retzel, E. F. 1994. Generalized suffix trees for biological sequence data: Applications and implementation. In Hunter, L., ed., *Proceedings of the 27th Annual Hawaii International Conference on System Sciences. Volume 5 : Biotechnology Computing*, 35–44. Los Alamitos, CA, USA: IEEE Computer Society Press.

Bockaert, J., and Pin, J. P. 1999. Molecular tinkering of G protein-coupled receptors: an evolutionary success. *EMBO Journal* 18(7):1723–1729.

Bourne, H. R. 1997. How receptors talk to trimeric G proteins. *Curr Opin Cell Biol* 9(2):134–142.

Brazma, A., and Vilo, J. 2000. Gene expression data analysis. *FEBS Letters* 480:17–24.

Brazma, A.; Jonassen, I.; Ukkonen, E.; and Vilo, J. 1996. Discovering patterns and subfamilies in biosequences. In States, D. J.; Agarwal, P.; Gaasterland, T.; Hunter, L.; and Smith, R., eds., *Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology*, 34–43. Menlo Park: AAAI Press.

Brazma, A.; Vilo, J.; Ukkonen, E.; and Valtonen, K. 1997. Data mining for regulatory elements in yeast genome. In Gaasterland, T.; Karp, P.; Karplus, K.; Ouzounis, C.; Sander, C.; and Valencia, A., eds., *Proceedings of the 5th International Conference on Intelligent Systems for Molecular Biology (ISMB97)*, 65–74. Menlo Park: AAAI Press.

Brazma, A.; Jonassen, I.; Eidhammer, I.; and Gilbert, D. 1998a. Approaches to automatic discovery of patterns in biosequences. *Journal of Computational Biology* 5(2):277–304.

Brazma, A.; Jonassen, I.; Vilo, J.; and Ukkonen, E. 1998b. Predictin gene regulatory elements *in silico* on a genomic scale. *Genome Research* 8(11):1202–1215.

Brazma, A.; Jonassen, I.; Vilo, J.; and Ukkonen, E. 1998c. Pattern discovery in biosequences. In Honavar, V., and Slutzki, G., eds., *Proceedings of Fourth International Colloquium on Grammatical Inference (ICGI-98)*, volume 1433, 257–270. Ames, Iowa, USA: Springer.

Brazma, A.; Robinson, A.; Cameron, G.; and Ashburner, M. 2000. One stop shop for microarray data. *Nature* 403:699–700.

Brazma, A.; Hingamp, P.; Quackenbush, J.; Sherlock, G.; Spellman, P.; Stoeckert, C.; Aach, J.; Ansorge, W.; Ball, C. A.; Causton, H. C.; Gaasterland, T.; Glenisson, P.; Holstege, F. C.; Kim, I. F.; Markowitz, V.; Matese, J. C.; Parkinson, H.; Robinson, A.; Sarkans, U.; Schulze-Kremer, S.; Stewart, J.; Taylor, R.; Vilo, J.; and Vingron, M. 2001. Minimum information about a microarray experiment (MIAME) toward standards for microarray data. *Nature Genetics* 29(4):365–371.

Brazma, A.; Sarkans, U.; Robinson, A.; Vilo, J.; Vingron, M.; Hoheisel, J.; and Fellenberg, K. 2002. Microarray data representation, annotation and storage.

In *Advances in Biochemical Engineering/Biotechnology,* volume Vol. 77 (Chip Technology), 113–139. Berlin: Springer-Verlag.

Brazma, A.; Ukkonen, E.; and Vilo, J. 1996. Discovering unbounded unions of regular pattern languages from positive examples. In Asano, T.; Igarashi, Y.; Nagamochi, H.; Miyano, S.; and Suri, S., eds., *Proceedings of 7th International symposium, ISAAC'96*, 95–104. Osaka, Japan: Springer.

Bussemaker, H. J.; Li, H.; and Siggia, E. D. 2000a. Building a dictionary for genomes: identification of presumptive regulatory sites by statistical analysis. *Proc. Natl. Acad. Sci. USA* 97:10096–10100.

Bussemaker, H. J.; Li, H.; and Siggia, E. D. 2000b. Regulatory element detection using a probabilistic segmentation model. In *Proc. of Eighth International Conference on Intelligent Systems for Molecular Biology (ISMB-2000)*, 67–74. La Jolla, California: AAAI Press.

Celis, J. E.; Kruhoffer, M.; Gromova, I.; Frederiksen, C.; M, M. O.; Thykjaer, T.; Gromov, P.; Yu, J.; Palsdottir, H.; Magnusson, N.; and Orntoft, T. F. 2000. Gene expression profiling: monitoring transcription and translation products using DNA microarrays and proteomics. *FEBS Lett.* 480(1):2–16.

Cho, R. J.; Campbell, M. J.; Winzeler, E. A.; Steinmetz, L.; Conway, A.; Wodicka, L.; Wolfsberg, T. G.; Gabrielian, A. E.; Landsman, D.; Lockhart, D. J.; and Davis, R. W. 1998. A genome wide transcriptional analysis of gene expression of the mitotic cell cycle. *Molecular Cell* 2:65–73.

Chomsky, N. 1956. Three models for the description of language. *IRE Trans. on Information Theory* 2:113–124. Also in 'Readings in mathematical psychology', R.D. Luce, R. Bush, and E. Galanter (eds.), New York: Wiley, pp. 105-124, 1965.

Chomsky, N. 1959. On certain formal properties of grammars. *Information and Control* 2:137–167.

Chu, S.; DeRisi, J. L.; Eisen, M.; Mulholland, J.; Botstein, D.; Brown, P. O.; and Herskowitz, I. 1998. The transcription program of sporulation in budding yeast. *Science* 282:699–705.

DeRisi, J. L.; Iyer, V. R.; and Brown, P. O. 1997. Exploring the metabolic and genetic control of gene expression on a genomic scale. *Science* 278:680–686.

Dopazo, J.; Zanders, E.; Dragoni, I.; Amphlett, G.; and Falciani, F. 2001. Methods and approaches in the analysis of gene expression data. *J Immunol Methods* 250(1–2):93–112.

Eisen, M.; Spellman, P. T.; Botstein, D.; and Brown, P. O. 1998. Cluster analysis and display of genome-wide expression patterns. *Proceedings of National Academy of Science USA* 95:14863–14867.

Everitt, B. S.; Landau, S.; and Leese, M. 2001. *Cluster Analysis*. London: Arnold, 4th edition edition.

Fleischmann, W.; Moller, S.; Gateau, A.; and Apweiler, R. 1999. A novel method for automatic functional annotation of proteins. *Bioinformatics* 15(3):228–233.

Gether, U. 2000. Uncovering molecular mechanisms involved in activation of G protein-coupled receptors. *Endocr Rev* 21(1):90–113.

Giegerich, R., and Kurtz, S. 1995. A comparison of imperative and purely functional suffix tree constructions. *Science of Computer Programming* 25(2–3):187–218.

Giegerich, R.; Kurtz, S.; and Stoye, J. 1999. Efficient implementation of lazy suffix trees. In *Proceedings of the Third Workshop on Algorithmic Engineering (WAE99)*, volume 1668, 30–42. Springer Verlag.

Goffeau, A.; Barrell, B. G.; Bussey, H.; Davis, R. W.; Dujon, B.; Feldmann, H.; Galibert, F.; Hoheisel, J. D.; Jacq, C.; Johnston, M.; Louis, E. J.; Mewes, H. W.; Murakami, Y.; Philippsen, P.; Tettelin, H.; and Oliver, S. G. 1996. Life with 6000 genes. *Science* 274:546–567.

Gonnet, G. H., and Knecht, L. 1996. *A tutorial introduction to computational biochemistry using Darwin.* Informatik Eidgenossische Technische Hochschule, Zurich, Switzerland.

Gusfield, D.; Landau, G.; and Schieber, B. 1992. An efficient algorithm for all the pairs suffix-prefix problem. *Inf. Process. Lett.* 41(4):181–185.

Gusfield, D. 1997. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge: Cambridge University Press.

Hart, R.; Royyuru, A. K.; Stolovitzky, G.; and Califano, A. 2000a. Systematic and automated discovery of patterns in PROSITE families. In Shamir, R.; Miyano, S.; Istrail, S.; Pevzner, P.; and Waterman, M., eds., *The Fourth Annual International Conference on Computational Molecular Biology RECOMB-2000*, 147–154. Tokyo, Japan: ACM Press.

Hart, R.; Royyuru, A. K.; Stolovitzky, G.; and Califano, A. 2000b. Systematic and fully automatic identification of protein sequence patterns. *J. Comp. Biol.* 7(3/4):585–600.

Hegde, P.; Qi, R.; Abernathy, K.; Gay, C.; Dharap, S.; Gaspard, R.; Hughes, J. E.; Snesrud, E.; Lee, N.; and Quackenbush, J. 2000. A concise guide to cDNA microarray analysis. *Biotechniques* 29(3):548–556.

Herrero, J.; Valencia, A.; and Dopazo, J. 2001. A hierarchical unsupervised growing neural network for clustering gene expression patterns. *Bioinformatics* 17:126–136.

Hertz, G. Z., and Stormo, G. D. 1994. Identification of consensus patterns in unaligned DNA and protein sequences: a large-devi ation statistical basis for penalizing gaps. In *Proc. of Third International Conference on Bioinformatics and Genome Research*.

Hertz, G. Z., and Stormo, G. D. 1999. Identifying dna and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics* 15(7-8):563–577.

Hoffmann, K.; Bucher, P.; Falquet, L.; and Bairoch, A. 1999. The PROSITE database, its status in 1999. *Nucleic Acids Research* 27(1):215–219.

Hughes, J. D.; Estep, P. W.; Tavazoie, S.; and Church, G. M. 2000. Computational identification of cis-regulatory elements associated with groups of functionally related genes in Saccharomyces cerevisiae. *Journal of Molecular Biology* 296(5):1205–1214.

Hunt, E.; Atkinson, M. P.; and Irving, R. W. 2001. A database index to large biological sequences. In *Proc. 27th Conf. on Very Large Databases*, 139–148. Morgan Kaufmann. ISBN 1-55860-804-4.

Iyer, V. R.; Horak, C. E.; Scafe, C. S.; Botstein, D.; Snyder, M.; and Brown, P. O. 2001. Genomic binding sites of the yeast cell-cycle transcription factors SBF and MBF. *Nature* 409(6819):553–558.

Jensen, L. J., and Knudsen, S. 2000. Automatic discovery of regulatory patterns in promoter regions based on whole cell expression data and functional annotation. *Bioinformatics* 16(4):326–333.

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

Jonassen, I.; Eidhammer, I.; and Taylor, W. R. 1999. Discovery of local packing motifs in protein structures. *Proteins: Structure, Function, and Genetics* 34(2):206–219.

Jonassen, I. 1997. Efficient discovery of conserved patterns using a pattern graph. *Comput. Appl. Biosci.* 13:509–522.

Kärkkäinen, J. 1995. Suffix cactus: A cross between suffix tree and suffix array. In Galil, Z., and Ukkonen, E., eds., *Combinatorial Pattern Matching, 6th Annual Symposium*, volume 937 of *Lecture Notes in Computer Science*, 191–204. Espoo, Finland: Springer.

Kemmeren, P.; van Berkum, N. L.; Vilo, J.; Bijma, T.; Donders, R.; Brazma, A.; and Holstege, F. C. 2002. Protein interaction verification and functional annotation by integrated analysis of genome-scale data. *Molecular Cell* 9(5):1133–1143.

Kivinen, J.; Mannila, H.; Ukkonen, E.; and Vilo, J. 1994. An algorithm for learning hierarchical classifiers. In Bergadano, F., and de Raedt, L., eds., *Proceedings of the European Conference on Machine Learning*, volume 784 of *LNAI*, 375–378. Berlin: Springer.

Kohonen, T. 1997. *Self-Organizing Maps*. Berlin, Heidelberg: Springer. (Second Extended Edition 1997).

Kruglyak, S., and Tang, H. 2001. A new estimator of significance of correlation in time series data. *Journal of Computational Biology* 8(5):463–470.

Lawrence, C. E.; Altschul, S. F.; Boguski, M. S.; Liu, J. S.; Neuwald, A. F.; and Wootton, J. C. 1993. Detecting subtle sequence signals: a gibbs sampling strategy for multiple alignment. *Science* 262:208–214.

Legendre, P., and Legendre, L. 1998. *Numerical Ecology*. Developments in Environmental Modelling. Elsevier.

Levy, S.; Compagnoni, L.; Myers, E. W.; and Stormo, G. D. 1998. Xlandscape: the graphical display of word frequencies in sequences. *Bioinformatics* 14(1):74–80.

Li, M., and Vitanyi, P. 1993. *An introduction to Kolmogorov complexity and its applications*. Texts and monographs in computer science. New York: Springer-Verlag.

Liang, M. F. 1983. *Word Hy-phen-a-tion by Com-put-er*. Ph.D. Dissertation, Stanford University.

Lonardi, S. 2001. *Global Detectors of Unusual Words: Design, Implementation, and Applications to Pattern Discovery in Biosequences*. Ph.D. Dissertation, Department of Computer Science, Purdue University.

Manber, U., and Myers, G. 1990. Suffix arrays: A new method for on–line string searches. In *Proceedings of the First Annual ACM–SIAM Symposium on Discrete Algorithms*, 319–327. SIAM.

Mannhaupt, G.; Schnall, R.; Karpov, V.; Vetter, I.; and Feldmann, H. 1999. Rpn4p acts as a transcription factor by binding to PACE, a nonamer box found upstream of 26S proteasomal and other genes in yeast. *FEBS Lett* 450(1–2):27–34.

Mannila, H.; Toivonen, H.; and Verkamo, A. I. 1994. Efficient algorithms for discovering association rules. In Fayyad, U. M., and Uthurusamy, R., eds., *AAAI*

Workshop on Knowledge Discovery in Databases (KDD-94), 181–192. Seattle, Washington: AAAI Press.

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.

Marsan, L., and Sagot, M.-F. 2000. Extracting structured motifs using a suffix tree – Algorithms and application to promoter consensus identification. In Shamir, R.; Miyano, S.; Istrail, S.; Pevzner, P.; and Waterman, M., eds., *The Fourth Annual International Conference on Computational Molecular Biology RECOMB-2000*, 210–219. Tokyo, Japan: ACM Press.

McCreight, E. M. 1976. A space–economical suffix tree construction algorithm. *Journal of the ACM* 23:262–272.

Moller, S.; Leser, U.; Fleischmann, W.; and Apweiler, R. 1999. EDIT-toTrEMBL: a distributed approach to high-quality automated protein sequence annotation. *Bioinformatics* 15(3):219–227.

Möller, S.; Croning, M. D.; and Apweiler, R. 2001. Evaluation of methods for the prediction of membrane spanning regions. *Bioinformatics* 17(7):646–653.

Möller, S.; Croning, M. D.; and *et al.* . 2001. 7TMHMM. *(Unpublished manuscript)*.

Möller, S.; Vilo, J.; and Croning, M. D. 2001. Prediction of the coupling specificity of G protein coupled receptors to their G proteins. *Bioinformatics* 17 Supplement (ISMB 2001):S174–81.

Morrison, D. R. 1968. PATRICIA - practical algorithm to retrieve information coded in alphanumeric. *Jrnl. A.C.M.* 15(4):514–534.

Neuwald, A. F.; Liu, J. S.; and Lawrence, C. E. 1995. Gibbs motif sampling: Detection of bacterial outer membrane protein repeats. *Protein Science* 4:1618–1632.

Ohler, U., and Niemann, H. 2001. Identification and analysis of eukaryotic promoters: recent computational approaches. *Trends in Genetics* 17(2):56–60.

Palin, K.; Ukkonen, E.; Brazma, A.; and Vilo, J. 2002. Correlating gene promoters and expression in gene disruption experiments. *Bioinformatics (European Conference on Computational Biology 2002)* 18:S172–180.

Paro, R., and Hogness, D. H. 1991. The polycomb protein shares a homologous domain with a heterochromatin-associated protein of drosophila. In *Proc. Natl. Acad. Sci. USA*, 263–267.

Pevzner, P., and Sze, S.-H. 2000. Combinatorial approaches to finding subtle signals in DNA sequences. In *Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology (ISMB),* 269–278. AAAI Press.

Pilpel, Y.; Sudarsanam, P.; and Church, G. M. 2001. Identifying regulatory networks by combinatorial analysis of promoter elements. *Nature Genetics* 29:153–159.

Quackenbush, J. 2001. Computational analysis of microarray data. *Nature Reviews Genetics* 2(6):418–427.

Ren, B.; Robert, F.; Wyrick, J. J.; Aparicio, O.; Jennings, E. G.; Simon, I.; Zeitlinger, J.; Schreiber, J.; Hannett, N.; Kanin, E.; Volkert, T. L.; Wilson, C. J.; Bell, S. P.; and Young, R. A. 2000. Genome-wide location and function of dna binding proteins. *Science* 290(5500):2306–2309.

Rigoutsos, I., and Floratos, A. 1998a. Motif discovery in biological sequences without alignment or enumeration. In Istrail, S.; Pevzner, P.; and Waterman, M., eds., *Proceedings of the 2nd Annual International Conference on Computational Molecular Biology (RECOMB-98)*, 221–227. New York: ACM Press.

Rigoutsos, I., and Floratos, A. 1998b. Combinatorial pattern discovery in biological sequences: the TEIRESIAS algorithm. *Bioinformatics* 14(1):55–67.

Rissanen, J. 1978. Modeling by the shortest data description. *Automatica-J.IFAC* 14:465–471.

Rocke, E., and Tompa, M. 1998. An algorithm for finding novel gapped motifs in DNA sequences. In Istrail, S.; Pevzner, P.; and Waterman, M., eds., *Proceedings of the 2nd Annual International Conference on Computational Molecular Biology (RECOMB-98)*, 228–233. New York: ACM Press.

Roth, F. R.; Hughes, J. D.; Estep, P. E.; and Church, G. M. 1998. Finding DNA regulatory motifs within unaligned non-coding sequences clustered by whole-genome mRNA quantitation. *Nature Biotechnology* 16(10):939–945.

Rousseeuw, P. J. 1987. Silhouettes: A graphical aid to the interpretations and validation of cluster analysis. *Journal of Computational and Applied Mathematics* 20:53–65.

Rubin, L. A., and Rubin, E. M. 2001. Genomic strategies to identify mammalian regulatory sequences. *Nature Reviews Genetics* 2(2):100–109.

Rust, A. G.; Mongin, E.; and Birney, E. 2002. Genome annotation techniques: new approaches and challenges. *Drug Discovery Today* 7:S70–S76.

Saitou, N., and Nei, M. 1987. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution* 4(4):406–425.

Scherf, M.; Klingenhoff, A.; Frech, K.; Quandt, K.; Schneider, R.; Grote, K.; Frisch, M.; Durner, V.; Seidel, A.; Werner, R.; and Werner, T. 2001. First pass annotation of promoters on human chromosome 22. *Genome Research* 11(3):333–340.

Scherf, M.; Klingenhoff, A.; and Werner, T. 2000. Highly specific localization of promoter regions in large genomic sequences by promoterinspector: a novel context analysis approach. *J Mol Biol* 297(3):599–606.

Schneider, T. D., and Stephens, R. M. 1990. Sequence logos: A new way to display consensus sequences. *Nucleic Acids Res.* 18:6097–6100.

Schneider, T. D.; Ehrenfeucht, A.; Stormo, G. D.; and Gold, L. 1986. Information content of binding sites on nucleotide sequences. *Journal of Molecular Biology* 188:415–431.

Schoneberg, T.; Schultz, G.; and Gudermann, T. 1999. Structural basis of G protein-coupled receptor function. *Mol Cell Endocrinol* 151(1-2):181–193.

Searls, D. B. 1992. The linguistics of DNA. *American Scientist.* 80(6):579–591.

Shimozono, S.; Shionohara, A.; Shinohara, T.; Miyano, S.; Kuhara, S.; and Arikawa, S. 1993. Finding alphabet indexing for decision trees over regular patterns: An approach to bioinformatical knowledge acquisition. In *Proceedings of the 26th Annual Hawaii International Conference on System Sciences*, 763–772. Los Alamitos, CA, USA: IEEE Computer Society Press.

Simon, M. I.; Strathmann, M. P.; and Gautam, N. 1991. Diversity of G proteins in signal transduction. *Science* 252(5007):802–808.

Sinha, S., and Tompa, M. 2000. A statistical method for finding transcription factor binding sites. In *Proc. of Eighth International Conference on Intelligent Systems for Molecular Biology (ISMB-2000)*, volume 8, 344–354. La Jolla, California: AAAI Press.

Spellman, P. T.; Sherlock, G.; Zhang, M.; Iyer, V. R.; Anders, K.; Eisen, M.; Brown, P. O.; Botstein, D.; and Futcher, B. 1998. Comprehensive identification of cell cycle-regulated genes of the yeast *saccharomyces cerevisiae* by microarray hybridization. *Molecular Biology of Cell* 9:3273.

Stormo, G. D. 2000. Dna binding sites: representation and discovery. *Bioinformatics* 16(1):16–23.

Tavazoie, S.; Hughes, D.; Campbell, M. J.; Cho, R. J.; and Church, G. M. 1999. Systematic determination of genetic network architecture. *Nature Genetics* 22:281–285.

The Chipping Forecast. 1999. *Nature Genetics* 21(1).

The Gene Ontology Consortium. 2000. Gene ontology: tool for the unification of biology. *Nature Genetics* 25:25–29.

Thompson, J.; Higgins, D. G.; and Gibson, T. J. 1994. Clustal W: improving the sensitivity of progressive multiple seq uence alignment through sequence weighting, position-specific gap penalties and weigh t matrix choice. *Nucleic Acids Research* 22:4673–4680.

Turi, T. G., and Loper, J. C. 1992. Multiple regulatory elements control expression of the gene encoding the *saccharomyces cerevisiae* cytochrome P450, lanosterol 14 alpha-demethylase (ERG11). *Journal of Biological Chemistry* 267:2046–2056.

Ukkonen, E. 1993. Approximate string-matching over suffix trees. In Apostolico, A.; Crochemore, M.; Galil, Z.; and Manber, U., eds., *Combinatorial Pattern Matching, 4th Annual Symposium*, volume 684 of *Lecture Notes in Computer Science*, 228–242. Padova, Italy: Springer.

Ukkonen, E. 1995. On-line construction of suffix trees. *Algorithmica* 14:249–260.

van Helden, J.; André, B.; and Collado-Vides, J. 1998. Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequen cies. *Journal of Molecular Biology* 281(5):827–842.

Vanet, A.; Marsan, L.; and Sagot, M.-F. 1999. Promoter sequences and algorithmical methods for identifying them. *Research in Microbiology* 150:779–799.

Vilo, J., and Kivinen, K. 2001. Regulatory sequence analysis: application to interpretation of gene expression. *European Neuropsychopharmacology* 11(6):399–411.

Vilo, J.; Brazma, A.; Jonassen, I.; Robinson, A.; and Ukkonen, E. 2000. Mining for putative regulatory elements in the yeast genome using gene expression data. In *Proc. of Eighth International Conference on Intelligent Systems for Molecular Biology (ISMB-2000)*, volume 8, 384–394. La Jolla, California: AAAI Press.

Vilo, J.; Kapushesky, M.; Kemmeren, P.; Sarkans, U.; and Brazma, A. 2003. Expression Profiler. In Parmigiani, G.; Garrett, E.; Irizarry, R.; and Zeger, S. L., eds., *The analysis of gene expression data: methods and software*. Springer. To appear.

Vilo, J.; Kapushesky, M.; and Kemmeren, P. 2002. Expression Profiler: tools and documentation *http://ep.ebi.ac.uk/*.

Wagner, A. 1999. Genes regulated cooperatively by one or more transcription factors and their identification in whole eukaryotic genomes. *Bioinformatics* 15(10):776–7784.

Wang, J.; Shapiro, B.; and Shasha, D., eds. 1999. *Pattern DiscoVery in Biomolecular Data: Tools, Techniques and Applications*. New York, NY: Oxford University Press.

Weiner, P. 1973. Linear pattern matching algorithms. In *Conference Record, IEEE* 14^{th} *Annual Symposium on Switching and Automata Theory*, 1–11.

Werner, T. 1999. Models for prediction and recognition of eukaryotic promoters. *Mammalian Genome* 10(2):168–175.

Werner, T. 2001. Cluster analysis and promoter modelling as bioinformatics tools for the identification of target genes from expression array data. *Pharmacogenomics* 2(1):25–36.

Wess, J. 1999. Molecular basis of receptor/g-protein-coupling selectivity. *Pharmacol Ther* 80(3):231–264.

Wingender, E.; Dietze, P.; Karas, H.; and Knuppel, R. 1996. TRANSFAC: a database of transcriptional factors and their DNA binding sites. *Nucleic Acids Research* 24:238–241.

Winzeler, E. A.; Shoemaker, D. D.; Astromoff, A.; Liang, H.; Anderson, K.; Andre, B.; ; Bangham, R.; Benito, R.; Boeke, J. D.; Bussey, H.; Chu, A. M.; Connelly, C.; Davis, K.; ; Dietrich, F.; Dow, S. W.; Bakkoury, M.; Foury, F.; Friend, S. H.; Gentalen, E.; ; Giaever, G.; Hegemann, J. H.; Jones, T.; Laub, M.; Liao, H.; Davis, R. W.; and et al. 1999. Functional characterization of the s. cerevisiae genome by gene deletion and parallel analysis. *Science* 285(5429):901–906.

Wolfertstetter, F.; Frech, K.; Herrmann, G.; and Werner, T. 1996. Identification of functional elements in unaligned nucleic acid sequences by a novel tuple search algorithm. *Computer Applications in Biosciences* 12:71–80.

Zhang, M. Q. 1998. Identification of human gene core promoters in silico. *Genome research* 8(3):319–326.

Zhang, M. Q. 1999. Promoter analysis of co-regulated genes in the yeast genome. *Computers and Chemistry* 23:233–250.

Zhu, J., and Zhang, M. Q. 1999. SCPD: a promoter database of the yeast *saccharomyces cerevisiae*. *Bioinformatics* 15(7/8):607–611.

TIETOJENKÄSITTELYTIETEEN LAITOS
PL 26 (Teollisuuskatu 23)
00014 Helsingin yliopisto

DEPARTMENT OF COMPUTER SCIENCE
P.O. Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki, FINLAND

JULKAISUSARJA **A**

SERIES OF PUBLICATIONS **A**

Reports may be ordered from:
Department of Computer Science, Library (A 214), P.O. Box 26, FIN-00014 University of Helsinki, FINLAND.

A-1990-1 K. Pohjonen & J. Tarhio (toim./eds.): Tietojenkäsittelyopin laitoksen tutkimusraportteja 1988–89 – Research reports at the Department of Computer Science 1988–89. 27 pp.

A-1990-2 J. Kuittinen, O. Nurmi, S. Sippu & E. Soisalon-Soininen: Efficient implementation of loops in bottom-up evaluation of logic queries. 14 pp.

A-1990-3 J. Tarhio & E. Ukkonen: Approximate Boyer-Moore string matching. 27 pp.

A-1990-4 E. Ukkonen & D. Wood: Approximate string matching with suffix automata. 14 pp.

A-1990-5 T. Kerola: Qsolver – a modular environment for solving queueing network models. 15 pp.

A-1990-6 Ker-I Ko, P. Orponen, U. Schöning & O. Watanabe: Instance complexity. 24 pp.

A-1991-1 J. Paakki: Paradigms for attribute-grammar-based language implementation. 71 + 146 pp. (Ph.D. Thesis).

A-1991-2 O. Nurmi & E. Soisalon-Soininen: Uncoupling updating and rebalancing in chromatic binary search trees. 12 pp.

A-1991-3 T. Elomaa & J. Kivinen: Learning decision trees from noisy examples. 15 pp.

A-1991-4 P. Kilpeläinen & H. Mannila: Ordered and unordered tree inclusion. 22 pp.

A-1991-5 A. Valmari: Compositional state space generation. 30 pp.

A-1991-6 J. Tarhio & M. Tienari (eds.): Computer Science at the University of Helsinki 1991. 66 pp.

A-1991-7 P. Jokinen, J. Tarhio & E. Ukkonen: A comparison of approximate string matching algorithms. 23 pp.

A-1992-1 J. Kivinen: Problems in computational learning theory. 27 + 64 pp. (Ph.D. Thesis).

A-1992-2 K. Pohjonen & J. Tarhio (toim./eds.): Tietojenkäsittelyopin laitoksen tutkimusraportteja 1990–91 – Research reports at the Department of Computer Science 1990–91. 35 pp.

A-1992-3 Th. Eiter, P. Kilpeläinen & H. Mannila: Recognizing renamable generalized propositional Horn formulas is NP-complete. 11 pp.

A-1992-4 A. Valmari: Alleviating state explosion during verification of behavioural equivalence. 57 pp.

A-1992-5 P. Floréen: Computational complexity problems in neural associative memories. 128 + 8 pp. (Ph.D. Thesis).

A-1992-6 P. Kilpeläinen: Tree matching problems with applications to structured text databases. 110 pp. (Ph.D. Thesis).

A-1993-1 E. Ukkonen: On-line construction of suffix-trees. 15 pp.

A-1993-2 Alois P. Heinz: Efficient implementation of a neural net α-β-evaluator. 13 pp.

A-1994-1 J. Eloranta: Minimal transition systems with respect to divergence preserving behavioural equivalences. 162 pp. (Ph.D. Thesis).

A-1994-2 K. Pohjonen (toim./ed.): Tietojenkäsittelyopin laitoksen julkaisut 1992–93 – Publications from the Department of Computer Science 1992–93. 58 s./pp.

A-1994-3 T. Kujala & M. Tienari (eds.): Computer Science at the University of Helsinki 1993. 95 pp.

A-1994-4 P. Floréen & P. Orponen: Complexity issues in discrete Hopfield networks. 54 pp.

A-1995-1 P. Myllymäki: Mapping Bayesian networks to stochastic neural networks: a foundation for hybrid Bayesian-neural systems. 93 pp. (Ph.D. Thesis).

A-1996-1 R. Kaivola: Equivalences, preorders and compositional verification for linear time temporal logic and concurrent systems. 185 pp. (Ph.D. Thesis).

A-1996-2 T. Elomaa: Tools and techniques for decision tree learning. 140 pp. (Ph.D. Thesis).

A-1996-3 J. Tarhio & M. Tienari (eds.): Computer Science at the University of Helsinki 1996. 89 pp.

A-1996-4 H. Ahonen: Generating grammars for structured documents using grammatical inference methods. 107 pp. (Ph.D. Thesis).

A-1996-5 H. Toivonen: Discovery of frequent patterns in large data collections. 116 pp. (Ph.D. Thesis).

A-1997-1 H. Tirri: Plausible prediction by Bayesian inference. 158 pp. (Ph.D. Thesis).

A-1997-2 G. Lindén: Structured document transformations. 122 pp. (Ph.D. Thesis).

A-1997-3 M. Nykänen: Querying string databases with modal logic. 150 pp. (Ph.D. Thesis).

A-1997-4 E. Sutinen, J. Tarhio, S.-P. Lahtinen, A.-P. Tuovinen, E. Rautama & V. Meisalo: Eliot – an algorithm animation environment. 49 pp.

A-1998-1 G. Lindén & M. Tienari (eds.): Computer Science at the University of Helsinki 1998. 112 pp.

A-1998-2 L. Kutvonen: Trading services in open distributed environments. 231 + 6 pp. (Ph.D. Thesis).

A-1998-3 E. Sutinen: Approximate pattern matching with the q-gram family. 116 pp. (Ph.D. Thesis).

A-1999-1 M. Klemettinen: A knowledge discovery methodology for telecommunication network alarm databases. 137 pp. (Ph.D. Thesis).

A-1999-2 J. Puustjärvi: Transactional workflows. 104 pp. (Ph.D. Thesis).

A-1999-3 G. Lindén & E. Ukkonen (eds.): Department of Computer Science: annual report 1998. 55 pp.

A-1999-4 J. Kärkkäinen: Repetition-based text indexes. 106 pp. (Ph.D. Thesis).

A-2000-1 P. Moen: Attribute, event eequence, and event type similarity notions for data mining. 190+9 pp. (Ph.D. Thesis).

A-2000-2 B. Heikkinen: Generalization of document structures and document assembly. 179 pp. (Ph.D. Thesis).

A-2000-3 P. Kähkipuro: Performance modeling framework for CORBA based distributed systems. 151+15 pp. (Ph.D. Thesis).

A-2000-4 K. Lemström: String matching techniques for music retrieval. 56+56 pp. (Ph.D. Thesis).

A-2000-5 T. Karvi: Partially defined Lotos specifications and their refinement relations. 157 pp. (Ph.D. Thesis).

A-2001-1 J. Rousu: Efficient range partitioning in classification learning. 68+74 pp. (Ph.D. Thesis).

A-2001-2 M. Salmenkivi: Computational methods for intensity models. 145 pp. (Ph.D. Thesis).

A-2001-3 K. Fredriksson: Rotation invariant template matching. 138 pp. (Ph.D. Thesis).

A-2002-1 A.-P. Tuovinen: Object-oriented engineering ofvisual languages. 185 pp. (Ph.D. Thesis).

A-2002-2 V. Ollikainen: Simulation texhniques for disease gene localization in isolated populations. 149+5 pp. (Ph.D. Thesis).