Approaches to the automatic discovery of patterns in biosequences.*

Alvis Brāzma[†]
EMBL Outstation – Hinxton, European Bioinformatics Institute
Wellcome Trust Genome Campus, Hinxton, Cambridge CB10 1SD, UK [‡]

Inge Jonassen[†§]
Department of Informatics, University of Bergen, HIB, N5020 Bergen, Norway
phone: (+47) 55584713, fax: (+47) 55584199, email: inge@ii.uib.no

Ingvar Eidhammer
Department of Informatics, University of Bergen, HIB, N5020 Bergen, Norway

David Gilbert
Department of Computer Science, City University,
Northampton Square, London, EC1V 0HB, UK

November 5, 1997

Running head: The automatic discovery of patterns in biosequences.

Abstract

This paper surveys approaches to the discovery of patterns in biosequences and places these approaches within a formal framework that systematises the types of patterns and the discovery algorithms. Patterns with expressive power in the class of regular languages are considered, and a classification of pattern languages in this class is developed, covering the patterns that are the most frequently used in molecular bioinformatics. A formulation is given of the problem of the automatic discovery of such patterns from a set of sequences, and an analysis is presented of the ways in which an assessment can be made of the significance of the discovered patterns. It is shown that the problem is related to problems studied in the field of machine learning. The major part of this paper comprises a review of a number of existing methods developed to solve the problem and how these relate to each other, focusing on the algorithms underlying the approaches. A comparison is given of the algorithms, and examples are given of patterns that have been discovered using the different methods.

*To be published in the Journal of Computational Biology
[†]The two first authors have contributed equally to this work
[‡]Work done whilst at the University of Latvia, Riga, Latvia, and the University of Helsinki, Finland
[§]Corresponding author

Keywords: automatic discovery, bioinformatics, biosequences, machine learning, patterns.

Introduction

Recently it has become relatively cheap and easy to determine nucleotide and protein sequences, and a considerable number of sequences has been amassed, representing a total length of several hundreds of millions of symbols. Moreover there are many different databases containing such sequence data. The emphasis in genome projects has moved from the acquisition of sequence data towards the analysis of this data. The aim of the analysis is the extraction of all sorts of biological "meaning" of these sequences, for example concerning the evolutionary history of the respective macromolecules and their three-dimensional structure and function. One way of analysing the sequences is to group them into *families*, each family being a set of sequences believed to be biologically (i.e., evolutionarily, structurally or functionally) related, and for each family to try to find common features or *patterns*. In this paper we survey methods for the automatic discovery of such patterns.

Different kinds of patterns can be used for characterising sequences and the corresponding macromolecules. For proteins and protein sequences we should distinguish between *sequence patterns* and *structure patterns*. Structure patterns describe features of the three-dimensional structures of the macromolecules, and may also include information about the corresponding sequences. Sequence patterns describe pure sequence (syntactic) properties. In this paper we consider only sequence patterns, and also regard *sequences* and *strings* to be synonymous terms. For instance, `C-x(2,4)-[DE]` is a sequence pattern matching any sequence containing a substring starting with `C`, followed by between two and four arbitrary symbols, followed by either a `D` or an `E`. This pattern is an example of a *deterministic* pattern (a deterministic pattern either matches or does not match a given sequence). Patterns may also be *probabilistic*, i.e., assigning a probability to the match between a sequence and the pattern. Examples of probabilistic patterns are profiles and Hidden Markov Models. Deterministic patterns are simple and pure mathematical concepts, and are easier to interpret than probabilistic patterns. On the other hand, probabilistic patterns have more modelling power.

The goal of automatic pattern discovery is, given a set (family) of sequences, to find new, *a priori* unknown patterns, that are common to (matches) all, or most of the sequences in the set. If it is discovered that a certain pattern matches all or most of the sequences in the famliy, then it is possible that the presence of this particular pattern plays a part in determining the biological function of the corresponding macromolecules. Also if we detect in a new sequence the presence of a pattern known to be characteristic for a certain family, then we can hypothesise that the new sequence belongs to that family, even if we do not know its biological properties yet. In this way patterns may be used for the classification of biosequences and for predicting their properties. A pattern is said to be *diagnostic* for a family if it matches all the known sequences in the family, and no other known sequences.

Many of the known protein families have been collected in the PROSITE database (Bairoch, 1992). For most of the families a diagnostic pattern is given; for some families, the pattern given

is not perfectly diagnostic — it may fail to match some sequences in the family, and/or it may match some known sequences outside the family. For example, accession number PS00028 in PROSITE (release 13, November 1995) gives the zinc finger c2h2 family containing 279 proteins in SWISS-PROT (release 32, November 1995). The pattern `C-x(2,4)-C-x(3)-[LIVMFYWC]-x(8)-H-x(3,5)-H` matches all 279 known member sequences, but also 26 other sequences in SWISS-PROT.

Powerful automatic pattern discovery algorithms may enable us to look for patterns in a large variety of potentially related biosequences. For example, such methods may be used to automatically construct patterns for the PROSITE database. These patterns are currently constructed semi-manually. Apart from the fact that this is a tedious process, this method does not guarantee that all the possible patterns are explored and that the best patterns are found. Another example of the use of pattern discovery algorithms could be the analysis of DNA sequences believed to be involved in gene regulation. More generally, one could possibly discover new and unexpected relationships and regularities in biological sequences. Thus pattern discovery methods may prove to be an important tool for knowledge discovery and data mining in biosequence databases.

In this paper we consider the problem of the automatic discovery of deterministic patterns in biosequences. This is a machine learning problem, namely that of extracting general rules from particular instances. In this context, the pattern is the general rule and sequences are the instances of the rule. Given a set of positive examples (sequences in some family), and possibly a set of negative examples (sequences not in this family), the problem is to find patterns matching the positive examples and not the negative examples (if given). Many nontrivial and interesting methods have been developed for this problem and it would appear that the field would benefit from some systematisation. For this reason, we have designed a framework consisting of the following five dimensions.

The first is based on formulating the problem of pattern discovery in the framework of machine learning, e.g., the pattern discovery problem is related to learning from only positive examples, from both positive and negative examples, and to learning from noisy data. Second, we define a pattern language that effectively includes most of the deterministic pattern languages used in biocomputing, and by restricting this general language we can obtain pattern languages studied by particular authors. The third dimension is based on the formulation and study of the concept of a fitness measure describing how well a discovered pattern fits the training data as well as rating our *a priori* belief in the patterns. Fourth, the pattern discovery algorithms can be rated according to whether they are guaranteed to find the patterns of the highest fitness or not. Finally, along the fifth dimension, we divide the pattern discovery algorithms themselves depending on the algorithmic paradigms underlying them. We distinguish between algorithms that are based on the enumeration of possible patterns and on choosing the fittest ones (*pattern-driven* or PD), algorithms based on looking for common parts in sequences (*sequence-driven* or SD), and algorithms based on a combination of these paradigms.

The structure of this paper is as follows. In section 2 we describe the first four dimensions of the framework. We also give a discussion of whether or not the pattern languages used give sufficient expressive power to describe the crucial biological features of the sequences. In section 3 we define the PD and SD approaches to pattern discovery, and also describe ways of

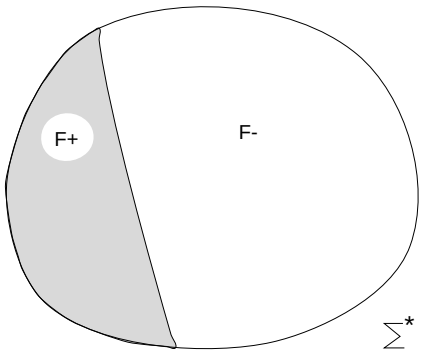

Figure 1: Schematic figure showing the relationships between the set F_+ of sequences in a family, F_- of sequences outside the family, and the set Σ^* of all possible sequences (strings).

combining these approaches. A survey is given of many of the currently known automatic pattern discovery methods. We place each of the surveyed methods within the common framework, and furthermore we identify the main algorithmic ideas of each method and show how these ideas relate to each other.

In Appendix A we present some basic information about the specific existing algorithms, and in Appendix B we give examples of the results of computational experiments, and samples of the patterns which have been discovered, for some of the existing methods. These experimental data are taken from the papers describing each of the methods. We conclude with a discussion of the possibility of establishing some benchmarks in the area of pattern discovery in biosequences.

2. Definition and discussion of the problem.

In this section we formulate the problems related to learning from biosequences. After giving a semi-formal specification of the problems, we sketch a three step approach to solving these problems, and discuss each step separately.

2.1 The formulation of the problems

Let F_+ be a set of sequences corresponding to a set of proteins sharing definite functional or structural properties (e.g., containing the same type of protein domain). We say that F_+ is a *family*. The set F_+ is a subset of the total set Σ^* of all possible strings over the amino-acid (or nucleotide) alphabet Σ. The sequences in $F_- = \Sigma^* - F_+$ constitute the set of sequences outside the family. This is sketched in figure 1. Note that while all sequences in F_+ correspond to biological macromolecules, there may be sequences in F_- not corresponding to such molecules. For example, not all sequences over the amino-acid alphabet correspond to foldable proteins.

Let g be a function $g : \Sigma^* \to \{\text{FALSE}, \text{TRUE}\}$ assigning boolean values to strings. Such a

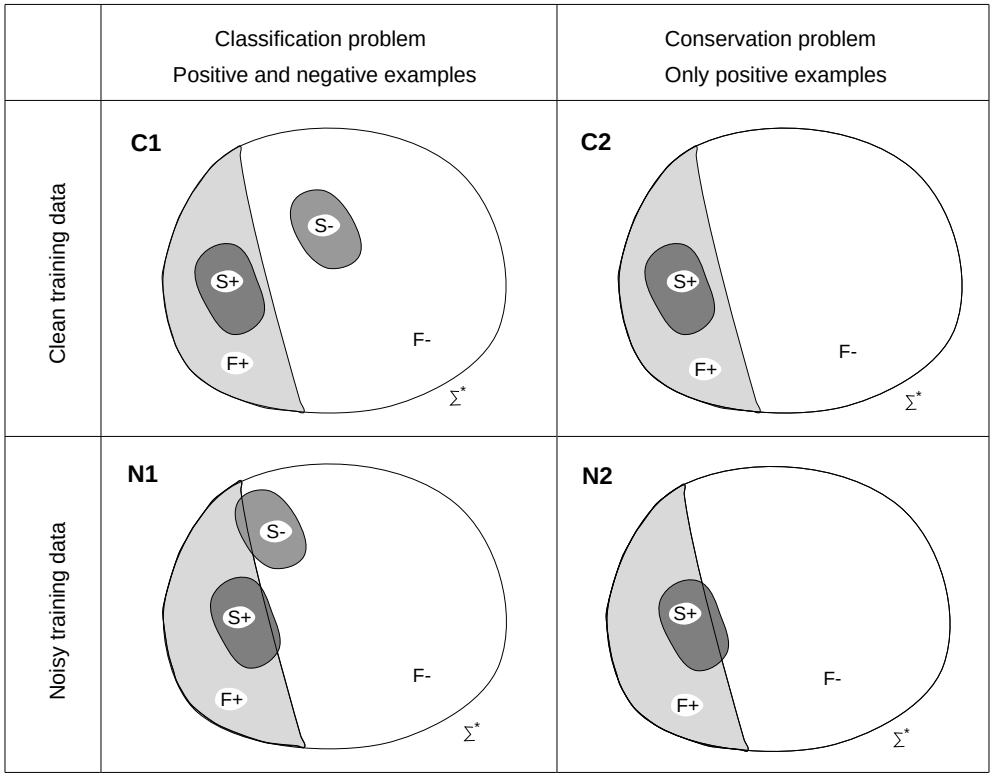

Figure 2: Schematic figure showing the training set for the different problems formulated in section 2.1. The training sets are in dark colour. In the left column, both positive and negative examples are given, while in the right column only positive examples are given. In the top row, clean training set can be assumed, while in the bottom row clean training set cannot be assumed.

function will be called a *string function*. Let g be a string function defined as follows:

$$g(s) = \begin{cases} \text{TRUE} & \text{if } s \in F_+ \\ \text{FALSE} & \text{if } s \in F_- \end{cases} \tag{1}$$

We will call g a *characteristic function* for the family F_+.

The basic problem studied in this paper is that of automatically finding string functions f approximating the characteristic function g for the family F_+. An algorithm for solving this problem takes as input a *training set* consisting of *positive examples*, which are sequences from F_+, and optionally *negative examples*, which are sequences from F_-. This is a machine learning problem, namely that of learning a general rule from a set of examples. When both positive and negative examples are given, we call it the *classification problem*, and when only positive examples are given, we call it the *conservation problem*. These problems are discussed below, and several more formal problem definitions are given, each with different assumptions about the training set. We discriminate between the case when the training set is assumed to be correct (clean data), and the case when there may be errors in the training set (noisy data). These different situations and the corresponding problem definitions are illustrated in figure 2.

2.1.1 Classification problem

Suppose we are given a set of sequences S_+ believed to be members of a family F_+, and a set S_- of sequences believed not to be members, i.e., $S_+ \subset F_+$ and $S_- \subset F_-$. The goal is to find a string function approximating the characteristic function for this family. Let us call such a function a *classifier function*. Assuming clean data the problem can be stated as follows.

C1: Suppose there exist two disjoint sets of sequences F_+ and F_- ($F_+ \cap F_- = \emptyset$) such that $F_+ \cup F_- = \Sigma^*$. Given two sets S_+ and S_- such that $S_+ \subset F_+$, and $S_- \subset F_-$, find *compact* string functions that return TRUE for sequences in S_+, FALSE for sequences in S_-, and have high likelihood of returning TRUE for sequences in F_+, and FALSE for sequences in F_-.

By compact we mean having a short description. We do not define precisely here what we mean by a 'short description' and by 'high likelihood', we discuss different ways of defining these notions later. As stated, **C1** consists of two parts:

C1a: find compact "explanations" of known sequences, and

C1b: try to predict the family membership of yet unknown sequences.

In reality sequences come from biological experiments and may contain errors (Kristensen, *et al.*, 1992), as well as may possibly have been wrongly included in the set of positive or negative examples. Therefore in general we cannot assume clean input and we should allow for some noise in the training set. In the noisy case, it is difficult to formulate all the aspects of the problem precisely. One possible definition is the following.

N1: Suppose there exist two disjoint sets of sequences F_+ and F_- ($F_+ \cap F_- = \emptyset$) such that $F_+ \cup F_- = \Sigma^*$. Given sets $S_+ \subset \Sigma^*$ and $S_- \subset \Sigma^*$ such that intersections $S_+ \cap F_-$ and $S_- \cap F_+$ are small, find compact string functions that return TRUE for most sequences in S_+, FALSE for most sequences in S_-, and have high likelihood of returning TRUE for sequences in F_+, and FALSE for sequences in F_-.

Unlike the case **C1**, we cannot select the classifier functions only from those which return correct TRUE/FALSE values for the entire training set. We need to find a balance between how well the classifier function fits the training set (i.e., how much is meant by "most"), and our *a priori* belief in how likely it is that different functions are able to correctly classify new sequences. The ways of assessing the latter are discussed later.

2.1.2 Conservation problem

Sometimes it is useful to find features common to a family of sequences, even if they are not unique to the family. In this case we do not want to construct a classifier function, but rather a function showing what is characteristic of the family. We call such a function a *conservation function*.

Let us say that a function is conserved in a set of sequences S if it returns TRUE for all sequences in S. Also, we say that a conservation function is *interesting* if it has a low probability of returning TRUE for random sequences[1], one function being more interesting than another if it has a lower probability of returning TRUE for random sequences.

C2: Suppose there exists a set of sequences F_+. Given a set S_+ such that $S_+ \subset F_+$, find interesting string functions (i.e., having low probability of returning TRUE for random sequences) that return TRUE for all sequences in S_+ and have high likelihood of returning TRUE for the sequences in F_+.

Note that an instance (S_+) of the conservation problem is closely related to an instance (S_+, S_-) of the classification problem where S_- consists of random sequences outside F_+. A method for solving the classification problem should be used if a classifier function is wanted in cases where there are sequences outside the family F_+ that are very similar to sequences in that family.

N2: Suppose there exists a set of sequences $F_+ \subset \Sigma^*$. Given a set $S_+ \subset \Sigma^*$ such that $S_+ \cap \overline{F_+}$ is small[2], find interesting string functions (i.e., having low probability of returning TRUE for random sequences) that return TRUE for most sequences in S_+ and have high likelihood of returning TRUE for the sequences in F_+.

In order to solve the problems **C1-N2** we will split them into three subproblems.

1. Find a good class of string functions from which the approximating function f is chosen for a particular real-world problem. We call this class the *solution space*, *hypothesis space*, or *target class*.
2. Define a ranking of the solution space, evaluating how good each function is for the given training set (i.e., how likely it is to approximate g). We call it a *fitness measure*.
3. Develop an algorithm returning those classifier functions from the given solution space that rate high enough according to the fitness measure.

The success in solving the prediction part of the problems depends on how successfully we chose the solution space and fitness measure, even if the algorithm is perfect in the sense of finding all the fittest patterns. In the next section we discuss possible solution spaces.

2.2 Solution spaces.

We have defined string functions as predicates returning TRUE or FALSE values. However, it is possible to generalise the definitions of string functions so that they return real values indicating the likelihood (or probability) that a given sequence belongs to a given family. A possible classification of different functions ranging from probabilistic to deterministic ones, has

[1] We assume some distribution for random sequences, for example we assume that the symbols in the sequences are independent and identically distributed (i.i.d.), i.e., $p_a = \frac{1}{|\Sigma|}$. Alternatively the frequencies of the symbols (in the training set, or in a database of nucleotide/protein sequences) can be used to define the symbol probabilities, i.e., $p_a = f_a$.

[2] $\overline{F_+}$ is the complement of F_+ with respect to Σ^*, i.e., $\overline{F_+} = \Sigma^* - F_+$

been given by Douglas Brutlag in a keynote address at the third international conference on Intelligent Systems for Molecular Biology (ISMB-95) as follows:

```
Deterministic                Consensus patterns
     |                       Alignments
     |                       Blocks or Weight Matrices
     |                       Templates or Profiles
     V                       Bayesian Networks
Statistical                  HMMs
```

The distinction between Hidden Markov Models (HMMs), Bayesian Networks, Templates and profiles is not strict, and depends on the detailed definitions of the models and the profiles used. Each of these models have application fields for which it is better suited. More on the statistical functions can be found in (Krogh *et al.*, 1994; Baldi *et al.*, 1994) (on Hidden Markov Models), (Gribskov, *et al.*, 1987; Bucher and Bairoch, 1994) (profiles), and (Chan, *et al.*, 1992) (alignments). Some authors have used maximum likelihood models for finding good *blocks*[3] (Lawrence and Reilly, 1990; Lawrence *et al.*, 1993; Bailey and Elkan, 1995; Bailey, 1995). Here we focus on the deterministic end, and particularly on patterns with expressive power within the class of regular languages.

2.2.1 Input alphabets

The ways of defining string functions will depend on the properties of the input alphabets. There are differences between nucleotide (DNA/RNA) and protein sequences that should be taken into account. Protein sequences are sequences over a 20-letter alphabet $\Sigma_p = \{$ `A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y` $\}$. Nucleotide sequences (DNA/RNA) are sequences over 4-letter alphabets $\Sigma_{DNA} = \{$`a,t,g,c`$\}$, and $\Sigma_{RNA} = \{$`a,u,g,c`$\}$.

The set of amino acids Σ_p may be grouped in classes $K_1, \ldots, K_n$ in different ways according to their physio-chemical properties, e.g., in AACC (Amino Acid Class Covering) hierarchical groups taken from (Smith and Smith, 1990) ($K_1 = \{$`D,E`$\}$, $K_2 = \{$`K,R,H`$\}$, $K_3 = \{$`N,Q`$\}$, $K_4 = \{$`S,T`$\}$, $K_5 = \{$`I,L,V`$\}$, $K_6 = \{$`F,W,Y`$\}$, $K_7 = \{$`C`$\}$, $K_8 = \{$`M`$\}$, $K_9 = \{$`A,G`$\}$, $K_{10} = \{$`P`$\}$, $K_{11} = \{$`D,E,K,R,H,N,Q,S,T`$\}$, $K_{12} = \{$`I,L,V,F,W,Y,C,M`$\}$, and $K_{13} = \Sigma_p$), or using Venn-diagrams (Taylor, 1986), or in some other way. Also, *substitution matrices*, e.g., PAM (Dayhoff, 1978), and Blosum (Henikoff and Henikoff, 1992), can be defined giving statistics for each pair of amino acids for how often they are found in equivalent positions in proteins that are within a certain evolutionary distance. Similar groupings and scoring matrices can be used in analysis of nucleotide sequences, however this, seems to play a less important role than in protein sequence analysis.

Both protein sequences and nucleotide sequences can be translated into smaller alphabets. Such a translation is called an *indexing*, and is obtained by making a partition[4] of the basic alphabet Σ, and translating symbols in the same partition into the same symbol in the reduced alphabet. For example, amino acids are either hydrophobic, neutral, or hydrophilic, and can be mapped onto a three-symbol alphabet $\Sigma_{hydro} = \{+, 0, -\}$ (e.g., (Karlind and Ghandour, 1985; Arikawa *et*

[3] A block is defined as a local ungapped alignment, i.e., it is a set of segments of identical length "put on top of each other" giving an alignment (Posfai *et al.*, 1989).

[4] A partition of a set A is a set B of disjoint subsets of A such that the union of the sets in B is A.

al., 1992)). Similarly nucleotide sequences can be translated into a purine-pyrimidine alphabet $\Sigma_{n_{red}} = \{R, Y\}$.

2.2.2. Generalised regular patterns

Pattern definitions are usually given by examples in the biocomputing literature. Here we give a more formal definition, and define a class of what we call *generalised regular patterns*, which will unify most of the deterministic pattern classes used in biocomputing. Various more restrictive pattern classes will be obtained as appropriate subclasses.

Let $\Sigma = \{a_1, \ldots, a_m\}$ be an alphabet called the *basic alphabet*. For protein sequences $\Sigma = \Sigma_p$, for nucleotide sequences $\Sigma = \Sigma_{DNA}$, or $\Sigma = \Sigma_{RNA}$; if an indexing is applied, Σ may be, for example, Σ_{hydro} or $\Sigma_{n_{red}}$, or any other abstract alphabet. Let $K_1, \ldots, K_n$ be some subsets of Σ, such that each subset contains more than one element ($|K_i| \geq 2$). Let $\Pi = \{b_1, \ldots, b_n\}$ be another alphabet disjoint with Σ, and let us define $L(b_i) = K_i$. For convenience let us also assume that $L(a_i) = \{a_i\}$, for $a_i \in \Sigma$. In practice $K_1, \ldots, K_n$ are classes of amino-acids (e.g., the AACC classes) or nucleic acids, and $b_1, \ldots, b_n$ are the characters denoting these classes. The character denoting a class $K_i = \{a_{i_1}, \ldots, a_{i_l}\}$ is usually denoted by $[a_{i_1} \ldots a_{i_l}]$. For instance, in our representation of the AACC hierarchy, b_1 denoting $K_1 = \{$`D,E`$\}$, would be denoted by [`DE`]. This does not apply, however, to the character b_i standing for the whole Σ, which is usually denoted by x (or sometimes by $\cdot$), effectively meaning the *wildcard* (or *don't-care*) character. Here we will use x for the wildcard. Wildcards are sometimes also referred to as *spacers*.

Let $x(p, q)$, where p and q are non-negative integers and $p \leq q$, be a wildcard of length from p to q, and let $L(x(p, q))$ be defined as a set of all words over Σ of length between p and q, i.e., $L(x(p, q)) = \{\alpha \in \Sigma^* | p \leq |\alpha| \leq q\}$. Let X be the set of all objects of the form $x(p, q)$. Finally, let $*$ be a character such that $* \notin \Sigma \cup \Pi$, and let $L(*) = \Sigma^*$, i.e., $*$ is the wildcard of an arbitrary length - $x(0, \infty)$.

A *generalised regular pattern* π is a string over an alphabet $(\Sigma \cup \Pi \cup X \cup \{*\})$. We define the language $L(\pi)$ of pattern π, where $\pi = c_1 \ldots c_r$, and $c_i \in \Sigma \cup \Pi \cup X \cup \{*\}$, as

$$L(\pi) = L(c_1) \ldots L(c_r),$$

the concatenation being defined as $L(c_1) \ldots L(c_k) = \{\gamma_1 \ldots \gamma_r | \gamma_1 \in L(c_1), \ldots, \gamma_r \in L(c_r)\}$. A string α *matches* a pattern π if $\alpha \in L(\pi)$. The class of languages that can be expressed using generalised regular patterns is a subset of regular languages. From now on, by 'pattern' we will understand a generalised regular pattern.

We introduce the following pattern classification based on limiting the pattern alphabet to different combinations of sets Σ, Π, X, and $\{*\}$, and limiting whether the symbol $*$ can appear in an arbitrary place in a pattern, or only at the beginning and at the end. For patterns where $*$ can appear only at the beginning and at the end, i.e., for patterns of the type $\pi = *\pi'*$, $\pi = *\pi'$, and $\pi = \pi'*$ we will distinguish between the following cases restricting the alphabet of π':

Class	definition	example pattern
A	$\pi' \in \Sigma^*$	t-c-t-t-g-a
B	$\pi' \in (\Sigma \cup \{x\})^*$	D-R-C-C-x(2)-H-D-x-C
C	$\pi' \in (\Sigma \cup \Pi)^*$	G-G-G-T-F-D-[ILV]-[ST]-[ILV]
D	$\pi' \in (\Sigma \cup \Pi \cup \{x\})^*$	V-x-P-x(2)-[RQ]-x(4)-G-x(2)-L-[LM]
E	$\pi' \in (\Sigma \cup X)^*$	G-C-x(1,3)-C-P-x(8,10)-C-C
F	$\pi' \in (\Sigma \cup \Pi \cup X)^*$	C-x(2,4)-C-x(3)-[ILVFYC]-x(8)-H-x(3,5)-H

In the second case when $*$ can appear in an arbitrary place we define classes:

Class	definition	example pattern
G	$\pi \in (\Sigma \cup \{*\})^*$	D-T-A-G-Q-E-*-L-V-G-N-K
H	$\pi \in (\Sigma \cup \Pi \cup \{*\})^*$	D-T-A-G-[NQ]-*-L-V-G-N-[KEH]
I	$\pi \in (\Sigma \cup \Pi \cup X \cup \{*\})^*$	D-T-A-x(2,5)-G-[NQ]-*-L-V-G-N-[KEH]

Note that these classes can be partially ordered in a lattice, **A** being the bottom element and **I** the top element. The class **A** corresponds to *substring patterns*, and the class **G** to *regular patterns* of Shinohara (1983). The class **F** corresponds to the class of patterns most frequently used in the PROSITE database. In PROSITE notation, the leading and closing $*$ symbols are not used, $*\pi*$ being written simply as π. The notation permits attachment of the pattern to the beginning or end of a sequence by using a leading $<$ or closing $>$ symbol, thus π becomes $<\pi>$.

Any of the classes containing the alphabet Π can be further refined by choosing a particular alphabet Π (i.e., particular subsets $K_1, \ldots K_n$). We can also restrict the solution space by considering patterns of a specific (or up to a specific) length.

2.2.3 Defining the string functions via patterns

The simplest way to define a *classification (conservation)* function using a pattern π is

$$f(\sigma) = \begin{cases} \text{TRUE} & \text{if } \sigma \in L(\pi) \\ \text{FALSE} & \text{otherwise.} \end{cases} \qquad \textbf{(a)}$$

An extension of this approach is to allow for approximate matching between the pattern and the string. A measure of the distance between two strings can be used (e.g., edit or Levenshteins distance (Levenshtein, 1966)). If $dist(\sigma_1, \sigma_2)$ is the distance between two strings σ_1, and σ_2, then the distance between a string and a pattern can be defined as $Dist(\pi, \sigma) = \min_{\sigma' \in L(\pi)} dist(\sigma', \sigma)$. Hence the definition of classifier/conservation function f can be generalised to include approximate matching:

$$f(\sigma) = \begin{cases} \text{TRUE} & \text{if } Dist(\pi, \sigma) \leq const \\ \text{FALSE} & \text{otherwise,} \end{cases} \qquad \textbf{(b)}$$

for some given constant *const*. More advanced distance measures, involving use of amino acid substitution matrices (Dayhoff, 1978; Henikoff and Henikoff, 1992) and different ways of penalising gaps, have also been used, for example in database similarity search programs, e.g., (Altschul *et al.*, 1990; Lipman and Pearson, 1985).

Patterns can also be used for defining classification/conservation functions in more complicated ways than just exact (**a**), or approximate (**b**) membership of a language. We are aware of two more ways reported in the literature:

c membership to a union of pattern languages, i.e., $f(\sigma) = \text{TRUE}$, if $\sigma \in L(\pi_1) \cup \ldots \cup L(\pi_n)$ where $\pi_1, \ldots, \pi_n$ are patterns of some class (Arimura *et al.*, 1994; Shoudai *et al.*, 1995; Brazma *et al.*, 1996b).

d using decision trees over patterns (for the definition of a decision tree see (Arikawa *et al.*, 1993)).

Method **c** is useful when a family contains several subfamilies and whilst there is a strong conserved pattern in each subfamily, the family as a whole share only a weak conserved pattern. Note that the learning of unions of patterns languages from positive examples is closely related to *unsupervised learning* – the task of simultaneously finding a set of patterns, and subfamilies (i.e., subsets) of the training set, so that each subfamily shares a distinct pattern.

Since most of the information about the function for the considered classes of string functions is given by the underlying patterns, we will from now on often refer to patterns instead of functions.

2.2.4 Do the solution spaces give sufficient expressive power?

The PROSITE database contains many examples of protein families which have diagnostic consensus patterns of the type defined above. For example, the zinc finger proteins all have cysteine or histidine amino acids in certain positions. In these cases, the class of generalised regular patterns may be sufficient. In many other families the features conserved between the sequences are more subtle. Probably there does not exist any consensus pattern diagnostic for the helix-turn-helix family. Patterns of correlations exist (e.g., pseudo-knots in RNA secondary structures) that give crossing dependencies taking us even beyond what can be compactly described using context-free grammars.

The choice of a target class $\mathcal{C}$ for our approximation functions f (i.e., the first part of our approach) is a trade-off between what classes of functions are expressive enough to allow description of the crucial biological features of sequences, and for what classes of functions we can develop an efficient algorithm for finding string functions from examples, i.e., what classes can be learned efficiently. For a particular problem, we want to choose a class $\mathcal{C}$ as small as possible for efficiency reasons, but at the same time $\mathcal{C}$ should be expressive enough to contain a function f approximating g as defined by equation (1) at the beginning of this section.

After we have chosen the solution space $\mathcal{C}$, the problem of finding the function f from positive and negative examples S_+ and S_- has some similarity to PAC-learning (probably approximately correct learning) (Valiant, 1984; Shinohara and Arikawa, 1995). The difference between our case and PAC learning is that PAC learning requires approximation within any given precision, while in our case it may not be possible. Also, a potentially infinite number of examples are assumed to be available in PAC learning, while in our case we have only a finite number of examples, and frequently this number is quite small. Moreover, it is possible that the shortest

description of any good approximation function f is simply a comprehensive collection of all the positive examples. In this case, the task of learning from biosequences is not so much that of approximating a hypothetical function g, but rather just discovering interesting properties of g.

2.3 Ranking discovered patterns and functions

In most of this subsection we assume that a conservation or classification function is defined directly from a single pattern using method **a** (exact match) or **b** (approximate match). In this way we are able to talk about ranking patterns instead of string functions, thus making it easier to relate the discussion to papers describing pattern discovery methods.

Our aim is to find a pattern π such that the string function f defined by π is as close as possible to the characteristic function of the family (as defined by equation (1)). However, as the characteristic function is unknown (note that we have only a part of the family known), we have to find an indirect way of evaluating patterns from the training set and their syntactic form. In practice, a fitness measure is defined as a function

$$F(\pi, S) \to \mathcal{R}$$

that takes two arguments and returns a real. The first argument is a pattern from a particular pattern class. The second argument is the training set. The value of the function should show how good the pattern is in respect to the training set. Sometimes the values can be normalised to $[0, 1]$.

If we have chosen a very restricted solution space (e.g., based on patterns of class **A** of a fixed length), the fitness measure can be very simple, particularly in the case of clean data. It can have just two values 0 and 1, assigning 1 to the patterns matching all the positive example sequences, and none of the negative example sequences for the classification problem. In the noisy case, we can use this simple two value fitness measure by assuming a certain maximum level of noise (say, 30%), and assigning 1 to the patterns correctly classifying at least the remaining portion (i.e., 70%) of the training set.

A weakness of using a 0/1-valued function is that the approach then depends heavily on the choice of the noise threshold. Instead of using 0/1-valued function we can use a function that increases continuously with the number of sequences matching the pattern. For instance, for the classifier problem such a fitness measure F_1 can be based on how many of the respectively positive and negative examples match the pattern P. For each of the candidate string functions $f_1, \ldots, f_n$, we count the number of *false positives* (FP), i.e., $s \in S_-$ and $f(s)$ =TRUE, and *false negatives* (FN), i.e., $s \in S_+$, and $f(s)$ = FALSE. We also define the number of *true positives* (TP) as the number of sequences s in S_+ for which $f(s)$ =TRUE, and the number of *true negatives* (TN) as the number of sequences in S_- for which $f(s)$ =FALSE. The *sensitivity* of a function f (Lathrop *et al.*, 1993) can be defined as

$$Sn(f) = \frac{TP}{TP + FP} \tag{2}$$

Similarly, the *specificity* of the function can be defined as

$$Sp(f) = \frac{TN}{TN + FP} \tag{3}$$

The proposed functions may be ranked according to their sensitivity, specificity, or to some combination of both[5].

We rely heavily on the choice of the solution space in the case of the described fitness measures, as they rate equally all the patterns matching the same portion of the given sequences. If we choose too general a solution space, for instance including the regular expression $\cup_{s \in S_+} s$, then, particularly for the conservation problem, it can lead to overfitting the training set and the string functions obtained may be not useful at all for prediction. The heavy burden of making the right choice of the solution space can be relieved by using a more subtle fitness measure that depends not only on how well the pattern fits the training data, but also includes as a factor some *a priori* rating of the patterns. For instance, we can express the fitness measure as $F(\pi, S) = F_1(\pi, S) \cdot F_2(\pi)$, where $F_1(\pi, S)$ shows how well the pattern π fits the training data and $F_2(\pi)$ gives an *a priori* rating of the pattern.

The *a priori* rating F_2 can be based on Occam's razor principle (see (Hutchinson, 1994)), which says that when several theories explain (past) observations equally well, the simplest theory is better. It is possible to prove formally that under certain assumptions the simplest patterns are likely "to predict the future better". In the case of the classification problem the evaluation can be based on Occam's Razor rather directly: simpler patterns correctly classifying the same number of examples should be *a priori* rated higher. Concretely, we can define $F_2(\pi)$ as the reverse to the number of bits needed to describe the pattern π. The dependence on the training data, in this case, can still be a 0/1-value function: $F_1(\pi, S) = 1$, if the function classifies (is conserved) the given percentage of sequences in the training set correctly, $F_1(\pi, S) = 0$ otherwise.

In the case of the conservation problem we want to rate *a priori* higher the patterns that are more "interesting" in the sense of the definition in section 2.1.2, i.e., that have smaller likelyhood of matching a random sequence. For this we can use the *information content* of the pattern (Jonassen, *et al.*, 1995) for *a priori* rating. Information content means the amount of information provided by the knowledge that the pattern matches a sequence. Amongst the patterns that match the same number n of sequences, we give a higher rating to the ones with a greater information content. This may paradoxically seem to be the opposite of Occam's Razor principle, since it will generally rate more complex patterns higher. Nevertheless the Occam's Razor is used here: since we are not given any negative examples, we assume that all random sequences are equally likely to serve as negative examples. A more obvious way of applying Occam's Razor principle for conservation problem is through the minimum description length principle, discussed later.

Another way of defining the fitness measure for the conservation problem is based on the *statistical significance* of the patterns (e.g., (Waterman, *et al.*, 1984; Staden, 1989a; Neuwald and Green, 1994; Sewell and Durbin, 1995)), defined as follows. Suppose $p_1, \ldots, p_n$ are patterns such that each p_i matches a subset S_i of S_+. Then, for a pattern p_i, the *pattern probability* is the probability that p_i matches at least $|S_i|$ out of $|S_+|$ random sequences (of the same length and

[5]One possible combined measure is the *correlation coefficient* between two sets; (1) the set of sequences in S_+, and (2) the set of sequences in the set $S_+ \cup S_-$ for which the function f returns TRUE. The correlation coefficient is $C = (TP \cdot TN - FP \cdot FN)/\sqrt{(TP + FP) \cdot (FP + TN) \cdot (TN + FN) \cdot (FN + TP)}$, which is 1.0 when there are no false positives or negatives, 0.0 when f is random with respect to S_+, and S_-, and -1.0 when there are only false positives and false negatives.

composition as the sequences in S_+) purely by chance. In this analysis the sequences and the sequence positions are assumed to be independent. The pattern significance can be defined as the reverse of the pattern probability, thus patterns having lower probability should be ranked higher. Note that statistical significance of the pattern will increase either if the information content of the pattern increases or if the pattern matches more sequences. In this way both aspects of the fitness measure are taken into account.

A closely related way is to measure the information content of the block defined by the substrings in the training set matching a pattern (Lawrence *et al.*, 1993; Staden, 1989b). A problem in this case is that a relatively small number of very similar sequences can result in a block of a large size. Therefore the assumption of some minimal number of sequences needed to match the pattern is still required. A way of avoiding this is considered in (Jonassen, *et al.*, 1996).

A direct way to apply Occam's Razor principle for the case of conservation problem is based on the *minimum description length* (MDL) principle (Rissanen, 1978; Li and Vitanyi, 1993; Brazma, *et al.*, 1996a), which gives a fitness measure easily applicable to the problem of unsupervised learning. Considering string functions defined from a pattern (**a** in our classification) or a set of patterns (**c**), the MDL principle says that the best pattern (or set of patterns) is the one that minimises the sum of

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

Such a fitness measure for conservation functions defined by unions of patterns is developed in (i.e., case **c**) (Brazma *et al.*, 1996b). A coding scheme is introduced using PROSITE type patterns for compression of sequences. The fitness value of a conservation function is defined to be proportional to the compression of the training set that can be achieved using the set of patterns defining the function. It is shown that this fitness measure can be expressed as the sum of the information contents of each pattern (Jonassen, *et al.*, 1995) times the number of sequences matching the respective pattern, minus a correction independent of the number of matched strings.

2.4 Search algorithms and guarantees

The specifications of the problems **C1** - **N2** are rather informal, leaving a number of notions undefined, including "high likelihood" and "most sequences". In practice, any learning algorithm for solving these problems is effectively designed for a specific target class $\mathcal{C}$ and uses a specific fitness measure F. The input of such an algorithm is the training set S, and the algorithm is required, given the training set S, to produce a set of patterns P from the class $\mathcal{C}$ such that the fitness value $F(\pi, S)$, for $\pi \in P$, is "relatively" high. If it can be proved that the algorithm will produce the specified portion of the patterns with the highest fitness value $F(\pi, S)$ among all the patterns in $\mathcal{C}$ (i.e., either the fittest pattern, or a given number or percentage of the fittest patterns, or all the patterns with a fitness higher than a given constant), then it is said to be *guaranteed* to find the best pattern (or best patterns).

The success in the prediction of the properties of unknown sequences (i.e., in finding a good approximation function f) relies on choosing an appropriate pattern class $\mathcal{C}$ and a good fitness measure F. Evaluating this is outside the scope of a mathematical definition, and should rely either on experimental evaluation or on the judgement of an expert. A set of sequences excluded from the training set may be used as a test set in order to evaluate how good the discovered patterns are at predicting family membership of unknown sequences. This is a standard evaluation method used in experimental machine learning, however, the number of known sequences is too small for the performance of this kind of experiment for most families known at present.

In order to test a new method for pattern discovery, it can also be applied to well known families to show that it is able to recover already known conserved patterns. The ultimate test of the significance of the pattern, however is its correspondance to some region conserved in the family for structural or functional reasons.

Various algorithms for finding classification or conservation functions having high fitness according to the given measure from the given class are discussed in the next section.

3. Algorithms for pattern discovery

In this section we first discuss algorithms for finding classification and conservation functions defined directly from generalised regular patterns using methods **a** and **b** (the exact and approximate match of a pattern) given in Section 2.2.3. Most of the algorithms described below, are for the discovery of patterns exactly matching some subset of the positive examples (see also the table given in appendix A). At the end of the section we briefly describe methods for finding functions defined using unions and decision trees of patterns (methods **c** and **d**).

At the highest level, we can divide the pattern discovery algorithms in two groups. The first, which we call pattern-driven (PD) approaches, is based on enumerating candidate patterns in a given solution space and picking out the ones with high fitness. The second, which we call sequence driven (SD) approaches, comprises algorithms that try to find patterns by comparing the given strings and looking for local similarities between them. For instance, an SD algorithm may be based on constructing a local multiple alignment of the given sequences and then extracting the patterns from the alignment by combining the segments common to most of the sequences.

The advantage of the PD approaches is that in this way it is possible to guarantee finding the best patterns up to some limited size (by *pattern size* or *length* we understand the minimal mumber of bits needed to describe the pattern), almost regardless of the total length of the examples. The reason for this is that it is usually possible to organise the algorithm so that it is linear-time in this length. On the other hand the size of the pattern-space is exponential in the length of the patterns. PD algorithms guaranteed to find the pattern with the highest fitness value, have worst case time complexity exponential in the length of the patterns. Thus, PD algorithms can be guaranteed to find only patterns of limited size. It is also possible to combine PD and SD approaches in a single algorithm.

It may be possible to discover patterns of an almost arbitrary size by SD algorithms. A weakness

of the SD approach is that in general it is impossible to guarantee optimality of the results without sacrificing efficiency. The reason for this is that the algorithmic problems on which precise comparison of multiple sequences can be based (e.g., the problem of constructing the optimal multiple alignment or finding the longest common subsequence) are NP-hard (Wang and Jiang, 1994; Garey and Johnson, 1979), and therefore SD approaches have to be based on heuristics. In general SD algorithms tend to work well if the sequences are sufficiently similar.

In the following subsections we describe in more detail the basic ideas of PD, SD and combined approaches to pattern discovery. When discussing time-complexity of the algorithms we will use n for the number of sequences, l for the average length of the sequences, and L for the total length of the sequences. We will not give precise time-complexity evaluations for all algorithms systematically, since this would often require to explain the algorithms in more detail than affordable for the lenght constraints, also detracting the reader from the main algorithmic ideas. We present the basic information about each algorithm separately in Appendix A, and present some sample patterns discovered by some of the algorithms in Appendix B.

3.1. Pattern driven approaches

The general framework of pattern driven algorithms can be formulated as follows:

- define the solution space $\mathcal{C}$ (i.e., a set of patterns) and the fitness measure,
- enumerate the patterns in the solution space,
- calculate the fitness of each pattern with respect to the given examples,
- report the fittest patterns.

The most straightforward implementation of the PD approach is explicit enumeration of all the patterns from the pattern space one by one. For instance, if the patterns are subwords (i.e., of the class **A** defined in section 2.2.2) of length 3 in the alphabet {`a, c, g, t`}, and if we use $\{0, 1\}$ as a value of fitness measure, assigning 1 to the patterns that correctly classify the given portions of the training sequences and 0 to the others, then the algorithm can enumerate all the words `aaa, aac, aag,` ..., `ttt`, calculate their fitness by counting in how many examples each is present, and output the words that have the fitness value 1.

In the simplest case of substring patterns (class **A**) this approach was first applied in the early 1980's (Queen, *et al.*, 1982; Waterman, *et al.*, 1984) and later in (Staden, 1989b). The search space is limited to fixed length patterns and the algorithms count the number of sequences in the training set that approximately match the pattern[6]. For *a priori* ranking of the patterns, Waterman *et al.* (1984) estimate the statistical significance of the discovered patterns, while Staden (1989b) calculates a measure of the information content of the block consisting of the segments that approximately match. Recently, Wolferstetter *et al.* (1996) have used a similar

[6]For this, a notion of the set of neighbours of a string, i.e., strings that are similar to the given string, is introduced, and a count is made of in how many examples each substring or any of its neighbours is present. The best pattern is selected on the basis of its fitness to the "neighbourhoods".

method (their fitness measure based on information content of the pattern) coupled with a user-friendly Web-interface for discovering conserved motifs in genome regulatory regions. The time complexity of these algorithms is linear in the total lenght of the sequences and exponential in the lenght of the patterns.

The straightforward enumeration approach can be easily extended to more complicated patterns. Smith *et al.* have used this approach for discovering patterns containing characters from the basic alphabet and wildcards (Smith, *et al.*, 1990). The algorithm enumerates all possible patterns consisting of three conserved positions with constant spacings within a pre-set range, i.e., patterns in class **B** from section 2.2.2. The patterns considered can be written in the form a_1-x(d_1)-a_2-x(d_2)-a_3, where a_1, a_2, and a_3 are characters from the basic alphabet, and d_1 and d_2 are the number of wildcard characters x in between them (for instance, H-x(2)-A-x(6)-G is such a pattern). In this case, only the numbers of sequences that exactly contain the pattern are counted. The user provides the minimum number of sequences that should contain the pattern in order for it to be considered. The patterns are evaluated using a heuristic fitness measure, and the patterns with the highest fitness are reported in the end. The time complexity of this part of the algorithm is $O(d^2L)$, where d is the miximal value of d_1 and d_2. (Smith, *et al.*, 1990) also uses elements of the SD approach to extend the patterns found by enumeration (see section 3.3 and appendix B). An interesting extension of this method has been reported in (Suyama, *et al.*, 1995), which in addition permits the discovery of patterns containing flexible length wildcards (i.e., patterns of the class **E**).

An obvious problem in this straightforward enumeration is that of efficiency. The size of the search space for patterns of length l grows as $O(|\Sigma|^l)$. However, the number of patterns can be reduced if we impose some restrictions on the pattern class. For instance, in the method of Smith *et al.* (1990) there are $20 \times 10 \times 20 \times 10 \times 20 = 800000$ candidate patterns of the type a_1-x(d_1)-a_2-x(d_2)-a_3 to be checked if the distance range is 10 (i.e., $0 \leq d_1 < 10$ and $0 \leq d_2 < 10$) and $a_i \in \Sigma_p$. However this number becomes impractical for more general pattern classes. Therefore some method for pruning the solution space, either by a provably accurate method or by using heuristics should be found if we want to increase the size and complexity of the patterns.

3.1.1. Methods for pruning during the search

A rigorous approach for pruning the search space can be based on representing it as a tree and pruning the subtrees rooted with patterns having a fitness under some threshold. For instance, if we are looking for simple patterns (i.e., of the class **A**) in the alphabet {`a,c,g,t`}, then a part of the pattern space can be represented as the tree in figure 3.

This tree can be traversed in either a breadth-first or a depth-first manner; both of these ways have been explored by Sagot *et al.* (1995b). The breadth-first approach can be more time-economic in practice because the pruning of the search tree can be more efficient than in depth-first search. For instance, for a substring `act` to be present in a sufficient number of sequences, both substrings `ac` and `ct` should have been found earlier if the search is done breadth-first. If depth-first search was used, we would only require that the substring `ac` should have been found earlier. Unfortunately breadth-first search can realistically only be applied to very short patterns, because the number of the patterns to be remembered grows very fast using

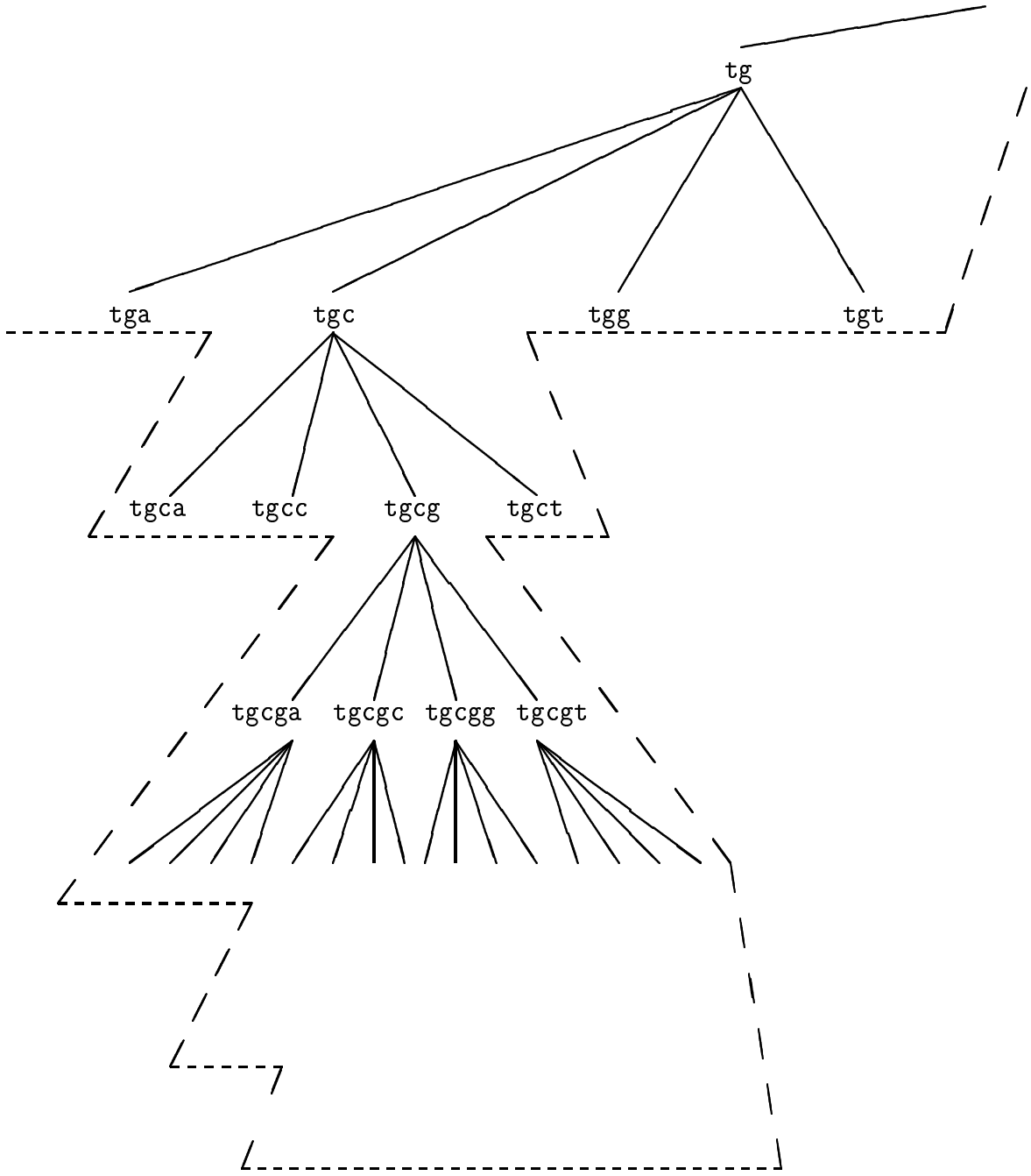

Figure 3: Pruning of the search tree. If, for instance, we have found that a substring `tga` is not present in sufficiently many sequences, then there is no need to look for any of `tgaa, tgac, tgag, tgat`. If, on the other hand, the substring `tgc` has been found in a sufficient number of the sequences, then one can hope that at least some of the substrings `tgca, tgcc, tgcg, tgct` may also be present in these sequences.

this method. Therefore for practical purposes depth-first search is used.

A very efficient implementation of this idea for substring patterns is the Karp-Miller-Rosenberg (KMR) (Karp, *et al.*, 1972) algorithm [7]. The idea is extended by Sagot *et al.* (1995b) to find more complicated patterns of class **C** containing symbols denoting groups of the basic alphabet. Unfortunately, the efficiency of the algorithm decreases when amino acid symbols are present in many groups. Also, if some groups are very large (containing many basic symbols) this slows the search because extending patterns with such symbols often produce conserved patterns, and hence large parts of the search tree have to be explored. For this reason (Sagot, *et al.*, 1995b) does not allow wildcard characters.

On the other hand it can be seen that dealing with wildcard characters should be quite easy. For instance, if we have found that the fitness of the pattern `tgc` is high enough, then by using wildcards the patterns can be potentially extended not only to patterns `tgca`, ..., `tgct`, but also to patterns $\texttt{tgc}x\texttt{a}, \ldots, \texttt{tgc}x\texttt{t}, \texttt{tgc}xx\texttt{a}, \ldots, \texttt{tgc}xx\texttt{t}, \texttt{tgc}xxx\texttt{a}, \ldots, \ldots$ and each of them should be checked for fitness. Neuwald and Green (1994) presents a method using this approach. They apply a pruning mechanism based on a measure of statistical significance of the patterns, avoiding to explore extensions of patterns with low significance thus speeding up the depth-first search significantly. Also, they introduce a new, so-called *block* data structure and use this to very efficiently find the set of substrings matching each pattern. Neuwald and Green allows group characters in patterns (i.e., the alphabet Π) for groups consisting of pairs of amino-acids, hence giving patterns of class **D**.

In a later paper, Sagot and Viari (1996) have presented an alternative approach, which uses a depth-first search to discover patterns containing ambiguous symbols as well as wildcards. In principle one need not specify beforehand which groups of letters are to be used in ambiguous pattern positions. In practice this works for nucleotide sequences, however, for protein sequences (where there are $2^{20} - 1$ possible non-empty groups) one needs to define a priori the character groups to be used. For each possible group one can set an upper limit on the number of occurrences of this group in a pattern. Together with a constraint on the minimum percentage of sequences to contain a pattern, this is used to prune the search. Also, if two patterns match the same sequence segments, and one is a generalisation of the other, only extensions of the least general pattern is explored further. The time-complexity of this algorithm is $O(Lkg^k)$, where g is the maximal number of character groups containing the same letter.

Jonassen *et al.* (1995) describes an algorithm where the use of a depth-first search strategy combined with the block data structure introduced in (Neuwald and Green, 1994), is pushed even further. This algorithm is able to discover patterns having both ambiguous positions (groups of amino acids) and flexible spacings (gaps), giving patterns in the class **F**. The user defines restrictions on the kind of patterns that can be discovered, effectively defining a subclass of **F**. The algorithm sets out to find all patterns in this subclass matching at least some minimum number N_{min} of the positive sequences. The search tree is pruned so that extensions of patterns matching less than N_{min} sequences, are not analysed. The algorithm works in two phases, and normally during the first phase only patterns consisting of single letter positions and wildcards

[7] The KMR algorithm can be adapted for finding all substrings present in at least k out of the given n sequences in time $O(N \log N)$, where N is the total length of the sequences. Note that the same task can be solved in time $O(N)$ by generalised suffix trees.

are considered. The best patterns found during the first phase are passed on to a second phase where they are subjected to an either exhaustive or heuristic search where ambiguous pattern symbols might be added. A fitness measure for patterns is defined, and the algorithm is guaranteed to find the highest scoring patterns within the subclass of **F** that match at least N_{min} of the positive examples.

In addition to using an SD-element to limit the search space (see section 3.3.2), Jonassen (1997) also introduces branch-and-bound and heuristics to make the pruning of the search tree more efficient. This speeds the search significantly, especially for sets of quite similar sequences. In both (Jonassen, *et al.*, 1995; Jonassen, 1997) an SD element is used to specialise patterns discovered in the depth-first search.

Experiments clearly show that pruning the search space in combination with efficient data structures substantially increases efficiency of the algorithms. Nevertheless the algorithms are still worst-case exponential in the length of the patterns, and no nontrivial speed-up over the straight-forward algorithms has been proved theoretically.

Note that although all the algorithms reported here are for the conservation problem, the same algorithms can be used for the classification problem by using an appropriate fitness measure. This has been used for instance by (Ogiwara *et al.*, 1992).

3.2. Sequence-driven approaches

The common elements of the sequence driven approaches can be summarised as follows:

- For sequences $s_1, \ldots, s_k \in S_+$, make sets $P_{\{s_1\}} = \{s_1\}, P_{\{s_2\}} = \{s_2\}, \ldots, P_{\{s_k\}} = \{s_k\}$.
- Iterate: choose i and j in some given way and combine the sets P_{S_i} and P_{S_j} into a new set $P_{S_i \cup S_j}$ such that $P_{S_i \cup S_j}$ is a set of patterns with high fitness that matches all (or most of the) sequences from $S_i \cup S_j$. In general more than two sets may be combined in one iteration step.
- In the end we obtain a set P_{S_+} of patterns conserved in S_+ (or in most of S+).

For example, suppose we are given three sequences:

$$s_1 = \texttt{AWCEFGHJKLM} \tag{4}$$

$$s_2 = \texttt{EFGOPAWRJKLS} \tag{5}$$

and

$$s_3 = \texttt{TAWUVOPHJKL} \tag{6}$$

According to the SD approach, we make three initial sets of patterns $P_{\{s_1\}} = \{\texttt{AWCEFGHJKLM}\}$, $P_{\{s_2\}} = \{\texttt{EFGOPAWRJKLS}\}$, and $P_{\{s_3\}} = \{\texttt{TAWUVOPHJKL}\}$. Suppose, for instance, that the chosen order for joining these sets is that first $P_{\{s_1\}}$ and $P_{\{s_2\}}$ are joined and then the result is joined with $P_{\{s_3\}}$. Further, let the method of choosing the common patterns be such that after the first step we get a set of two patterns $P_{\{s_1,s_2\}} = \{\texttt{*AW*JKL*}, \texttt{*EFG*JKL*}\}$ (note that both these

patterns match s_1 and s_2, and that both patterns are the longest in the sense that no extension matches s_1 and s_2). In the next step, joining $P_{\{s_1,s_2\}}$ with $P_{\{s_3\}}$, the algorithm may detect that only the first pattern: `*AW*JKL*` is shared by the third sequence and hence will obtain $P_{\{s_1,s_2,s_3\}} = \{$`*AW*JKL*`$\}$. This pattern is the fittest in the sense that it is the longest regular pattern that matches all three sequences.

Various SD methods differ in

1. the particular pattern space and the representation of patterns (e.g., local alignments may be used to represent the patterns),
2. the way the sets to be combined are chosen (i.e., in the methods for choosing i and j in the iteration step),
3. how the combination is done (dynamic programming, heuristics) and how the (fittest) patterns are chosen,
4. whether one, most, or all patterns/alignments are kept.

Joining the pattern sets can be done, for example, by using dynamic programming (Needleman and Wunsch, 1970; Smith and Waterman, 1981; Sankoff and Kruskal, 1983). Dynamic programming algorithms usually guarantee finding the fittest patterns common to the given pair of sets for a given fitness function. If negative examples are given (i.e., the classification problem), these can be taken into account in the iteration step (e.g., the patterns present in the negative examples may be excluded from the resulting set). The problems of pattern discovery and local multiple sequence alignment are very closely related, and some SD algorithms store local alignments instead of patterns during the iteration.

Note that SD algorithms that are based on combining pairs of pattern sets in each step, as in the example above (or, indeed, any limited number of sets) usually cannot be guaranteed to find the fittest patterns common to S_+ in the end, even if each iteration step is guaranteed to find the fittest pattern common to the pair of sets that are combined in the step. (This is similar to building multiple alignment by pairwise alignments, which also cannot guarantee finding the optimal alignment in the end.)

The earliest SD algorithms for finding a regular pattern (class **G**) common to a set of strings that we are aware of (Shinohara, 1983; Nix, 1983) were developed by the computational learning community, and do not have a direct relation to biocomputing. These algorithms are based on finding the longest common subsequence (LCS)[8] for pairs of sequences. The algorithm starts by finding the LCS of the two shortest sequences, and in the following steps takes the current shortest sequence and finds its LCS with the result of the previous steps. Although this algorithm is not guaranteed to find the LCS of the set of sequences, it has been proved in (Shinohara, 1983; Nix, 1983) that it *learns* the right regular pattern in the sense of *inductive inference* (Gold, 1967) [9] in polynomial time.

[8] By a subsequence of two sequences $a_1 \ldots a_n$, and $b_1 \ldots b_m$, we mean a sequence $c_1 \ldots c_k$, such that there exist $i_1 < \ldots < i_k$ and $j_1 < \ldots < j_k$ for which $c_1 \ldots c_k = a_{i_1} \ldots a_{i_k} = b_{j_1} \ldots b_{j_k}$.

[9] That is, if the sequences have been obtained from some given pattern by "filling-in" the wild-cards, and if the there are "many enough" and "wide enough variety" of such sequences given to the algorithm, then the algorithm

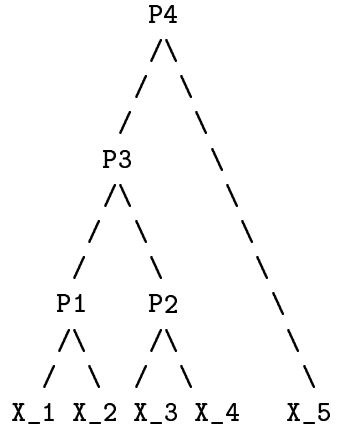

Figure 4: Example of dendrogram for sequences X_1, X_2, X_3, X_4, X_5. Pairs X_1, X_2, and X_3, X_4 are the most similar among themselves, and the sequence X_5 is the most different from any of the other sequences. The algorithm aligns X_1 to X_2, obtaining P_1, X_3 to X_4, obtaining P_2, then P_1 to P_2 obtaining P_3, and finally, P_3 to X_5 obtaining P_4. Patterns P_1, P_2, P_3, and P_4 match sequences which are below each of them, thus P4 matches all the sequences.

3.2.1. Best pair comparison based heuristics

An algorithm for finding patterns in biosequences based on pairwise comparison is given in Smith and Smith (1990). This approach uses the fact that pairs of sequences, as well as pairs of sequences and patterns, and pairs of patterns, can be aligned by dynamic programming algorithms. The algorithm also exploits the fact that the characters of the basic alphabet (i.e., Σ_p) can be organised in partially ordered hierarchical groups.

First, an estimated phylogenetic tree (so-called *dendrogram*) is built using the estimated relative distances among the sequences. For instance, a possible dendrogram of sequences X_1, X_2, X_3, X_4, X_5 where pairs X_1, X_2, and X_3, X_4 are the sequences most similar among themselves, but the sequence X_5 is the most different from any of the other, is given in figure 4. The pairs (sequence, sequence), or in later stages (sequence, pattern) or (pattern, pattern) are aligned at each node of the dendrogram starting bottom-up, and a common pattern is obtained from each pair via dynamic programming.

The result of aligning two characters is the character denoting the smallest possible group in the hierarchy containing both characters, which may already denote a group of basic characters. The scoring is positive, but decreases with groups higher up in the hierarchy[10]. Gaps are penalised as $w = w_0 + w_e \cdot k$, where w_0 is the gap opening penalty, w_e is the gap extension penalty, and k is the gap length. If, while aligning a pattern to a pattern, two gaps are aligned, only gap extensions (if needed) are penalised, but not the gap opening.

Note that pairwise alignments are guaranteed to give an optimal (i.e., the most specific) pattern

correctly restores the given pattern (or a pattern equivalent to the given) in time $O(l^4 n)$. Note that this in fact means that the approximation function f that is found by the algorithm is equivalent to the characteristic function g.

[10] In the AACC hierarchy, a match to a basic character is scored +3, at the next levels +2 and +1, and a match to a wildcard is scored 0.

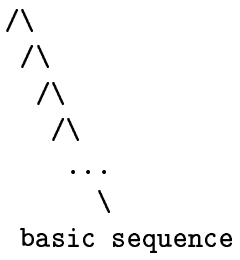

Figure 5: A dendrogram where one sequence is chosen as a basic sequence and all other sequences are aligned against it

common to the two sequences/patterns aligned, but this does not give any guarantee about the optimality of patterns higher up in the dendrogram with respect to all given sequences. In addition to the pattern that is common to all sequences, the algorithm also obtains patterns common to subsets of related sequences, therefore the algorithm can be also used for classification (in fact for unsupervised learning).

A different heuristic is developed by Roytberg (1992). One sequence is selected as the *basic sequence*, and all the other sequences (so-called *serial sequences*) are aligned against it. This approach corresponds to a dendrogram of the type given in figure 5. The algorithm finds the substrings in the basic sequence that have approximate matches in all, or in a specified percentage, of the serial sequences, additionally ensuring that the respective substrings from the serial sequences are similar to each other[11].

3.2.2. All pair comparison heuristics

A heuristic based on finding pairwise similarities between all pairs of sequences is described by Schuler *et al.*(1991). The algorithm begins by comparing all pairs of input sequences. It locates for each pair the substrings that score high enough, thus obtaining so-called 2-blocks[12]. Next, it attempts to extend such 2-blocks to three sequences. For this, it checks all pairs of 2-blocks having one sequence in common, and 3-blocks are extracted from those with similar enough parts in all three sequences. Then the same idea is applied to 3-blocks to extend them to 4-blocks and so on. Theoretically there may be exponentially many blocks to try, but in practice, if the threshold for similarity scores has been set high enough, the number of hypothesis is manageable. A very similar approach is also described in (Brodsky *et al.*, 1992).

A heuristic representing pairwise alignments by so-called *dot-matrices* is described by Vingron and Argos (1991). Given a pair of sequences $b_1 \dots b_l$ and $c_1 \dots c_m$, a dot matrix $A = [a_{i,j}]$ is a matrix of size $l \times m$ with elements $a_{i,j}$ defined as follows: $a_{i,j} = 1$ if $b_i = c_j$, otherwise $a_{i,j} = 0$.

[11] Note that the similarities are not necessarily transitive, i.e., the fact that some substring A from the basic sequence is similar to a substring B in a serial sequence X_b and to a substring C in a serial sequence X_c, does not necessarily mean that B is similar to C.

[12] By an *n-block* we mean an array of n substrings of equal length.

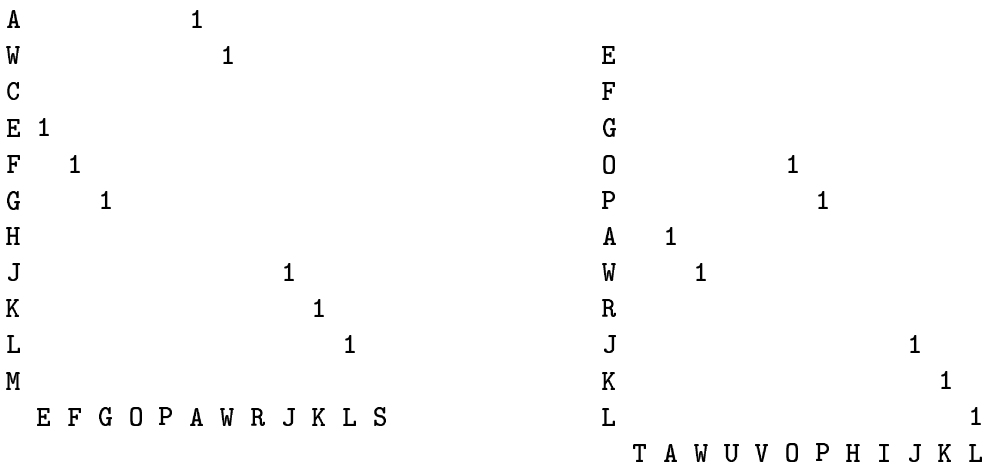

Figure 6: Dot-matrices of strings (4)-(5) and (5)-(6) (0 elements are left blank in the example)

For instance, the dot matrices 4-5 and 5-6 for the sequences (4), (5), and (6) are given in figure 6.

The algorithm calculates dot matrices for all pairs of sequences and filters out similarities (non-zero entries in the matrices) that are not consistent with the other dot matrices by using *Boolean multiplication*[13]. For instance, the result of Boolean multiplication for the matrices in figure 6 is given in figure 7. Note that the resulting matrix is different from the dot-matrix for the strings (4)-(6) in that only substrings present in all three sequences, namely `AW` and `JKL` have 1 in the respective positions (and not the matching character `H`).

In general, if sequences $X_1, \ldots, X_n$ are given, there exist $n(n-1)/2$ dot matrices $M_{1,1}$, $M_{1,2}$, $\ldots$, $M_{1,n}$, $M_{2,2}$, $M_{2,3}$, $\ldots, \ldots, M_{n,n}$. The matrix resulting from the Boolean multiplication $M^*_{k,m} = M_{k,l} \times M_{l,m}$ shows which substrings are common to all three sequences X_k, X_l, and X_m. By fixing k and m, and taking all possible l's (not equal to k or m), we can find substrings common to all strings. It is also possible to find all substrings common to X_k and X_l and at least a given number of other sequences by similar algebraic matrix manipulations. The time complexity of the algorithm is $O(l^3 n^4)$. Vingron and Argos (1991) describe a heuristic based on such matrix manipulations for finding "significant" (i.e., with relatively high fitness) substrings common to a majority of the sequences.

More general dot matrices can also be defined using real instead of boolean values representing the similarity scores between the positions. However, it should be noted that if a non-transitive similarity relation is used (e.g., defined from PAM or Blosum matrices), the algorithm may find sets of substrings some of which are not similar to each other (the algorithm guarantees 3-consistencies, but not k-consistencies for $k > 3$ (Freuder, 1978)).

After the dot matrices have been filtered, a directed graph is constructed with one node for each

[13] A *Boolean multiplication* of such matrices is defined the same way as ordinary matrix multiplication, except that the Boolean summation is used (i.e., 0+0=0, 0+1=1, 1+0=1, and 1+1=1).

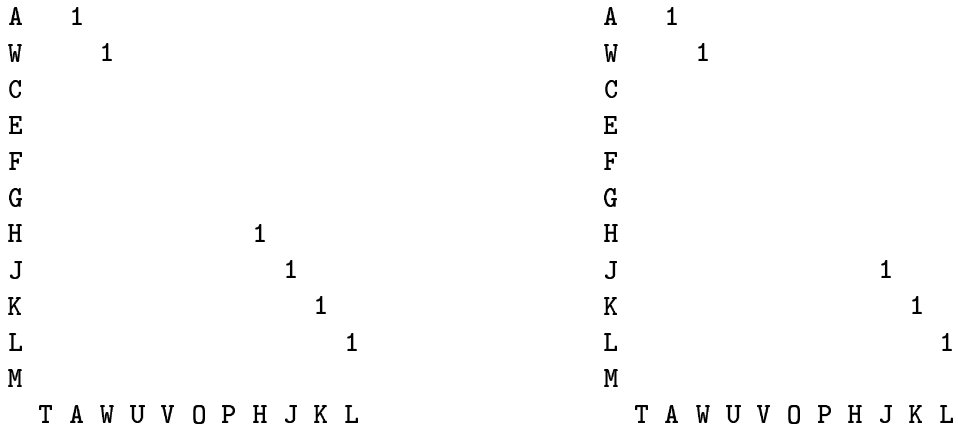

Figure 7: The dot-matrix for strings (4)-(6) (left), and the resulting dot-matrix from Boolean multiplication (4)–(5) $\times$ (5)–(6) (right)

possible local alignment of similar substrings, and edges between all nodes. Two special nodes (source and sink) are added, the source node corresponding to all sequence starts being aligned, and the sink node to all sequence ends being aligned. Each edge $u \rightarrow v$ in the graph is given a weight depending on the relative positions of the substrings corresponding to nodes u and v. Dijkstra's algorithm (Aho, *et al.*, 1983) is used to find the highest scoring path from the source to the sink node. This path defines a partial global alignment of the sequences.

It should be noted that an extremely efficient algorithm for finding the longest substrings common to at least k out of n given sequences can be based on *suffix-trees* (McCreight, 1976; Ukkonen, 1992; Hui, 1992). This gives an algorithm for the substring patterns (i.e., the patterns of class **Aa**), which is linear-time in the total length of the examples and independent of the length of the patterns. However, no very efficient suffix-tree based algorithms are currently known for finding approximately conserved substrings. A related approach use directed acyclic word graphs (DAWGs) instead of suffix trees (Clift *et al.*, 1986).

3.2.3. Algorithms for the classification problem

SD algorithms can also be used in the case when both positive and negative sequences are given (i.e., for the classification problem). This has been done by Kudo *et al.* (1992). The approach is primarily designed for finding patterns at gene splice-site 5' end, and since all such sites have a fixed position of 100% conserved GT, the sequences can be pre-aligned by aligning GT. The target language in (Kudo *et al.*, 1992) is a union of subwords, either with wildcards (i.e., **Bc** and without $*$), or in the more general case containing arbitrary combinations of basic characters (i.e., **Cc** and without $*$). The algorithm finds the least general set of patterns that covers all the positive examples, and does not contain any negative examples in iterative steps. Each step of the iteration introduces wildcard characters in order to unify some positive examples, but so that none of the negative examples is matched. More precisely, it attempts to unify by the

introduction of wildcards in non-matching positions, first pairs of positive examples, then triples from sequences contained in the successful pairs, then quadruples from successful triples, etc, until such unification is no longer possible without the inclusion of negative examples.

The problem of finding patterns from positive and negative examples has also recently been studied by Tateishi and co-workers (Tateishi and Miyano, 1995; Tateishi, *et al.*, 1995). They use a somewhat different definition of the classification problem. The positive and negative examples are provided in pairs $(pos_1, neg_1), (pos_2, neg_2), \ldots, (pos_n, neg_n)$, and the aim is to find a classification function able to distinguish between pos_i and neg_i for each i, but not necessarily between pos_i and neg_j for $i \neq j$. They show that the problem of finding a pattern maximising correctly classified pairs is still NP-hard. A greedy algorithm for approximating the solution for a simple pattern class $\mathbf{C}$ from pre-aligned sequences having the same length is given. Some heuristics for more complicated patterns are also proposed.

3.3. Combined approaches

The most obvious way to combine PD and SD approaches is to use SD for refining (expanding or combining) the patterns found by PD search. It is also possible to limit the search space prior to the search by using SD elements.

3.3.1. Using SD approach for refining PD found patterns

SD approach for refining PD found patterns can be used in a number of ways. The first way is to

- use a PD approach for spotting some candidate patterns,
- mark the position of the candidate patterns in the sequences,
- align the sequences so that the positions of candidate patterns are aligned together,
- finally extend the candidate pattern while the fitness of the emerging pattern is increasing.

This method is used in (Smith, *et al.*, 1990; Jonassen, *et al.*, 1995; Jonassen, 1997). A modification of this approach is used by Landraud, *et al.*(1989). In this algorithm first of all a variation of KMR is used to find all substrings present in at least k out of the given n substrings. In the next step, the substring having "the best" approximate similarities, in some precisely defined sense, in the remaining $n - k$ sequences, is picked out from the substrings found in the first step. The strings are aligned so that the respective substrings are aligned together in all the sequences. After that, the second step is repeated separately for the parts of the sequences that are to the left and to the right of the substrings used in the previous stage. This is repeated while possible, i.e., a divide-and-conquer strategy is used. A similar method has been used by Martinez (1988).

Another explicit way of combining PD and SD approaches is described by Ogiwara *et al.* (1992). The basic idea is to use a PD approach to find relatively short candidate substrings, to transform

the original sequences into different data structures consisting of these substrings joined by "gaps", and finally to align the data structures obtained and to extract the common patterns. Let us consider this approach in some more detail.

The algorithm is for the classification problem, i.e., it uses both positive and negative examples. All words of the given length are enumerated in a PD manner and a count is taken of in how many positive examples and how many negative examples each is present. In practice tetra-, penta-, and hexapeptide patterns (i.e., substrings of length 4 to 6) are counted. Only those strings that are present in at least f percent of positive examples, and in none of the negative examples are retained; in practice two cases: $f = 100\%$ and $f = 70\%$ are considered.

Next the positions of these words and their nearest neighbours are marked in the positive examples. In this case "nearest neighbour" means having no more than one difference (insertion, deletion or substitution). Thus the examples are transformed to new structures of the type: $p_{1,j}\, g_{1,j}\, p_{2,j}\, g_{2,j}\, \ldots p_{n,j}$, where $p_{i,j}$ are the frequent substrings, and $g_{i,j}$ are integers equal to the distance between the starting positions of i-th and $i+1$-th substrings in the j-th example. In the second stage the transformed examples are aligned by using heuristics of pairwise alignment. The output comprises consensus patterns of the type $p_1 - x(min_1, max_1) - p_2 - x(min_2, max_2) - \ldots - p_n$, where p_i are subwords, and $x(min_i, max_i)$ specifies the minimal and maximal distances (spacers) between the subwords.

Anoter way of refining patterns found by the PD approach is by grouping similar patterns together, aligning them, and trying to generalise from them. For instance, if substrings ... `aacaa` ... and ... `aagaa` ... are found to be frequently occurring in sequences, then a common pattern `aa`[`cg`]`aa` can be obtained from them. This kind of refinement is used in (Neuwald and Green, 1994; Saqi and Sternberg, 1994)

After having combined patterns, Neuwald and Green (1994) additionally calculate an *initial profile* from the (ungapped) alignment defined by the substrings matching a combined pattern (the simplest form of a profile is a position dependent scoring matrix, giving one score to each amino acid for each position in a segment to match the profile). The profile is iteratively refined by realigning the sequences to the profile, throwing away non-significant matches, and recalculating the profile.

In (Henikoff and Henikoff, 1991), a combined PD and SD algorithm is developed for finding frequent blocks in protein databases. The first stage simply uses the algorithm of (Smith, *et al.*, 1990), thus finding patterns and the respective blocks in a PD manner, and then extends them (see the beginning of this subsection). The positions of the patterns are marked on the initial examples. Next the "best" set of patterns that occur in the same order without overlapping in a critical number of sequences is found. Such an ordering is called a *path*. A graph is constructed where nodes represent patterns, and an arc extends from node b_1 to b_2 if pattern b_1 precedes pattern b_2 and does not overlap in at least the critical number of sequences. The graph is searched for the best path according to a defined scoring scheme. Note that this step is similar to the last step of Vingron and Argos' algorithm (Vingron and Argos, 1991).

3.3.2. Methods for limiting the search space prior to the search

A simple heuristic for limiting the search space can be based on an assumption that the patterns that are present approximately (within some distance) in many sequences are likely to be present in an exact form in at least some. This is not strictly true because the most fit pattern may be a kind of average, e.g., Steiner's sequence[14], itself not present in a single sequence. However if a sufficient number of sequences are given, it may be likely that at least one of the sequences will match the pattern exactly. Therefore a heuristic can be based on the enumeration of only those substrings that are present in at least one sequence. This reduces the search space drastically, since there are only $O(N^2)$ substrings for a set of strings with total length N. If the length of the substrings is bounded by l, then there are only $O(lN)$ patterns to be considered, instead of $O(|\Sigma|^l)$. This approach has been used by Saqi and Sternberg (1994), where a statistical significance measure has also been used for sorting out the interesting patterns. Additionally, after finding the most frequent substrings (Saqi and Sternberg, 1994) cluster the most similar ones, and generalise them to find more complicated patterns from their alignments, thus introducing the second SD element in their algorithm. The time complexity of this algorithm is $O(L^2 + t)$ where t is the time needed for clustering the similar substrings.

The described heuristic can be taken even further by using a random subset of the training set instead of the entire training set. If the number of sequences in the subset is large enough, then it is statistically likely that any substring that occurs approximately in sufficiently many sequences in the training set will occur in an exact form more than a certain number of times in the random subset[15]. Therefore it is sufficient to enumerate only those substrings that are present in many enough copies. Moreover, the strings in the subset can be represented as a *generalised suffix tree* (Hui, 1992), and then the potential candidates for the pattern can be selected in linear time (Wang *et al.*, 1994). Thus the algorithm becomes linear-time in the length of the sequences and the patterns.

A different way of limiting the search space is proposed by Jonassen (1997). Here a *pattern graph* is defined. A path in this graph corresponds to a set of patterns, and a depth-first search strategy is used to search for the paths corresponding to patterns matching at least N_{min} of the positive examples with the highest fitness. It is possible to derive a pattern graph from an existing multiple sequence alignment, for instance an alignment of a subset of the sequences in S_+, so that only patterns consistent with the alignment are considered. This gives a smaller search space, and can be considered as an SD-element.

3.4 Learning unions of patterns and decision trees

So far we have only considered algorithms for discovering string functions defined by one pattern from a subclass of generalised regular patterns (i.e., functions of type **a** and **b**). However, several algorithms have been reported that are able to discover more complex string functions based on unions of regular patterns or decision-trees over regular patterns, i.e., functions of type **c** and **d**.

[14] By Steiner's sequence for the set sequences $A_1, \ldots, A_k$, we understand a sequence B minimising $\sum_{i=1}^{k} distance(A_i, B)$. Note that B may not be any from $A_1, \ldots, A_k$.

[15] The necessarily size of the subset can be estimated by using random sampling theory.

Algorithms for discovering these classes of functions have been reported in (Arikawa *et al.*, 1993; 1992; Arimura *et al.*, 1994; Shoudai *et al.*, 1995), where the authors prove that the classes of these concepts can be learned from examples in polynomial time in the sense of inductive inference or PAC-learning. Unfortunately, the order of the polynomials is too high and therefore various heuristics have to be used in practical applications.

Arikawa *et al.* (Arikawa *et al.*, 1993; 1992) consider learning from both positive and negative examples, i.e., the classification problem. In (Arikawa *et al.*, 1993) the method for learning decision trees first introduced by Quinlan (Quinlan, 1986) is used. In (Arikawa *et al.*, 1992) an algorithm for learning so-called *elementary formal systems* has been developed. In practice only a special case of elementary formal systems is used, which in fact is the union of a bounded number of regular patterns. Aditionally a study is also made in (Arikawa *et al.*, 1993) of how the indexing of the basic alphabet can be performed automatically so that the classification of the positive and negative examples remains correct. The problem is proved to be NP-hard, and a heuristic for its approximation is given. Such an automated indexing allows to reduce the search space by reducing the alphabet.

In (Arimura *et al.*, 1994) a method for learning the union of an *a priori* bounded number of regular patterns from positive examples is developed. The algorithm finds the most specific union of pattern languages containing all positive examples. Note that in this approach the fact that the number of patterns in a union is bounded by some *a priori* given constant is essential, as otherwise the algorithm would simply return the union of all sequences in the training set. This sets an *a priori* limit on the number of subfamilies that can be discovered in the sequences. In the noisy data case this also means an assumption about the level of noise. Shoudai et. al. (Shoudai *et al.*, 1995) has similar limitations.

An algorithm for grouping the sequences and filtering out noise without an *a priori* assumption on the level of noise and the size of the groups has been developed by Wu and Brutlag (1995). On the other hand, this algorithm requires the sequences to be pre-aligned. The algorithm uses so-called *beam-search* method for splitting the training set into subfamilies in alternative ways and generating the candidate pattern.

An algorithm for discovering unions of an unbounded number of patterns and without any assumption on the level of noise from non-pre-aligned sequences is developed in (Brazma *et al.*, 1996b). The patterns are of the PROSITE type (i.e., class **F**). The algorithm uses a fitness measure based on the MDL principle (see section 2.3) to balance between how well a set of patterns covers the given examples, and how compact it is. The algorithm is based on the pattern discovery algorithm Pratt (Jonassen, 1997) and on a greedy set covering algorithm. The greedy algorithm guarantees finding the union of patterns within a logarithmic multiplier to the optimum. Experiments are reported showing that the algorithm correctly splits a family of biosequences into subfamilies discovering a strong pattern within each of the family, and thus effectively performs unsupervised learning (see section 2). For instance, the algorithm finds three subfamilies sharing common patterns in chromo domain family.

4. Conclusions

The aim of this work has been to give a survey of methods for the automatic discovery of patterns in biological sequences and to establish some systematisation of this area. For this reason we have designed a framework, which contains formalisations of the problems of pattern discovery and evaluation, and also classifications of pattern languages and algorithmic approaches. We have looked at algorithms for the discovery of deterministic patterns with expressive power inside the regular languages, chosing to describe some that we consider to be representative of the complete set. We have identified the main algorithmic ideas of each of these methods and shown how these ideas relate to each other. For practical purposes it would be valuable for each of the algorithms to define precisely what kinds of patterns it can "see" and to compare the existing algorithms on the basis of this criteria. We think that this survey is a step towards this goal.

While dealing with these problems we have noticed that different authors employ very different computational experiments to test their algorithms and to convince the reader of the usefulness or superiority of their algorithms. The number and lengths of the sequences used, the types of sequence families and the ways in which the results are presented vary greatly, although the algorithms are frequently intended to solve the same problem. We believe that it would be beneficial for the field if an attempt were to be made to establish some systematisation regarding which experiments could be used for testing the algorithms.

Although this survey shows that many nontrivial and efficient pattern discovery algorithms have been developed recently, biologists need considerably more powerful algorithms for efficient knowledge discovery in view of the growing volume of biosequence data. Algorithms are required which are able to discover more complicated or subtle patterns in larger training sets containing unknown levels of noise. We hope that this survey will help to move the field forward towards these aims.

Acknowledgements

The authors wish to thank Richard Lathrop, Darrel Conklin, Rein Aasland, and anonymous referees for helpful comments. Alvis Brāzma has been supported by the Finnish Centre for International Mobility (CIMO), the Latvian Council of Sciences (Grant Number 93.593), the Royal Society, and the Human Capital and Mobility programme of the European Union. Inge Jonassen and Ingvar Eidhammer have been supported by grants from the Norwegian Research Council. David Gilbert was supported by a grant from the British Council.

Appendix A: Algorithms and software

Key

Algorithms	
Pattern	Pattern type (see section 2.2.4)
Pre	Prealigned [Y/N]
G	Guaranteed [Y/N] (see section 2.4)
+/-	Uses positive and/or negative example training sets
Domain	DNA/protein/Not Applicable

Software	
Name	Name of the software
Src/Ex	Source or executable
Platform	Runs on what platform
Obtain	Obtain from: a/ftp=anonymous ftp; A=authors; n/a=not available; WWW=program can be run on the World-Wide-Web

Authors	Algorithms					Software			
	Pattern	Pre	G	+/-	Domain	Name	Src/Ex	Platform	Obtain
(Nix, 1983)	Ga	N	N	+	N/A				
(Shinohara, 1983)	Ga	N	Y	+	N/A				
(Waterman, *et al.*, 1984)	Ab	N	Y^{+}	+	protein, DNA				
(Martinez, 1988)	Gb	N	N	+	protein, DNA	GENALIGN			
(Landraud, *et al.* 1989)	Gb	N	N	+	protein, DNA				
(Staden, 1989b)	Ab	N	Y	+	DNA		Src (Fortran)	Vax VMS	n/a
(Smith and Smith, 1990)	Fa	N	N	+	protein				
(Smith, *et al.*, 1990)	Ba	N	Y	+	protein	MOTIF	Src (Turbo-C)	IBMPC	WWW*
(Vingron and Argos, 1991)	Ga [FILLOG/SUM]; Gb [FILMAXAV]	N	N	+	protein	unkown			A
(Kudo *et al.*, 1992)	Ba, Ca	Y	Y	+/-	DNA				
(Ogiwara *et al.*, 1992)	Ga	N	Y/N	+/-	protein				
(Roytberg, 1992)	Ab	N	N	+	protein, DNA	MuSCo		IBMPC, IBM/370	n/a avail
(Arikawa *et al.*, 1993)	Gd	N	N	+/-	protein				
(Neuwald and Green, 1994)	Da	N	N	+	protein	ASSET	Src	SPARC2	a/ftp
(Saqi and Sternberg, 1994)	Ca	N	N	+	protein				
(Wang *et al.*, 1994)	Gb	N	N	+	protein	DISCOVER, CLASSIFY	Ex	DOS, DEC Ultra, SunSPARC	A
(Jonassen, *et al.*, 1995)	Fa	N	Y/N	+	protein	Pratt	Src (C)	decalpha, sparc10	a/ftp
(Jonassen, 1997)	Fa	N	N	+	protein	Pratt2	Src (C)	UNIX, Linux, OS/2	a/ftp, WWW**
(Brazma *et al.*, 1996b)	Fc	N	N	+	protein	MDLPratt	Src (Perl/C)	UNIX, Linux	a/ftp
(Sagot, *et al.*, 1995b)	Ca	N	Y	+	protein				
(Sagot, *et al.*, 1995a)	Ab	N	Y	+	protein				
(Sagot and Viari, 1996)	Da	N	Y	+	protein, DNA				
(Shoudai *et al.*, 1995)	Fc, Fd	N	N	+/-	protein	BONSAI			n/a
(Suyama, *et al.*, 1995)	Ea	N	Y	+	protein	GAPE	Src (Fortran)	Sun	A
(Wu and Brutlag, 1995)	Ca	Y	N	+	protein	SEQCLASS	Sx, Common Lisp	Sun SPARC	n/a
(Wolferstetter *et al.*, 1996)	Ab	N	Y	+	DNA	CoreSearch	Src (C)	UNIX	a/ftp, WWW***

Notes:
+ Waterman *et al.*(1984) use a sliding window, and patterns are not only required to match a minimum number of sequences, but the matches are also required to be within this sliding window.
∗ URL: http://www.blocks.fhcrc.org/blockmkr/make_blocks.html
∗∗ URL: http://www.ii.uib.no/~inge/Pratt.html
∗ ∗ ∗ URL: http://www.gsf.de/biodv/consinspector.html

Appendix B: Input sequences (positive examples) of patterns discovered by some of the reported algorithms

Note: We base our notation on that of PROSITE, augmented with some additional symbols.

1. (Staden, 1989b)

Examples: 88 *E.coli* promoter sequences, varying in length from 47 to 64, having a total length of 5238 characters.
The patterns found most frequently to be approximately present in the sequences are:

t-t-t-t-t-t
t-t-a-t-a-a
t-t-g-a-c-a
t-c-t-t-g-a
t-a-t-a-a-t
a-c-t-t-t-a
a-a-a-a-a-a
a-g-t-a-t-a

2. (Smith and Smith, 1990)

Examples: 128 sequences of length between 141 and 147 from hemoglobin delta epsilon gamma beta major-chain sequences.
Pattern:

l-l-x(2)-a-x(3)-b-x(2)-c-x(5)-G-x-l-x-a-x-l-c-c-a-a-c-P-W-l-l-R-b- F-x(2)-F-G-x-c-x-l-x(3)-a-x(2)-l-x(2)-a-x(3)-G-x-i-a-x(3)-c-x(3)-c- x-l-c-l-x-a-x(3)-c-x(2)-L-S-l-x-H-x(3)-c-x(2)-l-x(2)-l-F-l-x-c-G- x(2)-c-a-x(2)-c-x(7)-F-x(4)-l-x(2)-c-l-i-c-x(3)-a-x(2)-p-L-x(3)-Y

Examples: 12 sequences from Trypsinogen/Venom serine proteases.
Pattern:

l-l-l-h-x-a-a-G-G-x(2)-C-x(2)-l-x(2)-P-b-x(3)-c-x(4)-i-x(0,1)-F-C- G-x-k-L-l-x(3)-W-V-a-k-A-p-H-C-x-l-x(2)-c-l-a-i-L-G-l-x(6)-l-x(2)- E-x-c-x(6)-c-x(2)-P-l-x-l-x(3)-c-l-l-x(0,1)-T-l-c-L-l-i-L-x(4)-l-x- l-a-x(2)-a-x-L-P-l-x(5)-G-l-x(3)-a-x-G-W-G-x(3)-l-g-x(5)-l-x(2)-l-C- x-l-x(2)-a-c-x-l-x(2)-C-l-x(2)-Y-x-G-x(0,1)-a-x(2)-l-x-c-C-x-G-c-c- l-G-G-x-D-k-C-x-G-D-S-G-G-P-a-a-x-l-G-x-c-Q-G-a-a-S-W-G-x(2,3)-C-A- x(4)-

P-p-c-x(2)-l-V-c-l-b-a-x-W-l-l-l-x-a-A

The lower case letters denote classes from the AACC hierarchy as follows: a=[ILV], b=[FWY], c=[ILVFWYCM], h=[DE], i=[HKR], j=[NQ], k=[ST], l=[DEHKRNQSTBZ], p=[AG].

3. (Smith, *et al.*, 1990)

Examples: 15 sequences from DNA integrases.
Pattern:

x(15)-H-x-L-R-H-x(2)-A-x(6)-G-x(6)-Q-x(2)-L-G-H-x(2)-l-x(2)-T-x(2)-Y-x(5)

4. (Kudo *et al.*, 1992)

Positive examples: 496 pre-aligned DNA segments of length 9 from around the $5'$ splice site (three in the exon and six in the intron).
Negative examples: 1123 DNA segments of length 9 (all containing gt in position 4-5).
Some of the best patterns discovered are (in class B):

<x-a-g-g-t-a-a-x-x>
<a-a-g-g-t-x-a-g-x>
<c-x-x-g-t-a-a-g-x>

and (in class C)

<x-[agc]-[agc]-g-t-a-a-g-x>
<[agc]-x-[agc]-g-t-a-a-g-x>
<x-[agc]-x-g-t-a-a-g-[tgc]>

5. (Ogiwara *et al.*, 1992)

Examples: sequences from cytochrome b5 family
A partially conserved pattern found:

H-P-G-G-E-E-V-L

Examples: sequences from a family of L-lactate dehydrogenase
A partially conserved pattern found:

P-V-D-[IV]-L-x(47)-G-[EQ]-H-G-D

Examples: sequences in a family of glyceraldehyde-3-phosphate dehydrogenases
A completely conserved pattern found:

G-F-G-R-I(0,1)-G-R-x(129,134)-S-N-A-S-C-T-T-N-[CS]-L-A-P- x(14)-[LM]-M-T-T-V-H-x(30,31)-T-G-A-A-[KR]-A-[VT]-x(92,95)- [SA]-W-Y-D-N-E

6. (Saqi and Sternberg, 1994)

Examples: a set of heat shock proteins
Some of the patterns found:

x-G-G-G-T-F-D-[ILV]-[ST]-[ILV]
x-[ILV]-[FWY]-D-L-G-G-G-T-F-D-[ILV]
D-[LF]-G-G-G-T-F-D

Examples: a set of toxin proteins
Some of the patterns found:

x(2)-C-C-x(4)-C-x
D-R-C-C-x(2)-H-D-x-C

7. (Neuwald and Green, 1994)

Examples: a set of 56 sequences of acyltransferases with an average length of 471.
Some of the patterns found:

V-x-P-x(2)-[RQ]-x(4)-G-x(2)-L-[LM]
N-x(2)-A-x(3)-Y-x(3)-G-F

8. (Wang *et al.*, 1994)

Examples: 47 sequences of length 190-780 in a group of cyclic proteins
Some of the patterns found:

L-Q-L
I-A-S-K-Y-E-E
D-T-A-G-Q-E-*-L-V-G-N-K

9. (Sagot, *et al.*, 1995a)

Examples: 80 proteins belonging to the elongation family
46 patterns found

10. (Shoudai *et al.*, 1995)

Examples: 3796 signal peptides indexed to the three-letter alphabet Σ_{hydro} of maximum length 32.
Classified in three groups of sizes 2205, 640, and 603, by patterns:

2-*-2-*-0-2-*-1-*-0-2
1-0-*-0-*-0-*-2-1-*-0
2-2-2-*-1-2-*-1-2

where 0,1,2 stands for different amino acid groups in different patterns (see (Shoudai *et al.*, 1995) for details).

11. (Jonassen, *et al.*, 1995)

Examples: 241 protein sequences from the zinc finger c2h2 family, average length 393

Pattern:

C-x(2,4)-C-x(3)-[ILVFYC]-x(8)-H-x(3,5)-H

Examples: 164 protein sequences from the snake toxin family, average length 64
Pattern:

G-C-x(1,3)-C-P-x(8,10)-C-C-x(2)-[EPDN]

Examples: 27 protein sequences containing PHD finger, average length 874
Pattern:

C-x(2,4)-C-[YCEPGSDNQR]-x-[VMFWHTAPGSN]-x-H-x(2)-C-[ILVMFYHTCA]-x(11)- [YWCEPGSDNQ]-x(2)-[IFHCAPGSDN]

12. (Brazma *et al.*, 1996b)

Examples: 31 chromo domain sequence segments
Union of patterns:
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] ∪
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 ∪
Y-x(0,2)-L-[IV]-K-W-x(6)-[HE]-x-[TS]-W-E-x(4)-[IL]

References

Aho, A. V.; Hopcroft, J. E.; and Ullman, J. D. 1983. *Data Structures and Algorithms.* Addison-Wesley.

Altschul, S. F.; Gish, W.; Miller, W.; Myers, E. W.; and Lipman, D. J. 1990. Basic local alignment search tool. *J. Mol. Biol.* 215:403–410.

Arikawa, S.; Kuhara, S.; Miyano, S.; Shinohara, A.; and Shinohara, T. 1992. A learning algorithm for elementary formal systems and its experiments on identification of transmembrane domains. In *Proc. 25th Hawaii Int. Conf. on System Sci.*, 675–684.

Arikawa, S.; Miyano, S.; Shinohara, A.; Kuhara, S.; Mukouchi, Y.; and Shinohara, T. 1993. A machine discovery from amino acid sequences by decision trees over regular patterns. *New Generation Computing* 11:361–375.

Arimura, H.; Fujino, R.; Shinohara, T.; and Arikawa, S. 1994. Protein motif discovery from positive examples by minimal multiple generalization over regular patterns. In *Proc. of the 5th Genome Informatics Workshop*, 39–48.

Bailey, T. L., and Elkan, C. 1995. The value of prior knowledge in discovering motifs with MEME. In *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology*, 21–29. AAAI Press.

Bailey, T. L. 1995. *Discovering motifs in DNA and protein sequences: the approximate common substring problem.* Ph.D. Dissertation, University of California, San Diego, USA.

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

Baldi, P.; Chauvin, Y.; Hunkapiller, T.; and McClure, M. M. 1994. Hidden Markov Models of Biological Primary Sequence Information. *Proc. Natl. Acad. Sci USA* 91:1059–1063.

Brazma, A.; Ukkonen, E.; and Vilo, J. 1996a. Discovering unbounded unions of regular pattern languages from positive examples. In *Proceedings of 7th Annual International Symposium on Algorithms and Computation (ISAAC-96), Lecture Notes in Computer Science 1178*, 95–104.

Brazma, A.; Jonassen, I.; Ukkonen, E.; and Vilo, J. 1996b. Discovering patterns and subfamilies in biosequences. In *Proc. of Fourth International Conference on Intelligent Systems for Molecular Biology*, 34–43. AAAI Press.

Brodsky, L. I.; Vassilyev, A. V.; Kalaydzidis, Y. L.; Osipov, Y. S.; Tatuzov, R. L.; and Feranchuk, S. I. 1992. Genebee: the program package for biopolymer structure analysis. In Gindikin, S., ed., *Mathematical methods of analysis of biopolymer sequences, DIMACS series in discrete mathematics and theoretical computer science, volume 8*. American Mathematical Society.

Bucher, P., and Bairoch, A. 1994. A generalized profile syntax for biomolecular sequence motifs and its function in automatic sequence interpretation. In *Proc. of Second International Conference on Intelligent Systems for Molecular Biology*, 53–61.

Chan, S. C.; Wong, A. K. C.; and Chiu, D. K. Y. 1992. A survey of multiple sequence comparison methods. *Bull. Math. Biol.* 54(4):563–598.

Clift, B.; Haussler, D.; McConnell, R.; Schneider, T. D.; and Stormo, G. D. 1986. Sequence landscapes. *Nucl. Acids Res.* 14(1):141–158.

Dayhoff, M. O. 1978. *Atlas of Protein Sequence and Structure*, volume 5. National Biomedical Research Foundation.

Freuder, E. C. 1978. Synthesizing constraint expressions. *Comm. ACM* 21(11):958–966.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY: W. H. Freeman.

Gold, E. M. 1967. Language identification in the limit. *Information and Control* 10:447–474.

Gribskov, M.; McLachlan, M.; and Eisenberg, D. 1987. Profile analysis: detection of distantly related proteins. *Proc. Natl. Acad. Sci. U.S.A* 84:4355–4358.

Henikoff, S., and Henikoff, J. G. 1991. Automated assembly of protein blocks for database searching. *Nucl. Acids Res.* 19(23):6565–6572.

Henikoff, S., and Henikoff, J. G. 1992. Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. USA* 89:100915–100919.

Hui, L. C. K. 1992. Color set size problem with application to string matching. In A.Apostolico, M. Chrochemore, Z., and U.Manber., eds., *Proc. of Combinatorial Pattern Matching*, 230–243. Springer-Verlag.

Hutchinson, A. 1994. *Algorithmic Learning*. Clarendon Press.

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

Jonassen, I.; Helgesen, C.; and Higgins, D. G. 1996. Scoring function for pattern discovery programs taking into account sequence diversity. Reports in Informatics 116, Dept. of Informatics, University of Bergen.

Jonassen, I. 1997. Efficient discovery of conserved patterns using a pattern graph. *Comput. Applic. Biosci.* in the press.

Karlind, S., and Ghandour, G. 1985. The use of multiple alphabets in kappa-gene immunoglobulin DNA sequence comparison. *The EMBO Journal* 4:1217–1223.

Karp, R. M.; Miller, R. E.; and Rosenberg, A. L. 1972. Rapid identification of repeated patterns in strings, trees and arrays. In *4th ACM Symposium on Theory of Computing*, 125–136.

Kristensen, T.; Lopez, R. S.; and Prydz, H. 1992. An estimate of the sequencing error frequency in the DNA sequence databases. *DNA Seq.* 2:343–346.

Krogh, A.; Brown, M.; Mian, I. S.; Sjoelander, K.; and Haussler, D. 1994. Hidden Markov model in computational biology. Applications to protein modelling. *J. Mol. Biol.* 235:1501–1531.

Kudo, M.; Kitamura-Abe, S.; Shimbo, M.; and Iida, Y. 1992. Analysis of context of $5'$-splice site sequences in mammalian mRNA precursors by subclass method. *Comput. Applic. Biosci.* 8(4):367–376.

Landraud, A. M.; Avril, J.-F.; and Chretienne, P. 1989. An algorithm for finding a common structure shared by a family of strings. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 11(8):890–895.

Lathrop, R.; Webster, T.; Smith, R.; Winston, P.; and Smith, T. 1993. Integrating AI with sequence analysis. In Hunter, L., ed., *Artificial Intelligence and Molecular Biology.* AAAI Press/The MIT Press. 211–258.

Lawrence, C. E., and Reilly, A. A. 1990. An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. *Proteins: Struct. Funct. Genet.* 7:41–51.

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.

Levenshtein, V. I. 1966. Binary codes capable of correcting deletion, insertions, and reversals. *Cybernetics and Control Theory* 10:707–710.

Li, M., and Vitanyi, P. 1993. *An Introduction to Kolmogorov Complexity and its Applications.* New York: Springer-Verlag.

Lipman, D. J., and Pearson, W. R. 1985. Rapid and sensitive protein similarity searches. *Science* 227:1435–1441.

Martinez, H. M. 1988. A flexible multiple sequence alignment program. *Nucl. Acids Res.* 16(5):1683–1691.

McCreight, E. M. 1976. A space–economical suffix tree construction algorithm. *J. ACM* 23:262–272.

Needleman, S., and Wunsch, C. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.* 48:443–454.

Neuwald, A. F., and Green, P. 1994. Detecting patterns in protein sequences. *J. Mol. Biol.* 239:689–712.

Nix, R. P. 1983. *Editing by Example.* Ph.D. Dissertation, Yale University, Xerox Palo Alto Research Center, California, USA.

Ogiwara, A.; Uchiyama, I.; Seto, Y.; and Kanehisa, M. 1992. Construction of a dictionary of sequence motifs that characterize groups of related proteins. *Protein Engng.* 5(6):479–488.

Posfai, J.; Bhagwat, A. S.; Posfai, G.; and Roberts, R. J. 1989. Prediction motifs derived from cytosine methyltransferases. *Nucl. Acids Res.* 17(7):2421–2435.

Queen, C.; Wegman, M. N.; and Korn, L. J. 1982. Improvements to a program for DNA analysis: a procedure to find homologies among many sequences. *Nucl. Acids Res.* 10:449–456.

Quinlan, J. R. 1986. Induction of decision trees. *Machine Learning* 1:81–106.

Rissanen, J. 1978. Modeling by the shortest data description. *Automatica-J.IFAC* 14:465–471.

Roytberg, M. A. 1992. A search for common patterns in many sequences. *Comput. Applic. Biosci.* 8(1):57–64.

Sagot, M. F., and Viari, A. 1996. A double combinatorial approach to discovering patterns in biological sequences. In Hirschberg, D., and Myers, G., eds., *Combinatorial Pattern Matching*, 186–208. Springer-Verlag.

Sagot, M.-F.; Viari, A.; and Soldano, H. 1995a. A distance-based block searching algorithm. In et al, C. R., ed., *Proc. of Third International Conference on Intelligent Systems for Molecular Biology*, 322–331. Menlo Park, California: AAAI Press.

Sagot, M.-F.; Viari, A.; and Soldano, H. 1995b. Multiple sequence comparison: a peptide matching approach. In Galil, Z., and Ukkonen, E., eds., *Proc. of 6th Annual Symposium on Combinatorial Pattern Matching*, 366–385. Springer-verlag.

Sankoff, D., and Kruskal, J. B. 1983. *Time Warps: String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison.* Addison-Wesley.

Saqi, M. A. S., and Sternberg, M. J. E. 1994. Identification of sequence motifs from a set of proteins with related function. *Protein Engng.* 7(2):165–171.

Schuler, G. D.; Altschul, S. F.; and Lipman, D. J. 1991. A workbench for multiple alignment construction and analysis. *Proteins: Struct. Funct. Genet.* 9:180–190.

Sewell, R. F., and Durbin, R. 1995. Method for calculation of probability of matching a bounded regular expression in a random data string. *J. Comp. Biol.* 2:25–31.

Shinohara, T., and Arikawa, S. 1995. Pattern inference. In Jantke, K. P., and Lange, S., eds., *Algorithmic learning for knowledge-based systems, GOSLER final report.* Springer-Verlag. 259–291.

Shinohara, T. 1983. Polynomial time inference of extended regular pattern languages. *Lecture Notes in Computer Science* 147:115–127.

Smith, R. F., and Smith, T. F. 1990. Automatic generation of primary sequence patterns from sets of related protein sequences. In *Proc. Natl. Acad. Sci. USA*, 118–122.

Smith, T., and Waterman, M. 1981. Identification of common molecular subsequences. *J. Mol. Biol.* 147:195–197.

Smith, H. O.; Annau, T. M.; and Chandrasegaran, S. 1990. Finding sequence motifs in groups of functionally related proteins. In *Proc. Natl. Acad. Sci. USA*, volume 87, 826–830.

Staden, R. 1989a. Methods for calculating the probabilities of finding patterns in sequences. *Comput. Applic. Biosci.* 5:89–96.

Staden, R. 1989b. Methods for discovering novel motifs in nucleic acid sequences. *Comput. Applic. Biosci.* 5(4):293–298.

Suyama, M.; Nishioka, T.; and Oda, J. 1995. Searching for common sequence patterns among distantly related proteins. *Protein Engng.* 8(11):1075–1080.

Shoudai, T.; Lappe, M.; Miyano, S.; Shinohara, A.; Okazaki, T.; Arikava, S.; Uchida, T.; Shimozono, S.; Shinohara, T.; and Kuhara, S. 1995. BONSAI Garden: parallel knowledge discovery system for amino acid sequences. In et al, C. R., ed., *Proc. of Third International Conference on Intelligent Systems for Molecular Biology*, 359–366. Melno Park, California: AAAI Press.

Tateishi, E., and Miyano, S. 1995. A greedy strategy for finding motifs from positive and negative examples. Technical Report RIFIS-TR-CS-118, Research Institute of Fundamental Information Science, Kyushu University, Japan.

Tateishi, E.; Maruyama, O.; and Miyano, S. 1995. Extracting best consensus motifs from positive and negative examples. Technical Report RIFIS-TR-CS-115, Research Institute of Fundamental Information Science, Kyushu University, Japan.

Taylor, W. R. 1986. The classification of amino-acid conservation. *J. Theoret. Biol.* 119(2):205–218.

Ukkonen, E. 1992. Constructing suffix trees on–line in linear time. *Information Processing* 1:484–492.

Valiant, G. L. 1984. A Theory of the Learnable. *Comm. ACM* 27(11):1134–1142.

Vingron, M., and Argos, P. 1991. Motif recognition and alignment for many sequences by comparison of dot–matrices. *J. Mol. Biol.* 218:33–43.

Wang, L., and Jiang, T. 1994. One the complexity of multiple sequence alignment. *J. Comp. Biol.* 1(4):337–348.

Wang, J. T. L.; Marr, T. G.; Shasha, D.; Shapiro, B. A.; and Chirn, G.-W. 1994. Discovering active motifs in sets of related protein sequences and using them for classification. *Nucl. Acids Res.* 22(14):2769–2775.

Waterman, M. S.; Arratia, R.; and Galas, D. J. 1984. Pattern recognition in several sequences: Consensus and alignment. *Bull. Math. Biol.* 46(4):515–527.

Wolferstetter, F.; French, K.; Herrmann, G.; and Werner, T. 1996. Identification of functional elements in unaligned nucleic acid sequences by a novel tuple search algorithm. *Comput. Applic. Biosci.* 12(1):71–80.

Wu, T. D., and Brutlag, D. L. 1995. Identification of protein motifs using conserved amino acid properties and partitioning techniques. In et al, C. R., ed., *Proc. of Third International*

Conference on Intelligent Systems for Molecular Biology, 402–410. Melno Park, California: AAAI Press.