REPORTS
IN
INFORMATICS

ISSN 0333-3590

Approaches to the automatic discovery of patterns in biosequences

Alvis Brāzma, Inge Jonassen,
Ingvar Eidhammer, David Gilbert

REPORT NO 113 December 1995

Department of Informatics

UNIVERSITY OF BERGEN

Bergen, Norway

Approaches to the automatic discovery of patterns in biosequences.

Alvis Brāzma
Department of Computer Science, University of Helsinki, Finland

Inge Jonassen, Ingvar Eidhammer
Department of Informatics, University of Bergen, Norway

David Gilbert
Department of Computer Science, City University, London, England

Abstract

This paper is a survey of approaches and algorithms used for the automatic discovery of patterns in biosequences. Patterns with the expressive power in the class of regular languages are considered, and a classification of pattern languages in this class is developed, covering those patterns which 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 presented of the ways in which an assessment can be made of the significance and usefulness of the discovered patterns. It is shown that this problem is related to problems studied in the field of machine learning. The largest part of this paper comprises a review of a number of existing methods developed to solve this 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.

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

Introduction

Biological macromolecules, DNA's, RNA's, and proteins, are chains of relatively small organic molecules. The different types of these organic molecules are few – there are 4 different bases for DNA's and RNA's and 20 different amino-acids for proteins. A macromolecule can be coded as a string over an alphabet of size 4 (for DNA/RNA), or 20 (for proteins) starting from one end of the chain and moving towards the other. The strings for DNA/RNA molecules are called *nucleotide sequences*, and each element in such a sequence is called a *base*. Similarly, the strings for protein molecules are called *protein sequences*, and each element in such a sequence is an *amino-acid* (*residue*). Collectively nucleotide and protein sequences, are called *bio-sequences*, or simply *sequences*.

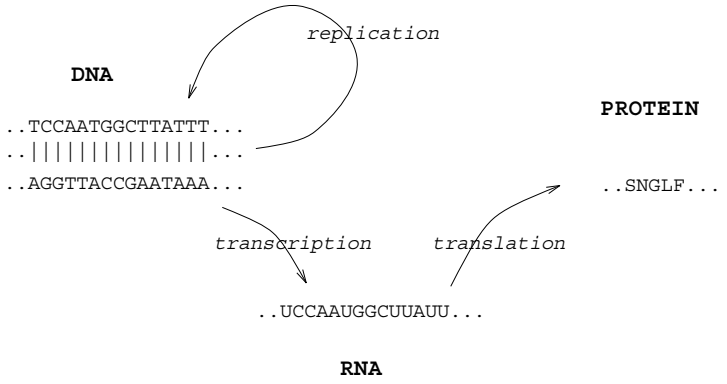

Figure 1: *The central dogma* of molecular biology illustrated. The DNA-molecules contain genes. Each gene encodes a macromolecule, either a protein or an RNA-molecule. When a gene codes for a protein, it is first used for generating a single stranded RNA-molecule (called a messenger RNA – mRNA), which contains the codes for producing the protein. The process of producing a protein using the information in a mRNA molecule, is called translation. A triple of consecutive bases in the mRNA, code for one amino-acid, and consecutive triples code for amino-acids that will be consecutive in the resulting protein. The figure shows an example of the process of making one protein.

The genome of a living organism can be given as a set of nucleotide sequences, which will contain amongst other things the organism's genes. Figure 1 shows the relationship between DNA and proteins in living cells. Protein and RNA molecules fold up into three-dimensional structures which are determined by the sequences. The three-dimensional structure determines the function of the macromolecule. (For an introduction to molecular biology, see [ABL+94], for more information on protein structures; see [BT91])

Recently it has become relatively cheap and easy to determine nucleotide and protein sequences, and a considerable number of sequences has been amassed, with a total length of several hundreds of millions of characters. There is a large number of different databases containing sequence data. For instance, the EMBL nucleotide sequence database (release 44.0, November 1995) contains more than 180,000 nucleotide sequences, and the total number of bases in this release is nearly 200 millions; the number is estimated to double every 18 months. The SWISS-PROT protein sequence database [BB92] (release 32.0) contains more than 49,000 entries, with more than 17 million amino-acids in total. There are also databases of three dimensional structure of the proteins, but they are much smaller. For instance, the PDB (Protein Data Bank) contains descriptions of three-dimensional structures of biological macromolecules (DNA, RNA, and proteins); release 95.11 (December 1995) contains about 4,000 entries.

At present the main bottle-neck to progress in molecular biology is the analysis of data, and not the acquisition of sequence data. The aim of this analysis is the extraction of all kind of biological "meaning" of these sequences, for example the evolutionary history of the respective macromolecules, and their three-dimensional structure and function. Unfortunately no general solution to this problem is currently known. One particularly important problem is that of determining the three dimensional structure of proteins, and since performing this experimentally is very work-intensive and expensive, a key problem is to try to predict the structure from its sequence. No general and accurate method is known for solving this problem.

One way of analysing the sequences, is to group them in *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 that can be expressed purely in terms of the sequences. We refer to descriptions of such common "syntactic" features as *patterns*.

Different kinds of patterns can be used for characterising sequences. We focus on deterministic patterns, i.e. patterns for which it is always possible to determine purely from the sequence if it has this property or not. We say that a sequence *matches* the pattern, if it has this property. For instance, a pattern may be a substring, and a sequence match this substring pattern, if it contains the substring.

If a common pattern is discovered in a set of biologically related sequences it is possible that the presence of this particular pattern is important for the biological function of the corresponding macromolecule. For example, it may be essential for the tree-dimensional structure of the molecule. Also if we detect the presence of an earlier discovered pattern in a new sequence, we can infer that the new sequence belongs to the same family, even if we do not know its biological properties yet. In this way patterns may be used for the classification of bio-sequences and for predicting their properties. A pattern is said to be *diagnostic* for the family if it matches all the known sequences in the family, and no other known sequences. And a pattern is said to be a *motif* for a specific family if it matches every sequence in the family, and such a pattern is said to be *conserved* in the family.

Many of the known protein families have been collected in the PROSITE database [Bai92]. For most of the families in PROSITE, 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 gives the zinc finger c2h2 family containing 236 proteins in SWISS-PROT. The pattern `C`-x(2,4)-`C`-x(3)-[`LIVMFYWC`]-x(8)-`H`-x(3,5)-`H` given, matches all 236 known member sequences, and also 20 other sequences in SWISS-PROT.

Currently PROSITE patterns are extracted semi-manually. Apart from the fact that this is a tedious process, in this way, there is no guarantee that all the possible patterns are explored and that the "best" patterns are found. Methods for the automatic discovery of patterns may provide better (more diagnostic) patterns [JCH95], and such methods also make it possible to test many more hypotheses on the family relationships of sequences. Moreover, the problem of automatically discovering patterns goes beyond just discovering patterns for PROSITE database. Efficient methods for solving this problem can provide molecular biologists with very valuable tools and may help in understanding the biological properties of the macromolecules. A number of such methods have already been developed and it seems that the field would benefit from some systematisation.

The problem of the automatic extraction or discovery of patterns common to a set of sequences is in essence 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 extract a general rule from these examples. Thus the pattern discovery algorithm is in essence a machine learning algorithm, and the set of examples (both positive and negative) is the *training set*.

In this survey we develop a common framework underlying most of the existing approaches to automatic pattern discovery that we are aware of. We partly formalise the problem, and link it to related problems studied in the field of machine learning. We present a classification of the patterns that the existing methods are able to discover, and we discuss possible and reported methods for the ranking of discovered patterns, and ways to assess their relevance to biological "reality". The main part of the survey is a description of existing algorithms for pattern discovery. We try to do this in a systematic way, and to show how the different approaches are related.

In Appendix A we present some basic information about the specific existing algorithms, and in Appendix B we give, for some of the existing methods, examples of the results of computational experiments, and samples of the patterns which have been discovered. 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 of biosequences. We regard this survey as a step towards the systemisation of the area, as well as an attempt to present this problem to the wider community, including the machine and algorithmic learning community and the computer science community in general.

2. Definition and discussion of the problem.

Here we discuss in general terms the problem of learning from biosequences. First we describe two different, but related problems of learning family descriptions. Next we go on to discuss different possible solution spaces, i.e. different classes of family descriptions, focusing on consensus patterns. We discuss how to evaluate and rank a set of discovered patterns, and finally we discuss whether family descriptions based on consensus patterns have enough expressive power to allow description of the crucial biological features defining the families.

2.1 The two problems

We describe two different, but related problems. The first is how to find a *classifier function* for a family of biosequences. This is a function which takes a sequence as argument, returning TRUE for members of the family, and FALSE for non-members. Both positive examples (members of the family) and negative examples (sequences not in the family) are given as a training set. In the second problem only positive examples (family members) are given, and the goal is to extract a description of features *conserved* in (i.e., characterising) the family. This is to be encapsulated into what we will call a *conservation function*. Both problems involve finding functions having specific properties which we define and discuss below. In machine learning terminology the problem corresponds to "learning from positive and negative examples", and "learning from only positive examples". Different algorithms that have been constructed for solving each of these problems are described in section 3.

2.1.1 Classification problem

Suppose F_+ is a family of related sequences. The biologist may want to find a function f, which can be used to decide for a new sequence, whether or not it belongs to F_+. Unfortunately not all family members will be known. The biologist will have a set of sequences S_+ believed to be members of the family, and a set of sequences S_- believed not to be members of the family.

Assuming that all sequences in S_+ are correct and from F_+, and that no sequences in S_- are from F_+ (clean data), the problem can be stated:

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

By compact we mean having a short description. What is meant by a 'short description' and what is meant by 'high likelihood' is not defined precisely here, we will discuss the ways of defining these notions later.

As stated, **C1** in fact consists of two parts:

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

C1b: try to predict the properties of sequences not yet known.

Note that this resembles very closely one of the classic interpretations of machine learning (inductive inference), where a theory is viewed as a compact description of past observations together with predictions of future ones [Sol64]. Normally the sets F_+ and F_- together will define the "total sequence set" $U = F_+ \cup F_-$, for example the set of all sequences in SWISS-PROT [BB92]. In this case we will define the complement $\overline{F}$ of a set F ($F \subset U$) to be $U - F$.

The solution to problem **C1** depends on two things. The first is to find a good class (for the particular problem) of classification functions, we call this class the *target class*, *solution space*, or *hypothesis space*. The second is to design an efficient algorithm which, given S_+ and S_-, searches the solution space and tries to find functions returning TRUE for sequences in S_+ and FALSE for sequences in S_-. In this case solving the "prediction" part would depend on the choice of target class. If we chose too general target class, for instance including the regular expression $\cup_{s \in S_+} s$, it could lead to overfitting the training set. Such over-fittings should be rejected by using a less general class of target functions.

A more subtle way for solving this problem is additionally to define a ranking of the solution space evaluating how good each function is in respect to the training set, and to develop an algorithm returning those classifier functions that rate high enough according to the ranking. For the classification problem the ranking of the solution space is likely to work well, if it rates the simpler (i.e., having shorter description) functions higher. This is a well known principle in machine learning – called the Occam's Razor principle (see [Hut94]).

In general we cannot assume perfect (clean) input. The data comes from biological experiments, and may contain errors, the sequences themselves may contain errors [KLP92], and sequences may have been wrongly included in the set of positive or negative examples. Ideally, algorithms should allow for some noise in the input (training set).

When allowing for noisy data, the problem becomes:

N1: Suppose there exist two disjoint sets of sequences F_+ and F_- ($F_+ \cap F_- = \emptyset$). Given sets $S_+ \subset F_+, S_- \subset F_-$, find compact classifier functions of sequences such that they return TRUE for most sequences in S_+, FALSE for most sequences in S_-, and have high likelihood of returning TRUE for the sequences in F_+, and high likelihood of returning FALSE for the sequences in F_-.

The modification of "for all" to "for most" requires the learning algorithm to find classification functions when the training set contains errors. This complicates the situation, since now we cannot choose the functions only among ones returning correct TRUE/FALSE for the entire training set. We need to find a balance between how well the classifier function fits the training set (i.e., how well it explains the past observations), and how high is its ranking according to some ranking of the solution space (e.g., how short a description it has – Occam's Razor principle). This problem is often solved by assuming a certain level of noise (say, 30%), and then, among the functions correctly classifying at least this portion of the training set, the ones with the highest ranking is chosen. An alternative to this could be using of the Minimum Description Length (MDL) principle [LV95].

2.1.2 Conservation problem

Sometimes it is interesting 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 conserved in the family. This can give valuable information about sequence patterns being conserved between the family members.

Let us call such a function a *conservation function*, and 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, one function being more interesting than another if it has a lower probability of matching random sequences[1].

C2: Suppose there exists a set of sequences F_+. Given a subset $S_+ \subset F_+$, find interesting conservation functions of sequences being conserved in S_+, such that they have high likelihood of returning TRUE for the sequences in F_+.

How do we decide which is the best of several a conservation functions? What we want is in some sense the *least general*, or *most specific*, function returning TRUE on F_+, which is

[1] For random sequences we assume some distribution, for example assuming 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., $p_a = f_a$.

characterising at least certain aspects of this family. Unfortunately the entire F_+ is not known, and if we over-fit S_+, this may lead to a trivial hypothesis, for instance $\cup_{s \in S_+} s$. This can be avoided by defining a sufficiently narrow solution space. An alternative, more sophisticated way, would be to define a ranking of the solution space simultaneously reflecting how simple and how interesting the functions are, and design algorithms finding the best functions according to this ranking. Such a ranking may be based on the MDL principle.

Allowing for noisy data, we need to find functions not necessarily conserved in the complete set S_+. Maybe some sequences have been included in S_+, that do not belong to the family, or maybe some of the sequences contain errors. Allowing for some errors (noise), we still want to be able to find the functions conserved in the family. This means that we need to find functions conserved in subsets of S_+.

More formally:

N2: Suppose there exist a set of sequences F_+. Given a subset $S_+ \subset F_+$, find interesting conservation functions of sequences being conserved in most of S_+, such that they have high likelihood of returning TRUE for the sequences in F_+.

Allowing for noisy data, it becomes even more difficult to decide between different functions. Different functions may work for different subsets of S_+. And they may have different probabilities of returning TRUE random sequences. The algorithm should be able to report a set of "good" solutions, leaving to the domain expert (biologist) to interpret the results.

2.1.3 The two problems are related

The classifier and the conservation problem are closely related. Let an instance of the classifier problem **C1** be given as (S_+, S_-), and let the function f be a solution with high fitness. Is f a solution to the instance (S_+) of the conservation problem **C2**? If $F_+ \cup F_- = U$, i.e., if F_- is the set of sequences in the total set which are not in the family, then f will be a conservation function for the family F_+. The function f will be conserved in the set S_+, and it will have a high likelihood of returning TRUE for sequences in F_+, because it is a classifier function for (F_+, F_-). Because $F_- = U - F_+$, and because f has a low likelihood of returning TRUE for sequences in F_-, f will be interesting (in the sense defined above). If the set $F_- \neq U - F_+$, the solution function f need not be interesting. That is, it can have a high likelihood of returning FALSE for sequences in F_-, and still not have a low probability of returning TRUE for random sequences. Normally the set F_- will be the set of all sequences not in the family F_+.

On the other hand, let f be a conservation function for the family F_+ found by using only the positive in examples S_+. Will f be a classifier function for F_+ and hence have a high likelihood of returning FALSE for non-members? If f is interesting, f will have a low probability of returning TRUE for random sequences. And if the set of non-member sequences is random, then f will be a classifier function for F_+. There may be sequences outside F_+ that are very similar to sequences in F_+. If one wishes for classifier functions returning FALSE for these, an algorithm for the classifier problem should be used, and negative examples similar to members in F_+ should be included in S_-.

As an analogy, imagine that we are making algorithms for learning classifier and conservation functions for physical (macro-) structures. We may want the algorithm to find a classifier function for 'chairs'. The positive examples will be a set of descriptions of chairs (as diverse as possible), and the negative examples should include descriptions of random structures and of structures similar to chairs, like stools and tables. The resulting classifier function would also be a good conservation function for chairs. On the other hand, a conservation function found from only positive examples of chairs, might not be able to discriminate between chairs and stools or tables, even if it returns FALSE for most non-chair structures. If F_- does not contain a random set of physical structures, e.g., if F_- contains only stools, then the resulting classifier functions may not be conservation functions.

The same is true for the domain of bio-sequences. If one wants a classifier function for alpha-globins, beta-globins should be included as negative examples. A conservation function for alpha-globins might not be good at discriminating between alpha- and beta-globins.

This means that the set of conservation functions for a family F_+ is a superset of the set of classifier functions for F_+ (vs some set F_-) when F_- contains a "random" subset. When this is the case, all possible classifier functions are conservation functions, while there may be conservation functions that are not classifier functions. This is not surprising since it is natural that using both positive and negative examples, can make learning "sharper" than just from positive examples.

2.2 Solution spaces.

Here we discuss different ways of defining the functions.

2.2.1 DNA vs protein

So far we have discussed conservation/classifier functions for bio-sequences in general. 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

$$\Sigma_p = \{\texttt{D,E,K,R,H,Q,S,T,I,L,V,F,W,Y,C,M,A,G,P}\}. \tag{1}$$

- Nucleotide sequences (DNA/RNA) are sequences over 4-letter alphabets

$$\Sigma_{DNA} = \{\texttt{a,t,g,c}\}, \tag{2}$$

  and

$$\Sigma_{RNA} = \{\texttt{a,u,g,c}\}. \tag{3}$$

The set of amino acids Σ_p may be grouped (possibly overlapping groups) in different ways according to their physio-chemical properties, e.g., AACC hierarchical groups from [SS90] ($K_1 =$

{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,M}, K_8 = {A,G}, K_9 = {D,E,K,R,H,Q,S,T}, K_{10} = {I,L,V,F,W,Y,C,M}, and $K_{11} = \Sigma_p$). The Venn-diagram given in [Tay86] specifies a number of overlapping groups of amino-acids, each group containing the amino-acids having a certain physio-chemical property in common (for instance, there are groups of tiny, small, hydrophobic, and polar amino-acids). Also, matrices have been defined giving statistics for each pair (a, b) of amino acids, how often a and b are found in equivalent positions in proteins at a certain evolutionary distance (the evolutionary distance is a measure of how far back in history the proteins have a common ancestor). These matrices are called *substitution matrices*, and the most frequently used are probably the PAM [Day78], and the Blosum [HH92] matrices. (A range of other matrices have also been defined, e.g. based on similarity of physio-chemical properties between the amino-acids.) The nucleotides may be divided into two groups; pyrimidines (c and t) and purines (a and g), and scoring matrices may also be defined in this way. However, scoring matrices and groupings seem to play a more important role in analysis of protein sequences.

Both protein sequences and nucleotide sequences can be translated into smaller (some times called abstract) alphabets. Such a translation is called an *indexing*, and is obtained by making a partition[2] 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 $\Sigma_{hydro} = \{+, 0, -\}$. Similarly nucleotide sequences can be translated into a purine-pyrimidine alphabet $\Sigma_{n_{red}} = \{R, Y\}$.

2.2.2 Hierarchy of solution spaces

Douglas Brutlag gave a keynote address with the title "Where is the information in biological sequences", at the third international conference on Intelligent Systems for Molecular Biology (ISMB-95) in Cambridge, UK. Among other things, he discussed different ways of modelling families using functions corresponding to what we have called classifier and conservation functions. He made a classification of different functions ranging from statistical to deterministic functions:

```
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 are not strict. The exact ordering between these depends on the detailed definitions of the models and the profiles used. Each of these models has application fields for which is better suited. More on the statistical functions can be found in [KBM+94, BCHM94] (on Hidden Markov Models), [GME87, BB94] (profiles), and [CWC92] (alignments). In this review we focus on the deterministic end, consensus patterns in the class of regular languages.

[2] 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.

2.2.4 Consensus patterns

Let us define a class of what we call *generalised regular patterns* (GRP), which effectively includes all the classes of patterns we will consider in this paper. The specific pattern class of each particular algorithm will be obtained taking a particular subclass of GRP, and we will introduce a classification of patterns for this purpose.

Let $\Sigma = \{a_1, \ldots, a_m\}$ be *basic alphabet*. When protein sequences are analysed, $\Sigma = \Sigma_p$, and when nucleotide sequences are analysed, $\Sigma = \Sigma_{DNA}$, or $\Sigma = \Sigma_{RNA}$. If an indexing is applied, Σ may be for example Σ_{hydro} or $\Sigma_{n_{red}}$.

Let $K_1, \ldots, K_n$ be subsets of Σ, such that each contain at least two elements ($|K_i| \geq 2$). Let $\Pi = \{b_1, \ldots, b_n\}$ be another alphabet disjoint with Σ, and let us define a function $L_1(b_i) = K_i$. For convenience, let us assume, that $L_1(a_i) = \{a_i\}$, for $a_i \in \Sigma$. In practice $K_1, \ldots, K_n$ are classes of amino-acids or nucleic acids, and $b_1, \ldots, b_n$ are the characters denoting these classes. For instance, the AACC classes can be used. In practise, 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 AACC hierarchy, b_1 denoting $K_1 = \{\texttt{D,E}\}$, 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.

Let X be a set of all objects of the type $x(p, q)$, where p and q are nonnegative integers, such that $p \leq q$ (i.e., $X = \{x(p, q) | p \in \mathbf{N}, q \in \mathbf{N}, 0 \leq p \leq q\}$). Let

$$L_1(x(p, q)) = \{\alpha \in \Sigma^* | p \leq |\alpha| \leq q\},$$

i.e., $x(p, q)$ effectively means wildcard of a flexible length from p to q.

If $\pi = c_1 \ldots c_r$ is a string over an alphabet $\Sigma \cup \Pi \cup X$, then define $L_1(\pi) = L_1(c_1) \ldots L_1(c_r)$, where $L_1(a)L_1(b) = \{\alpha\beta | \alpha \in L_1(a), \beta \in L_1(b)\}$. A *GRP* is a string of the type

$$*\pi_1 * \pi_2 * \ldots * \pi_k *,$$

or of one of the types $\pi_1 * \pi_2 * \ldots * \pi_k *$, or $*\pi_1 * \pi_2 * \ldots * \pi_k$, or $\pi_1 * \pi_2 * \ldots * \pi_k$, where $\pi_1 \ldots \pi_k$ are strings over an alphabet $\Sigma \cup \Pi \cup X$. We define the language

$$L(*\pi_1 * \pi_2 * \ldots * \pi_k *) = \{\gamma_1\beta_1\gamma_2\beta_2\gamma_3 \ldots \gamma_k\beta_k\gamma_{k+1} | \beta_i \in L_1(\pi_i), i = 1, \ldots, k; \gamma_j \in \Sigma^*, j = 1, \ldots, k+1\}.$$

Note that $*$ actually means $x(0, \infty)$, i.e., a wildcard of arbitrary length. We will introduce a classification scheme based on this definition along two dimensions.

Along the first dimension we will distinguish between patterns of the types:

1. π_1,
2. $*\pi_1 *$ (or $\pi_1 *$, or $*\pi_1$)
3. $*\pi_1 * \pi_2 * \ldots * \pi_k *$ (or $\pi_1 * \pi_2 * \ldots * \pi_k *$, or $*\pi_1 * \pi_2 \ldots * \pi_k$, or $\pi_1 * \pi_2 \ldots * \pi_k$).

where $\pi_i \in \Sigma \cup \Pi \cup X$ $(i = 1, \ldots, k)$.

Along the second dimension we distinguish between case when π_i (from 1-3) are restricted to the alphabets:

A: $\pi_i \in \Sigma^*$,

B: $\pi_i \in (\Sigma \cup \{x\})^*$,

C: $\pi_i \in (\Sigma \cup \Pi)^*$,

D: $\pi_i \in (\Sigma \cup X)^*$,

E: $\pi_i \in (\Sigma \cup \Pi \cup X)^*$

Note that these alphabets define a partial ordering of A to E. A is contained in B, C, D, and E, B is contained in D and E, C is contained in E, and D is contained in E. This means that, for instance all patterns in 2B are also in 2D and 2E.

For example, 1A is the set of strings over Σ, 2A is the set of substring patterns (i.e., a presence of a certain subword in a string), 3A means extended regular patterns [Shi83]. 2B are substring patterns with wildcards [SAC90, NG94]. 2E are the patterns of the type used in PROSITE database [Bai92]. Formally, this definition does not cover all the cases permitted in the description of PROSITE patterns[3]. Nevertheless, it covers the PROSITE type patterns used most frequently in practice. 1E is effectively the set of PROSITE patterns attached to both the start and the end of the sequence.

In PROSITE notation the leading and closing $*$ symbols are not used. So $*\pi*$ is written simply as π, and PROSITE notation allows for attaching the pattern to the beginning or the end of a sequence by using leading $<$ or closing $>$ symbol, thus π becomes $<\pi>$. In PROSITE the individual symbols from the pattern alphabet are separated by dash symbols $-$. For instance, the PROSITE pattern `C-x(2,4)-C-x(3)-[LIVMFYWC]-x(8)-H` matches any sequence containing a substring starting with `C`, followed by between 2 and 4 arbitrary symbols, followed by `C`, and 3 arbitrary symbols, followed by one of `L, I, V, M, F, Y, W`, or `C`, followed by 8 arbitrary symbols, followed by `H`.

Now let us see how classification (conservation) functions can be defined by using these patterns. The simplest way to define such a function is

$$f(\sigma) = \begin{cases} \text{TRUE} & \text{if } \sigma \in L(P) \\ \text{FALSE} & \text{otherwise,} \end{cases}$$

where P is a pattern.

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 distance [Ste94]).

[3] The syntax defined for PROSITE patterns allows for description of a flexible number of non-wildcard symbols, for instance [DE](2,3). We have not found any PROSITE patterns using this kind of construction.

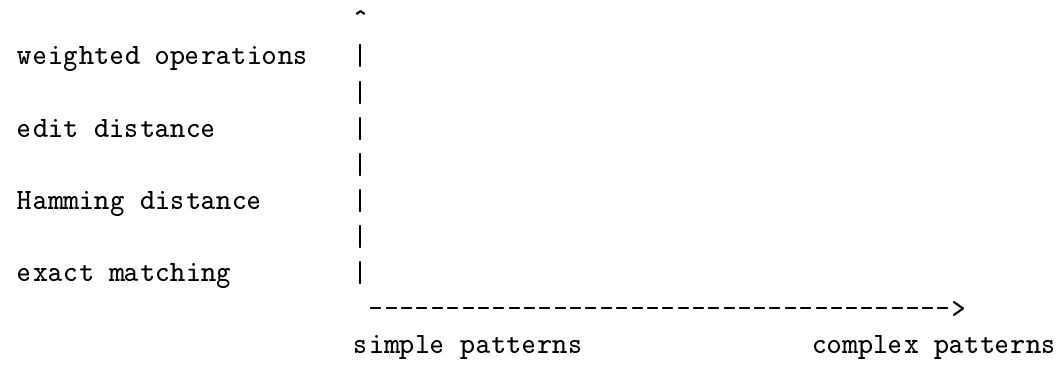

Figure 2: The diagram below illustrates two different axis measuring the modelling power of classifier functions based on patterns. The first axis represents the expressional power of the pattern language itself, and the second axis represents the way a string is matched against the pattern.

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

$$Dist(P, \sigma) = \min_{\sigma' \in L(P)} dist(\sigma', \sigma).$$

Now the definition of classifier/conservation function f can be generalised to include approximate matching:

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

for some given constant *const*.

The distance measure used can allow for only substitution operations (corresponding to matching with k mismatches), or for both substitutions and insertions/deletions (corresponding to matching with k differences). The operations can be weighted, and instead of the number of operations applied, one can limit the total weight of the operations.

Exact/approximate matching.
There are at least two different approaches to describing the variation allowed in biosequences; using a simple pattern language (for example, class 2A, substring patterns) and approximate matching, or a pattern language with more expressive power (for example class 2E) and exact matching. The two approaches can be combined; approximate matching may be necessary even if the pattern language is strong enough to describe the allowed variation. See figure 2.

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

c membership of a union of pattern languages (i.e. $f(\sigma) = \text{TRUE}$, if $\sigma \in L(P_1) \cup \ldots \cup L(P_n)$ where $P_1, \ldots, P_n$ are patterns of some class.)

d by using decision trees over patterns (for definition of a decision tree see [AMS+93]).

2.3 Ranking discovered patterns

As noted above, it is important to have some "good" ranking of the solution space, in respect to the training set, so that the best (one or few) patterns may be presented to the user. We will call a measure used for ranking the patterns a *fitness measure*. Below we discuss how such measures can be defined.

Formally a fitness measure can be defined as a function

$$F(p, T) \to \mathcal{R}$$

which takes two arguments and returns a real. The first argument is a pattern from a particular pattern class $\mathcal{C}$. The second argument is a training set (i.e., a set of sequences in the conservation function case, and a two sets of sequences in the classification case). The real returned by the function shows how good the pattern is in respect to the training set. The reals can be normalised to $[0, 1]$.

The fitness function has two aspects. The first is how well the pattern fits the training set. In the case of clean data (i.e., for problems **C1** and **C2**), this aspect is easy and is essentially a "black and white" situation. If the pattern is good for the training set (i.e., if the pattern matches all sequences in S_+, and for the classifier problem; no sequences in S_-), then $F(p, T) = 1$, otherwise $F(p, T) = 0$. In the case of noisy data it is not so easy, since we should not reject the patterns making some "mistakes" on training set. A simple way to deal with the problem, which is used in most of the existing discovery algorithms, is to assume a certain upper bound on the level of noise, say 30%, and to reject all the functions which are not good for more than this percentage of the training set.

An alternative way is to define the fitness of a pattern as the number of positive examples matching the pattern (exactly or approximately). For instance, if the patterns are subwords (i.e., of the type 2A) of length 3 in the alphabet Σ_{DNA}, then all for each word `aaa`, `aac`, `aag`, ..., `ttt`, we can count in how many positive examples it is present. This number can be regarded as the fitness. If negative examples are given, the fitness should also depend on the number of negative examples matched.

If the wildcard character (x) or characters denoting groups of characters from the basic alphabet, are allowed in the patterns, the fitness of a pattern P should be defined in a more complicated way. Therefore, a more subtle rating of the functions is done by using the second aspect of the fitness function, which should rate patterns "as such". In the case of classification functions this is usually done by using some kind of Occam's Razor, which basically means rating simpler patterns higher. (Simplicity can be defined as some approximation to Kolmogorov complexity, basically meaning the length of the shortest possible description of the object [Hut94].) In case of conservation functions, on the other hand, it is common to choose a sufficiently narrow solution space (avoiding over-fitting the training set), and to rate higher more specific patterns, which basically leads to rating longer patterns higher.

In this way, the fitness functions can be computed as in two steps, the first being to assess how

well it fits the training set, and the second, how good the pattern is as such. Some way of partially merging these steps (for the conservation problem) is considering the size (i.e., area) of the "block" covered by the pattern in the training set. Another, more subtle way to balance these two aspects, which is not yet fully explored in context of bio-pattern discovery would be using MDL principle [LV95, BUV95].

Some reported fitness functions.
Next we discuss in some more detail, how some authors have defined fitness functions. Let us start with the conservation case. An example of a fitness function based on the pattern itself, is the function used in [JCH95]. They define a measure of information contents of the identified pattern. Basically the measure increases with increasing pattern length, more specific pattern positions contributing more. Others have used the information contents of the *block*[4] defined by the segments in the training set matching a pattern, as a ranking function [LAB+93, Sta89b]. In all of these papers a pattern has to match some minimum number of the positive examples, and the number of sequences matching a pattern are not be taken into consideration when the patterns are to be ranked.

If the discovered patterns match more or less arbitrary subsets of the examples, more sophisticated fitness measures should be defined. Suppose $p_1, \ldots, p_n$ are patterns such that each p_i matches a subset S_i of the training set S_+. Methods for calculating the probability that a pattern (of different types) matches a random sequence (assuming some distribution) have been reported (e.g.,[WAG84, Sta89a, NG94, SD95]). We hypothesise that the sequences are independent (i.e., unrelated), and that sequence positions (symbols in the sequences) are independent (call this *the null model*). Then, for a pattern p_i, the *pattern probability* is the probability that p_i matches at least $|S_i|$ out of $|S_+|$ sequences purely by chance (under the null model). This pattern probability can be used to rank the patterns, patterns having lower probability should be ranked higher. Note that this measure contain elements of both aspects (the pattern as such, and the training set).

When the patterns are being chosen from a big set P of patterns, we can define the *significance* of a pattern p as the probability of (under the null model) finding at least one pattern in the set P having pattern probability less than (or equal to) that of p [WAG84, NG94]. In other words, the significance value v for p is the probability of finding a pattern as improbable as p in P, and is lower when a bigger set of patterns is used. Note that the significance of a pattern p will be a function of the pattern itself, the training set, and the set P of patterns from which p was chosen. The same pattern p discovered to match a subset of the sequences in S_+, will get different significance values when different sets of patterns (all containing p) are used. Specifically, if p is chosen from a set P of patterns, it will get a lower (better) significance value than if p was discovered when a bigger class P' is explored. Note that both significance values and pattern probabilities are real values (in $\mathcal{R}$) between 0 and 1, a lower value meaning that the pattern is more "surprising", and hence should be ranked higher.

These measures were defined for the conservation problem, but can also be applied to solutions of the classifier problem, when the two problems are related (see section 2.1.3). Alternatively, the *diagnostic power* of a pattern can be quantified. Assume that the the total set $U = F_+ \cup F_-$ is

[4] 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.

defined. Let (S_+, S_-) be an instance of the classification problem, and let $p_1, \ldots, p_n$ be patterns produced by a pattern discovery program for this problem instance. For each pattern p_i we can count the number of *false positives* and *false negatives.* A false positive is a sequence in F_- ($F_- = \overline{F_+}$) matching P_i (i.e. $s \in F_-$ and $f(s)$ =TRUE, where f is the classification function), and a false negative is a sequence in F_+ *not* matching P_i (i.e., $s \in F_+$, and $f(s)$ =FALSE). If a pattern has no false positives or negatives, it is *diagnostic* for the family. The following is from [LWS+93]. The *sensitivity* of a pattern can be defined as

$$Sn(P_i) = \frac{|F_+|}{|F_+| + fn} \tag{4}$$

where F_+ is the set of known sequences in the family, and where $|F|$ is the cardinality of the set F, and fn is the number of false negatives. Similarly, the *specificity* of a pattern can be defined as

$$Sp(P_i) = \frac{|F_-|}{|F_-| + fp} \tag{5}$$

where fp is the number of false positives, and where F_- is the set of known sequences not in te family. Recall that $F_- = \overline{F_+} = U - F_+$. A pattern gets maximum sensitivity score (1.0) if it matches all family sequences, and it gets maximum selectivity score (1.0) if it matches no sequences outside the family. The discovered patterns may be ranked according to sensitivity, specificity, or some combination. One possible combined measure is the *correlation coefficient* between two sets; 1) the set of sequences in F_+, and 2) the set of sequences in the total set U that matches the pattern P_i. The correlation coefficient is

$$C = \frac{|F_+| \cdot |F_-| - fp \cdot fn}{\sqrt{(|F_+| + fp) \cdot (fp + |F_-|) \cdot (|F_-| + fn) \cdot (fn + |F_+|)}}, \tag{6}$$

which is 1.0 when there are no false positives or negatives, and decreases towards zero as the number of false positives and negatives grow. Note that the measures (4)-(6) depend on the size of F_+ and the set of sequences matching the pattern, and also on the number of sequences in F_-.

2.4 Search algorithms and guarantees

The specification of the problems **C1**, **C2**, **N1**, and **N2** were rather informal, leaving undefined a number of notions including 'high likelihood' and 'most sequences'. Therefore, these specifications cannot be considered as precise specifications for pattern discovery algorithms. On the one hand it may not be compulsory to give a precise specification for this problem, since the ultimate test of these algorithms will in any case be, whether the results produced are interesting for biologists and helpful for solving some problems in biology. Still, it is helpful to understand better what might be a more formal specification for pattern discovery algorithms.

One of the requirements in the problems **C1**, **C2**, **N1**, and **N2** is to predict "future". It is difficult to use this requirement in a formal specification directly, therefore in reality more modest requirement is used as a specification for pattern discovery algorithms.

The algorithm is designed for a specific pattern class $\mathcal{C}$ and uses a specific fitness function F. The input of the algorithm is the training set T, and the algorithm is required, given the training

set T, to produce a set of patterns P from the class $\mathcal{C}$ such that the fitness value $F(p,T)$, for $p \in P$ is "relatively" high. If the algorithm can be proven to produce the specified portion of the patterns with the highest fitness value $F(p,T)$ among all the patterns in $\mathcal{C}$ (i.e., either the one fittest pattern, or a given number or percentage of the fittest patterns, or all the patterns with the 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 future mostly relies on choosing an appropriate pattern class $\mathcal{C}$ and a good fitness function F. In reality, this means, that we need to chose a pattern class, and a fitness function, in some sense reflecting the biological significance.

How well do the patterns predict the future?
If the family is big enough, a random subset of the family (and a subset of the set of sequences outside the family) can be used as a training set. The resulting patterns can be tested against the rest of the sequences, evaluating for example the sensitivity, or specificity, of the patterns. If a function performs well on the sequences not in the training set, then it is more likely to perform well on new sequences. This is a standard evaluation method used in experimental machine learning. For most families known at present, the number of known sequences is too small for performing experiments like this. As more sequences become known, this approach can be applied more widely.

Biological significance
An expert user (biologist with knowledge about the family) may be able to assess how likely a classifier function is to provide correct classification for unknown family members. Hence it is an advantage if the algorithm produce several alternative classifier functions to be presented to the user.

When new methods for pattern discovery are presented, the authors often apply it to well known families to show that it is able to recover already known conserved patterns. This also gives us a way to decide between different fitness functions; a fitness function should give the highest value to patterns that are already known to be of biological significance. Biological and statistical significance are not always the same. The ultimate test of a pattern is to check experimentally whether it corresponds to some region conserved in the family for structural and/or functional reasons. Biologists knowing the family well, may be able to decide between patterns which are likely to be of biological importance.

2.5 Do the solution spaces give sufficient expressive power?

Suppose there exists some function $g : \Sigma^* \to \{\text{TRUE, FALSE}\}$ (i.e., a classification function for sequences from some subset of Σ^*). Furthermore, suppose that this function is "computed" (in some sense) by a biological system. For instance, the function may be the classification of all primary structures of protein molecules in *transmembrane* proteins and *non-transmembrane* proteins. Such a classification is usually "performed" by a biological system (living cell) correctly, although the TRUE/FALSE value may not be defined for all sequences. Suppose, we are given sets S_+ and S_- of examples of sequences (i.e., strings from Σ^*) such that for $g(s)$ =TRUE for all $s \in S_+$, and $g(s)$ =FALSE for $s \in S_-$ (this assumes clean data, but our considerations can

be easily generalised for the noisy case).

Now, our aim is to find some class $\mathcal{F}$ of computable functions, and some function $f \in \mathcal{F}$, such that f approximates g in a way that f and g returns the same value for as many sequences s as possible. Note that in general we cannot hope to find the function g directly, since it may not be computable on a digital computer. Thus, in fact, our problem consists of two steps, first to find a good class of functions $\mathcal{F}$ and second, to find a good approximation function $f \in \mathcal{F}$. To find a good class of approximation functions is very important since there is always a trade-off between what classes of functions can be learned efficiently, and what classes are expressive enough to allow description of the crucial biological features of sequences. If we take too general a target class (e.g., the class of all context sensitive grammars), then the task of learning may be hopeless. On the other hand, if we take too simple target class, it may not be expressive enough for expressing the interesting properties, i.e. approximate g. Different classes $\mathcal{F}$ may be appropriate for different problems, i.e., for different kinds of function g. We want to chose $\mathcal{F}$ as "sharp" as possible for the particular problem. After a class $\mathcal{F}$ has been chosen, it is also important to find a good ranking of the functions in this class, reflecting the biological significance of different functions from this class.

Note that after we have chosen the class $\mathcal{F}$, the problem of finding the function f approximating g from the given sets of positive and negative examples S_+ and S_- can be regarded as the PAC-learning (Probably Approximately Correct) problem in machine learning. A good survey regarding PAC-learning of regular pattern languages and the applications to molecular bioinformatics is given in [SA95].

Let us focus on the problem of finding classifier/conservation functions for protein families. The proteins in a family share some common structure and/or function. As the function and structure of the proteins are determined by the sequences, the sequences of the proteins in a family may share some features, the kind depending on the specific protein family. There are many examples in the PROSITE database [Bai92] of families having diagnostic consensus patterns. Typically the proteins in such a family share a function or structure, that depends critically on the sequences having specific amino acids in certain positions. For example, the zinc finger proteins all have cysteine or histidine amino acids in certain positions. In these cases generalised regular patterns may be sufficient.

In many other families there are no completely conserved positions, meaning that features conserved between the sequences are more subtle. For example, helix-turn-helix (HTH) domains are formed by sequences with great variability. The pattern of conservation is very subtle. There does probably not exist any consensus pattern diagnostic for the HTH family. For such families, it may be possible to find statistical functions (e.g. profiles or HMMs) describing or classifying the family. Recently use has begun to be made of profiles to classify PROSITE families [BB94], implying that patterns in the class of regular languages may not be sufficient to classify these families.

In some families, the correlations between sequence elements play an important role in determining the structure and function of the macromolecules. For example, in ribosomal RNA there are patterns of secondary structure involving the single stranded nucleotide chain folding up and forming Watson-Crick base pairs with itself. Correlations probably also play a role in proteins [KB94]. To make adequate classifier functions for this kind of families, we may need to use

models describing the correlations. This takes us beyond the class of regular languages, and also beyond what is describable using HMMs, profiles. Models describing correlations have been made using stochastic context free grammars and covariance models [SBU+93, ED94]. Patterns of correlations exist (e.g. pseudo-knots in RNA secondary structures) that give crossing dependencies taking us beyond what can be described using context-free grammars.

3. Algorithms

In this section we describe the main ideas behind the algorithms used for solving the problems formulated in previous sections, i.e., for automatically finding conservation or classification functions for a set of sequences. We will not review each algorithm and each paper separately, instead, we will try to give a common thread winding trough all (or the majority) of the algorithms.

On the highest level we can divide the basic ideas in two groups. The first, which we call bottom-up (BU) approaches, is based on enumerating the candidate patterns. For this, we should:

- define the pattern space (i.e., the solution space $\mathcal{C}$),
- enumerate all (or at least many of) the patterns in the solution space,
- calculate the fitness of each pattern in respect to the given examples,
- report the fittest patterns.

For instance, if the patterns are subwords (i.e., of the type A2) of length 3 in the alphabet {`a, c, g, t`}, then all the words `aaa, aac, aag,` ..., `ttt` can be enumerated and for each of them it can be counted in how many examples it is present. This number can be regarded as the fitness of the pattern. The algorithm should either report patterns above some fitness threshold, or a given number of the fittest patterns.

The second class of approaches, which we call top-down (TD) approaches comprise algorithms trying to find patterns by comparing the strings and looking for local similarities. For instance, if two sequences

`AWCDEFGHIJKLM` (7)

and

`NEFGOPQAWRJKLS` (8)

are given, then by comparing them it can be noticed, that they share three continuous substrings: `AW`, `EFG`, and `JKL`. If we wish to arrange these substrings so that a common regular pattern of the type A3 appears, then only two of them can be used at a time. Either we can have: `*AW*JKL*`, or `*EFG*JKL*`. The second one may be preferable, since it is longer, and therefore more specific and less likely to appear by chance if the frequencies of occurrence of all the characters are assumed to be equal. If a third sequence:

`TAWUVOPHIJKLYZ` (9)

is also given, then there remain only two substrings common to all three sequences, namely `AW` and `JKL`, leading to the regular pattern: `*AW*JKL*`.

As we will see later, it is also possible to combine the two approaches in a single algorithm.

The advantage of the BU approaches is that in this way it is possible to guarantee to find the best patterns up to some limited size, in general 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 the total length of the examples. On the other hand the size of the search-space is exponential in the length of the patterns, implying that the algorithms are as a rule exponential in the size of the patterns. Consequently usually only patterns of limited size can be found by pure BU algorithms.

It may be possible to discover patterns of almost arbitrary size by the use of TD algorithms. At the same time the weakness of these algorithms is that in this way it is impossible in general to guarantee that some pattern has not been missed. The reason for this is that the algorithmic problems related to finding local similarities for n sequences is NP-complete. Therefore, precise TD algorithms are usually exponential-time in the length of examples, meaning that some heuristics has to be used. In general TD algorithms tend to work well if the sequences are sufficiently similar.

In the following subsections we will describe in more detail the basic ideas of BU, TD and combined approaches to pattern discovery. We will present the basic information about each algorithm separately in Appendix A. Some sample patterns discovered by some of the algorithms are presented in Appendix B. Subsections 3.1 and 3.2 are organised so that we start with considering algorithms for conservation problem, and end by considering algorithms for classification problem. In subsection 3.3 we also start with conservation problem, continue with classification problem, and then consider some related algorithms and applications.

3.1. Bottom-up approaches

The most straightforward implementation of the BU approach is the explicit enumeration of all the patterns from the pattern space one by one, as in the example at the beginning of this section. In the simplest case when the patterns are simple subwords (case A2) this approach has been first applied to pattern discovery in biosequences in the early 1980's [QWK82, WAG84]. In order to calculate fitness in this case it is reasonable to count not only the number of examples containing the pattern precisely, but also, optionally with some decreasing weight, the number of examples approximately[5] containing the pattern. An efficient way to deal with this is to define for each pattern a set of neighbours, i.e. substrings which are similar to the given substring, and count in how many examples each substring together with its neighbours is present. The best pattern is picked out from the "neighbourhoods" having a sufficiently high level of fitness. A similar approach has been used by Staden [Sta89b]. For ranking the patterns, Waterman et al. [WAG84] estimates the statistical significance of the discovered patterns, while Staden [Sta89b] calculates a measure of *information content* of the block consisting of the segments

[5] in the sense of some distance measure

which approximately match[6].

This straight-forward approach can be easily extended for more complicated patterns. Smith et al. [SAC90] has used this approach for discovering patterns consisting of basic alphabet characters and wildcards. The algorithm enumerates all the patterns consisting of three non-wildcard characters and up to 24 wildcards between each of them, i.e. the patterns of the type $a_1\, d_1\, a_2\, d_2\, a_3$, where a_1, a_2, and a_3 are characters from the basic alphabet, and d_1 and d_2 the number of wildcards in between them. Since the method is applied to proteins, the basic alphabet is of size 20. In this case, the patterns allow for a lot of variation, and it is both sufficient and reasonable to count only the numbers of examples which contain the pattern exactly. The user provides the minimum number of examples that should contain the pattern for it to be considered as 'fit'. Apart from this, a heuristic estimate for determining the relative fitness of each of these patterns is introduced and only some of these patterns (those with high scores) are reported in the end. [SAC90] also uses elements of the TD approach to extend the patterns found by enumeration (see section 3.3).

An obvious problem in this straight-forward enumeration is that of efficiency. The size of the search space for patterns of length l grows as $O(|\Sigma|^l)$. The number of patterns can be reduced if we put some restrictions on pattern class. For instance, in Smith's et al. method [SAC90] there are $20 \times 10 \times 20 \times 10 \times 20 = 800000$ candidate patterns of the type $a_1 d_1 a_2 d_2 a_3$ to check if the distance range is 10 (i.e., $0 \leq d_1 < 10$ and $0 \leq d_2 < 10$). However for more general pattern classes this number becomes unpractical. 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. Heuristics for BU approaches

One simple heuristic for limiting the search space is based on an assumption that the patterns which are present approximately (within some distance) in many examples 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, for example Steiner's sequence[7], itself not present in a single example. However, if sufficient examples are given this is not very likely. Therefore only those substrings which are present at least in one example need to be enumerated. This reduces the search space drastically, since for a set of strings with total length N, there are only $O(N^2)$ substrings. If the length of the substrings is bounded by l, then there is only $O(lN)$ number of substrings to consider (instead of $O(|\Sigma|^l)$). After finding the most frequent substrings we can cluster the most similar ones, and generalise them to find more complicated patterns from their alignments (this is a TD element, see 3.3). This approach has been used by Saqi and Sternberg [SS94], where also a kind of statistical significance has been used for sorting out the interesting patterns.

If the number of examples is large enough this heuristic can be taken even further. We can select a random subset of examples, and if the number of examples in both the total set and the

[6] In fact the first two of these approaches assume that the sequences are approximately prealigned and apply the described algorithm within a fixed window over the approximately aligned sequences.

[7] By Steiner's sequence of 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 $A_1, \ldots, A_n$.

subset is large enough, it is statistically likely that any substring which occurs approximately in a sufficient number of examples should occur more than some number of times in the random subset in an exact form. This number can be estimated by using random sampling theory. Therefore, we can enumerate only these substrings that are present in the subset more than a certain number of times. Moreover, the strings in the subset can be represented as a *generalised suffix-tree* (GST) [Hui92], and then the potential candidates for the pattern, can be selected in linear time. GSTs are a generalisation of the notion of suffix-trees [McC76, Ukk92]. Thus the algorithm becomes linear time in the length of the examples and the patterns. This approach has been used by Wang et al. [WMS+94]. It can also be used for finding more complex patterns, from the classes A3 or D2, but then the method looses some of its efficiency because suffix-trees are not that well suited for these kind of patterns.

Note that GSTs provide an extremely efficient algorithm for finding the longest substrings common to at least k out of n given sequences. First of all, a GST can be constructed in linear time in the total length of sequences. After the GST is constructed, the query to find the longest substring common to at least k strings can be executed also in linear time, thus the whole problem can be solved in linear time (see [Hui92] for details). However we note that the construction of GST in linear time is technically very complicated and in practice often simpler and theoretically less efficient algorithms are used for constructing GSTs. Moreover, currently no very efficient algorithms using GSTs for approximate substring finding are known. Nevertheless it is worth noting that in case of substring patterns and exact occurrence, GST gives the most efficient pattern discovery algorithm.

3.1.2. Tree representation of the solution space

A rigorous approach for pruning the search space, which can be applied in the case of more general patterns and is still guaranteed to find all patterns from the solution space, is based on the observation, that if a subpatterns of a certain larger pattern is not present in the given number of sequences, then, the larger pattern itself, cannot be present in this (or a larger) number of sequences. This can be easily seen, if we consider the space of all the patterns represented as a tree. For instance, if we are looking for simple patterns (i.e., of the type A2) in alphabet {`a,c,g,t`}, then a part of the pattern space can be represented as a 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. [SVS95b]. The breadth-first approach may seem more time economic since the the length of the patterns can be extended not only by one character in each step but by factor of two. For instance, for a pattern `actg` to be present in a sufficient number of examples, both patterns `ac`, and `tg` should have been found earlier, if the search is done in the breadth-first manner. Unfortunately this approach can be realistically applied only for very short patterns, because the number of the patterns to be remembered in this way grows exponentially. Therefore for practical purposes the depth-first search is used.

A very efficient implementation of this idea applied to substring patterns is the so called Karp-Miller-Rosenberg (KMR) [KMR72] which can be adapted to find all substrings present in at least k out of the given n sequences in time proportional to $N \log N$, where N is the total length of the sequences. Note that by using generalised suffix-trees [Hui92] this can be done in

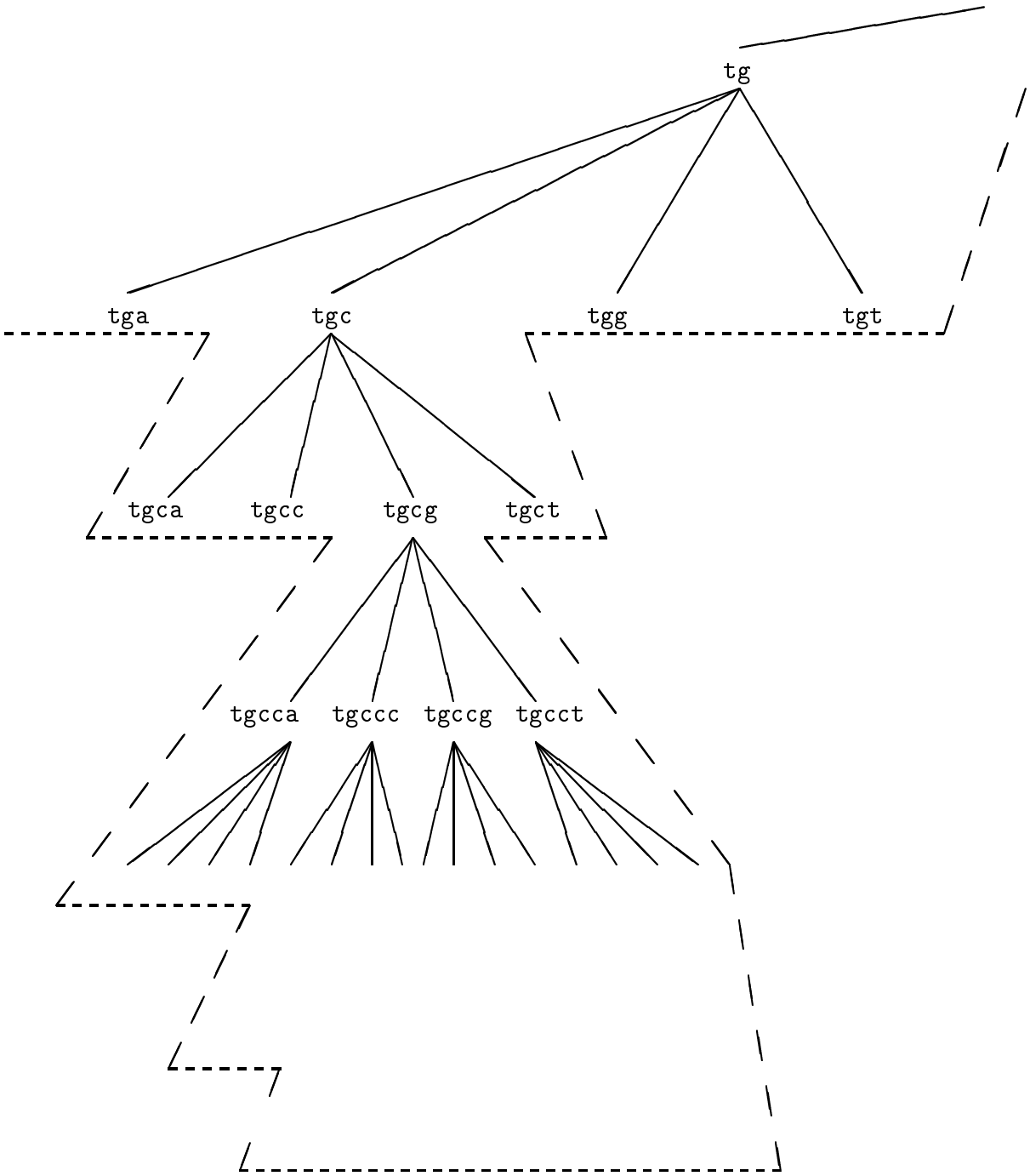

Figure 3: Pruning of the search tree. If, for instance, we have found, that a substrings `tga` is not present in sufficiently many sequences, then there is no need to look for any of `tgaa, tgac, tgag, tgat`. If, on the other hand, the substring `tgc` has been found in sufficiently many sequences, then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

linear-time.

The idea is extended by Sagot et al. [SVS95b] to finding the approximate presence of patterns, who also considers the application of this approach to more complicated patterns of the type 2C containing symbols denoting groups of symbols of the basic alphabet. Unfortunately the efficiency of the algorithm decreases if any of the groups is large (containing many basic symbols), because extension of these patterns is often possible. Therefore [SVS95b] does not allow wildcard characters.

On the other hand it can be seen that dealing with the wildcard characters should be quite easy. The fact that, for instance, we know that the pattern `tgc` is present in a sufficient number of examples enables us not only to find patterns `tgca`, ..., `tgct`, but also `tgc`x`a`, ..., `tgc`x`t`, `tgc`xx`a`, ..., `tgc`xx`t`, `tgc`xxx`a`, ..., ... This observation has been used by Neuwald and Green [NG94] where a new, so called *block* data structure has been introduced, and applied in a very efficient way to find the set of substrings matching each pattern, speeding up the depth-first search. They define the significance of a pattern, which is effectively a function of the pattern itself and the number of matching substrings[8]. This measure is used to heuristically prune the search tree. In patterns containing some minimum number of symbols from the basic alphabet (Σ_p), they allow for ambiguous pattern positions consisting of two symbols from Σ_p, which means that this algorithm can discover patterns in class 2C. They also use a TD approach to analyse the set of substrings matching a discovered pattern (see section 3.3).

Jonassen et al. [JCH95] describes an algorithm where the use of a depth-first search strategy combined with the block data structure 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 2E. The user defines restrictions on the kind of patterns that can be discovered, effectively defining a subclass of 2E. The algorithm sets out to find all patterns in this subclass matching at least some minimum number N_{min} of the positive examples. The search tree is pruned so that extensions of patterns matching less than N_{min} sequences, are not analysed. A fitness measure for patterns is defined, and the algorithm is guaranteed to find the highest scoring patterns within the subclass of 2E that match at least N_{min} of the positive examples.

A modification of the algorithm in Jonassen et al [JCH95], is proposed by Jonassen and Eidhammer [JE95]. 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 conserved patterns, matching at least N_{min} of the positive examples, with the highest fitness. The pattern graph may optionally be derived from an existing multiple sequence alignment, for instance of a subset of the sequences in $S1$, so that only patterns consistent with the alignment are considered. This gives a smaller search space, and can be considered as a TD-element. Branch-and-bound and heuristics are introduced to make the pruning of the search tree more efficient, speeding the search significantly especially for sets of quite similar sequences. In both [JCH95, JE95] a TD element is used to specialise patterns discovered in the depth-first search.

Experiments clearly show that using the tree representation of the pattern space combined with pruning and the use of efficient data structures substantially increases efficiency of the

[8] And, to be precise, the class of patterns explored, see section 2.3

algorithms; longer patterns can be discovered this way than by the straight-forward search. 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.

3.1.3. More complicated solution spaces

The BU approaches can also be used for finding more complex conservation or classification functions based on unions of regular patterns or decision-trees over regular patterns. This has been studied in [AMS+93, AKM+92, ASO94]. There the authors prove that the classes of these concepts can be *learned* from examples in polynomial time in sense of inductive inference or PAC-learning (see [SA95]). Unfortunately, the order of the polynomials is too high and therefore various heuristics have to be used in practical applications. This is not surprising, since the search space in case of these more complicated functions is much larger.

In [ASO94] a method for learning the union of a bounded number of regular patterns from only positive examples is developed and implemented, and interesting experimental results are presented. Arikawa [AMS+93, AKM+92] considers learning from both positive and negative examples, i.e. the classification problem. In [AMS+93] the method for learning decision trees first introduced by Quinlan [Qui86] is used. In [AKM+92] the authors have developed algorithm for learning so called *elementary formal systems*. 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. How the indexing of the basic alphabet can be performed automatically so that the classification of the examples in positive and negative remains correct is also studied in [AMS+93]. The problem is proved to be NP-complete, and a heuristic for its approximation is given.

3.2. Top-down approaches

By TD approaches we mean algorithms based on finding local similarities by comparing the sequences. Two quite obvious ways how this can be done are looking for sufficiently long common (or similar) substrings or subsequences, or by trying to align the given sequences to minimise the mismatches. By a subsequence of two sequences $a_1 \dots a_n$, and $b_1 \dots b_m$ we mean $c_1 \dots c_k$, such that there exists $i_1 < \dots < i_k$ and $j_1 < \dots < j_k$ for which $c_1 \dots c_k = a_{i_1} \dots a_{i_k} = b_{j_1} \dots b_{j_k}$. A common subsequence of more than two strings can be similarly defined. Note the distinction between common subsequence and common substring which we define as $d_1 \dots d_k = a_i \dots a_{i+k-1} = b_j \dots b_{j+k-1}$, i.e., a continuous common substring. We say that the subsequence is the longest common subsequence (LCS) if k is the maximal possible for any common subsequence of the involved sequences. For instance `EFGJKL` is LCS of the sequences (7) and (8) on page 18. If we align the strings (7) and (8) to maximise the number of aligned identical characters, the alignments can be featured as in Figure 4. From these alignments the pattern `*EFG*JKL*` can be easily extracted. The problems of finding the LCS and the optimal alignment are closely related, particularly if all mismatches of symbols (including alignment with a gap) are "penalised" equally. In general not all matches or mismatches of symbols should be scored equally. Gaps are usually scored as a *gap opening penalty* plus a *gap extension penalty* for each extension. The mismatches between different characters are penalised according to some

```
AWCDEFGHI    JKLM
----|||------|||-
   NEFGOPQAWRJKLS
```

Figure 4: Possible alignments of strings AWCDEFHIJKLM and NEFGOPQAWRJKLS.

scoring matrix (e.g., PAM or BLOSUM). Both problems are also closely related to the problem of finding a *distance* between the strings.

It is nontrivial to define what is the optimal alignment for the simultaneous alignment of more than two sequences. We will not go into any details, since the problem of multiple alignment is a wide area itself (see for example [CWC92]), and, although closely related to pattern discovery, it is not exactly the same problem. Given a good alignment between strings, the underlying pattern can be extracted. On the other hand, if a pattern common to a set of strings is found, it can be used as a kind of "anchor" point for aligning these strings.

Whether we use the simple scoring scheme or a more complicated one, we are faced with the same problem. It becomes infeasible to calculate either the LCS or the optimal alignment as the number of sequences grows. In the case of two sequences an optimal alignment, according to the scoring scheme used, can be found efficiently by using dynamic programming [SK83, Ste94, CLR90]. However, as the time complexity of multiple alignment using dynamic programming is $O(l^n)$ for n sequences of average length l, this solution quickly becomes infeasible as the number of sequences grow. Pruning techniques exist making it possible to optimally align 6-8 sequences [LAK89]. It has been shown that the problem of finding LCS for n sequences is NP-complete [GJ79] (as is that of multiple sequence alignment [WJ94]), and therefore a rigorous solution is unrealistic for more than a very small number of sequences and the use of some kind of heuristics in TD approaches is inevitable.

Most of the known heuristics are based on using a pairwise sequence comparison. The first TD algorithms which we are aware of to find a regular pattern (type 3A) common to a set of strings were developed by the machine learning community, and do not have a direct relations to bio-computing [Shi83, Nix83]. These algorithms are based on finding the LCS for pairs of sequences. The algorithm begins 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, Shinohara [Shi83] and Nix [Nix83] prove that in some rigorously defined machine learning model (the inductive inference model) the method will produce the "right" pattern in polynomial-time in the total length of examples.

3.2.1. Best pair comparison based algorithms

An algorithm for finding patterns in biosequences based on pairwise comparing is given in Smith et al. [SS90]. This approach uses the fact that not only pairs of sequences can be aligned by

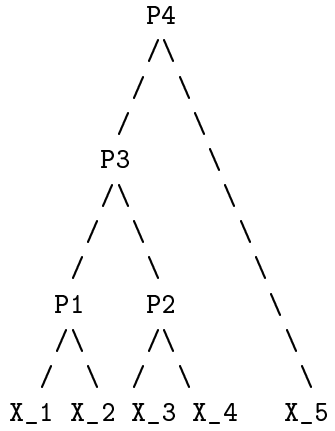

Figure 5: Example dendrogram for sequences X_1, X_2, X_3, X_4, X_5. Pairs X_1, X_2, and X_3, X_4 are the most similar among themselves, but the sequence X_5 is the most different from any of the other sequences. The algorithm aligns X_1 to X_2, obtaining P_1, X_3 to X_4, obtaining P_2, then P_1 to P_2 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 P_4 is the pattern matching all sequences.

dynamic programming algorithms, but also that sequences can be aligned to regular patterns, as well as patterns to patterns. The algorithm exploits the fact that the characters of the basic alphabet (i.e., Σ_p) can be organised in partially ordered hierarchical groups. Although the AACC hierarchy, is used in the algorithm, in principle the method can be applied for any hierarchy, including the flat hierarchy, i.e., when there is only one group containing all basic alphabet characters.

An estimated phylogenic tree (the so called *dendrogram*) is built featuring 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 5. The pairs (sequence, sequence), or in later stages (sequence, pattern) or (pattern, pattern) are aligned at each node of the dendrogram starting from bottom-up, and a common pattern is obtained from each pair.

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. 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. Gaps are penalised by the formula $w = w_0 + w_e \cdot k$, 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 extension (if needed) are penalised, but not the gap opening.

The algorithm generates for a given dendrogram the most specific pattern for each node common to the sequences below that node. The pattern is common to the whole set of the given strings in the root. Each step (i.e., pairwise alignment) is guaranteed to give an optimal (i.e. the most specific) pattern 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. An interesting property of this approach is that besides the pattern which

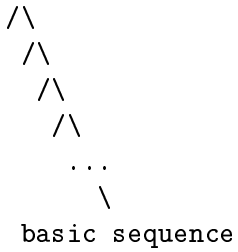

Figure 6: A dendrogram where one sequence is chosen as a basic sequence and all other sequences are aligned against it

is common to all sequences, we also obtain patterns common to subsets of related sequences, therefore the algorithm can be also used for classification.

A different heuristic is developed by Roytberg [Roy92]. One sequence is selected as the *basic sequence*, and all the other sequences (the so called *serial sequences*) are aligned against it. This approach would correspond to a dendrogram of the type given in Figure 6. The substrings in the basic sequence are found that have approximate matches in all, or in a specified percentage, of the serial sequences. The algorithm additionally ensures, that the respective substrings from the serial sequences are similar to each other[9]. The respective substrings from the basic sequence that have similar substrings in a sufficient number of mutually similar sequences are called *fundamental*. Finally the algorithm finds the fundamental substrings which are maximal in the sense, that they cannot be extended. A disadvantage of this method is that it is sensitive to the selection of the basic sequence.

3.2.2. All pair comparison heuristics

A heuristic based on finding pairwise similarities between all pairs of sequences is described by Schuler [SAL91]. The algorithm begins by comparing all pairs of input sequences. It locates for each pair the local similarities (i.e. substrings) which score high enough, thus obtaining 2-blocks[10]. 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 the 4-blocks and so on. Theoretically there may be exponentially many blocks to try, but in practise if the threshold for similarity scores has been taken high enough the number of hypothesis is manageable. A very similar approach is also described by Brodsky et al. [BVK$^+$92].

A heuristic using representation of pairwise alignments by so called *dot-matrices* is described in [VA91]. 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

[9]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.

[10]By an *n-block* we mean an array of n substrings of approximately matching symbols.

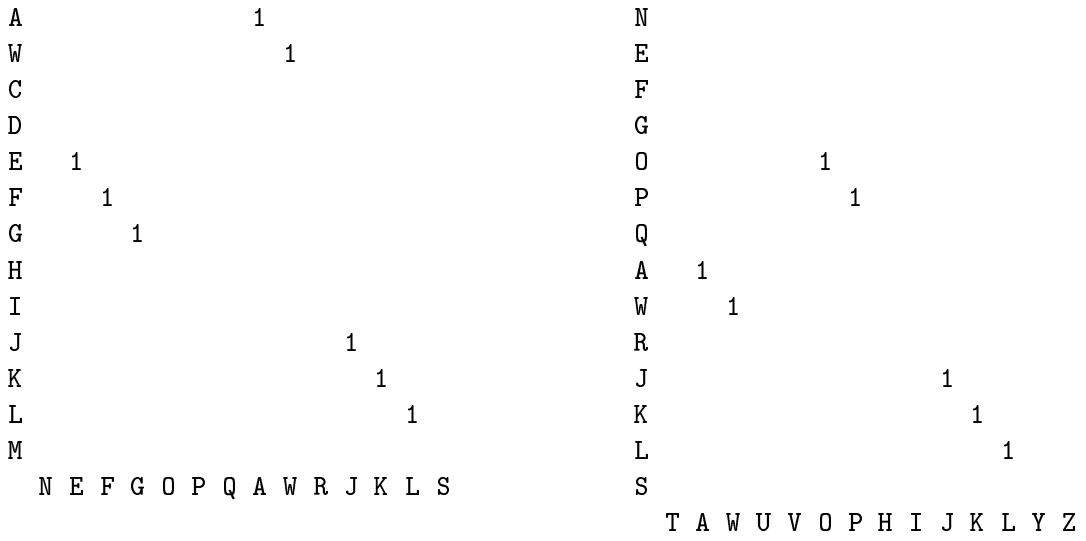

Figure 7: Dot-matrices of strings (7)-(8) and (8)-(9) (0 elements are left blank in the example)

size $l \times m$ with elements $a_{i,j}$ defined as follows: $a_{i,j} = 1$ if and only if $b_i = c_j$, otherwise $a_{i,j} = 0$. For instance, the dot matrices 7-8 and 8-9 for the sequences (7), (8), and (9) on page 18 are given in Figure 7. More general dot matrices using real values instead of boolean representing the similarity scores between the positions, can also been defined.

A *Boolean multiplication* of such matrices is performed in the same way as normal matrix multiplication, except that instead of summation, the Boolean summation is used (i.e. 0+0=0, 0+1=1, 1+0=1, and 1+1=1). For instance, the result of Boolean multiplication for the matrices in Figure 7 is given in Figure 8. Note that the resulting matrix is different from 1-3 in that only substrings present in all three sequences, namely `AW` and `JKL` have 1 in the respective positions. In general, if sequences $X_1, \ldots, X_n$ are given, there exist $n(n-1)/2$ dot matrices $M_{1,1}, M_{1,2}, \ldots M_{n,1}, M_{2,2}, M_{2,3}, \ldots, \ldots, M_{n,n}$. The matrix resulting from the Boolean multiplication $M^*_{k,m} = M_{k,l} \times M_{l,m}$ shows which sequences are similar 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. Vingron et al. [VA91] 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.

3.2.3. Algorithms for classification problem

TD algorithms can also be used in the case when both positive and negative examples are given (i.e. for the classification problem). An interesting approach is described by Kudo et al. [KKASI92]. On one hand, the problem in [KKASI92] is simpler since it is assumed that

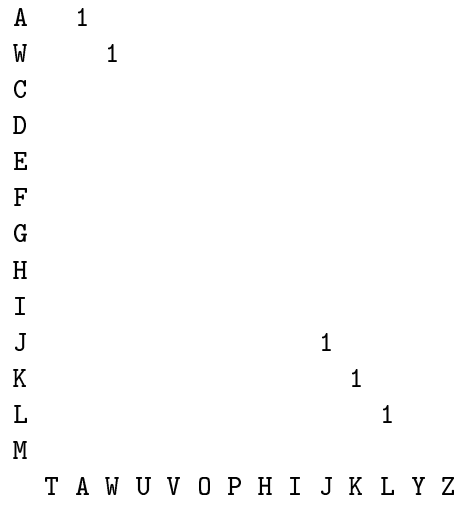

Figure 8: Resulting dot-matrix from Boolean multiplication (7)-(8) $\times$ (8)-(9)

the positive examples are given correctly pre-aligned. The approach is primarily designed for finding patterns at gene splice-site 5-th end, and since all such sites have a fixed position of 100% conserved GT, the examples can be aligned by aligning GT. On the other hand, the target language in [KKASI92] is relatively powerful: a union of subwords, either with wildcards (i.e. B1c), or in the more general case containing arbitrary combinations of basic characters (i.e. C1c).

The algorithm finds the least general set of patterns that cover all the positive examples, and does not contain any negative examples. In principle this could be achieved by a simple BU exhaustive search, but it would not be practical. Kudo et al. [KKASI92] proposes an algorithm for more efficient search. It uses iterative steps, each step of the iteration introducing wildcard characters 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 firstly pairs of positive examples, then triples from sequences contained in the successful pairs, then quadruples from successful triples, etc, while such unification is possible without including any negative examples. Note that this algorithm is guaranteed to find the smallest set of patterns compatible with all positive examples, and excluding negative examples. In some sense this approach can also be rated as BU, although the candidate patterns for search has been generated directly from examples.

The problem of finding patterns from positive and negative examples has also recently been studied by Tateishi et al. [TM95, TMM95]. 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-complete. A greedy algorithm for approximating the solution for a simple pattern class 1C from prealigned examples having the same length is given. Some heuristics for more complicated patterns are

also proposed.

Finally we note that in some sense the work of Saqi and Sternberg [SS94] can also be regarded as a TD approach, since they take the hypothesis directly from the examples. Nevertheless this approach is based on the direct enumeration of the candidate patterns, and therefore we rate it as BU. The last examples show that the distinction between BU and TD approaches is not always very exact. Moreover, as the next subsection will show, very explicit BU and TD approaches can be combined into a single algorithm.

3.3. Combined approaches and some related works

The most obvious way to combine BU and TD is to use TD for refining (expanding or combining) the patterns found by BU search. This can be done in two ways. The first way is to

- use BU 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 pattern are aligned together (thus the candidate patterns become a kind of "anchors"),
- finally extend the pattern to the left and to the right from these "anchor" points while the fitness of the emerging pattern is increasing.

This method is used in [SAC90, JCH95]; the latter expands only to the right because of the asymmetric nature of the block data structures.

The second way of refining found by the BU approach is by grouping the 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 examples, then a common pattern `aa`[`c,g`]`aa` can be obtained from them (where [`c,g`] stands for a `c` or `g`). This kind of refinement is used in [NG94, SS94]. After having combined patterns, Neuwald and Green [NG94] calculate an initial profile[11] from the (ungapped) alignment defined by the substrings matching a combined pattern. This profile is iteratively refined by realigning the sequences to the profile, throwing away non-significant matches, and recalculating the profile.

A modification of the first combined approach is used by Landraud et al. [LAC89]. 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.

[11] 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. Additionally, a profile may contain position-dependent gap-penalties.

Note that the algorithm is "greedy" and is likely to produce good results if the sequences are sufficiently similar.

Another explicit way of combining BU and TD approaches is described by Ogiwara et al. [OUSK92]. There the basic idea is to use BU approach to find relatively short candidate substrings, to transform the original sequences to different data structures consisting of these substrings, joined by 'gaps', and finally to align the obtained data structures and to extract the common patterns. Let us consider this approach in some more detail.

The algorithm is for the classification problem, ie. it uses both positive and negative examples. All words of the given length are enumerated in a BU 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 which are present in at least f percent of positive examples, and in none of the negative examples are retained; in practice two cases: $f = 1$ and $f = 0.7$ 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. 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. The transformed examples are the data for the second stage, which is that of multiple alignment. Multiple alignment is performed trough pairwise alignments and dynamic programming. In this way the 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.

Note that in principle this method could also be used the same way for the conservation problem, i.e. if only positive examples are given. However the negative examples allow to use shorter substrings $p_{i,j}$ as a starting point for the alignment.

In Henikoff et al. [HH91] a combined BU and TD algorithm is developed for finding frequent blocks in protein databases. The first stage is simply using the algorithm of [SAC90], thus finding patterns and the respective blocks in a BU manner, and then extending them (see the beginning of this subsection). The positions of the patterns are marked on the initial examples. Next the "best" set of patterns which occur in the same order without overlapping in a critical number of examples is found. Such ordering is called a *path*. To find the best path firstly 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. After this all paths are searched and some "scores" are calculated (for details of the scoring scheme see [HH91]).

Wu et al. [WB95] use the TD approach to extract patterns from (assumingly) correctly prealigned examples, in combination with a BU heuristic search for correctly grouping the examples in subclasses, and excluding the noise. In some sense this may be seen as an attempt to deal with the noise without *a priori* assumptions about its level.

4. Conclusions

The aim of this survey has been to establish some systematisation of the area of pattern discovery in biosequences. In order to achieve this we have given a partially formalised specification of the problem of the automatic discovery of 'good' patterns from a set of sequences, and we have partially formalised the problem of ranking the patterns in respect to the sequences. A classification of pattern languages has been introduced, and we have given a systematic overview of the currently known algorithmic ideas used in algorithms for pattern discovery in biosequences.

We have described most of the algorithms that we are aware of for biosequence pattern discovery whose expressive power lies inside the regular languages. We have not dealt with algorithms for probabilistic patterns (e.g. Gibbs samplers) nor those for patterns outside the regular languages. Moreover, this survey should not be regarded as comprehensive even for regular language patterns.

While dealing with these problems we have noticed that different authors implement 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 are very different, although the algorithms are frequently intended to solve the same problem.

We believe that it would be beneficial for the field that an attempt is made to establish some systematisation about which experiments could be used for testing the algorithms. This would be useful for comparing the algorithms and choosing the best algorithm for a particular problem. It may also be useful to adopt some standard manner of how to present those experimental results about new sequence families which have not yet explored by earlier methods. The standard could include a standard pattern language, for example the PROSITE pattern language, and some statistics, for example the number and lengths of the sequences in the family, and for each pattern the number of matching sequences inside and outside the family. This would mean the establishment of some benchmarks or even some, possibly informal, 'theory of correct experiment' for the area.

Acknowledgements

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 is paid by a grant from the Norwegian Research Council.

Appendix A: Algorithms and software

Key

Algorithms	
Pattern	Pattern type (see section 2.2.4)
G	Guaranteed [Y/N]
+/-	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

Authors	**Algorithms**				**Software**			
	Pattern	G	+/-	Domain	Name	Src/Ex	Platform	Obtain
[Nix83]	3A	N	+	N/A	none			
[Shi83]	3A, exact	Y	+	N/A	none			
[WAG84]	2A, approx	Y	+	protein, DNA	none			
[LAC89]	3A, approx	N	+	protein, DNA	none			
[Sta89b]	2A, fuzzy	Y	+	DNA	unknown	Fortran77	Vax VMS	n/a
[SS90]	2E, exact	N	+	protein	none			
[SAC90]	2E, exact, before TD element	Y	+	protein	MOTIF	Turbo-C	IBMPC	n/a
[VA91]	3A [FILLOG/SUM]; 3A, approx [FILMAXAV]	N	+ align	protein	unkown			A
[KKASI92]	1C exact	Y	+/-	DNA	none			
[OUSK92]	3A, exact	Y/N	+/-	protein	none			
[Roy92]	2A, approx	N	+	protein, DNA	MuSCo		IBMPC, IBM/370	n/a avail
[AMS+93]	3A, decision trees	N	+/-	protein	none			
[NG94]	2C	N	+	protein	ASSET	Src	SPARC2	a/ftp
[SS94]	2C	N	+	protein	none			
[WMS+94]	3A, approx	N	+	protein	DISCOVER, CLASSIFY	Ex	DOS, DEC Ultra, SunSPARC	A
[JCH95]	2E	Y/N	+	protein	Pratt	Src (C)	dec-alpha, sparc10	a/ftp
[JE95]	2E	N	+	protein	Pratt2	Src (C)		n/a
[SVS95b]	2C	Y	+	protein	none			
[SVS95a]	2A, approx.	Y	+	protein, DNA	none			
[TSLM+95]	3A, decision trees and unions	N	+/-	protein	BONSAI			n/a
[WB95]	1C	N	+	protein	SEQCLASS	x, Common Lisp	Sun SPARC	n/a

Appendix B: Selected 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 [Sta89b]

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 [SS90]

Examples: 128 sequences of length between 141 and 147 from hemoglobin delta epsilon gamma beta major-chain chains.
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-I-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-I-c-L-I-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)-I-V-c-l-b-a-x-W-I-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. [SAC90]

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)-I-x(2)-T-x(2)-Y-x(5)
```

4. Kudo et al. [KKASI92]

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 1B):

```
<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 1C)

```
<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. [OUSK92]

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 [SS94]

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 [NG94]

Examples: a set of 56 sequences with an average length 471 from acyltransferases
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. [WMS+94]

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 [SVS95a]

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

10. Shoudai et al. [TSLM+95]

Examples: 3796 signal peptides indexed to 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
```

11. Jonassen et al. [JCH95]

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]
```

References

[ABL+94] D. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts, and J. D. Watson. *Molecular biology of the cell.* Garland Publishing Inc, New York, 3 edition, 1994.

[AKM+92] S. Arikawa, S. Kuhara, S. Miyano, A. Shinohara, and T. Shinohara. A learning algorithm for elementary formal systems and its experiments on identification of transmembrane domains. In *Proc. 25th Hawaii Int. Conf. on System Sci.*, pages 675–684, 1992.

[AMS+93] S. Arikawa, S. Miyano, A. Shinohara, S. Kuhara, Y. Mukouchi, and T. Shinohara. A machine discovery from amino acid sequences by decision trees over regular patterns. *New Generation Computing*, pages 361–375, 1993.

[ASO94] H. Arimura, T. Shinohara, and S. Otsuki. Finding minimal generalizations for unions of pattern languages and its application to inductive inference from positive data. In *Proc. of the 11th STACS, Lecture Notes in Comp. Sci., 755*, pages 649–660. Springer, 1994.

[Bai92] A. Bairoch. Prosite: a dictionary of sites and patterns in proteins. *Nucleic Acids Research*, 20:2013–2018, 1992.

[BB92] A. Bairoch and P. Boeckmann. The swiss-prot protein sequence data bank. *Nucleic Acids Research*, 20:2019–2022, 1992.

[BB94] P. Bucher and A. Bairoch. 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*, pages 53–61, 1994.

[BCHM94] P. Baldi, Y. Chauvin, T. Hunkapiller, and M. M. McClure. Hidden markov models of biological primary sequence information. *Proc. Natl. Acad. Sci USA*, 91:1059–1063, Feb 1994.

[BT91] C. Branden and J. Tooze. *Introduction to protein structure.* Garland Publishing Inc, New York, 1991.

[BUV95] A. Brazma, E. Ukkonen, and J. Vilo. Finding a good collection of patterns covering a set of sequences. Technical Report C-1995-60, University of Helsinki, Department of Computer Science, University of Helsinki, Finland, December 1995.

[BVK+92] L. I. Brodsky, A. V. Vassilyev, Y. L. Kalaydzidis, Y. S. Osipov, R. L. Tatuzov, and S. I. Feranchuk. Genebee: the program package for biopolymer structure analysis. In S. Gindikin, editor, *Mathematical methods of analysis of biopolymer sequences, DIMACS series in discrete mathematics and theoretical computer science, volume 8*. American Mathematical Society, 1992.

[CLR90] T. H. Cormen, C. E. Leicerson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[CWC92] S. C. Chan, A. K. C. Wong, and D. K. Y. Chiu. A survey of multiple sequence comparison methods. *Bulletin of Mathematical Biology*, 54(4):563–598, 1992.

[Day78] M. O. Dayhoff. *Atlas of protein sequence and structure*, volume 5. National Biomedical Research Foundation, 1978.

[ED94] S. R. Eddy and R. Durbin. Rna sequence analysis using covariance models. *Nucleic Acids Research*, 22:2079–2088, 1994.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, NY, 1979.

[GME87] M. Gribskov, M. McLachlan, and D. Eisenberg. Profile analysis: detection of distantly related proteins. *Proc. Natl. Acad. Sci. U.S.A*, 84:4355–4358, 1987.

[HH91] S. Henikoff and J. G. Henikoff. Automated assembly of protein blocks for database searching. *Nucleic Acids Research*, 19(23):6565–6572, 1991.

[HH92] S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. USA*, 89:100915–100919, 1992.

[Hui92] L. C. K. Hui. Color set size problem with application to string matching. In *Proc. of Combinatorial Pattern Matching*, pages 230–243. Springer-Verlag, 1992.

[Hut94] A. Hutchinson. *Algorithmic learning*. Clarendon Press, 1994.

[JCH95] I. Jonassen, J. F. Collins, and D. G. Higgins. Finding flexible patterns in unaligned protein sequences. *Protein Science*, 4(8):1587–1595, 1995.

[JE95] I. Jonassen and I. Eidhammer. Discovering patterns conserved in sets of related protein sequences. In *Proceedings of Norwegian Informatics Conference*, pages 95–112, Norway, 1995. Tapir.

[KB94] T. M. Klinger and D. L. Brutlag. Discovering structural correlations in alpha-helices. *Protein Science*, 3:1847–1857, 1994.

[KBM+94] A. Krogh, M. Brown, I. S. Mian, K. Sjoelander, and D. Haussler. Hidden markov model in computational biology. applications to protein modelling. *Journal of Molecular Biology*, 235:1501–1531, 1994.

[KKASI92] M. Kudo, S. Kitamura-Abe, M. Shimbo, and Y. Iida. Analysis of context of 5′-splice site sequences in mammalian mrna precursors by subclass method. *CABIOS*, 8(4):367–376, 1992.

[KLP92] T. Kristensen, R. S. Lopez, and H. Prydz. An estimate of the sequencing error frequency in the dna sequence databases. *DNA Seq.*, 2:343–346, 1992.

[KMR72] R. M. Karp, R. E. Miller, and A. L. Rosenberg. Rapid identification of repeated patterns in strings, trees and arrays. In *4th ACM Symposium on Theory of Computing*, pages 125–136, 1972.

[LAB+93] C. E. Lawrence, S. F. Altschul, M. S. Boguski, J. S. Liu, A. F. Neuwald, and J. C. Wootton. Detecting aubtle sequence signals: A gibbs sampling strategy for multiple alignment. *Science*, 262:208–214, Oct 1993.

[LAC89] A. M. Landraud, J-F. Avril, and P. Chretienne. 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, Aug 1989.

[LAK89] D. J. Lipman, S. F. Altschul, and J. D. Kececioglu. A tool for multiple sequence alignment. *Proc. Natl. Acad. Sci. USA*, 86:4412–4415, 1989.

[LV95] M. Li and P. Vitanyi. Computational machine learning in theory and praxis. Technical Report NC-TR-95-052, Royal Holloway, University of London, UK, Department of Computer Science, Egham, Surrey TW20 0EX, England, September 1995. (To appear in Lecture Notes in Computer Science 1000).

[LWS+93] R. Lathrop, T. Webster, R. Smith, P. Winston, and T. Smith. Integrating ai with sequence analysis. In L. Hunter, editor, *Artificial Intelligence and Molecular Biology*, pages 211–258. AAAI Press/The MIT Press, 1993.

[McC76] E. M. McCreight. A space–economical suffix tree construction algorithm. *Journal of the ACM*, 23:262–272, 1976.

[NG94] A. F. Neuwald and P. Green. Detecting patterns in protein sequences. *Journal of Molecular Biology*, 239:689–712, 1994.

[Nix83] R. P. Nix. *Editing by Example.* PhD thesis, Yale University, Xerox Palo Alto Research Center, California, USA, August 1983.

[OUSK92] A. Ogiwara, I. Uchiyama, Y. Seto, and M. Kanehisa. Construction of a dictionary of sequence motifs that characterize groups of related proteins. *Protein Engineering*, 5(6):479–488, 1992.

[Qui86] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[QWK82] C. Queen, M. N. Wegman, and L. J. Korn. Improvements to a program for dna analysis: a procedure to find homologies among many sequences. *Nucleic Acids Research*, 10:449–456, 1982.

[Roy92] M. A. Roytberg. A search for common patterns in many sequences. *CABIOS*, 8(1):57–64, 1992.

[SA95] T. Shinohara and S. Arikawa. Pattern inference. In K. P. Jantke and S. Lange, editors, *Algorithmic learning for knowledge-based systems, GOSLER final report*, pages 259–291. Springer-Verlag, 1995.

[SAC90] H. O. Smith, T. M. Annau, and S. Chandrasegaran. Finding sequence motifs in groups of functionally related proteins. In *Proc. Natl. Acad. Sci. USA*, pages 826–830, Jan 1990.

[SAL91] G. D. Schuler, S. F. Altschul, and D. J. Lipman. A workbench for multiple alignment construction and analysis. *PROTEINS: Structure, Function, and Genetics*, 9:180–190, 1991.

[SBU+93] Y. Sakakibara, M. Brown, R. C. Underwood, I. S. Mian, and D. Haussler. Stochastic context-free grammars for modeling rna. Technical Report UCSC-CRL-93-16, University of California Santa Cruz, 1993.

[SD95] R. F. Sewell and R. Durbin. Method for calculation of probability of matching a bounded regular expression in a random data string. *Journal of Computational Biology*, 2:25–31, 1995.

[Shi83] T. Shinohara. Polynomial time inference of extended regular pattern languages. *LNCS*, 147:115–127, 1983.

[SK83] D. Sankoff and J. B. Kruskal. *Time warps: string edits, and macromolecules: the theory and practice of sequence comparison.* Addison-Wesley, 1983.

[Sol64] R. J. Solomonoff. A formal theory of inductive inference, part 1 and part 2. *Information and Control*, 7:1–22 and 224–254, 1964.

[SS90] R. F. Smith and T. F. Smith. Automatic generation of primary sequence patterns from sets of related protein sequences. In *Proc. Natl. Acad. Sci. USA*, pages 118–122, Jan 1990.

[SS94] M. A. S. Saqi and M. J. E. Sternberg. Identification of sequence motifs from a set of proteins with related function. *Protein Engineering*, 7(2):165–171, 1994.

[Sta89a] R. Staden. Methods for calculating the probabilities of finding patterns in sequences. *CABIOS*, 5:89–96, 1989.

[Sta89b] R. Staden. Methods for discovering novel motifs in nucleic acid sequences. *CABIOS*, 5(4):293–298, 1989.

[Ste94] G. A. Stephen. *String searching algorithms.* World Scientific, 1994.

[SVS95a] M-F. Sagot, A. Viari, and H. Soldano. A distance-based block searching algorithm. In C. Rawlings et al, editor, *Proc. of Third International Conference on Intelligent Systems for Molecular Biology*, pages 322–331, Menlo Park, California, July 1995. AAAI Press.

[SVS95b] M-F. Sagot, A. Viari, and H. Soldano. Multiple sequence comparison: a peptide matching approach. In Z. Galil and E. Ukkonen, editors, *Proc. of 6th Annual Symposium, CPM (Lecture Notes in Computer Science 937)*, pages 366–385. Springer, July 1995.

[Tay86] W. R. Taylor. Identification of protein sequence homology by consensus template alignment. *Journal of Molecular Biology*, 188:233–258, 1986.

[TM95] E. Tateishi and S. Miyano. 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, Aug 1995.

[TMM95] E. Tateishi, O. Maruyama, and S. Miyano. Extracting best consensus motifs from positive and negative examples. Technical Report RIFIS-TR-CS-115, Research Institute of Fundamental Information Science, Kyushu University, Japan, Aug 1995.

[TSLM+95] M T. Shoudai, M. Lappe, S. Miyano, A. Shinohara, T. Okazaki, S. Arikava, T. Uchida, S. Shimozono, T. Shinohara, and S. Kuhara. Bonsai garden: parallel knowledge discovery system for amino acid sequences. In C. Rawlings et al, editor, *Proc. of Third International Conference on Intelligent Systems for Molecualar Biology*, pages 359–366, Melono Park, California, Jouly 1995. AAAI Press.

[Ukk92] E. Ukkonen. Constructing suffix trees on–line in linear time. *Information Processing 92*, 1:484–492, 1992.

[VA91] M. Vingron and P. Argos. Motif recognition and alignment for many sequences by comparison of dot–matrices. *Journal of Molecular Biology*, 218:33–43, 1991.

[WAG84] M. S. Waterman, R. Arratia, and D. J. Galas. Pattern recognition in several sequences: Consensus and alignment. *Bulletin of Mathematical Biology*, 46(4):515–527, 1984.

[WB95] T. D. Wu and D. L. Brutlang. Identification of protein motifs using conserved amino acid properties and partitioning techniques. In C. Rawlings et al, editor, *Proc. of Third International Conference on Intelligent Systems for Molecular Biology*, pages 402–410, Melno Park, California, July 1995. AAAI Press.

[WJ94] L. Wang and T. Jiang. One the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1(4):337–348, 1994.

[WMS+94] J. T. L. Wang, T. G. Marr, D. Shasha, B. A. Shapiro, and G-W. Chirn. Discovering active motifs in sets of related protein sequences and using them for classification. *Nucleic Acids Research*, 22(14):2769–2775, 1994.