Techniques for comparison, pattern matching and pattern discovery
From sequences to protein topology

David GILBERT[1], David WESTHEAD[2] and Juris VIKSNA[3]
(1) Bioinformatics Research Centre, Department of Computing Science
University of Glasgow, Glasgow G12 8QQ, Scotland, UK
(2) School of Biochemistry and Molecular Biology, University of Leeds,
Leeds, West Yorkshire, LS2 9JT, U.K.
(3) Institute of Mathematics and Computer Science, University of Latvia, Latvia

Abstract. In this chapter we review techniques for sequence based pattern discovery and comparison, and show how these can be extended to RNA structures and abstract representations of protein structure at the fold level. We first define deterministic patterns over sequences and distinguish pattern matching from string comparison, reviewing the use of dynamic programming to compute edit distance, and longest common sub-sequence. We then describe approaches to pattern discovery in sequences and describe methods for evaluating the goodness of patterns. Next we show how string pattern languages can be extended to be applied to more complex data structures which can sequence and structural information, and give some algorithms for pattern discovery over certain classes of biosequences with structural content, with specific application to RNA. Finally we describe our work on pattern discovery and structure comparison for topological descriptions of protein structures, and show how these approaches can form the basis for practical and useful computational systems.

1. Biological background

This chapter assumes familiarity with molecular genetics, i.e. the composition and function of DNA and RNA and the processes of transcription from DNA to RNA and translation from RNA to amino acids. We refer the reader to any good undergraduate text on biochemistry for further details, e.g. [1] or [2].

2. Motivation

Given a particular target sequence/structure in which we are interested, and about which we are lacking certain information, we often wish to find homologous sequences/structures in order to make some hypotheses about the function of that sequence/structure. In general we will have access to a set of reference sequences/structures which are suitably annotated with organism of provenance, biological function(s) etc., and which may be grouped into *families* according to certain criteria, e.g. biological function or phylogenetic relationship. The reference sets may be very large, e.g. all known nucleotide or amino-acid sequences – 16 million or 100,000 records respectively, or all publicly available protein structures – 17,000 (February 2002). The task is thus to use some effective method to relate the target to the reference set, i.e. to perform a search with the target, where effectiveness is measured both by

biological usefulness of the results as well as 'speed' of operation. In principle there are two main approaches to searching

(1) *pair-wise compare* the target with each member of the reference set,
(2) group the reference set into families, extract common features of each family, and to *match* the target with these common descriptions.

In each case, we will need to *rank* the results in some way in order to be able to consider the most significant. The two approaches can be regarded as being the same if each member of the reference set forms one singleton family in (2). However, in general, the advantage of (2) is that each family usually comprises several members and thus there are fewer families than the reference individuals, and hence less matching operations have to be made than comparison operations. Moreover, matching may be faster than comparison, depending on the detail and form of the common descriptions. The disadvantages of (2) are firstly how the choice is made to form family groupings, and how characteristic are the common descriptions. Of course, the groupings and generation of common descriptions is usually performed infrequently, and is certainly not carried out each time a search is made.

3. Comparison of sequences or structures

If we *compare* a (new) sequence or structure with another sequence or structure, then we can obtain a measure of distance or similarity between the two objects; the distance measure should ideally be a *metric*, i.e.

- distances should be positive and the distance from an object to itself should be zero
- distances should be symmetric
- distances should respect the *triangle inequality* (the direct distance is the shortest distance between two objects)

Comparison of two sequences/structures should also produce a set of largest common sub-sequences or sub-structures (LCS) - it is not guaranteed that the set is a singleton - and a correlation between the two sequences/structures and each LCS. The sequences/structures can then be aligned using the LCS. In fact, pair-wise comparison can generalised to *n* objects, although the complexity of a naïve implementation of n-wise comparison can be very high. A form of n-wise distance can be obtained by computing the mean of all the pair-wise distances between the *n* objects.

In general comparison is a more expensive operation than deterministic matching, and more closely related in complexity to probabilistic matching. The most common use of comparison is to pair-wise compare a new sequence or structure *s* with the members of a set *T* of sequences or structures, each of which has some known biological attributes (function, or at least organism of provenance) – see above. As with probabilistic matching, the result of the comparisons will be the association with each member of *T* of a comparison value for *s*; the task is then to interpret these values. Again, these values can be ordered and also associated with some measure of significance (e.g. E-values or P-values**)**. Thus the comparisons can be ordered, and only those deemed to be significant considered.

4. Sequences, structures, patterns and motifs

There are many terms used to describe common similarities between sequences or structures, for example pattern, motif, fingerprint, template, fragment, core, site, alignment, weight matrix, profile. For our purposes we will regard a *pattern* as a description of some properties of a sequence or structure, and a *motif* as a pattern associated with some biological

meaning [3]. Moreover, if in a new sequence we detect the presence of a pattern known to be characteristic to a certain family, then we can hypothesise that the new sequence belongs to that family, even if we do not know its biological properties yet. In this way patterns may be used for the classification of bio-sequences and for predicting their properties. A pattern is said to be *diagnostic* for a family if it matches all the known sequences in the family, and no other known sequences. In general, patterns may be characteristic (match most of the sequences in a family and few other sequences), or classificatory (used to decide to which family a sequence belongs).

Patterns can be *deterministic*, i.e. can be used to decide if a sequence or structure matches the pattern or not, or *probabilistic*, when a value can be assigned to the match. For instance `C-x(2,4)-[DE]` is a sequence pattern matching any sequence containing a substring starting with `C` followed by between two and four arbitrary symbols followed by either a `D` or an `E`. Examples of probabilistic patterns are profiles and Hidden Markov Models. Deterministic patterns are simple and pure mathematical concepts, and are easier to interpret than probabilistic patterns; they are also easier to discover from scratch, especially if the data is noisy (contaminated). On the other hand, probabilistic patterns have more modelling power since they permit weights to be attributed to alternatives [4].

More generally, we will often wish to classify a new sequence or structure *s* using a library or set of *M* motifs. If the motifs are deterministic then matching *s* against the members of *M* will result in a subset of motifs which may be diagnostic for *s*. If *M* comprises probabilistic patterns then the result of the matching will be the association with each pattern of a match value for *s*; the task is then to interpret these values. It is usual to associate an ordering relation with match values, for example total ordering over integers or reals, and also to associate some measure of significance with match value, for example E-values (expectation values) or P-values (probability values). Thus the matches can be ordered, and only those deemed to be significant considered.

The use of such motif libraries is predicated on the prior identification of meaningful families, the selection of (possibly representative) family members, and the ability to generate patterns (either by hand, or automatically) which are sufficiently characteristic of the family.

5. Protein family analysis using patterns

Thus, the general protocol for family analysis is to

(1) collect sequences (structures) into a family based on biological function or phylogenetic relationships
(2) make family description by local multiple alignment, global multiple alignment or pattern discovery
(3) use the description to identify more family members
(4) analyse the extended set to see if the members are biologically related to the original family members

Regular expressions

We briefly remind the reader of regular expression notation:

Symbol: for each symbol **a** in the alphabet of the language, the regular expression **a** denotes the language containing just the string `a`

Alternation: Given 2 regular expressions M and N then M | N is a new regular expression. A string is in language (M|N) if it is language M or language N. The language (**a**|**b**) = {a,b} contains the 2 strings `a` and `b`.

Concatenation: Given 2 regular expressions M and N then M•N is a new regular expression. A string is in language (M•N) if it is the concatenation of two strings α and β such that α is in language M and β is in language N. Thus the regular expression **(a|b)•a** = {aa, ba} defines the language containing the 2 strings `aa` and `ba`

Repetition: M* stands for zero or more times repetition of M, M+ one or more times, and M? for zero or one occurrences of M.

Character ranges: [a-zA-Z] character set alternation, '.' any single character except a new-line (i.e. a wild card),

Regular expressions and biosequences

In general we have the following basic alphabets: Σ = { a, t, c, g} for DNA nucleotides, and Σ = { a, u, c, g} for RNA nucleotides. In the case of proteins we have a 20 character alpabet of amino acids Σ = {A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y}. We can also have general character group alphabet $\Pi = \{g_1 \ldots g_n\}$ - e.g. amino-acid class – and wild cards $X = \{ x(n_1,n_2) \mid n_1<n_2 \in N\}$. Thus $V(x(c_1,c_2))$ is the set of all words over Σ of length between c_1 and c_2. A pattern $P=p_1 \ldots p_n$, $p_i \in \Sigma \cup \Pi \cup X$. It is also common practice to separate pattern alphabet characters by a dash "-" [4]. Thus, for example, given a pattern *P*=A-x(2,6)-[LI]-x(0,∞) and a string *S*=ACDEFLGHJKL, we can obtain a match since S = A•CDEF•L•GHJKL (where '•' means concatenation) and $A \in V(A)$, $CDEF \in V(x(2,6))$, $L \in V([LI])$, $GHJKL \in V(x(0,\infty))$.

PROSITE patterns which are regular expressions over amino-acids use the following notation: '**x**' any amino acid; ambiguities: **[ALT]** =Ala or Leu or Thr, negation: **{AM}** any amino acid except Ala and Met, '-' as the separator, ` <` for the N-terminal, ` >` f or the C-terminal, and '.' for the end of the pattern. Repetition is indicated in general by e.g. x(2,4) = x-x *or* x-x-x *or* x-x-x-x and in the special case x(3) = x-x-x. An example of a prosite pattern is `[AC]-x-V-x(4)-{ED}.`, which can be expanded to`[Ala or Cys]-x-Val-x-x-x-x-{any but Glu or Asp}`. Another example is `<A-x-[ST](2)-x(0,1)-V.` which starts at the N-terminal of the sequence and is can be expanded to `Ala-x-[Ser or Thr]-[Ser or Thr]-(x or none)-Val`.

Structural sequence patterns

Eidhammer et al [5] introduce CBSDL, a constraint-based structure description language, where *structural* patterns contain constraints on the

(1) length of a substring to match a specific component;
(2) distance (in the input string) between substrings to match the different components of a pattern;
(3) contents of a substring to match a component, e.g. the second symbol should be an `a` or a `t`;
(4) positions on the input string where a particular component can match;
(5) correlation between two substrings matching different components, e.g. the substrings should be identical, or the reverse of each other.

This definition thus includes purely sequential patterns, which do not include a correlation constraint and are within the class of regular languages, for example PROSITE patterns. Structural patterns have at least one correlation constraint, for example repetitions or palindromes, and may describe context-free languages or even languages beyond the expressive power of context-free grammars, although there may be some languages in these classes which they cannot describe. From the point of view of bioinformatics, sequential patterns can thus describe sequences, whereas structural patterns can describe 'folded' strings, i.e. RNA structures such as stem-loops and pseudo-knots, and some topological descriptions of protein structures. Some examples of patterns which can be described by this language are shown below. We use Greek letters to indicate sub-patterns, possibly superscripted by *r* for reverse and *c* for complement, and underline corresponding sections of sequences.

Complements **a-u g-c**, **g-u** (weaker)

Table 1: Examples of structural sequence patterns

Tandem repeat	α-α	acg-acg
Simple repeat	α-β-α	acg-aaa-acg
Multiple repeat	α-β-α-δ-α	acg-aa-acg-uu-acg
Palindrome	α-α^{r}	acg-gca
Stem loop	α-β-α^{rc}	acg-aa-cgu
Pseudoknot	α-γ_1-β-γ_2-α^{rc}-γ_3-β^{rc}	augg-cuga-aggc-cgau-c-ucag-ggcau-aucg-ccgu

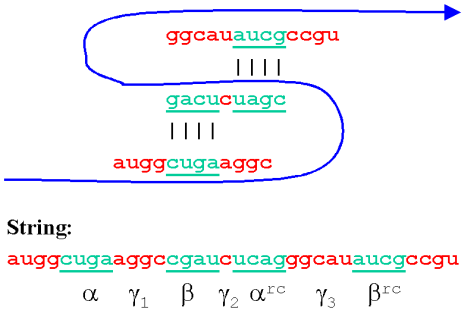

Figure 1: pseudoknot

6. Learning – pattern discovery

Brazma et al [4] have surveyed pattern discovery in biosequences; in the following we generalise their definitions to biostructures as well as biosequences.

A *protein family* F+ is a set of protein sequences or structures sharing definite functional or structural properties. If we have a *language L* of strings or structures then F^{+} is a subset of the total set of all possible sequences/structures that can be generated by the grammar of *L*, and F- are all those sequences/structures in *L* which do not belong to the family. Pattern discovery is then the problem of automatically finding functions approximating the characteristic function for the family F+. An algorithm for solving this problem takes as input a *training set* consisting of *positive examples*, which are sequences from F+, and optionally *negative examples* which are sequences from F-. This is a machine learning problem, namely that of learning a general rule from a set of examples. When both positive and negative examples are given, it is called the *classification problem*, and when only positive examples are given, it is called the *conservation problem*.

In the classification problem, we are given a set of sequences/structures S+ believed to be members of a family F+, and a set S- of sequences/structures believed not to be members of

F+, i.e. S+ ⊂ F+ and S- ⊂ F-. We also assume that F+ and F- are disjoint. The goal is then to find *compact* "explanations" of known sequences, i.e. functions that return true for all s ∈ S+ and false for all s ∈ S-, and have a high likelihood for returning true for s ∈ F+ and false for s ∈ F-. Furthermore, we would like to try to predict the family relationship of yet unknown sequences. In reality the data may be dirty, i.e. some members of S+ may be members of F- and some of S- may be in F+. Such contamination may be due to mis-classification by the field biologists or else to errors in data processing. If S+ ∩ F- and S- ∩ F+ are small, then the task is to find *compact* string functions that return true for *most* s ∈ S+ and false for *most* s ∈ S- , and have a high likelihood for returning true for s ∈ F+ and false for s ∈ F-.

In the *conservation problem* we are only given positive examples, i.e. a set S+ of sequences/structures believed to be members of family F+. The goal is then to find *interesting* string functions that return true for all s ∈ S+ and have a high likelihood for returning true for s ∈ F+. If the data are contaminated, hopefully to a small extent, then we need to find *interesting* string functions that return true for *most* s ∈ S+, and have a high likelihood for returning true for s ∈ F+. We can define *interesting* functions as having a low probability for returning true for random sequences.

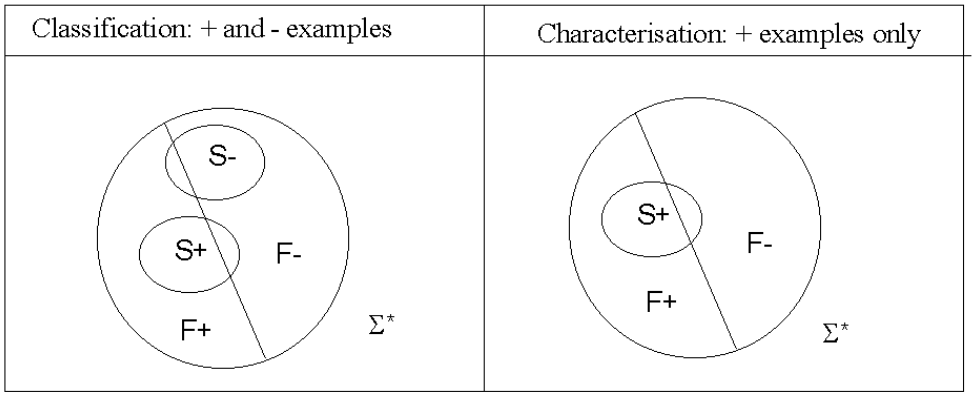

Figure 2: Classification and characterisation, noisy case. (Adapted from [4])

When carrying out pattern discovery we need to construct training and test sets of positive and optionally of negative examples. Since in practice we may not know all members of F+ and F-, we can randomly divide an initial set of positive examples into a training set S+ and a test set T+, similarly for S- and T-. The goal is to accurately describe "new" members of F+ and F- when we come across them.

In order to solve these problems, we need to:

(1) Find a good class of functions from which the approximating function f will be chosen for a particular real-world problem. This class is called the *solution space*, *hypothesis space*, or *target class*.
(2) Define a *fitness measure*, giving a ranking of the solution space, and thus evaluating how good each function is for the given training set.
(3) Develop an algorithm returning those classifier functions from the given solution space that rate high enough according to the fitness measure.

The success in solving the prediction part of the problems will depend on how successfully we will have chosen the solution space and fitness measure, even if the algorithm is perfect in the sense of finding all the fittest patterns.

7. Pattern discovery algorithms

Pattern discovery algorithms can be divided into two groups: pattern driven and sequence driven. In the following we adapt from [4].

Pattern-driven (PD) approaches are based on enumerating candidate patterns and selecting those with the best fitness; the general framework of these algorithms is:

(1) define the solution space, i.e. a set of patterns, and the fitness measure
(2) enumerate the patterns in the solution space
(3) calculate the fitness of each pattern with respect to the given examples
(4) report the fittest patterns

The most straightforward implementation is to limit the solution space by the size of the patterns, and to explicitly enumerate all the patterns from this space one by one. The advantage of this approach is that it is possible to guarantee finding the best patterns up to some limited size, almost regardless of the total length of the examples. This is because it is usually possible to organise the algorithm so that it is linear-time in this length. On the other hand the size of the pattern-space is exponential in the length of the patterns - for example there are more than 10^{13} different sub-string patterns of length 10 over the amino acid alphabet. PD algorithms guaranteed to find the pattern with the highest fitness value, have worst case time complexity exponential in the length of the patterns. Thus, guaranteed PD algorithms can only find patterns of limited complexity. An example of a program which uses a PD algorithm is PRATT [6,7].

Sequence or structure-driven (SD) methods find patterns by comparing given strings or structures and then looking for local similarities between them. For instance an SD algorithm may be based on constructing a local multiple alignment of given sequences and then extracting the patterns from the alignment by combining the segments common to most of the sequences. This may be achieved by

(1) grouping the sequences according to sequence similarity
(2) finding a common pattern, e.g. by dynamic programming that matches all or most of the sequences described by the parent groups
(3) grouping similar patterns together and repeating step (2) until only one group is left.

In general, more than two patterns may be combined in step (3). SD methods also differ in how the sets to be combined are chosen, how combination is performed (dynamic programming, heuristics) and how the (fittest) patterns are chosen, how patterns are represented and how many patterns are kept from stage (2). It may be possible to discover patterns of an almost arbitrary size by SD algorithms. However, since the construction of an optimal alignment or finding the longest sub-sequence are NP-hard problems, SD methods have to be based on heuristics and hence cannot guarantee optimal results. In general SD algorithms tend to work well if the sequences are sufficiently similar.

8. Sequence comparison

Comparison of sequences or structures is widely used in bioinformatics. In general, a *probe* sequence or structure is compared with a set of example sequences or structures annotated with attributes established by experiment or trusted computation. The probe sequence can then be putatively attributed annotations derived from the most closely matching examples. All against all pair-wise comparison of a set of sequences or structures can be used to generate data which can then be used as input to a clustering algorithm.

The thesis underlying these approaches is that evolution proceeds by mutations of nucleotide sequences whose products (RNA or proteins) are subject to selection, and that sequence or structure homology is a good indication of evolutionary relationship. The basic mutations that can occur are *substitution*, *insertion*, and *deletion* of nucleotides. These operations may then affect the amino-acid sequence that is derived from the nucleotide sequence via *transcription* and *translation*. However, the effect of one nucleotide mutation may not be manifested in the resulting amino-acid sequence and is termed a "silent" mutation. This is due to the triplet nature of codons - 3 nucleotides code for one amino-acid - and hence the redundancy of the genetic code: 64 triplet combinations are possible on the 4-letter nucleotide alphabet, but only 20 amino-acids are used in proteins. Since one amino-acid can be coded for by several codons it is possible that a change in a single nucleotide in a codon will merely result in a alternative codon for the same amino-acid. Of course, the insertion or deletion of one nucleotide will cause a *frame-shift* with the result that there may be mis-coding in all codons downstream of the mutation. The fact that the same sequence can evolve from different ancestors by convergent evolution means that the sequence homology is not a straight-forward indicator of evolutionary relationship. Moreover, since structure is more preserved than sequence (function is heavily dependant on structure, and evolutionary selection is exercised on function) means that some evolutionary relationships can only be determined by structure comparison.

Edit distance and LCS

Levenshtein [8] is attributed with first formalising the concept of edit distance over strings, which is the minimum number of edit operations - insertion, deletion or substitution of one symbol - to transform one string into another. Two identical strings have an edit distance of zero; the higher the score, the less similar are the strings. Given two strings $V=v_1..v_i$ and $W=w_1..w_j$ then the edit distance is given by the following recurrence relation:

$$d(i,0) = i,$$

$$d(0, j) = j \quad \text{else}$$

$$d(i, j) = \min \begin{cases} d(i-1, j)+1 \\ d(i, j-1)+1 \\ d(i-1, j-1) \text{ if } v_i = w_j \\ d(i-1, j-1)+1 \text{ if } v_i \neq w_j \end{cases}$$

In bioinformatics we are generally interested in *indels* (insertions and deletions) only since a computation omitting substitution will give us the longest possible common subsequence. Thus the last term of the minimum relation, $d(i\text{-}1,j\text{-}1)+1$ if $v_i \neq w_j$, is omitted. Moreover in practice, we often wish to compute the length of the *longest common subsequence* (LCS) for two strings as well as the LCS itself, since this will enable us to rate the match by the size of the LCS. The recurrence relation for the length of the LCS for two strings can simply be derived from that for edit distance:

$$l(i,0) = 0,$$

$$l(0, j) = 0 \quad \text{else}$$

$$l(i, j) = \max \begin{cases} l(i-1, j) \\ l(i, j-1) \\ l(i-1, j-1) + 1 \text{ if } v_i = w_j \end{cases}$$

The complexity for a naïve implementation of the recurrence relation is exponential in the length of the two strings, and thus a dynamic programming approach is used to compute these relations which reduces to complexity to *O(i,j)* for two strings length *i* and *j*. In addition, the LCS can be directly derived from the dynamic programming table, as can the associated alignment of the two strings. Details of the application of dynamic programming to determine edit distance, LCS and extract alignments can be found in e.g. [9,10]. Note that in general there is not one unique LCS or maximal alignment; the alternatives can be derived from alternative tracebacks.

The recurrence relations given above can be parameterised for different gap penalties and match/mismatch scores:

$$l(i, j) = \max \begin{cases} l(i-1, j) + g \\ l(i, j-1) + g \\ l(i-1, j-1) + \mathrm{t}(v_i, w_j) \end{cases}$$

where g is a gap penalty and

$\mathrm{t}(v_i, w_j)$ is a table of match/mismatch penalties

Multiple aligments

Multiple alignments can be used to analyse *gene families* and reveal subtle conserved family characteristics. Such alignments can be obtained by simultaneous N-wise alignment, generalising the *pairwise* approach above and dynamic programming will require an N-dimension matrix. The complexity of an n-wise dynamic programming is $O(m^n)$ for n sequences of length m and hence this approach is only useful for short sequences. In practice heuristic algorithms are used, for example the progessive approach of ClustalW in which pairwise sequence identities are computed, similar sequences are aligned in pairs, add distantly related sequences aligned later.

Profiles can be generated from multiple alignments by adding the observed frequencies of characters at each position of the alignment. These can be stored in a databases of profiles, for example the Prosite database [11] and we can then search with a sequence against this database. The advantages of such a search are that it is faster than sequence against sequence and gives a more general result, e.g. “the input sequence matches globin profile”, rather than “the input sequence matches X,Y and Z sequences with functions Fx, Fy and Fz”. We can also search with a profile against a database of sequences - this is used in PSI-BLAST which constructs a matrix from the top sequence hits for searching a probe sequence against a sequence database, and then iteratively uses the matrix as the probe.

9. Protein structures - TOPS representations.

In the following sections we show how techniques from sequence processing can be applied to representations of protein structure at the fold, or topology, level. For details of

pattern discovery and structure comparison for more detailed representations of protein structure the reader is referred to the excellent survey paper by Eidhammer et. al. [3].

Protein structures can be described at a highly simplified 'topological' level using TOPS cartoons [12,13,15]. These are schematic abstractions of protein three-dimensional structures in two dimensions. An example is shown in Figure 3. TOPS cartoons were originally drawn manually [12]; recently an algorithm that produces the cartoons automatically from protein structures has been devised [13,14,15] and implemented as a computer program (Figure 3).

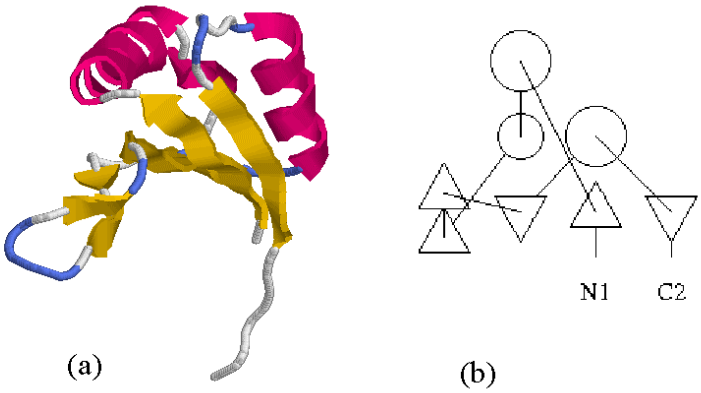

Figure 3: 2BOP: (a) Rasmol view and (b) TOPS cartoon

Although the cartoons do not explicitly display the information about hydrogen bonding between strands or chirality connections between SSEs, the program outputs a data structure containing, among other things, the information about these bonds and connections. We refer to this richer representation as a TOPS *diagram*; it is from this form that the cartoons are produced, which can then stored in graphical format (for example postscript or gif files). The following definitions for TOPS diagrams and simple patterns are adapted from [17].

A TOPS diagram is a formalisation of a cartoon, based on the underlying topological information from which the cartoon is generated - secondary structure elements (SSEs), H-bonds and chiralities. Such a diagram is a sequence of secondary structures and two associated sets defining relations, or constraints, over elements in the sequence; relations can be of the form of H-bonds or chiralities.

More formally, a TOPS diagram is a triple $T=(S,H,C)$ where $S=(S_1,..., S_k)$ is a sequence of length k of secondary structure elements (SSEs), H is the set of topological representations of hydrogen bonds, C the set of chirality connections. In this description an "H-bond" refers to a ladder of individual hydrogen bonds between adjacent strands in a sheet and chiralities are a subset of those generated by Slidel' s algorithm [16].

An SSE can be either α (helix) or β(strand) respectively. Since each SSE in a TOPS diagram is associated with a direction up or down we associate a direction symbol, + or -, with each SSE as appropriate. An H-bond constrains the types of the two SSE' s involved to be strands, and each bond is associated with a relative direction $D\in\{P,A\}$, indicating whether the bond is between parallel or anti-parallel strands. Chiralities are associated with handedness $C\in\{L,R\}$ (left and right respectively), and only occur between pairs of SSEs of the same type.

Diagram = (S, H, C), $\Sigma = \{\alpha+, \alpha-, \beta+, \beta-\}$ where

$S=(S_1,..., S_k)$, $1\leq i\leq k$, $S_i \in \Sigma$

$H = \{(S_i,\delta,S_j), \mid S_i,S_j, \in\{\beta+, \beta-\}\ \delta = P \leftrightarrow S_i=S_j, \delta = A \leftrightarrow S_i\neq S_j\}$

$C = \{(S_1, \chi,S_j) \mid S_i,S_j, \in \Sigma, , \chi \in \{L,R\}$

As an example, we give a TOPS diagram for 2bop in Figure 4, and represent this TOPS diagram in our notation as follows:

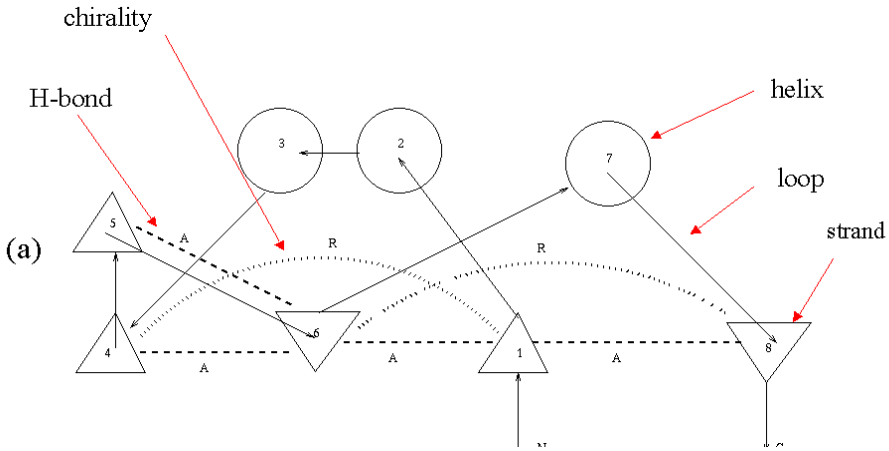

Figure 4: TOPS diagram of 2BOP

2bop = (*S*,*H*,*C*)
S = (β_1+, α_2-, α_3-, β_4+, β_5+,β_6-, α_7+, β_8-)
H = { (β_1+, A β_6-), (β_1+, A β_8-), (β_4+, A β_6-),(β_5+, A β_6-)}
C = {(β_1+,R, β_4+),(β_6-,R, β_8-)}

Simple TOPS patterns

A *simple* TOPS pattern (or motif) is similar to a TOPS diagram, but is a generalisation describing several diagrams which conform to some common topological characteristics. This generalisation is achieved by specifying the insertion of SSEs (and any associated H-bond and chiralities) into the sequence of secondary structure elements; indeed a diagram is just a pattern where no inserts are permitted. An insert is indicated by the length of its sequence.

Formally a simple TOPS pattern is a triple *P*=(*T*,*H*,*C*) where *T* (referred to as a *T*-pattern) is a sequence T= (I_0,V_1, I_1,V_2, ...,I_{k-1},V_k,I_k) comprising secondary structure elements indicated by V_i and between each of these an insert description. Each insert description I_m denotes the number of SSEs that can be inserted at that position, and ranges from zero to 30 in practice. Moreover I_m does not have to be a compact range, i.e. we can explicitly enumerate the number of inserts permitted. Thus for example we can have $I_3 \in \{0,1,3,6\}$ which is more discriminatory than $I_3 \in 0..6$ (i.e. $I_3 \in \{0,1,2,3,4,5,6\}$).

In principle, just as for TOPS diagrams, each SSE in a TOPS pattern is associated with an orientation and is a character from the alphabet {α,β}. Since TOPS diagrams exhibit rotational invariances of 180° about the x and y-axes, orientations (indicated by + and -) indicate relative opposite directions rather than absolute directions.

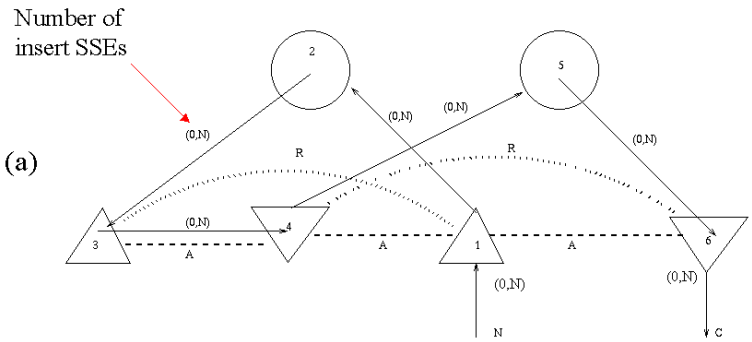

Figure 5: Plait motif

For example a TOPS pattern which describes plaits (2bop is an instance of a plait) is illustrated in Figure 5; arrows between SSEs in the sequence have been annotated with their corresponding insert descriptions. The formal definition of this TOPS pattern is below:

Plait = (*T*,*H*,*C*)

T = (I_0 , β_1+, I_1, α_2-, I_2, β_3+, I_3, β_4-, I_4, α_5+, I_5, β_6-, I_6)

H = { (β_1+, A β_4-), (β_1+, A β_6-), (β_3+, A β_4-)}

C= {(β_1+,R, β_3+),(β_4-,R, β_6-)}

Pattern matching for TOPS diagrams

TOPS graphs (diagrams and patterns) are totally vertex ordered, reflecting the underlying biology whereby an amino acid sequence gives rise to a sequence of secondary structure elements (helices and strands). We have previously developed two methods for matching TOPS patterns to TOPS diagrams. In [17] we describe a finite domain constraint-based algorithm which exploits the total ordering of the secondary structure elements and in [18] we describe a more sophisticated algorithm. Two additional ideas are used in this latter method to make this process more efficient. Firstly, we assign a number of additional labels to vertices and edges. Secondly, if an edge *e* can not be mapped according to the existing mapping for previous edges, then the next place where this edge can be mapped according to the labels is found, and the minimal match positions of previous edges are advanced in order to be compatible with the minimal position of *e*.

Pattern discovery for simple TOPS patterns

We have previously [19,20] described a method which, given a set of TOPS diagrams, efficiently discovers a simple pattern that characterizes that set. The essential difference between pattern discovery techniques for sequences [4] and TOPS diagrams is that techniques for the former assume a regular grammar, whilst the grammar of the latter is context-sensitive, owing to the fact that H-bond and chirality arcs may cross (i.e. they describe a “copy language”). In a naive version of a pattern-driven approach for TOPS diagrams not only would we have to enumerate all the possible combinations of the SSEs (and their orientations) in a pattern of length k, but also all the possible H-bond and chirality connections over them. Our method is very simple: by starting with an empty pattern we try to extend it in all possible ways and discard extensions that do not match in the set of positive examples, until we find the largest pattern that cannot be extended further. Our algorithm discovers patterns of H-bonds (and chiralities) based on the properties of sheets for TOPS diagrams; we also derive T-patterns, i.e. the associated sequences of SSEs and insert sizes. Briefly, the algorithm attempts to discover a new sheet by finding, common to all the target

set of diagrams, a (fresh) pair of strands, sharing an H-bond with a particular direction. Then it attempts to extend the sheet by repeatedly inserting a fresh strand that is H-bonded to one of the existing strands in the (current) sheet. The algorithm then finds all further H-bonds between all the members of the current sheet. The entire process is repeated until no more sheets can be discovered; any chirality arcs between the SSEs in the pattern are then discovered by a similar process. The numbers of inserts at each insert position in the pattern and their probabilities of occurrence are then computed over all the members of the learning set, and combined with the SSE sequence to give the T-pattern. The result is the least general common TOPS pattern characterizing the target set of TOPS diagrams.

Composite TOPS patterns

A *composite* TOPS pattern C-Patt comprises a disjunction of simple TOPS patterns, which we write as a set.

C-Patt = $\{p_1, \ldots, p_n\}$ where p_i , $i \in 1..N$, is a *simple* TOPS pattern.

A single simple TOPS pattern p can be written as the composite pattern {p}.

Compression

The compression of a pattern with respect to a set of structures is computed in a standard way, and adapted from [21], by reference to the size of the pattern and the total size of the components of the structures which are not included in the pattern. This value is normalised to the range 0 (best) to 1 (worst). In the following, for simplicity, we define the compression function for the beta sheet content (Hbonds) of TOPS diagrams; in practice we use compression computed over SSEs, Hbonds and chirality arcs.

Given:

(I) A TOPS pattern P = (T,Hp,Cp) where

T is a T-pattern, Hp is a set of Hbonds, |Hp| is the cardinality of Hp and Cp is a set of chiralities,

(II) A set of n examples $E = \{e_1,...,e_n\}$ that is described by P, where e_i ($1 =< i =< n$) is a diagram (S_i,H_i,C_i) with S_i is a sequence of SSEs, H_i is a set of Hbonds, $|H_i|$ is the cardinality of H_i, and C_i is a set of chiralities,

Then

(1) Raw Compression:

$$Comp_{raw}(P,E) = (\sum_{i \in 1..n} |H_i|) - (n-1)*|Hp|$$

(2) Normalised compression:

$$Comp_{norm}(P,E) = 1 - ((\sum_{i \in 1..n} |H_i|) - Comp_{raw}) / ((\sum_{i \in 1..n} |H_i|) - min(|H_i|))$$

which is a measure of the goodness of compression, varying from 0 (best, the pattern completely describes all the examples) to 1 (no compression).

Coverage

The *coverage* cov(P,S,E) of a pattern P which matches a subset S containing *n* diagrams out of a set E of *m* examples is *n/m*.

Goodness of a pattern

The goodness G of a pattern P is computed with respect to a subset S that P matches from a set of examples E, and is a function on the normalised compression and the coverage of the

pattern. In practice we have found the best results for our pattern discovery algorithm to be given when using G defined as follows:

$G(P,S,E) = \ln(Comp_{norm}(P,S)) * Cov(P,S,E)$

A "good pattern" is below some given value ("pruneval" or PV in the following) of G.

Algorithm to derive a cover of a set of examples (division into subsets and associated patterns)

Given: a set of domains E, and a cutoff prunevalue PV
Initialise count i:=1, covering set K=∅
while E is not empty do
Set pattern P_i = empty, hence match set S_i = E (since the empty pattern matches any example)
While $G(P_i, S_i, E) > PV$ /*find a "good" pattern P_i for an acceptable subset S_i of the examples E */
Extend P_i using the simple pattern discovery algorithm
$S_i = \{d \mid d \in E \text{ and } match(P_i,d)=true\}$
end_while
$E := E - S_i$
$K := K \cup \{(P_i,S_i)\}$
i := i+1
end_while
Output: K, the cover set of (Pattern,DomainSet) pairs

We assume the existence of a function *match* s.t. match(P,d) is true if pattern P matches protein domain description d, according to the definition in [17].

Note: it is not guaranteed that any P_i exclusively matches domains from its match set S_i and to no other domains from any other set S_j ($j \neq i$). I.e. the grouping generated on the example set is not a partition, and P_i is therefore *characteristic* of S_i, not a classifier function. We refer to these groups as *clusters* in the manner of [22] [23].

We can trivially extract the following from K:

(1) The composite pattern P for the initial set of examples:
$P = \{ P_i \mid (P_i,S_i) \in K\}$

(2) The set of clusters C for K
$C = \{ S_i \mid (P_i,S_i) \in K\}$

Complexity of the algorithms

The task of matching a TOPS pattern and a TOPS diagram is essentially that of sub-graph isomorphism, and hence NP-complete since the maximal clique problem is NP-complete; this result is not changed for vertex ordered graphs. Also, the relatively small number of edges cannot be exploited to obtain polynomial algorithms, since in [24] and [25] similar graph structures are considered that are even simpler (the vertex degree is 0 or 1) and for such graphs the subgraph isomorphism problem is proven to be NP -complete. There are several non-polynomial algorithms for subgraph isomorphism problem, the most popular being by Ullmann [26] and McGregor [27]. Our algorithm can be regarded as a variant of McGregor's method which is based on constraint satisfaction; however there is an additional mechanism of re-computing constraints which is periodically invoked. For practical purposes it is also worth noting the complexity of graphs that have to be dealt with in TOPS formalism – the maximal number of vertices is around 50 and the number of edges is comparatively small and similar to the number of vertices. A similar class of graphs has been considered by Koch et

al. [28] whose authors describe a maximal common subgraph algorithm based on searching for maximal cliques in a vertex product graph. Although this method would seem to be applicable to TOPS, it is only practical for finding maximal common subgraphs for two graphs and is not directly useful for finding motifs for larger sets of proteins.

The worst time complexity for the simple pattern extension algorithm for n TOPS diagrams with k secondary structure elements (helices and strands) is $O(n*k^k)$, i.e. proportional to the size of the set of examples in the learning set [18]. For a set of n TOPS diagrams, which eventually can be grouped into j clusters, our clustering algorithm has complexity order n^2/j.

Comparison with other clustering techniques

Traditional clustering techniques based on pair-wise comparison and then hierarchical clustering have complexity order n^2 for n objects, based on the comparison operation, and an additional penalty for the clustering. We could have used our protein topology distance method, described in [19] to provide the data for such clustering. However, if characteristic patterns are to generated for significant clusters, then having identified in some way these sets, pattern discovery still has to be performed, and thus there is a further overhead order at least linear in the number of topology diagrams. The advantage of the method presented in this paper is that clustering and pattern discovery are performed concurrently, with the lower complexity described above.

Evaluation on NAD binding domains

We selected the set of 14 NAD binding domains listed in Table 2 as examples, and constructed TOPS diagrams for each one. The sequence identity for these domains varied from 5% to 95% We then generated a cover of this set using our algorithm, and extracted the clusters (i.e. the DomainSet from each (Pattern,DomainSet) pair in the cover). In order to evaluate the groupings we performed a pairwise comparison of the domains using our structure comparison program described in [19], and then performed single linkage clustering on the pairwise distances.

Characterising CATH using pattern unions

We have generated covers for CATH H-level superfamilies based on the Nreps non-redundant set from CATH version 2.0 (http://www.biochem.ucl.ac.uk/bsm/cath). Since our pattern discovery method is designed to work with domains with significant beta sheet content, we have restricted our data set to families with significant beta sheet content, i.e. CATH classes 2 and 3. Covers for each superfamily were generated using various pruneval values and the corresponding composite patterns extracted from the cover; we also generated composite pattern comprising one simple pattern for each family, indicated in the following by pruneval=g1. We then evaluated the patterns for each family against the entire CATH database, computing the sensitivity and specificity as follows:,

sensitivity sn = TP/(TP+FN) and specificity sp = TN/(TN+FP)

where TP stands for true positive, FP for false positive, TN for true negative and for FN false negative. Given a composite pattern $P = \{p_1, \ldots, p_n\}$ In this case,

- TP indicates a match of any simple pattern $p_i \in P$ to a domain of the correct Hfamily
- FP indicates a match of any simple pattern $p_i \in P$ to a domain of an incorrect Hfamily
- TN indicates a failed match of all simple patterns $p_i \in P$ to a domain of an incorrect Hfamily

- FN indicates a failed match of all simple patterns $p_i \in P$ to a domain of the correct Hfamily

Note that a false negative can result because the patterns were generated from a training set, but evaluated over the entire database.

Results

NAD binding domains

With a pruneval cutoff value of 5, we generated 5 clusters, shown below with their functions and organisms, and illustrated in Figure 6.

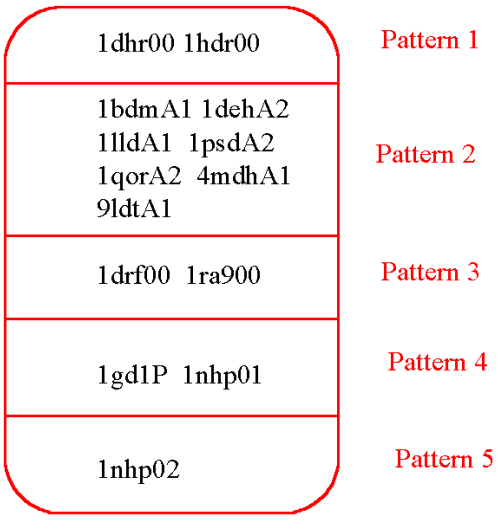

Figure 6: Grouping data by discovered patterns

The groupings generally respect those arrived at by considering function:

Table 2: clustering of NAD/NADP binding domains

Domain	Function & organism	Group
1hdr00	Dihydropteridine reductase (human)	1
1dhr00	Dihydropteridine reductase (rat)	1
9ldtA1	Lactate dehydrogenease (pig)	2
1lldA1	Lactate dehydrogenase (bacterial)	2
4mdhA1	Malate dehydrogenase (pig)	2
1bdmA1	Malate dehydrogenase (bacterial)	2
1qorA2	Quinone oxido-reductase (bacteria)	2
1dehA2	Alcohol dehydrogenase (human)	2
1psdA2	D-3-phosphoglycerate dehydrogenase (bacteria)	2
1ra900	Dihydrofolate reductase (bacterial)	3
1drf00	Dihydrofolate reductase (human)	3
1nhp01	NADH peroxidase FAD binding (bacteria)	4
1gd1P1	D-glyceraldehyde-3-phosphate dehydrogenase bacteria	4
1nhp02	NADH peroxidase NAD binding (bacteria)	5

In Figure 7 we illustrate the pattern for Group 2.

1bdmA1,1dehA2, 1lldA1, 1psdA2,
1qorA2, 4mdhA1, 9ldtA1

10 nodes, 5 Hbonds, 4 chiralities

Pattern sequence :
[[0,1,3],e(1),h(2),e(3),[0,1,2],e(4),h(5),[0,1],e(6),h(7),e(8),
[0,1],h(9),[0,1],e(10),[0,1,2]]

Hbond pattern :
[3-h(1)-4,8-h(1)-10,6-h(1)-8,1-h(1)-6,1-h(1)-3]

Chirality pattern : [(1,1,3),(4,1,6),(6,1,8),(8,1,10)]

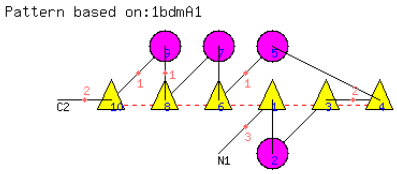

Figure 7: Group 2

Figure 8 shows a dendrogram, generated with the OC program [29] of the complete linkage analysis of the pairwise comparison data, annotated with the functions and organisms of the domains. We have marked on the dendrogram the clusters obtained using our pattern discovery method in red, showing that the groups found using our pattern method correspond well to the functional groupings and also clusters determined by pairwise comparison.

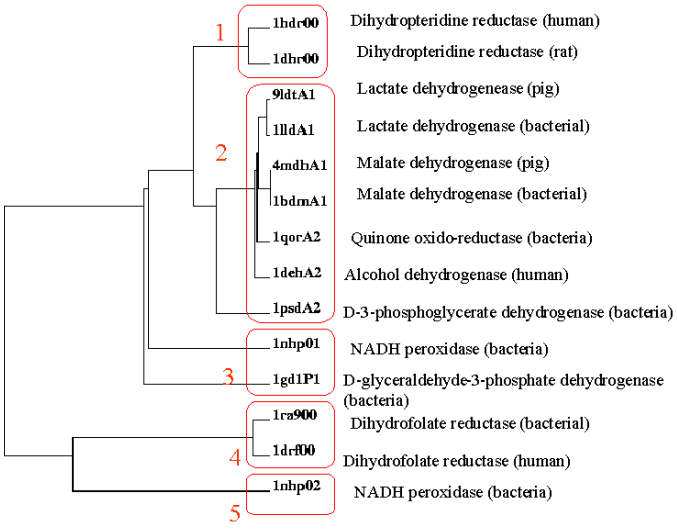

Figure 8: Dendrogram of pairwise comparisons annotated by discovered groups

CATH H-level superfamilies

Figure 9 shows the specificity of automatically discovered patterns for various values of pruneval.

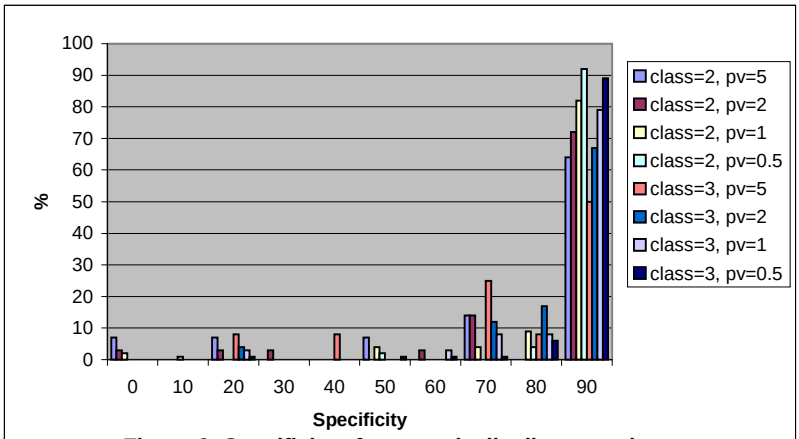

Figure 9: Specificity of automatically discovered patterns

These results are summarised in Table 3, and show clearly that the specificity of the patterns increases as the pruneval decreases.

Table 3: Pattern specificities for CATH classes, generated by different prunevals

		Pruneval		
Class	***0.5***	***1***	***2***	***5***
2	96.6	93.0	85.6	79.0
3	95.9	92.9	90.4	77.8
2&3	94.3	89.4	85.9	77.8

In Figure 10 we have plotted sensitivity against specificity for composite patterns discovered for the CATH H-level superfamilies, using different prunevalues.

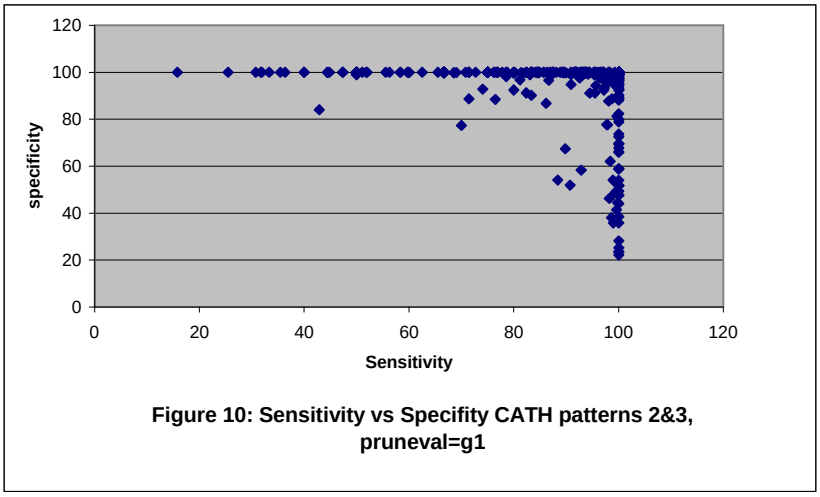

Figure 10: Sensitivity vs Specifity CATH patterns 2&3, pruneval=g1

It is of interest to note that patterns rarely exhibit both poor sensitivity and poor specificity, but there are many families whose patterns have high sensitivity but low specificity, or vice-versa. This result is likely to be cause by the way in which the CATH families are generated, and for example the training set (N-reps) not being topologically representative of the test set (entire superfamily). In Figure 11 we have plotted the number S-level subfamilies for each H-level superfamily against the number of sub-patterns for that H-level superfamily. The results show that as the pruneval is decreased, the relationship

becomes more linear, giving an indication that our algorithm is dividing up the superfamilies into the same number of subfamilies as defined by the CATH group.

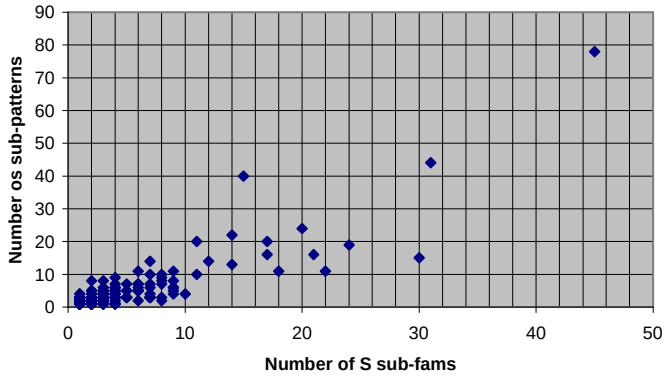

Figure 11: Number of sub-patterns versus number of groups in CATH Hfams

Summary

Bioinformatics is often about adapting existing techniques to new problem domains, and in doing so coming up with novel solutions. In this tutorial paper we hope to have illustrated this process. We have reviewed some pattern description, pattern discovery and string comparison techniques which have been designed to analyse biological sequences, and showed how essential ideas from these algorithms these can be adapted for application to abstract representations of protein structures.

Acknowledgements

DG's work has been partially supported by an EPSRC Visiting Research Fellowship at the European Bioinformatics Institute (1998) and a Leverhulme Research Fellow at University College, University of London (2000). JV's work has been partially supported a Wellcome Trust International Research Development Award.

References

1 W. H. Elliott and D. P. Elliott, Biochemistry and Molecular Biology, OUP, 1997
2 L. Stryer. Biochemistry. W. H. Freeman & Co, 2001
3 I. Eidhammer, I. Jonassen and W. Taylor: Structure Comparison and Structure Patterns, Journal of Computational Biology, 7:5 pp 685-716, 2000
4 A. Brazma, I. Jonassen, I. Eidhammer and D. R. Gilbert: Approaches to the automatic discovery of patterns in biosequences, Journal of Computational Biology. 5:2 277-303, 1998
5 I. Eidhammer, D. R. Gilbert, I. Jonassen, M. Ratnayake and S. V. Grindhaug: A Constraint Based Structure Description Language for Biosequences, Journal of Constraints 6:2/3, 2001
6 I. Jonassen, J. F. Collins and D. G. Higgins: Finding flexible patterns in unaligned sequences. Protein Science 4(8):1587-1595, 1995.
7 I. Jonassen: Efficient discovery of conserved patterns using a pattern graph. CABIOS, 13:509-522, 1997.
8 V.I. Levenshtein: Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Akademii nauk SSSR (in Russian)*, 163(4):845-848, 1965. Also in Cybernetics and Control Theory, 10(8):707-710, 1996.
9 D. Gusfield. Algorithms on strings, trees and sequences, CUP 1997
10 P. A. Pevzner. Computational Molecular Biology. MIT press, 2000

11 A. Bairoch, P. Bucher, and K. Hofman. The PROSITE database, its status in 1995. *Nucleic Acids Research*, 24(1):189-196, 1996.
12 M.J.E. Sternberg and J.M. Thornton. On the conformation of proteins: The handedness of the connection between parallel beta strands. *Journal of Molecular Biology*, 110:-283, 1977
13 T.P. Flores, D.M. Moss, and J. M. Thornton: An algorithm for automatically generating protein topology cartoons. *Protein Engineering*, 7(1):31-37, 1994
14 D. R. Westhead, T. W. F. Slidel, T. P. J. Flores, and J. M. Thornton. Protein structural topology: automated analysis and diagrammatic representation. Protein Science, 8(4):897-904, 1999.
15 D. R. Westhead, D. C. Hutton, and J. M. Thornton. An atlas of protein toplogy cartoons available on the World Wide Web. *Trends in Biochemical Sciences*, 23, 1998
16 T.W.F. Slidel and J.M. Thornton, Chirality in Protein Structure , in Protein Folds: a distance-based approach, Henrik Bohr and Soren Brunak (eds), CRC press, 1996, 253-264
17 D. R. Gilbert, D. R. Westhead, N. Nagano, and J. M. Thornton. Motif-based searching in tops protein topology databases. *Bioinformatics*, 15(4):-326, 1999
18 J. Viksna and D. R. Gilbert, Pattern matching and pattern discovery algorithms for protein topologies, WABI 2001: 1st Workshop on Algorithms in BioInformatics, August 2001, , LNCS 2149 pp 98-111, ISBN 3-540-42516-0
19 D. R. Gilbert, D. R. Westhead, J. Viksna and J. M. Thornton, Topology-based protein structure comparison using a pattern discovery technique, Journal of Computers and Chemistry, 26:1, 23-30, 2001
20 D. R. Gilbert and D. R. Westhead, A topology-based pattern discovery method for protein structure characterization and comparison, submitted to *Proteins*
21 A. Brazma, I. Jonassen, J. Vilo and E. Ukkonen, Pattern Discovery in Biosequences., Proceedings of Fourth International Colloquium on Grammatical Inference (ICGI-98) (1433) (pp. 255--270) July 1998. *Springer.*
22 B. Mirkin (1998) Least-Squares Structuring, Clustering, and Data Processing Issues, The Computer Journal, 41, no. 8, 519-536.
23 B. Mirkin and I. Muchnik (1998) Combinatorial Optimization in Clustering, in D.-Z. Du and P.Pardalos (Eds.) Handbook of Combinatorial Optimization, 2, Boston, Ma.: Kluwer Academic Publishers, 261-329.
24 P. A. Evans, Finding common subsequences with arcs and pseudoknots. Proceedings of Combinatorial Pattern Matching 1999, LNCS 1645 (1999) 270–280.
25 K. Zhang., L. Wang, B. Ma: Computing similarity between RNA structures. Proceedings of Combinatorial Pattern Matching 1999, LNCS 1645 (1999) 281–293.
26 J. R. Ullmann,. An algorithm for subgraph isomorphism. Journal of the ACM 23 (1976) 31–42
27 J. J. McGregor,. Relational consistency algorithms and their application in finding subgraph and graph isomorphisms. Information Science 19 (1979) 229–250.
28 I. Koch., T. Lengauer, E. Wanke.: An algorithm for finding maximal common subtopologies in a set of protein structures. Journal of Computational Biology 3 (1996) 289–306.
29 G.J. Barton,. The OC program: http://barton.ebi.ac.uk/new/software.html, 1997