Protein Science (1995), *4*:1587-1595. Cambridge University Press. Printed in the USA.
Copyright © 1995 The Protein Society

Finding flexible patterns in unaligned protein sequences

INGE JONASSEN,[1] JOHN F. COLLINS,[2] AND DESMOND G. HIGGINS[3]

[1] Department of Informatics, University of Bergen, HIB, N5020 Bergen, Norway
[2] Biocomputing Research Unit, ICMB, Darwin Building, King's Buildings, Mayfield Road, Edinburgh EH9 3JR, United Kingdom
[3] European Bioinformatics Institute, Hinxton Hall, Hinxton, Cambridge CB10 1RQ, United Kingdom

(Received March 16, 1995; Accepted May 23, 1995)

Abstract

We present a new method for the identification of conserved patterns in a set of unaligned related protein sequences. It is able to discover patterns of a quite general form, allowing for both ambiguous positions and for variable length wildcard regions. It allows the user to define a class of patterns (e.g., the degree of ambiguity allowed and the length and number of gaps), and the method is then guaranteed to find the conserved patterns in this class scoring highest according to a significance measure defined. Identified patterns may be refined using one of two new algorithms. We present a new (nonstatistical) significance measure for flexible patterns. The method is shown to recover known motifs for PROSITE families and is also applied to some recently described families from the literature.

Keywords: algorithm; flexible gaps; patterns; protein families; PROSITE

A common problem in protein sequence analysis is to search for common sequence patterns or motifs in groups of functionally related proteins. Such patterns may be the result of common ancestry combined with conservative evolutionary pressure to maintain important residues at active sites and other functionally important parts of the protein. It is not always possible to identify conserved patterns in protein families. When they do occur, however, they can be very simple and useful tools in helping to identify new members of the families and in trying to understand the relationship between sequence, structure, and function.

One situation where the identification of shared patterns is of great practical importance is where one has a set of functionally related sequences and one wishes to know if the common function is reflected in the sequences. This can be tested by attempting to align the sequences and looking for any conserved blocks of alignment, e.g., bacterial and bacteriophage DNA binding proteins of the lambda repressor family will show a conserved block of 22 amino acids corresponding to the helix-turn-helix DNA binding domain (Dodd & Egan, 1990). This works well when the sequences are easy to align. In some cases, however, the alignment is very difficult to obtain or evaluate. The conserved regions may be very short or repeated within the proteins. An alternative is to take the unaligned sequences and use a pattern searching program to look for conserved patterns. Such patterns show up as exactly or highly conserved positions separated by fixed or variable spacing.

A second practical use of patterns is to find diagnostic signatures for families. This is well illustrated by the PROSITE database of protein patterns (Bairoch & Bucher, 1994). Here, groups of functionally and evolutionarily related proteins are listed along with patterns that can be used to distinguish each family from all (or most) other sequences in the SWISS-PROT protein sequence database (Bairoch & Boeckmann, 1992). These patterns are extremely fast and simple to use in order to identify new members of the families. The diagnostic power of each pattern can be assessed readily by the numbers of false-positive and false-negative examples found by the pattern (the number of sequences that contain the pattern that are not members of the family and the number of sequences that do not contain the pattern but which are members of the family, respectively). These numbers and the corresponding SWISS-PROT sequence identifiers are listed for each pattern in PROSITE. Currently, these patterns are extracted semimanually.

There are several computer programs available for identifying conserved patterns in sets of unaligned sequences. All have disadvantages. In this paper, we describe some improvements on the available methods that allow more biologically realistic patterns to be identified. Ideally, one would like to use a measure of pattern significance (nonrandomness) and to select the most significant patterns from the sequences, allowing for ambiguity at each position and variable spacing between all of the elements. Computationally, this is a very difficult problem as the number of possible patterns to search for (and examine for

Reprint requests to: Inge Jonassen, Department of Informatics, University of Bergen, HIB, N5020 Bergen, Norway; e-mail: inge.jonassen@ii.uib.no.

significance) is enormous. Further, the estimation of significance for very general patterns in protein sequences is still an open problem.

All of the existing methods impose some constraint on the type of patterns that can be found. The simplest constraint is to look for short conserved words or k-tuples, for example (Ogiwara et al., 1992; Saqi & Sternberg, 1994; Wang et al., 1994). A conserved word is a consecutive series of, say, three to five conserved residues. This is algorithmically simple because one can generate a table of all occurring words in advance. It is then a relatively simple matter to search this table for words occurring in all of the sequences. Conserved words can then be joined by flexible regions (variable spacing) to make larger and more interesting patterns (Ogiwara et al., 1992; Wang et al., 1994). Ambiguity can be introduced by searching for all words that are within some preset number of differences (either substitutions or insertions and deletions) from each other (Wang et al., 1994). This can be fast and simple to do but only works when there are some conserved (or largely conserved) words to begin with. Patterns composed of isolated conserved residues separated by totally ambiguous positions are hard to find.

A second approach is to look for small numbers (e.g., 3) of exactly or highly conserved positions, separated by short fixed spacing. This was first done by Smith et al. (1990) and was used to provide the initial conserved segments for the BLOCKS database of conserved sequence blocks (Henikoff & Henikoff, 1991). Here, one make a table of all triplets of conserved residues with all spacings between the residues up to a preset maximum. The algorithm of Neuwald and Green (1994) allows for any number of fixed positions and fixed spacings between them up to a maximum total pattern length that is set by the user. This algorithm combines a significance measure for patterns and a depth first search strategy with a data structure (the "block" data structure) that allows one to quickly check the occurrences of any potential pattern. This method is fast and usually guaranteed to find any patterns over a significance threshold. The algorithm does allow for very limited ambiguity at some of the pattern positions (e.g., the most common conserved substitutions) but does not allow for variable length spacing between the main pattern elements.

In this paper, we describe some improvements to the method of Neuwald and Green that allow for greater ambiguity at partially conserved pattern positions and that allow for limited variable spacing between pattern elements. Biologically, variable spacing is important, because even in well-conserved regions, variable loop sizes can occur. This allows one to quickly and automatically generate patterns from unaligned sets of protein sequences that are very similar to those used in the PROSITE database. It is still not possible to search for totally general patterns in reasonable time, but the improvements described here are significant improvements over existing methods. We demonstrate the usefulness of the software with some examples from PROSITE and with some recently published examples from the literature.

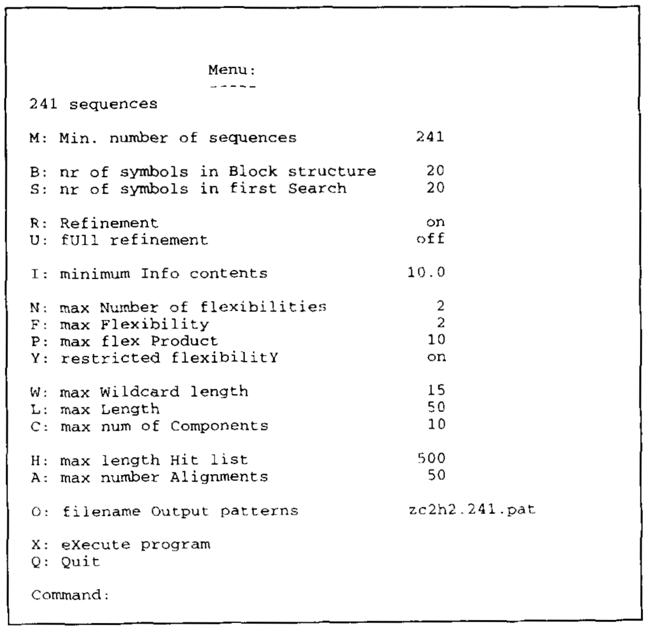
Results

The Pratt program

We have developed a program called Pratt that, given a set of unaligned protein sequences, finds patterns matching a minimum number of these sequences. The user specifies the minimum number of sequences to be matched and the class of pattern to be searched for. We describe the Pratt program and then give results of running the program on some test cases. We adopt PROSITE (Bairoch & Bucher, 1994) notation for describing patterns. For example D-x(2,3)-[DE] is a pattern matching four- and five-segments starting with D and ending with D or E. The middle part of the pattern matches any two or three amino acids and is called a wildcard region. We say that a pattern matches a sequence if it matches a segment from the sequence. PROSITE is a collection of such patterns, containing approximately 1,000 entries most of which contain a pattern (not always perfectly) diagnostic for a family of protein sequences.

The program accepts sequences in FASTA (Pearson & Lipman, 1988), SWISS-PROT (Bairoch & Boeckmann, 1992), and GCG (Devereux et al., 1984) formats. The opening menu is shown in Figure 1. The parameters and their meaning will be described below, and the algorithmic details of how the parameters affect the working of Pratt are described in the Methods section.

Pratt first searches the space of patterns, as constrained by the user, and compiles a list of the most significant patterns (according to our nonstatistical significance measure) found to be matching at least the user-defined minimum number of sequences. If the user has not switched off the refinement (option R on the menu), these patterns will be input to one of the pattern refinement algorithms. The most significant patterns resulting from this are then output to a file. An overview of the algorithm is given in Figure 2.



```
                    Menu:
                    -----
241 sequences

M: Min. number of sequences              241

B: nr of symbols in Block structure       20
S: nr of symbols in first Search          20

R: Refinement                             on
U: fUll refinement                       off

I: minimum Info contents                10.0

N: max Number of flexibilities             2
F: max Flexibility                         2
P: max flex Product                       10
Y: restricted flexibilitY                 on

W: max Wildcard length                    15
L: max Length                             50
C: max num of Components                  10

H: max length Hit list                   500
A: max number Alignments                  50

O: filename Output patterns          zc2h2.241.pat

X: eXecute program
Q: Quit

Command:
```

Fig. 1. Pratt's menu, when run on a file containing 241 sequences in the ZINC FINGER C2H2 PROSITE family, showing default parameters. The minimum number of sequences is, by default, the number of sequences in the set given, and the file name for the output is, by default, the input file name appended with the minimum number of sequences and the extension pat. Other parameters are described in the Results section.

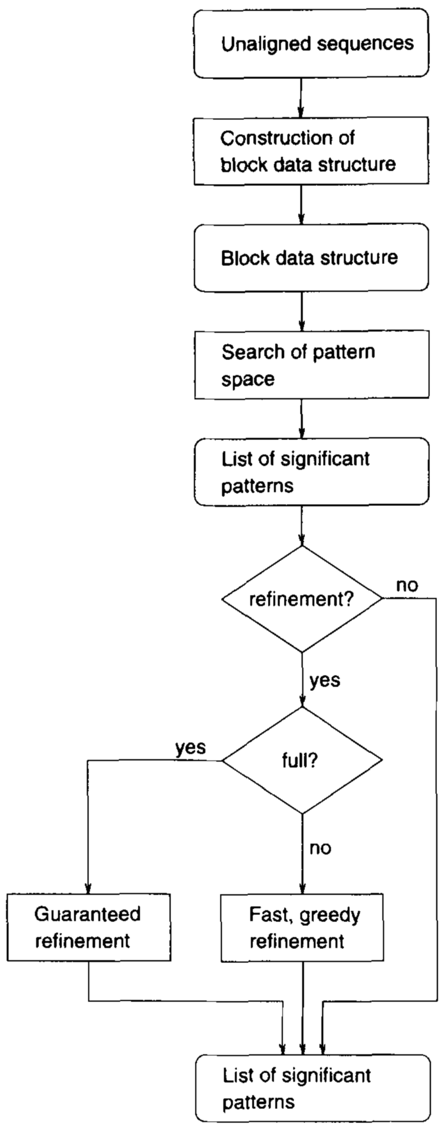


Fig. 2. Outline of program structure. Pratt reads sequences into memory and then allows the user to set parameters controlling Pratt's behavior using the menu in Figure 1. Pratt constructs the internal data structure to be used during the search and then searches the space of patterns (as restricted by the user). The most significant patterns found to be matching the minimum number of sequences (as specified by the user) are input to a refinement algorithm. The user has the choice between two different refinement methods.

Terminology

Using PROSITE notation to describe patterns, a pattern P in the class considered can be written as

$$P = A_1\text{-}x(i_1,j_1)\text{-}A_2\text{-}x(i_2,j_2)\text{-}\ldots\text{-}x(i_{p-1},j_{p-1})\text{-}A_p \qquad (1)$$

where $A_1, \ldots, A_p$ are nonempty sets of amino acids, and $i_1 \le j_1$, $i_2 \le j_2$, ..., $i_{p-1} \le j_{p-1}$ are integers. We call A_1, A_2, ..., A_p the components of P, so p is the number of components in P. A_k is called an identity component if it represents one amino acid and an ambiguous component if it represents more than one. For example, if P is D-x(2)-E, A_1 is the set consisting of amino acid D, $i_1 = j_1 = 2$, and A_2 is the set consisting of amino acid E, and the number of components is $p = 2$. A wildcard region $x(i_k, j_k)$ is fixed if $i_k = j_k$ and hence matches exactly i_k amino acids in a sequence. A fixed wildcard region matching any i_k amino acids can also be written $x(i_k)$. If $j_k > i_k$, we call $x(i_k, j_k)$ a variable or flexible wildcard region, matching between i_k and j_k amino acids in a sequence. We call $j_k - i_k$ the flexibility of the region. For example $x(2,3)$ and $x(8,9)$ have a flexibility of 1, and $x(2,4)$ has a flexibility of 2. A fixed pattern is a pattern containing only fixed wildcard regions; a flexible pattern may contain one or several flexible wildcard regions. The length $L(P)$ of a pattern P is the maximum length of a sequence segment matching P. The length of D-x(2)-E is 4 and the length of P in (Equation 1) is $L(P) = p + j_1 + j_2 + \ldots + j_{p-1}$. The product of flexibilities for the pattern P above is defined as $(j_1 - i_1 + 1)(j_2 - i_2 + 1) \ldots (j_{p-1} - i_{p-1} + 1)$. For example, the product of flexibilities for D-x(2,3)-E-x(2,4)-F is $(3 - 2 + 1) \times (4 - 2 + 1) = 6$. This quantity is used for restricting the memory and time used by the algorithm (see Methods section).

Specifying the class of patterns to be searched

Pratt has parameters defining maximum values for many of the quantities described above. The menu, shown in Figure 1 with default values, allows the user to change these. This allows the user to restrict the class of patterns to be searched for. The less restrictive the limits are, the more subtle the patterns that can be found. However, more memory and time will be needed (see Methods section for more details).

The user can change the maximum length of patterns to be searched for (option L), the maximum number of components in a pattern (option C), and the maximum length of a wildcard region (option W). The user can also change the maximum number of flexible wildcard regions (option N), the maximum flexibility of a wildcard region (option F), and the maximum product of the flexibilities (option P). The option Y allows the user to avoid searching for patterns having two consecutive flexible wildcard regions. Setting the maximum number of flexible wildcard regions to 0, the user instructs Pratt only to look for fixed patterns. We examined the patterns in PROSITE to get an idea of what kind of flexibility we should allow. We identified all variable length wildcard regions in patterns in PROSITE, of the form $x(i,j)$, having flexibility $j - i$. Figure 3 shows the distribution of flexibilities found in PROSITE Release 12.0 (October 1994). This shows that most (85%) of the flexibilities were 1 or 2. Pratt easily deals with this kind of flexibility. Using option M the user can also change the minimum number of sequences that patterns should match.

The search for patterns is done in two phases. The first phase exhaustively searches the space of patterns (as restricted by the user), and then the most significant patterns found in the first phase are refined. The set of possible values for pattern components, both during the initial search and for the refinement step, is read from a user-modifiable text file Pratt.sets. This file contains one value (a set of amino acids) on each line. During the initial search, components of a pattern can take on the values corresponding to the first s lines in this file (s can be set by the user using option S). The first 20 lines in Pratt.sets should contain all the amino acids, one per line, and then the next lines contain sets of amino acids that share some physiochemical

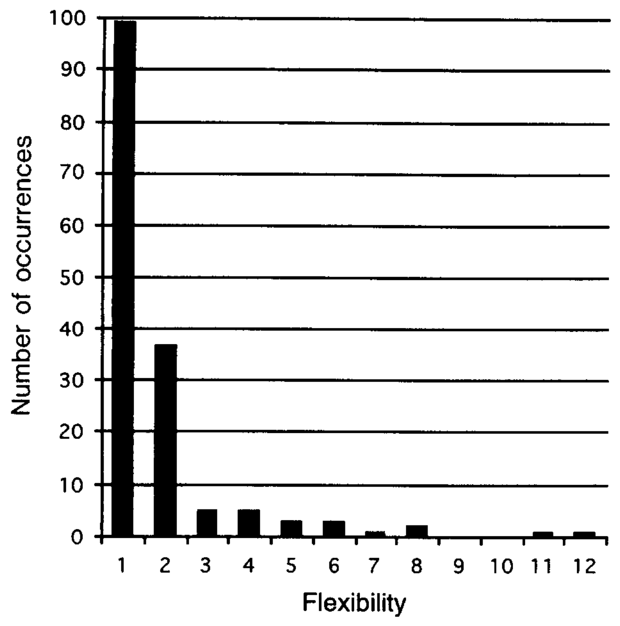


Fig. 3. Distribution of lengths of flexibilities found in PROSITE release 12.1 (October 1994). A wildcard region $x(i,j)$ has flexibility $j-i$, and the histogram shows how many flexible wildcard regions occur for a range of flexibility values. All flexible wildcard regions in PROSITE are represented in the histogram, except for one having flexibility 50.

properties or that one expects to be interchangeable in constrained positions. By default $s = 20$, and initially Pratt searches for patterns having only identity components. For example, the initial search may find the pattern D-*x*(2)-E, which next could be refined to D-*x*-[ILVF]-E-*x*-[DE].

Choosing refinement algorithm

For the refinement step, the user has a choice between two different algorithms. The fastest, which is used by default, can introduce new ambiguous symbols into fixed length wildcard regions in the already identified patterns and it can append the patterns with fixed length wildcard regions and ambiguous components. It is greedy and is therefore not guaranteed to identify all refined conserved patterns. The set of allowed ambiguous symbols is read from Pratt.sets. This refinement algorithm is fast and requires little memory.

The second refinement algorithm is used when the user switches on the full refinement option (U on the menu). This method is guaranteed to find all conserved refined patterns in the class of patterns defined by the parameter values. It can introduce new ambiguous symbols into both fixed and flexible wildcard regions and can also append the identified pattern with wildcard regions and ambiguous symbols. It requires all ambiguous symbols to be represented in Pratt's internal data structure, which requires extra memory (see Methods section). The user specifies the number of lines from Pratt.sets to be represented in the block data structure using option B.

For example, if the pattern D-*x*(1,2)-[ILV]-E-*x*(2)-[DE]-F matches all the sequences, the initial search (using default parameters) will be guaranteed to find the pattern D-*x*(2,3)-E-*x*(3)-F. Using the simple refinement algorithm this may be refined to D-*x*(2,3)-E-*x*(2)-[DE]-F (if there is a line in Pratt.sets containing both D and E). If the second refinement algorithm is used, and if both ILV and DE are sets included in the block data structure, this will be guaranteed to find the pattern D-*x*(1,2)-[ILV]-E-*x*(2)-[DE]-F.

Ranking patterns

A measure of significance is calculated for each of the identified conserved patterns. The significance of a pattern is calculated as the sum of the information contents of the components minus a penalty for each flexibility. The significance measure used is not a statistical one. A more detailed description is given in the Methods section. If the significance of a pattern is above the significance threshold (which can be changed by the user), the pattern is added to a list that is sorted by significance. The maximum number of elements in the list is by default 500 and can be changed by the user. Both patterns found by the initial search and patterns found during the refinement phase are added to such lists. Finally Pratt outputs the identified patterns sorted by significance.

How to obtain the program

Pratt is written in ANSI C. It has been compiled and tested on DEC Alpha, Sun Sparc 10, and Silicon Graphics Challenge M workstations and should be portable to other platforms having an ANSI C compiler. The program is available from anonymous ftp servers ftp.ebi.ac.uk and ftp.ii.uib.no.

Test cases

We demonstrate how Pratt works on real sequence data, using some examples of protein sequence families, three of them from PROSITE (release 12.0). The sequences in SWISS-PROT corresponding to each of the PROSITE families were obtained using SRS (Etzold & Argos, 1993). The Pratt.sets file used for the test cases contains: (1) 20 lines each containing one single amino acid symbol; (2) symbols from the amino acid class hierarchy, described in Smith and Smith (1990): DE, KRH, NQ, ST, ILV, FWY, AG; and (3) sets with 10 or fewer members listed in Table 1 of Taylor (1986). The sequences in SWISS-PROT (release 29) matching the identified patterns (and the PROSITE patterns) were found using the MacPattern program (Fuchs, 1994). The run times given were obtained using a DEC alpha workstation.

Example families from PROSITE

We use two zinc finger families and one snake toxin family from PROSITE as test cases. Data about the PROSITE entries are summarized in Table 1 and results from the program being run on these families are summarized in Table 2. The patterns in PROSITE are intended to be diagnostic for the three families of protein sequences.

Zinc finger C2H2 (accession number PS00028)

This is a family of eukaryotic and viral DNA binding proteins. The PROSITE pattern describes the DNA binding domain itself. There are 241 family members listed in PROSITE, including one false negative (does not contain the pattern but is a C2H2

Table 1. *Summary of the three test cases from PROSITE*

	PROSITE ID	No. of sequences	Average length	PROSITE pattern
1	Zinc finger C2H2	241	393	C-*x*(2,4)-C-*x*(3)-[LIVMFYWC]-*x*(8)-H-*x*(3,5)-H
2	Zinc finger C3HC4	47	644	C-*x*-H-*x*-[LIVMFY]-C-*x*(2)-C-[LIVMYA]
3	Snake toxin	164	64	C-P-*x*(6,8)-[LIVYST]-*x*-C-C

zinc finger-containing protein) and four sequences of unknown status (might be zinc finger proteins). The PROSITE pattern also matches 20 other sequences (false positives). Using default parameters, the pattern C-*x*(2,4)-C-*x*(12)-H-*x*(3,5)-H (matching 292 sequences in SWISS-PROT) was found in 29 s. Setting the minimum number of sequences (option M on the menu) to 240, we find the pattern C-*x*(2,4)-C-*x*(3)-[ILVFYC]-*x*(8)-H-*x*(3,5)-H, which is a restriction of the PROSITE pattern. This second run takes 36 s. This pattern finds 260 sequences in SWISS-PROT (release 29), giving the same number of false positives as the original PROSITE pattern.

Zinc finger C3HC4 (accession number PS00518)

This is a family of eukaryotic and viral proteins containing a conserved cysteine-rich domain, probably involved in zinc-dependent binding to DNA. The PROSITE entry lists 47 members, 46 matching the PROSITE pattern. Using default parameters, we find the pattern C-*x*-H-*x*-[ILVMFYC]-C-*x*(2)-C, in 3 min and 55 s. This pattern matches 59 sequences in SWISS-PROT (11 false positives), whereas the PROSITE pattern gives three false positives. Disallowing flexibilities, the same pattern is found in 9 s. Reducing the minimum number of sequences to 46, Pratt finds the pattern C-*x*-H-*x*(2)-C-*x*(2)-C-[ILVMY] matching 49 sequences in SWISS-PROT (release 29), giving three false positives.

Snake toxin family (accession number PS00272)

This family includes 164 cytotoxins, neurotoxins, and venom peptides. The proteins all have four disulfide bridges, one of which is covered by the PROSITE pattern. This pattern matches 155 out of the 164 family members and 13 other sequences (false positives). Running the program using default parameters, no conserved pattern (with significance above the threshold) is found (running time: 1 s). Setting the minimum number of sequences to 155, the most significant pattern found was G-C-*x*(1,3)-C-P-*x*(8,10)-C-C-*x*(2)-[EPDN], which is quite similar to the PROSITE pattern (see Table 1). This second run took 51 s. The fuzzy position [LIVYST] in the PROSITE pattern was not identified. The reason is that it is within a flexible region, and the greedy refinement procedure does not search for fuzzy symbols within flexible regions. The pattern identified by Pratt matches 155 sequences in SWISS-PROT (no false positives) and is therefore a more diagnostic pattern for the family than the PROSITE pattern.

Other test cases

We also tested Pratt on two further protein families from the recent literature: the SH3 domain (Musacchio et al., 1992) and the PHD finger domain (Aasland et al., 1995). These test cases are more difficult than the ones above and we therefore included all the sets from Table 1 in Taylor (1986) in the file Pratt.sets.

SH3 domain

These are peptide binding domains found in many eukaryotic signal transduction proteins. The domain is short (roughly 63 amino acids long) and weakly conserved between the different examples. In the structure-based alignment of Thompson et al. (1994a), there is only one exactly conserved residue. We ran Pratt on the 64 sequences in Musacchio et al. (1992). Only the SH3 domains of the sequences were included in the search. Default parameters gave no patterns in less than 1 s. Including the seven amino acid sets from Smith and Smith (1990), DE, KRH, NQ, ST, ILV, FWY, and AG, in the initial search, we found no significant patterns using 7 s. Setting the minimum number of sequences to 55, the program runs for 18 s and finds some significant patterns. The two most significant patterns are A-*x*(3)-[FWY]-*x*(7,9)-[ILV]-*x*(4)-[GVSDN]-*x*(2)-[ILVMFYW]-*x*(3)-[GREAKSDNQ], and G-*x*-[IVF]-P-*x*(2,4)-[ILV], both corresponding to columns in the alignment of Thompson et al. (1994a). Next we used Pratt to find conserved patterns on a

Table 2. *Summary of results obtained when running Pratt on the three families in Table 1*[a]

	Parameters used	Time used	Most significant pattern identified
1	Default	0:29	C-*x*(2,4)-C-*x*(12)-H-*x*(3,5)-H
	NMIN = 240	0:36	C-*x*(2,4)-C-*x*(3)-[ILVFYC]-*x*(8)-H-*x*(3,5)-H
2	Default	4:47	C-*x*-H-*x*-[ILFMVYC]-C-*x*(2)-C
	No flexibilities	0:09	C-*x*-H-*x*-[ILFMVYC]-C-*x*(2)-C
	NMIN = 46, no flex.	0:18	C-*x*-H-*x*(2)-C-*x*(2)-C-[ILVMY]
3	Default	0:01	No pattern
	NMIN = 155	0:51	G-C-*x*(1,3)-C-P-*x*(8,10)-C-C-*x*(2)-[EPDN]

[a] The numbers in column one refer to the same column in Table 1. NMIN is the minimum number of sequences required to match the pattern (option M from the menu). Times are given as minutes and seconds.

larger data set of 70 SH3 domain containing proteins. The proteins were selected from the results of a profile search (Thompson et al., 1994b) against SWISS-PROT. In this case, the complete proteins were used (average length 721 residues). A search with default parameters gave no significant patterns in 11 min. Setting the minimum number of sequences to 65 and otherwise using default parameters, we found among others the patterns G-[ILMFYWKQR]-[ILVMFAN]-P-*x*(0,2)-Y-[ILVP]-[VKTCEAGSQR] and W-*x*(12,14)-P-[VKTCAGSDR]-[VMKTAPSN]-Y-[ILVMF]-[VKTEASQR]. This run took 49 min. Sixty-seven sequences in SWISS-PROT (release 30) match both patterns.

PHD finger domains

We analyzed 27 sequences containing the PHD finger (Aasland et al., 1995). The average length of the sequences is 874 amino acids. Using default parameters, Pratt takes 18 s but outputs no patterns. Setting the minimum number of sequences to 24 and otherwise using default parameters, Pratt uses 13 min, and outputs the pattern C-*x*(2,4)-C-[YCEPGSDNQR]-*x*-[VMFWHTAPGSN]-*x*-H-*x*(2)-C-[ILVMFYHTCA]-*x*(11)-[YWCEPGSDNQ]-*x*(2)-[IFHCAPGSDN] (which is a refinement of the pattern C-*x*(2,4)-C-*x*(4)-H-*x*(2)-C) and many variations of it.

Discussion

The program Pratt is a flexible tool for finding conserved patterns in a set of unaligned protein sequences. It allows the user to specify the type of patterns to be searched for, and it is guaranteed to find all conserved patterns in the specified class. Using both flexible wildcard regions and ambiguous positions, Pratt can find biologically interesting patterns. The test cases show that it can recover known patterns for some PROSITE families, and in one case it detected a more selective pattern than the one given in PROSITE (snake toxin family). It was also shown to identify interesting conserved patterns in some recently described sequence families not yet in PROSITE.

In many cases the program is very fast. For example it finds the pattern C-*x*(2,4)-C-*x*(3)-[ILVFYC]-*x*(8)-H-*x*(3,5)-H conserved in 240 zinc finger sequences in 36 s. The algorithm for finding patterns with variable wildcard regions is an extension of the depth first search strategy described in Neuwald and Green (1994). The more ambiguous positions are detected during the refinement phase. Each of the most significant patterns detected during the search of pattern space is analyzed to check if new ambiguous positions can be added. This two-phase strategy allows Pratt to detect weakly conserved positions without sacrificing too much efficiency. If Pratt cannot find any significant conserved pattern in the specified class, this is normally reported quite quickly, and the user can easily rerun the program using more permissive parameter values.

Pratt also has weaknesses. It is not very well suited for finding patterns conserved in a small subset of the sequences input. It will be guaranteed to find all such patterns, but it may take a long time. Also it is not able to find patterns having no (or just one or two) well-conserved positions. The significance measure used by Pratt is not well justified theoretically. This is a common problem with all the methods that allow gaps (Roytberg, 1992).

How does Pratt compare to other programs for finding conserved patterns in a set of unaligned protein sequences? As Pratt has been specifically designed to find patterns of conserved residues having variable spacing, we are particularly interested in whether other existing programs can find this type of pattern. The two programs for finding patterns of constantly spaced conserved positions described in Smith et al. (1990) and Neuwald and Green (1994) do not have this ability. Neither do they allow for ambiguous positions in the same general way as does Pratt. However the second method seems to be well suited for finding patterns conserved in an arbitrary sized subset of the sequences; Pratt is not well suited for this purpose.

The method described in Roytberg (1992) identifies a set of sequence segments, one from each sequence, so that all pairwise distances between the segments are below a threshold. The distance measure allows for substitutions and insertions/deletions, which means that the method can find conserved patterns allowing for gaps. It will be hard for this method to efficiently find patterns of conserved residues with constant or variable spacing. It does not recognize and therefore cannot exploit cases where some positions within such a pattern are conserved and others are allowed to vary freely. Without this information the similarity between two segments matching a pattern may not seem significant. Using a liberal distance threshold, this method may be able to find this kind of pattern, but not very efficiently.

The methods in Ogiwara et al. (1992) and Wang et al. (1994) are based on the detection of more or less strictly conserved k-tuples and can find patterns consisting of k-tuples with variable length spacing between them. These methods work best for k having a value of at least 3, and for these values of k they will probably not find a pattern like the one that Pratt found in the zinc finger sequences. On the other hand, these methods, as well as Roytberg (1992) and Saqi and Sternberg (1994), are likely to find patterns that Pratt will not find. The Gibbs sampler-based method (Lawrence et al., 1993) is superior to Pratt at aligning weakly conserved regions lacking strongly conserved positions but is unable to find patterns having variable length spacing between conserved positions.

Conserved patterns in a set of sequences can also be found using multiple sequence alignment programs like Clustal W (Thompson et al., 1994a), followed by manual inspection to find conserved blocks of alignment. In cases where the sequences are difficult to align, this method may miss short conserved motifs. Pratt can be used to search very efficiently for conserved patterns in such sequences. The two approaches are complementary. No one tool will be best for finding all types of patterns. Pratt is a flexible tool for finding conserved patterns and it allows the user to search for patterns of conserved positions with limited variable length spacing. It should be used together with other tools when analyzing a set of sequences believed to be related.

Pratt can be further developed in different directions. The methods can trivially be modified to find repeated patterns in one sequence. This has not been implemented. In this modified version the user would input the minimum number of matches to a pattern in one sequence instead of the minimum number of sequences to match a pattern. Another possibility is to use Pratt as a search engine as part of a larger pattern finding system. This could be used to search for different classes of conserved patterns, running Pratt several times on the same set of sequences using different parameter values. Each time it should allow for more general patterns and/or decrease the minimum number of sequences to be matched. The wrap around could repeatedly run Pratt in this way until a sufficiently significant pattern is identified or until the system decides to give up. This would free the user from having to experiment with Pratt's parameters and would also make it possible to use Pratt in a fully automated fashion.

Methods

The algorithm for searching and pruning the space of possible patterns is an extension of the method described by Neuwald and Green (1994). Their method searches the space of a restricted type of pattern and reports the most significant identified patterns that match any number of sequences, where the significance is a function of both the number of sequences matched and the pattern itself. They are able to find fixed patterns having a minimum number of identity components and a number of ambiguous components (consisting of pairs of amino acids) in a restricted way.

We restrict the search of pattern space to patterns matching more than a minimum number of sequences. This makes it possible for us to prune the search space very efficiently, which allows us to extend the class of patterns that can be found. We briefly describe the block data structure and the basic search algorithm. Both are similar to the ones described in Neuwald and Green (1994), where a more detailed description is given. Next we describe how the algorithm has been modified to allow for more general ambiguous positions and for variable length wildcard regions, and we outline the algorithms for refining patterns.

The basic algorithm

Given N sequences $S^1, S^2, \ldots, S^N$ over some alphabet Σ (typically the set of one letter codes for the amino acids), having lengths $L_1, L_2, \ldots, L_N$, $(S^i = S_1^i, \ldots, S_{L_i}^i)$, we define B_k^S as the set of all k-segments (a k-segment is a substring of length k) from the sequences. To be able to detect patterns near the ends of sequences, k-segments are constructed also at the ends by padding the sequences with dummy symbols not in Σ. Then for all i between 1 and k inclusive, and for each symbol a in Σ, the block data structure contains the set $b_{i,a} \subseteq B_k^S$ of all k-segments having a in position i. This can then be used to efficiently find all segments in B_k^S matching a fixed pattern. For example, the set of segments from the sequences $S^1, S^2, \ldots, S^N$ matching the pattern D-x(2)-E is $b_{1,\mathrm{D}} \cap b_{4,\mathrm{E}}$, i.e., the set of all segments having D in the first position and E in the fourth position.

The basic algorithm uses the block data structure when exploring the space of a restricted class of patterns. This is a variation of the algorithm of Neuwald and Green (1994). We want to find all fixed patterns with only identity components, matching at least N_{min} of the sequences. The search is done in a recursive way. The recursion starts with the empty pattern (denoted e), which matches all k-segments. Let P be the pattern and let M_P be the set of k-segments matching P. So, initially $P = e$ and $M_P = B_k^S$. At each level of recursion, a pattern P (which does match at least N_{min} sequences) is considered. All simple extensions of P, giving patterns within the defined class, are generated. A simple extension of P is P appended with (1) a fixed wildcard region, followed by (2) an amino acid symbol. When extending the empty pattern, no wildcards are appended. So, P may be extended to the pattern $P' = P$-$x(i)$-a where i is an integer and a is an amino acid symbol. The set of segments matching P' is $M_{P'} = M_P \cap b_{L(P)+i+1,a}$. It can be efficiently calculated using M_P and the block data structure. The extended patterns having matches in at least N_{min} sequences are recursively analyzed in the same way. If no simple extension of P matches at least N_{min} sequences, and if P has significance above the threshold, P is added to a list of patterns to be refined and reported.

The algorithm may be used to find fixed patterns with ambiguous symbols by allowing the pattern components to take on either the value of a single amino acid or an ambiguous symbol specifying a set of alternative amino acids. This makes it possible to find patterns such as D-x(2)-[DE]. Neuwald and Green allow for ambiguous positions in a restricted way using a similar approach to the one described here. We let the user specify a set of possible values for the pattern components in the initial search. This is done by specifying the number of lines (s) from the file Pratt.sets to be included. The number of ambiguous symbols to be included in the block data structure is also specified by the user and should be the same as s or bigger than s (if the guaranteed refinement algorithm is used). For each set A (corresponding to an ambiguous symbol) to be included in the block data structure, and for each i between 1 and k inclusive, the block data structure contains the set $b_{i,A}$ of all k-segments having an amino acid in set A in position i. The recursive search algorithm above can be used to search this space of patterns. A simple extension is now a fixed length wildcard region and a symbol corresponding to one of the first s lines in Pratt.sets.

Flexible patterns

We now describe how the method has been extended to find flexible patterns of the form defined in Equation 1. To be able to find all segments matching a flexible pattern in an efficient way, we define for each flexible pattern P a corresponding (finite) set $F(P)$ of fixed patterns. $F(P)$ is defined so that an $L(P)$ segment $s = s_1, \ldots, s_{L(P)}$ matches P (A_1 matching s_1) if and only if s matches at least one pattern Q in $F(P)$ (s_1 matching the first component in Q). The set of fixed patterns corresponding to P, is

$$F(P) = \bigcup_{i_1 \le k_1 \le j_1, \ldots, i_{p-1} \le k_{p-1} \le j_{p-1}} \{A_1\text{-}x(k_1)\text{-}A_2\text{-} \ldots \text{-}A_{p-1}\text{-}x(k_{p-1})\text{-}A_p\}. \quad (2)$$

The number of fixed patterns in $F(P)$ is equal to the product of flexibilities for P defined under Results. See Figure 4 for an

```
P= D-x(2,3)-[DE]-x(4)-F-x(3,4)-G

          F(P)= { D-x(2)-[DE]-x(4)-F-x(3)-G,
                  D-x(2)-[DE]-x(4)-F-x(4)-G,
                  D-x(3)-[DE]-x(4)-F-x(3)-G,
                  D-x(3)-[DE]-x(4)-F-x(4)-G }
```

Fig. 4. Example of a flexible pattern and the corresponding set of fixed patterns.

example of a flexible pattern and the corresponding set of fixed patterns. The set of segments matching P is $M_P = \bigcup_{Q \in F(P)} M_Q$, which can be calculated as above, because all patterns in $F(P)$ are fixed patterns. Below we give an efficient way to calculate M_P in the recursive search procedure.

The recursive procedure exploring the space of fixed patterns is modified to search the space of flexible patterns. Let P be a flexible pattern, let $F(P)$ be the corresponding set of fixed patterns, and let M_P be the set of k-segments matching P. Initially, $P = e$, $F(P) = \{e\}$, and $M_P = B$. At each level of recursion we are considering a pattern P and all its simple extensions. A simple extension of P is P appended with (1) a (possibly flexible) wildcard region $x(i,j)$, and (2) an amino acid set a in V_A (V_a is the set of allowed pattern components), which makes a new pattern $P' = P\text{-}x(i,j)\text{-}a$. The set of fixed patterns corresponding to P' is $F(P') = \bigcup_{i \le k \le j, Q \in FJ(P)} \{Q\text{-}x(k)\text{-}a\}$, and the set of segments matching P' is $M_{P'} = \bigcup_{Q' \in F(P')} M_{Q'}$, which can be efficiently calculated using

$$M_{P'} = \bigcup_{i \le k \le j, Q \in F(P)} \{M_Q \cap b_{L(Q)+k+1,a}\}. \tag{3}$$

To be able to calculate $M_{P'}$ in this way, we store for each Q in $F(P)$ the length of Q, $L(Q)$, and the set of segments matching Q, M_Q. Note that if $F(P)$ contains m fixed patterns, then $F(P')$ will contain $(j - i + 1) \cdot m$ patterns. Having calculated $M_{P'}$, we check if it contains segments from at least N_{min} sequences. If it does, this means that the new pattern P' matches the minimum number of sequences, and it is analyzed recursively in the same way. If no simple extension of P matches at least N_{min} sequences, and if P has significance above the threshold, P is added to a list of patterns to be refined and reported.

Guaranteed refinement algorithm

The most significant conserved patterns discovered during the search of pattern space can be refined using either of two refinement algorithms. The guaranteed refinement algorithm is used by Pratt if the user switches on the full refinement option. It takes, as input, a conserved pattern P and carries out a new search of pattern space using the above recursive algorithm but allowing for more ambiguous pattern components. It uses P to direct the search and matches patterns only against segments in M_P. Because the refinement search is constrained to a subset of the segments, we can afford to allow for more ambiguous symbols. Using the menu, the user specifies how many ambiguous symbols to be included in the block data structure and how many of these to be allowed as pattern components during the initial search. During the refinement search, pattern components are allowed to take on the value of any symbol included in the block data structure. As a result, the refinement procedure can substitute any wildcard position in P with a symbol included in the block data structure and can also expand the pattern to the right (but not to the left). The algorithm is guaranteed to find all refined patterns in the class considered matching at least N_{min} sequences. It is quite expensive computationally, requiring more space for the block data structure, and one new search of pattern space for each of the most significant patterns identified during the initial search.

Fast and greedy refinement algorithm

The second refinement algorithm considers all wildcard positions in fixed wildcard regions and checks if the pattern obtained by substituting the wildcard with an ambiguous symbol matches at least N_{min} sequences. This is the procedure used by Pratt unless the user switches on the full refinement option. In order to refine a pattern P, we first append wildcard symbols to the end of P so that the refinement step may extend the pattern to the right. Extension to the left cannot be done in the same way because of the asymmetric nature of the block data structure. Each ambiguous symbol is one of, or a of subset of one of, a list L of allowable amino acid sets. L is given by the user as the file Pratt.sets, one line per amino acid set.

We consider one wildcard position w_i in P at a time starting with the leftmost wildcard w_i in a fixed wildcard region. For example, if P is D-x(2)-E, we first consider the wildcard position immediately after the D, matching the second position in segments matching P. For each amino acid symbol r, we count the number of times $n(r)$ that r in a segment matching P matches w_i. We then sort the amino acid symbols according to the number of matches, so that $n(r_1) \ge n(r_2) \ge \ldots n(r_{20})$. We want to find a set R of amino acid symbols, so that (1) R is a subset of a set in L (a set is a subset of itself), and (2) so that when w_i in P is substituted with the ambiguous symbol corresponding to R, the resulting pattern still matches at least N_{min} sequences. We greedily try finding such a set by including the amino acids most frequently matched with w_i. Initially we let $R = \{r_1\}$. Then, until at least N_{min} sequences are matched, or until further expansion is not possible, we expand R with the first r_i so that (1) r_i is not already included in R, and (2) there is a set u in L so that $(R \cup \{r_i\}) \subseteq u$. If a set R with the desired properties is found, we make a new pattern P' from P by substituting the wildcard w_i with R. Then both P and P' are recursively analyzed in the same way, only considering wildcard positions to the right of w_i. This is a greedy algorithm, and it is not guaranteed to find a refined pattern with the desired properties, even if one exists.

Significance of a flexible pattern

Others (e.g., Karlin & Altschul, 1990; Neuwald & Green, 1994) have defined the significance of a fixed pattern matching a number of sequences according to the probability of the sequences sharing the pattern by chance. This is not easily extended to flexible patterns. We define a significance measure in another way. What we need is a significance measure that we can use to rank the identified patterns and also to define a lower threshold of significance for patterns to be reported. Because all patterns identified are to match at least N_{min} sequences, we do not take into account the number of sequences matched when evaluating the significance of a pattern.

We define a measure of significance for a pattern that is the sum of the information contents of the pattern's components (Shannon, 1948). To deal with flexibility, we simply subtract a constant number of bits for each flexibility. This measure is used to rank the identified patterns. A pattern with high information content is considered significant.

More formally, for a pattern P as given in Equation 1, we define the information content of P to be

$$I(P) = \sum_{i=1}^{p} I(A_i) - c \int_{k=1}^{p-1} (j_k - i_k). \qquad (4)$$

The information content of a single position A_i is the decrease in uncertainty given that only symbols from A_i occur in this position:

$$I(A_i) = -\sum_{a \in S} [p_a \log_2(p_a)] + \sum_{a \in A_i} \left[\frac{p_a}{p_{A_i}} \log_2\left(\frac{p_a}{p_{A_i}}\right)\right], \qquad (5)$$

where p_a is the probability of amino acid a, S is the set of all amino acids, and $p_{A_i} = \sum_{a \in A_i} p_a$. The p_a values were calculated from the frequency of amino acid a in SWISS-PROT. For the constant c, we have used the value 0.5. Note that the information content is the same for all identity components. For test cases we have looked at, the significance measure defined above seems to give a reasonable ranking of identified patterns.

Time and space complexity

The block data structure and the data structures needed when doing the search of pattern space can require a lot of memory. All sets are implemented using bit-vectors. Given N sequences of average length L. The block data structure with set of symbols V_B will require on the order of $k|V_B|\lceil NL/8 \rceil$ bytes. For example, if $N = 100$, $L = 200$, $k = 50$, and $|V_B| = 20$, on the order of 2.5 Mbyte is needed for the block data structure.

The search itself also requires some memory. If M is the maximum number of components, F is the maximum product of flexibilities, G is the maximum length of a wildcard, and V_A is the set of symbols to be used in the search, on the order of $M \cdot F \cdot \max(G+1, |V_A|+1)\lceil NL/8 \rceil$ bytes are needed. For example, if $N = 100$, $L = 200$, $|V_A| = 20$, and $M = F = G = 10$, more than 5.25 Mbytes are needed for the search. Using reasonably powerful modern workstations, the memory requirement is not a problem for analyzing on the order of hundreds of protein sequences of typical length.

As an alternative to storing sets $b_{i,A}$ for ambiguous symbols A in the block data structure, we could choose to store only the sets $b_{i,a}$ for all single amino acid symbols and, when $b_{i,A}$ is needed, A being an ambiguous symbol, generate it from the sets for single amino acids $\bigcup_{a \in A} b_{i,a}$. During the search, this operation will be done many times, which can justify the extra time and memory needed for generating and storing each set $b_{i,A}$ separately. When limited system resources are available, and big sets of sequences are to be analyzed, this solution may be the preferred.

The time complexity is difficult to analyze. The time used will depend on the sequence lengths. The longer the sequences the more patterns will randomly match N_{min} sequences, and therefore we will prune the search at a lower level. The more closely related the sequences are, the less efficiently can the search space be pruned. The smaller N_{min} is relative to N, the longer time the search will take, and the more patterns will be found. The time complexity dependency on N, the number of sequences, is linear. Worst case behavior will arise with long, closely related sequences.

The time used also depends on the class of patterns that the user wants to search. The more flexibility allowed, the more ambiguous the symbols, and the bigger the sets the ambiguous symbols define, the longer time the search will take. The refinement step may also be time consuming, and the time used refining depends on how many patterns to be refined and which algorithm is chosen for the refinement step. The guaranteed refinement algorithm will be very time consuming if, again, the set of ambiguous symbols is big, and if the ambiguous symbols define big sets. The time used by the faster refinement algorithm depends on the set L of allowable ambiguous symbols, the number of sets in L, and the size of each set.

Acknowledgments

Inge Jonassen was paid by a grant from the Norwegian Research Council. We thank Toby Gibson and Rein Aasland for suggesting test cases and Amos Bairoch for advice on using PROSITE.

References

Aasland R, Gibson TJ, Stewart AF. 1995. The PHD finger: Implications for chromatin-mediated transcriptional regulation. *Trends Biochem Sci 20*:56–59.

Bairoch A, Boeckmann B. 1992. The SWISS-PROT protein sequence data bank. *Nucleic Acids Res 20*:2019–2022.

Bairoch A, Bucher P. 1994. PROSITE: Recent developments. *Nucleic Acids Res 22*:3583–3589.

Devereux J, Haeberli P, Smithies O. 1984. A comprehensive set of sequence analysis programs for the VAX. *Nucleic Acids Res 12*:387–395.

Dodd IB, Egan JB. 1990. Improved detection of helix-turn-helix DNA-binding motifs in protein sequences. *Nucleic Acids Res 18*:5019–5026.

Etzold T, Argos P. 1993. SRS—An indexing and retrieval tool for flat file data libraries. *Comput Appl Biosci 9*:49–57.

Fuchs R. 1994. Predicting protein function: A versatile tool for the Apple Macintosh. *Comput Appl Biosci 10*:171–178.

Henikoff S, Henikoff JG. 1991. Automatic assembly of protein blocks for database searching. *Nucleic Acids Res 19*:6565–6572.

Karlin S, Altschul SF. 1990. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proc Natl Acad Sci USA 87*:2264–2268.

Lawrence CE, Altschul SF, Wootton JC, Boguski MS, Neuwald AF, Liu JS. 1993. Detecting subtle sequence signals: A Gibbs sampling strategy for multiple alignment. *Science 262*:208–214.

Musacchio A, Gibson T, Lehto VP, Saraste M. 1992. SH3—An abundant protein domain in search of a function. *FEBS Lett 307*:55–61.

Neuwald AF, Green P. 1994. Detecting patterns in protein sequences. *J Mol Biol 239*:698–712.

Ogiwara A, Uchiyama I, Yasuhiko S, Kanehisa M. 1992. Construction of a dictionary of sequence motifs that characterize groups of related proteins. *Protein Eng 5*:479–488.

Pearson WR, Lipman DJ. 1988. Improved tools for biological sequence comparison. *Proc Natl Acad Sci USA 82*:3073–3077.

Roytberg MA. 1992. A search for common patterns in many sequences. *Comput Appl Biosci 8*:57–64.

Saqi MAS, Sternberg MJE. 1994. Identification of sequence motifs from a set of proteins with related function. *Protein Eng 7*:165–171.

Shannon CE. 1948. A mathematical theory of communication. *Bell System Tech J 27*:379–423, 623–656.

Smith HO, Annau TM, Chandrasegaran S. 1990. Finding sequence motifs in groups of functionally related proteins. *Proc Natl Acad Sci USA 87*:826–830.

Smith RF, Smith TF. 1990. Automatic generation of primary sequence patterns from sets of related protein sequences. *Proc Natl Acad Sci USA 87*:118–122.

Taylor WR. 1986. Identification of protein sequence homology by consensus template alignment. *J Mol Biol 188*:233–258.

Thompson JD, Higgins DG, Gibson TJ. 1994a. Clustal W: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res 22*:4673–4680.

Thompson JD, Higgins DG, Gibson TJ. 1994b. Improved sensitivity of profile searches through the use of sequence weights, gap excision and the BLOSUM62 matrix. *Comput Appl Biosci 10*:19–29.

Wang JTL, Marr TG, Shasha D, Shapiro BA, Chirn GW. 1994. Discovering active motifs in sets of related protein sequences and using them for classification. *Nucleic Acids Res 22*:2769–2775.