

User Guide to AMPS (Alignment of Multiple Protein Sequences)

Geoffrey J. Barton

School of Life Sciences
University of Dundee
Dundee DD1 5EH
UK

Tel: 01382 345860
e-mail: geoff@compbio.dundee.ac.uk

Contents

1	Update History	1
2	Introduction to Version D 2.0	1
3	Related Programs	1
4	Background	2
5	Introduction	2
5.1	Needleman and Wunsch Algorithm	2
5.2	Predicting the likely quality of alignment	2
5.3	Multiple alignment strategy	3
6	Description of the AMPS package	3
7	MULTALIGN - Instructions for Use under Unix	4
7.1	Pairwise Alignment	4
7.2	Time considerations	6
8	Multiple Alignment Just using sequence information	6
8.1	Single order alignment	6
8.2	Tree based alignment	7
8.2.1	Comments	8
8.3	Multiple alignment - building on an existing alignment	8
8.4	Examples	8
8.4.1	Aligning without additional information	9
8.4.2	Secondary structure dependent gap_penalties	9
8.4.3	Defining explicit weights to align ONE extra sequence	10
8.4.4	Lookup tables	10
9	User Guide to the program ORDER	11
10	Flexible Pattern Matching and Database Scanning	12
11	Introduction	12
11.1	Program limits	13
12	Defining a Pattern	13
12.1	Defining the scoring scheme	13
12.2	Alternative scoring systems	14

13	Sorting the results of a scan using PROGRAM SORTER	15
13.1	To sort the output file and store the entire sorted list	15
13.2	To sort the output and store only the top N scoring results	15
13.3	Extracting the IDentifier codes for the top scoring sequences	16
13.4	Specifying a cutoff score	17
13.5	Generating Prolog Clauses	18
13.6	Generating a Histogram of the score distribution	18
13.7	Getting Help	18
14	Extracting Sequences from a PIR Database Using PROGRAM SELECT	19
15	Generating Alignments of the Pattern With Interesting Sequences	20
16	Generating multiple pattern alignments with each sequence	21
17	APPENDICES	22
17.1	Summary of Commands for MULTALIGN	22
17.2	MULTALIGN file formats	25
17.2.1	seq_file	25
17.2.2	matrix_file	26
17.2.3	order_file	26
17.2.4	tree_file	26
17.2.5	block_file	27
17.2.6	Save_lookup format	28
17.2.7	Format for Input to program PATT	29
17.3	VMS installation guide	29
17.4	Unix Installation Guide	29
17.4.1	Unix usage	30
18	References	31
19	Known Bugs	31

1 Update History

Preliminary User Guide: February 1988
First Update: June 1990
Second Update: Sept 1991
Third Update: August 1993 some new commands - see APPENDIX
Minor Changes to docs and some bug fixes: January 1994
Version D 2.0

Minor Changes and some bug fixes - should now work under OSF/1: March-June 1994
Version D 2.1

Very minor changes to make Version D2.2.

November 2003: Minor changes to fix linux distribution

Copyright (c) 1988,1991,1994,1997 G. J. Barton.

2 Introduction to Version D 2.0

This is the first major update to the AMPS package since its initial distribution in 1988 (Version D 1.0). No major changes have been made to the basic multiple alignment functions, except the addition of a dendrogram drawing facility in programme ORDER and the provision of identifier codes of up to 20 characters (including the '>' symbol) (program PATT is limited to 10 characters). Existing users should not experience problems - I hope... The major addition is the availability of the flexible pattern matching and database scanning features of the programmes described in Barton, 1990 and Barton & Sternberg 1990. Sections have been added to this guide to explain how these features may be used.

All development of this package is now carried out on a Sun Microsystems SPARCstation under the Unix operating system. The most stable version is therefore the SPARCstation one. With this release is the first option for Silicon Graphics Personal Iris and compatibles. The core multiple alignment programs will compile under VAX/VMS - see the instructions in BUILD.COM.

If you have problems with the programs and your local Guru cannot solve them, then e-mail me on gjb@bioch.ox.ac.uk stating exactly what the problem is, together with example files.

3 Related Programs

ALSCRIPT: Takes the output of AMPS (block file format) and allows prettyfication of the alignment: boxing, shading etc.

AMAS: Performs a hierarchical analysis of a multiple alignment.

STAMP: Does multiple alignment of protein three dimensional structures.

See our [www](http://barton.ebi.ac.uk) pages for details.

<http://barton.ebi.ac.uk>

4 Background

AMPS is a suite of programs designed for the multiple alignment of protein sequences and flexible pattern matching. Initial development of the algorithms programmed in this suite was performed whilst I was in the Department of Crystallography at Birkbeck College, University of London. The programs as distributed are the result of refinements and improvements on the the

original Birkbeck programs during my time at the Imperial Cancer Research Fund Laboratories in London and further development at the Laboratory of Molecular Biophysics, Oxford.

These programs are available for academic, teaching and non-commercial purposes. **The programs are supplied “as is” with no warranty whatsoever as to functioning, performance or effect on hardware of other software, express or implied. The author disclaims any implied warranties of merchantability of fitness for any particular purpose.**

For commercial use, a commercial licence is required. Please contact the author at the above address for details.

5 Introduction

5.1 Needleman and Wunsch Algorithm

One of the most widely applied sequence alignment algorithms is the method of Needleman and Wunsch (1970). When supplied with two protein sequences and a matrix of pair-scores (eg. an identity matrix, or Dayhoff's mutation data matrix) the algorithm identifies the best alignment(s) between the two sequences and a score for the alignment. Whilst this approach may be used with virtually any pair of sequences, extending the method to more than 3 sequences is impractical. This computer package implements a heuristic approach to multiple sequence alignment that has been shown to be effective for many protein families. (See Barton and Sternberg 1987b for a description of the algorithm and assesment of accuracy, See Zvelebil et al, 1987, for an application in secondary structure prediction)

5.2 Predicting the likely quality of alignment

Barton and Sternberg (1987a,b) considered families of proteins for which the three dimensional structure of at least two members had been solved by x-ray crystallography. The alignment obtained by comparing the tertiary structures was used as a 'standard' against which to test the Needleman and Wunsch (1970) sequence alignment algorithm.

In summary; if the significance score obtained for the pairwise comparison of two sequences is greater than 5-6 S.D. then there is a high probability that the sequences will be correctly aligned over most of the regions that belong to core secondary structures. Whilst this observation does not mean that alignments of sequence pairs scoring less than 5.0 S.D. are meaningless, alignments in this region should be treated with caution and additional evidence in support of the particular alignment should be sought. Conversely, if two sequences score greater than 15.0 S.D. then the alignment is likely to be correct over most of its length.

5.3 Multiple alignment strategy

1. All pairwise comparisons between the sequences are performed. The information obtained may be used to construct a dendrogram that visualises the groups of similar residues.

2. Given a group of similar sequences, the order in which they should be aligned is determined (ie most similar pair through to least similar).
3. The multiple alignment algorithm is then applied to the sequences. Firstly the most similar pair are aligned, then the next most similar sequence is aligned to the alignment of the most similar pair. Then the next most similar sequence is aligned and so on..... An optional process is to re-align each sequence with the completed alignment in order to refine the alignment.

6 Description of the AMPS package

Version D2.0 consists of the following programs.

MULTALIGN (AMULT/FSCAN) performs a number of functions. These include pairwise sequence alignment and assessment of statistical significance, multiple alignment by the methods of Barton and Sternberg (1987b) and a method similar to that of Feng and Doolittle (1987), and additional functions that allow the inclusion of variable gap-penalties and specific weighting schemes. Each aspect of this program and its use is described below.

ORDER uses the output from a MULTALIGN pairwise run to perform cluster analysis (tree generation), ordering of the sequences by similarity and general format changes.

DAYB takes a sequence in seq_file format and creates a file in block_file format. Just type DAYB and follow the instructions.

SORTER takes the output of a database scan and sorts the results into descending-score order. SORTER also allows construction of histograms and extraction of the identifying codes for the top scoring proteins.

SELECT takes a list of protein identifier codes and a sequence database and extracts the sequences from the database that correspond to the codes. Unlike some programs (e.g. PSQ) the sequences are output in the same order as the codes are presented. SELECT is normally used in conjunction with SORTER.

PATT Processes the output of a flexible pattern analysis to produce a more readable output.

7 MULTALIGN - Instructions for Use under Unix

The program is run from a command file that contains a series of keyword commands that tell the program which files to use for what, how to format the output and above all, which type of alignment to perform.

Keyword commands are all followed by an equals sign '=' which may in turn be followed by a string of arguments in free format. The precise formats for each command are described in APPENDIX 1. Keyword commands may be in upper or lower case or mixed case. See APPENDIX 2 for a full description of input/output file formats.

NOTE: All filenames must be fully qualified. The examples shown in this manual give filenames as e.g. "ampsdir:md.mat", you need to change this to indicate the exact path for the file, e.g. "/home/geoff/gjb/md/md.mat", or wherever the file is on your computer system.

Once you have created the command file, for example “test.com”, then run multalign by typing:

```
multalign < test.com
```

All results and error messages will be written to the output file, so look here to see if all has worked well.

7.1 Pairwise Alignment

An example command file for this mode is globin_pairs.com. This is shown below:

Commands	Explanation number
-----	-----
output_file=globin_pairs.out	1.
mode=pairwise	2.
matrix_file=amspdirt:md.mat	3.
pairwise_random=100,100,1	4.
gap_penalty=8.0	5.
constant = 8	6.
seq_file=globin.seq	7.

1. Command to amult. set the file for output of results to 'globin_pairs.out' This command must ALWAYS be the first. Note: The output file should NOT be set to the log file name.
2. Specify that pairwise mode is to be used. If this command is not included, the program defaults to mode=multiple.
3. Define the matrix file to be used in the comparisons. This file is the Dayhoff mutation data matrix or similar file containing pairscore values for each amino acid pair.
4. Specify that 100 randomizations of each sequence pair are to be performed in order to estimate the statistical significance of the alignment obtained.
5. define a gap penalty of 8.0
6. define a constant of 8 to be added to the matrix.
7. specify the file containing the sequences to be aligned pairwise.

This command file causes all pairwise comparisons to be performed on the sequences in the file 'globin.seq'. In other words, for the 7 sequences sequence 1 is aligned with 2, 1 with 3 and so on. For N sequences there are $N*(N-1)/2$ comparisons performed.

For each sequence pair (eg. 1 and 2), a full Needleman and Wunch sequence comparison is performed. Then the sequences are shuffled and recompared (in this case 100 times) in order to find the expected distribution of scores that would be obtained if the sequences were unrelated but have the same length and composition as 1 and 2. Various statistics on the comparisons are then output to the specified output_file (globin_pairs.out).

The output file from this sequence alignment run contains the following information:

A banner describing the program name, source and limitations of use. Information on the maximum length and number of sequences allowed. Information on the commands specified to the program, files etc. A list of the sequences defined in the sequence file. Finally, a set of numerical results of the run in a series of fields as follows.

Field	Description
-----	-----
1 and 2	number describing the sequences aligned on this row.
3 and 4	The lengths of the sequences described in 1 and 2.
5	The match score obtained for the comparison of the two sequences.
6	The number of internal gaps in the alignment (overhangs at the ends are not counted)
7	The number of positions at which two amino acids are aligned.
8	The number of positions at which identical amino acids are aligned
9	Percentage identity (8/7)
10	Normalised Alignment score - The match score divided by the number of aligned positions * 100. (5/7)*100.
11	Alternative Normalised Alignment score - The match score minus the number of gaps times the gap penalty all divided into the number of aligned positions. (7/((5-6)*gap_penalty))*100.
12	Number of randomizations performed
13	mean score for the randomizations
14	standard deviation of the random scores
15	Significance score for the alignment. Given by the mean random score minus the match score all all divided by the standard deviation. (13-5)/14.
16	Comparison number.

This output file is in the correct format for input by the program ORDER. Clearly, however the actual alignments have not been output. If the alignments are required you must include the command line(s)

print_horizontal= (for horizontal format)

print_vertical= (for vertical format)

Note that a file containing alignments cannot be read directly by ORDER. (the alignments would first have to be deleted using a text editor).

7.2 Time considerations

Performing randomizations for all pairs of sequences can be very time consuming, particularly if a large number of sequences are to be compared prior to multiple alignment. A good guide to the relative similarity between the sequences is given by the normalised alignment scores. These do not require randomization in order to calculate and so considerably speed the process. Simply delete the line 'pairwise_random' from the command file to avoid performing randomizations.

8 Multiple Alignment Just using sequence information

8.1 Single order alignment

This is the method described in Barton and Sternberg (1987b). The sequences are aligned progressively in one order. If the sequences are all clearly similar to each other - say > 50% identity, then the actual order will make very little difference to the final alignment. In general however, it is better to establish the order by first performing all pairwise comparisons for the sequences as described in the previous section, then using program ORDER to define an order (described below) before proceeding to multiple alignment.

The ORDER program generates an order_file. This specifies the order in which the sequences present in the seq_file are to be aligned. A typical command file for multiple alignment is given in globin_mult.com and shown below.

Commands	Explanation number
-----	-----
output_file=globin_mult.out	1.
mode=multiple	2.
matrix=file=amspdir:md.mat	3.
gap_penalty=8.0	4.
constant = 8	5.
consplot=mz	6.
print_vertical=	7.
seq_file=globin.seq	8.
order_file=globin.ord	9.

1. Command to MULTALIGN. set the file for output of results to 'globin_mult.out' This command must ALWAYS be the first. Note: The output file should NOT be set to the log file name.
2. Specify that multiple mode is to be used. If this command is not included, the program defaults to mode=multiple.
3. Define the matrix file to be used in the comparisons. This file is the Dayhoff mutation data matrix or similar file containing pairscore values for each amino acid pair.
4. define a gap penalty of 8.0
5. define a constant of 8 to be added to the matrix defined in 5.
6. optional request to perform a conservation analysis using the algorithm of Zvelebil et al (1987) on the resulting alignment (only works if the print_vertical command is also present).
7. specify vertical format output
8. define the file containing sequences to be aligned as 'globin.seq'

9. define the order file obtained from running the program ORDER on the results of a pairwise sequence comparison run. If this command is absent then the program aligns the sequences in the order that they are present in the seq_file. Note. the order_file command must always appear after the seq_file command.

Optionally a process of iteration can be performed. Once all the sequences have been added to the alignment, the first sequence can be realigned with the ALIGNMENT of the sequences 2-N. Then the second sequence is aligned with 1,3-N and so on. To specify that iteration should be performed include the command 'iterations=int' where int is an integer greater than 0. In general iteration values of 3 can refine the alignments slightly and are of greatest use if there are relatively few sequences to be aligned (say ≤ 10).

The result of a run using this command file is illustrated in the file globin_mult.out.

8.2 Tree based alignment

This option allows the alignment to be performed in a manner similar to that described by Feng and Doolittle (1987). Rather than using a single order, the sequences are aligned according to a defined evolutionary tree. In my implementation, the tree can be obtained by running program ORDER, this produces two files, an order_file and a tree_file. The order file specifies an order that preserves the clustering of the sequences by similarity. The tree_file gives specific instructions to the program as to which comparisons to make and in which order.

For example, given four sequences A,B,C,D. the tree_file may specify that A and B should be aligned first, then C with D and finally the AB alignment is aligned with the CD alignment using average scores at each amino acid position.

Example command file. globin_mult_tree.com

Commands	Explanation number
-----	-----
output_file=globin_mult_tree.out	1.
mode=multiple	2.
matrix=file=ampsdire:md.mat	3.
gap_penalty=8.0	4.
constant = 8	5.
consplot=mz	6.
print_vertical=	7.
seq_file=globin.seq	8.
order_file=globin_pairs_tree.ord	9.
tree_file=globin_pairs.tree	10.

This is the same as globin_mult.com with the exception of two items.

order_file defines the cluster order. This is usually different to the 'simple' order defined in globin_pairs.ord.

tree_file defines the tree_file generated by program ORDER, or manually typed in.

The example output file from this run is globin_mult_tree.out.

8.2.1 Comments

It is currently not possible to say which approach to multiple alignment gives the best results. The two methods give approximately equivalent alignments when the sequences are quite similar (as judged by significance scores from pairwise comparisons). If the sequences consisted of two distinct families then the tree based alignment will give better alignments **WITHIN** the families but roughly equivalent alignments **BETWEEN** the families. In the current implementation the single order method will run faster so is probably the general method of choice.

8.3 Multiple alignment - building on an existing alignment

Frequently more is known about some of the sequences to be aligned than others. For example, if two or more proteins in the family of interest have had their three-dimensional structures solved by x-ray crystallography, then they may be aligned by inspection of the three dimensional models with much greater confidence than by using sequence information alone. MULTALIGN therefore allows one or more extra sequences to be added to a preexisting alignment.

At a lesser level, certain residues may be expected to align on functional grounds - for example binding studies may suggest that a particular Histidine residue is involved in the catalytic mechanism of two or more of the proteins to be aligned. In this situation it is important to be able to include this information in the alignment. This is particularly important if the overall similarity between the sequences is low since it is likely to lead to a biologically more reasonable alignment.

Certain regions of the sequence(s) may be expected to experience fewer gap events than others. For example the core secondary structural elements (beta strands and alpha helices) generally do not tolerate large insertions or deletions. This feature can be encoded when using MULTALIGN by specifying a variable gap-penalty. (See Barton and Sternberg 1987a for a discussion of the benefits of secondary structure dependent gap-penalties).

8.4 Examples

In order to align sequences to an existing alignment (in this context, an existing alignment may consist of a minimum of only one sequence) the alignment must be in 'block_file' format. Essentially this is the same as the file produced by the print_vertical command and may even contain conservation information which is ignored when the file is read in. The minimum requirements for a block_file are given in appendix 2.

8.4.1 Aligning without additional information

Example: aligning myoglobin sequences with an existing alignment derived from three dimensional structure superposition.

Command file: bash_mult.com

```
output_file=bash_mult.out
matrix=file=ampsdire:md.mat
```

```
gap_penalty=8.0
constant=8
print_vertical=
block_file=bash.bloc,1
seq_file=myoglobins.seq
```

These commands will mostly be familiar by now!! The only new addition is the `block_file` command. This defines a block file to be read, the filename must be followed by an integer that identifies the iteration number specified in the `block_file`. In this example, the iteration number is set to 1.

This `block_file` contains the alignment described by Bashford et al (1987) for 7 globin sequences aligned on structural grounds. The command file aligns all 60+ myoglobin sequences with this structure based alignment. See file `bash_mult.out` for the result.

8.4.2 Secondary structure dependent gap_penalties

Example: Aligning with an alignment derived from three-dimensional structure superposition but including secondary structure dependent gap_penalties.

Secondary structure dependent penalties are included by inserting range delimiting characters into the `block_file`. This is illustrated by the `block_file` 'bash_vg.bloc'. A start of range is indicated by the ">" character following the alignment, an end of range by the '#' character.

A typical command file that takes advantage of this information is 'bash_mult_vg.com'. In this example, the delimited ranges correspond to the conserved alpha helices in the proteins, the gap penalty is therefore INCREASED within the ranges. (An alternative strategy would be to define the ranges as the non-helix bits and then decrease the gap penalty)

Command file: `bash_mult_vg.com`

```
output_file=bash_mult_vg.out
matrix=file=ampsdire:md.mat
gap_penalty=8.0
constant=8
gap_factor=100
print_vertical=
block_file=bash_vg.bloc,1
seq_file=myoglobins.seq
```

The additional command is `gap_factor`. This defines a value by which the standard `gap_penalty` is multiplied WITHIN the defined ranges. Clearly, a number greater than one decreases the likelihood of a gap within the range, a number less than one increases the likelihood of a gap.

In this example there is little advantage in this approach since the alignment is very good without the use of secondary structural dependent gaps. When aligning less similar sequences, the benefits become more apparent.

8.4.3 Defining explicit weights to align ONE extra sequence

One further refinement that may be incorporated when aligning to a `block_file` is to include specific residue weights. Before explaining how this is achieved it is necessary to understand some of the program's entrails.

8.4.4 Lookup tables

When the program reads a `block_file` it generates a lookup table that consists of an array of dimensions (length of the alignment)*(the number of amino acid types). Each element of the lookup table represents the weight assigned to aligning a particular amino acid at that position. MULTALIGN allows you to print out this Table and modify it to increase/decrease the weights for particular amino acids at particular alignment positions. The Table may then be read in by the program and used to subsequently align another amino acid sequence.

Note. This option is meant for application when using MULTALIGN to scan sequence databases with flexible patterns, hence it is only possible to align ONE further sequence with the `block_file`. (The subject of flexible patterns and database scanning is the basis of a paper shortly to be submitted. The necessary commands to make use of this feature will be added as an appendix once the paper is in print.)

The command `'save_lookup=bash_day.look'` causes the lookup table to be written out to the file `bash_day.look`.

The Table may be modified, then read in by the command `'read_lookup=bash_day.look'`

An example command file to generate and save a lookup table is shown below (file `'bash.look.com'`)

```
output_file=bash.out
matrix=file=amspdir:md.mat
gap_penalty=8.0
constant=8
gap_factor=100
save_lookup=bash_day.look
block_file=bash_vg.bloc,1
non_alignment=
seq_file=globins.seq
```

The `seq_file` command must be present, and at least one sequence present in the `seq_file`.

The Lookup_table (`bash_day.look`) might be edited to increase specific residue weights before using it to align another sequence to the block.

An example command file is `'bash_rlook.com'`

```
output_file=bash2.out
gap_penalty=8.0
gap_factor=100
read_lookup=bash_day.look
block_file=bash_vg.bloc,1
seq_file=myo.seq
```

This will cause the lookup table to be read and used to score the alignment of the sequence in file `myo.seq` with the `block_file`. If more than one sequence is defined in the `seq_file` then a garbage alignment will be generated for sequences 2-N.

9 User Guide to the program ORDER

Order is a program designed to be run interactively. Its functions are to process the output of a MULTALIGN in pairwise mode and produce the following files.

1. Order_file.
2. tree_file/order_file. (cluster analysis)
3. graphical output - a simple dendrogram is produced.

Type order.

1. The program will then ask you for the name of a file produced by MULTALIGN in pairwise mode (but not including any alignments).
2. Does the input file have timings? If you specified pairwise_timer= when running MULTALIGN then answer Yes, otherwise N (Default=N).
3. Enter filename for comparisons to exclude - press <return>.
4. Options: Five possible results of pairwise comparison may be used to perform the ordering or cluster analysis.

```

NGAPS(1)    (Use the number of gaps - totally useless)
PIDENT(2)   Use percentage identities - possibly useful.
NAS(3)      Normalised alignment score (see description of Multalign pairwise
mode for explanation)
NASAL(4)    As for NAS
RSCORE(5)   Use S.D. scores calculated from randomizations.

```

Answer with an integer 1-5 (Usually 3 or 5) 5 is best if you've calculated it.

5. Produce an order file? [Y] Answering Yes to this option prompts for a filename, then exits the program. The order is defined in a manner suitable for the single order multiple alignment algorithm.
6. Answering No to 5. gives the question Perform cluster analysis ? [Y] Answer Yes then you get
 - (a) Save full cluster details? (ie do we want to generate a tree_file and a compatible order_file for subsequent use by MULTALIGN to perform tree based multiple alignment?? The default is YES. If we answer Yes, then we are next prompted for an order filename and tree filename to be written.

(This is followed by a request for plotting details - note, there is a bug in the programme, you MUST supply a title and axis title, but these won't appear on the output - The page width requested, refers to the number of plotting characters typically 80 or 132).
7. Answering No to 6. gives the question 'Give ADDTREE filename' This allows the pairwise scores to be written out in a form that can be read by the cluster analysis program ADDTREE (See Corter(1982))

10 Flexible Pattern Matching and Database Scanning

11 Introduction

This section describes how to use AMPS to perform flexible pattern matching by the method of Barton and Sternberg (1990). This approach can help identify weaker similarities between proteins that would otherwise be missed by conventional sequence comparison methods. AMPS may also be used to scan the database with a multiple sequence alignment using the Needleman-Wunsch algorithm - normally this is not as effective as deriving a pattern and scanning with that.

PLEASE NOTE: It is important that you are familiar with the AMPS alignment features described above, before attempting flexible pattern matching.

AMPS allows the following operations to be performed:

1. Definition of a pattern representative of a particular protein fold including the explicit description of allowed flexibility in gap length between defined regions.
2. A variety of scoring systems for each element of the pattern. eg. based on frequency weights, Dayhoff's matrix, conservation or fully user defined weights.
3. Scanning of the pattern against a database of protein sequences, subsequent rank ordering and display of the results of the scan.
4. Detailed analysis of a single sequence for the presence of multiple occurrences of the pattern. Calculation of the significance of the best matching pattern by reference to randomized sequences.

These points will be illustrated by reference to examples.

A typical pattern analysis might follow the following steps:

1. Define a pattern and choose a scoring scheme.
2. Scan the pattern against the database (PROGRAM MULTALIGN).
3. Sort the results (PROGRAM SORTER).
4. Get ID's of interesting proteins (PROGRAM SORTER).
5. Extract sequences of interesting proteins from the database (PROGRAM SELECT).
6. Align pattern to the proteins - including alternative alignments (PROGRAM MULTALIGN).
7. Produce compressed output for inspection (PROGRAM PATT)

Not every stage need be performed. For example, if we already know the subset of the protein database that is interesting, then steps 1-4 can be avoided.

11.1 Program limits

The normal MULTALIGN program is limited to up to 500 sequences not exceeding 1200 amino acids length. As a consequence, the maximum sequence length considered in a database scan is also 1200. Sequences of greater than 1200 amino acids in length will be IGNORED. The program reports how many sequences were actually compared in the scan at the end of the output file.

IMPORTANT: THE AMULT PROGRAM MUST HAVE BEEN RECOMPILED FOR LONG SEQUENCES TO USE THIS OPTION. THIS IS NOT THE DEFAULT INSTALLATION. PLEASE SEE THE NOTES IN THE FILE CALLED BUILD.

12 Defining a Pattern

The flexible pattern is defined around the block_file format described in APPENDIX II. The only additions necessary are if flexible gaps are to be included, when the following syntax is used.

```
>test  this is a test flexible pattern
>test2  this is also a flexible pattern
*iteration 1
ag                each line in this file
v1 <23:28         defines a pattern "ELEMENT"
pg
ww <1:3
kr
.
.    further pattern elements here
.

etc

*
```

This file means: score against ag at position 1, v1 at position 2 then allow between 23 and 28 gaps (inclusive) then score against pg, then ww, then allow a gap of 1 to 3 residues, then score kr etc etc.

12.1 Defining the scoring scheme

When the pattern is read by the program, a lookup table is defined that specifies the score for aligning each possible amino acid with each element of the pattern. There are several possible ways of deriving this table, each of which is defined by commands to the program.

The most flexible scoring system is to define a LOOKUP table with the READ_LOOKUP command (see above). A simpler approach is to calculate a lookup table based upon the observed amino acids in the pattern.

Example command file (bash_ge_4_scan1.com) to scan with the pattern defined in the file bash_ge_4.bloc.

```
output_file=bash_ge_4_scan1.out    1. output file
mode = scan                        2. set the mode to scan
block_file/pattern=bash_ge_4.bloc,1 3. define the pattern file
matrix_file=md.mat                 4. define dayhoff matrix scoring
database=protein.seq              5. define the database to scan
```


This scan takes 977 seconds (16 minutes) on a Sun SPARCstation 1 (6721 sequences scanned out of a total of 6858). The result of this scan is shown in file `bash_ge_4_scan1.out`.

The new commands are:

MODE=SCAN This tells the program to compare a pattern or alignment to the DATABASE rather than perform multiple or pairwise sequence alignment.

/PATTERN in the **BLOCK_FILE** command. This tells the program to use the flexible pattern matching algorithm rather than a conventional Needleman and Wunsch.

DATABASE This defines the sequence database to be scanned (must be in PIR format as defined in APPENDIX II).

By specifying the `md.mat` MATRIX file we are defining DAYHOFF scoring for the scan with simple averages used to define the lookup table.

12.2 Alternative scoring systems

IDENTITY change the `matrix_file` command to: `MATRIX_FILE=ampsdire:UP.MAT` the presence of an amino acid in an element will indicate a score of 1 with the same amino acid, 0 with all others.

FREQUENCY delete the `MATRIX_FILE` command and insert: `FREQSCORE=` The frequency of occurrence of each amino acid will be used to score.

CONSERVATION delete `FREQSCORE` and replace with `CONSCORE`.

WEIGHT (Dodd and Egan's approach). define `FREQFILE=fname` where `fname` is name of a file containing amino acid abundancies.

EXPLICIT Write a LOOKUP file, then read this into the program with the `READ_LOOKUP=fname` command.

13 Sorting the results of a scan using PROGRAM SORTER

The results of a scan are written to the `OUTPUT_FILE`. However, the results are in database order, so need to be sorted with the highest scoring (most interesting) sequences first. The program **SORTER** performs this function.

EXAMPLES: Type `sorter` to start the program. Some messages appear - to get help, enter `/help`. Some times for operations will also be printed to the screen (not shown in the examples below).

13.1 To sort the output file and store the entire sorted list

```
ENTER FILE TO PROCESS: bash_ge_4_scan1.out

ENTER FILE FOR SORTED OUTPUT [.sorted]: <return>
1=MATCH, 2=NAS, 3=RMEAN, 4=RSD, 5=SCORE

ENTER CHOICE :1          (Always answer 1)

READING DATA FILE
  6721 DATASETS READ IN
PERFORMING SORT
DATA SORTED
WRITING SORTED FILE
SORTED FILE WRITTEN

Generate prolog clauses? [N]: <return>
```

This Example created the file `bash_ge_4_scan1.sorted`. The file is identical in format to `bash_ge_4_scan1.out`, but has the comparisons sorted in decreasing order of similarity to the pattern.

13.2 To sort the output and store only the top N scoring results

(For the sake of brevity in the following example, we only store the top 20)

```
ENTER FILE TO PROCESS: bash_ge_4_scan1.out/S=20
I WILL REPORT THE TOP  20 SCORES

ENTER FILE FOR SORTED OUTPUT [.sorted]: bash_ge_4_scan1.top20
1=MATCH, 2=NAS, 3=RMEAN, 4=RSD, 5=SCORE

ENTER CHOICE :1          (Always answer 1)

READING DATA FILE
  6721 DATASETS READ IN
CALCULATING SCORE DISTRIBUTION
USING ITEM 1
  STATISTICS COMPLETE
NUMBER OF POINTS  6721
MEAN                40.8484
SD                  26.2812
SKEW                2.19682
KURTOSIS            6.44562
CALCULATING SIGSCORES
PERFORMING SORT
DATA SORTED
WRITING SORTED FILE
SORTED FILE WRITTEN

Generate prolog clauses? [N]: <return>
```

The `/S=` option calculates various statistics on the scores obtained, and expresses each reported score in SD units from the mean. For example part of the file `bash_ge_4_scan1.top20`:

```
>P1;HZPG          : Hemoglobin zeta chain - Pig
  46  141    153.14    3.33    0.00    0.00    4.27
   A   B      C      D      E      F      G
```

The numbers shown after each protein identifier and title line refer to the following:

A Number of elements in the pattern (46).
B The length of the sequence being scanned (141).
C The score for the best alignment of the pattern
and the sequence (153.14).
D the score divided by the number of pattern elements (3.33).
E The mean (if randomisations are performed).
F The Standard Deviation (if randomisations are performed).
G The Distance the score (C) is from the mean of the distribution in
standard deviation units (i.e. $(153.14 - 40.84)/26.28$).

If randomisations are performed using the DATABASE,N option where N= the number of randomisations to be performed, then E,F and G refer to the mean and s.d. of the comparisons to randomised sequences.

Although S.D. values are shown, these are not normally very useful. It is generally better to look at the distribution of scores visually using the /hist option described below.

13.3 Extracting the IDentifier codes for the top scoring sequences

The /ID option allows the identifier codes to be extracted from a file. The resulting file contains SCORE ID pairs with the score expressed in an integer form suitable for input to the program SELECT.

For example:

```
ENTER FILE TO PROCESS: bash_ge_4_scan1.top20/ID
ENTER NUMBER OF IDS TO WRITE OUT: 20
ENTER FILE FOR ID'S: bash_ge_4_scan1.top20id
1=MATCH, 2=NAS, 3=RMEAN, 4=RSD, 5=SCORE

ENTER CHOICE :1
USING:      1

READING DATA FILE
  20 DATASETS READ IN
PERFORMING SORT
DATA SORTED
WRITING SORTED FILE
SORTED FILE WRITTEN

Generate prolog clauses? [N]: <return>
```

Giving the following file: bash_ge_4_scan1.top20id

```
15314 HZPG
15257 HZCZ
15257 HZHU
15142 HACHPE
```

```

14957 HADKP
14900 HEMSY2
14800 HBRB3
14728 HBPY
14714 HBF3T
14700 HBMSH0
14686 HBTG
14686 HBTP
14671 HGMQJ
14671 HGMQP
14671 HGMQR
14671 HGBAY
14671 HGHUA
14671 HGMKS
14657 HBOR
14657 HEGT1

```

13.4 Specifying a cutoff score

All results above some cutoff score may be selected, rather than sorting the entire file:

```

ENTER FILE TO PROCESS: bash_ge_4_scan1.out/cutoff

ENTER CUTOFF [real]: 101.0      (for example)
I will report sequences scoring .gt. :    101.00

ENTER FILE FOR SORTED OUTPUT [.sorted]: bash_ge_4_scan1.gt101
1=MATCH, 2=NAS, 3=RMEAN, 4=RSD, 5=SCORE

ENTER CHOICE :1
USING:      1

READING DATA FILE
    327 DATASETS READ IN
PERFORMING SORT
DATA SORTED
WRITING SORTED FILE
SORTED FILE WRITTEN

Generate prolog clauses? [N]: <return>

```

13.5 Generating Prolog Clauses

The output of any scan may be converted to Prolog clauses by answering Y to the Generate prolog clauses? question. An example of the format is shown in the file: bash_ge_4_scan1.pl. This option is probably of limited interest!

13.6 Generating a Histogram of the score distribution

The score distribution may be examined using the /HIST option as follows:

```

ENTER FILE TO PROCESS: bash_ge_4_scan1.out/HIST
1=MATCH, 2=NAS, 3=RMEAN, 4=RSD, 5=SCORE

```

```

ENTER CHOICE  :1
USING:        1

READING DATA FILE
  6721 DATASETS READ IN

ENTER Graph FILENAME [.graph]: <return>

Rescale Graph 1-100 [Y]: N
Enter the bucket interval required
10

```

The rescale graph option, scales the values reported to a 1-100 unit scale to allow comparison between different scan runs. The bucket interval refers to the range for each bucket in the histogram, you must choose a suitable value to give a clear representation of the scan. The result of the above analysis are shown below and in the file: `bash_ge_4_scan1.graph`.

Bucket	Score	Frequency	
1	9.430	384	=====
2	19.430	614	=====
3	29.430	1016	=====
4	39.430	1486	=====
5	49.430	1737	=====
6	59.430	910	=====
7	69.430	209	=====
8	79.430	26	
9	89.430	1	
10	99.430	7	
11	109.430	16	
12	119.430	17	
13	129.430	36	=
14	139.430	85	==
15	149.430	171	====
16	159.430	5	

13.7 Getting Help

Brief help is available by typing `/help` at the program prompt.

14 Extracting Sequences from a PIR Database Using PROGRAM SELECT

If you have access to a database management program (e.g. NBRF PSQ) then you may use that to extract the sequences of interest from the database. However, PROGRAM SELECT provides a method of extracting sequences from any PIR format sequence file. For example, given the file: `'bash_ge_4_scan1.top20id'` obtained above and the PIR database file `'protein.seq'`.

```

Program S E L E C T

```

```

Extracts sequences from PIR database

Author: G. J. Barton (1990)
Maximum Allowed Sequence Length: 8000
Maximum Allowed Number of Sequences: System Dependent

Enter name of file containing SCORE ID pairs: bash_ge_4_scan1.top20id

Opening File: bash_ge_4_scan1.top20id

Enter Database Filename: protein.seq

Opening File: protein.seq

Just Extract Identifiers/titles (no sequences) ?[Y/N]: N

Enter Output Filename: bash_ge_4_scan1.top20seq

Opening File: bash_ge_4_scan1.top20seq

Searching for: 20 Sequences
Found: HACHPE      1
Found: HADKP       2
Found: HZHU        3
Found: HZCZ        4
Found: HZPG        5
Found: HGHUA       6
Found: HGMQP       7
Found: HGMQR       8
Found: HGMQJ       9
Found: HGBAY      10
Found: HGMKS      11
Found: HBRB3      12
Found: HBMSH0     13
Found: HEGT1      14
Found: HEMS2      15
Found: HBTG       16
Found: HBOR       17
Found: HBPY       18
Found: HBTTP      19
Found: HBFG3T     20
Extracted: 20 Sequences

```

The information displayed to the screen signals when each sequence is found, the sequences are sorted into descending score order and output to the chosen file (bash_ge_4_scan1.top20seq).

15 Generating Alignments of the Pattern With Interesting Sequences

We now have a pattern and a file containing sequences with which it gives a good score. In order to see the alignments of the pattern and each sequence, we must re-run the MULTALIGN program with the print_horizontal (or _vertical) command. This produces a VERY verbose output. For example, the command file bash_ge_4_scan1_top20.com:

```
output_file=bash_ge_4_scan1_top20.alig
```

```

mode = scan
block_file/pattern=bash_ge_4.bloc,1
matrix_file=md.mat
database=bash_ge_4_scan1.top20seq
print_horizontal=

```

produces the output shown in file bash_ge_4_scan1_top20.alig.

A far more compact form of output may be obtained by using the PRINT_HORIZONTAL/PATTERN=fname command. This produces a file in a special format for the program PATT.

```

output_file=bash_ge_4_scan1_top20.out
mode = scan
block_file/pattern=bash_ge_4.bloc,1
matrix_file=md.mat
database=bash_ge_4_scan1.top20seq
print_horizontal/pattern=bash_ge_4_scan1_top20.patt

```

We can now run the program PATT on the resulting file:

```

-----
Program      P A T T E R N
-----

Processes FSCAN print_horizontal/pattern output

Author:  Geoff Barton

Maximum Pattern Length:      2000
Maximum Pattern Hits + Pattern to Display:      2000

Enter Pattern file: bash_ge_4_scan1_top20.patt

Enter Output file: bash_ge_4_scan1_top20.pattout

Enter page width for horizontal output (>50, Def:132):
Output width:      132
Reading Pattern Description
---Initializing
---Done
Reading Pattern Alignment
153.14 >HZPG      141    1    1 Hemoglobin zeta chain - Pig
152.57 >HZCZ      142    2    1 Hemoglobin zeta-1 chain - Chimpanzee
152.57 >HZHU      141    3    1 Hemoglobin zeta chain - Human
151.43 >HACHPE     141    4    1 Hemoglobin pi' chain - Chicken
149.57 >HADKP      141    5    1 Hemoglobin pi' chain - Muscovy duck
149.00 >HEMSY2     146    6    1 Hemoglobin epsilon-y2 chain - Mouse
148.00 >HBRB3      147    7    1 Hemoglobin gamma (beta-3) chain - Rabbit
147.29 >HBPY       146    8    1 Hemoglobin beta chain - Pigeon
147.14 >HBFG3T     146    9    1 Hemoglobin beta chain - Bullfrog tadpole
147.00 >HBMSH0     147   10    1 Hemoglobin beta-h0 chain - Mouse
146.86 >HBTG       146   11    1 Hemoglobin beta chain - Australian echidna
146.86 >HBTPP      146   12    1 Hemoglobin beta chain - Western painted turt
146.71 >HGMQJ      146   13    1 Hemoglobin gamma chain - Japanese macaque
146.71 >HGMQR      146   14    1 Hemoglobin gamma chain - Rhesus macaque
146.71 >HGBAY      146   15    1 Hemoglobin gamma chain - Yellow baboon
146.71 >HGMQP      146   16    1 Hemoglobin gamma chain - Pig-tailed macaque
146.71 >HGHUA      146   17    1 Hemoglobin gamma chains - Human and chimpanz
146.71 >HGMKS      146   18    1 Hemoglobin gamma chain - Spider monkey
146.57 >HBOR       146   19    1 Hemoglobin beta chain - Duckbill platypus

```

```
146.57 >HEGT1      147   20   1 Hemoglobin epsilon-I chain - Goat
```

```
Sort the scores? [Y]
Formatting Alignments
Adding Flexible Gap Details
Writing Vertical Format Alignment
Writing Horizontal Format Alignment
```

The only options in this program are the output width for the results and the option to sort the scores that are displayed. The sorting option is necessary if multiple patterns per sequence are calculated using the `PATTERN_LEVEL=N` option described below.

The output file from the program `PATT` contains vertical and horizontal format multiple alignments of the pattern with all the sequences in the list. This is a considerable compression of data over the `print.horizontal=` format (file `bash_ge_4_scan1_top20.alig`). The vertical format output may be used to define a further pattern for database scanning.

16 Generating multiple pattern alignments with each sequence

In the previous examples, only the best scoring alignment of the pattern with the sequence entries is output. However, a pattern may match equally well at more than one position in a sequence (for example, if a repeat is present). The following example, illustrates the principal of generating multiple pattern alignments, though the results are of limited interest.

(file `bash_ge_4_scan1_top20_patt2.com`)

```
output_file=bash_ge_4_scan1_top20.junk2
mode = scan
block_file/pattern=bash_ge_4.bloc,1
matrix_file=md.mat
pattern_level=5
database=bash_ge_4_scan1_top20seq
print_horizontal/pattern=bash_ge_4_scan1_top20.patt2
```

The only new command necessary is the `PATTERN_LEVEL=N` command. This specifies that the top N ways of aligning the pattern to EACH sequence will be output.

As before, the program `PATT` may be used to compress the results and store them in the file `bash_ge_4_scan1_top20.patt2out`. Inspection of this file shows that in this example, the alternative pattern alignments have identical scores and differ only in the positioning of the first group of pattern elements.

17 APPENDICES

17.1 Summary of Commands for MULTALIGN

Commands may be entered in mixed case. All commands must include the '=' delimiter. All commands may use a maximum of 80 characters including their arguments, all commands are free format.

output_file=STRING

STRING is the filename to be used for output of results and error messages. Must be the first command issued.

mode=OPTIONS

OPTIONS are one of MULTIPLE, PAIRWISE or SCAN

MULTIPLE is the default - signifies 'normal' multiple alignment and tree based alignments may be used.

PAIRWISE signifies conventional pairwise comparisons to be made for all sequences in the seq_file.

SCAN specifies that an alignment or pattern are to be compared to a group of sequences.

matrix_file=STRING

STRING is the filename of the pairscore matrix chosen (one of MD.MAT, UP.MAT, GC.MAT etc)

seq_file=STRING

STRING is the filename of the file containing sequences.

order_file=STRING

STRING gives the name of the order file

constant=INT

INT is a number added to each element of the matrix defined by matrix_file. (Default=8)

gap_penalty=REAL

REAL is a number used as the length independent gap penalty. (Default=8.0).

iterations=INT

INT defines the number of iterations performed in single order multiple alignment.

print_vertical=

Switch to specify vertical format output of alignments.

print_horizontal=

Switch to specify horizontal format output of alignments.

print_horizontal/pattern=STRING

Specify creation of an output file with a suitable format for program PATT. STRING is the name of the file.

max_horiz=INT

INT defines the number of residues output per line when the print_horizontal flag is set. (Default=100).

show_matrix=

Flag to specify that the pairscore matrix should be written to the output file after any constant has been added.

block_file=STRING,INT

Specifies the block_file name defined by STRING. INT specifies which iteration to read from the block_file. (INT MUST BE PRESENT OR AN ERROR WILL RESULT).

block_file/pattern=STRING,INT

Specifies a block file with flexible pattern ranges.

consplot=OPTION

Signals that a conservation profile should be plotted. Only works if the print_vertical command is set.

OPTIONS are MZ for conservation profiles according to Zvelebil et al (1987).

GB for a profile that is based upon the $N*(N-1)/2$ comparisons possible between all pairs of residues at a given alignment position.

gap_factor=REAL

When aligning to a block_file, this command specifies the factor to be applied to the gap_penalty within the defined regions.

conscore=

Flag to specify that the alignment will be scored using conservation values (a la Zvelebil et al 1987) instead of using the specified pairscore matrix.

show_lookup=

Flag to specify that the lookup_table should be written out at each cycle of the alignment. The format produced cannot subsequently be read back in by the program.

save_lookup=STRING

STRING is the name of a file to save the first generated lookup table to. Once saved, the program stops. The lookup table saved in this way may be read in by the program (see read_lookup) for subsequent processing.

s_lookup_comment=STRING

May be used in conjunction with save_lookup to specify a title for the saved lookup Table.

read_lookup=STRING

Defines the filename of a lookup table to be read in by the program. The Table overrides any table generated by pairscore matrices.

pairwise_random=INT1,INT2,INT3

Specifies the number of randomizations to be performed in pairwise per sequence pair in pairwise mode.

INT1 = initial number of randomizations,

INT2 = max number of randomizations,

INT3 = increment.

Normally one only requires a single value for randomizations - say 100.

This would be defined thus: pairwise_random=100,100,1

pairwise_timer=

Flag, if present then system statistics are recorded for each pairwise alignment.

pairwise_self=

Flag, if present then in pairwise mode a self comparison is also made for each pair of sequences. (eg. for ABC, AvA, BvB etc.).

pairwise_align_file=STRING

Specifies that alignments produced in pairwise mode be written to a file other than the output_file.

tree_file=STRING

Specifies the name of the file containing the tree to follow for tree based multiple alignment. Also tells the program to perform a tree alignment rather than a single order alignment.

numerate=

If print_vertical has been defined, then this will number each aligned position of the alignment.

non_alignment=

Flag, signals that no alignment is performed. This can be useful to switch formats from vertical to horizontal, calculate conservation values or numerate the sequences. Note, if a block file is being read in then a seq_file must also be defined).

database,N= STRING

STRING specifies the name of the database to scan using the SCAN mode.

N specifies how many randomisations to perform for each database entry

- normally do not include a value for N (very expensive in CPU and not very useful).

gap_char_dash=

Specifies that all gaps will be output as dashes '-' rather than spaces.

gap_char_dot=

Specifies that all gaps will be output as dots '.' rather than spaces.

use_end_pen=

Weights gaps at the ends of the alignment (overhangs). Only effective in tree_file alignments.

last_it=

In single order mode with multiple iterations, this requests that only the last iteration alignment is output. (Normally alignments for each iteration are displayed).

scansps_output=

In scan mode, this outputs a more condensed version of the output file which only has a score, id pair on each line rather than the verbose output produced by default. This file must be sorted using the Unix sort utility rather than SORTER. (e.g. for a file called RESULTS.SCAN type:

```
sort +0 -1 -n -r < RESULTS.SCAN > RESULTS.SORTED
```

If you take the top N results from the .SORTED file, you can then extract the sequences using the SELECT program.

17.2 MULTALIGN file formats

Where possible, MULTALIGN reads files in free format. In some cases however, files are fixed format- usually because these are generated by programs such as ORDER and would not normally be typed in by hand.

17.2.1 seq file

This file contains one or more protein sequence in ONE LETTER CODE. The format follows that of the NBRF (PIR) databases .SEQ file as follows.

```
>P1;IDENT
this is the title line
a s dhjAALLDKHGDK(D,K,L).P
L
WWPGS*
>P1;IDENT2
this is another title line
a R G S DF SDSDDSSDAKKKFG
*
etc...
```

Each sequence entry starts with a '*;*'. This is followed by an identification code (max 10 characters). The next record is a title line (max 500 characters). The following record(s) give the one letter code sequence. Maximum of 500 characters to a line. The end of the sequence is identified by a '*' character. Note that lowercase letters are put into uppercase, and only alphabetic characters are read on the sequence lines, any alphabetic characters that are not standard one_letter codes are translated as 'X' (unknown).

Minimum requirements are a record with '>', a title line and one residue followed by a '*'.

Examples: globin.seq, myoglobins.seq. Source: PSQ 'COPY' command, or typed in manually.

17.2.2 matrix_file

This follows the conventions of the NBRF (PIR) programs.

```
line1: Title of matrix
line2: 23 characters representing the one-letter codes and defining the order
that the matrix is stored in
lines3 to 25 are format(23I3).
```

MULTALIGN reads the one_letter code line and uses this as an index into the matrix that follows. Examples: md.mat,up.mat,gc.mat. Source: the above example files.

17.2.3 order_file

This is Fixed format file that defines the order of alignment.

Each line gives three numbers, A, B, C.

```
A is an index to which sequence in the seq_file should be aligned next,
B gives the alignment score that was obtained with...
C which is the sequence that gave the score with A.
```

Format(1x,i10,f20.2,i10)

MULTALIGN must have already read the seq_file before the order file is defined. the sequences are then re-ordered within the program to conform to the order specified by the order_file.

Examples: globin_pairs.ord

Source: output from program ORDER as a result of processing the results of a pairwise sequence comparison.

17.2.4 tree_file

This defines the order in which a tree based multiple alignment is performed. The format is FIXED and very simple. The aim of tree based alignment at each stage is to perform a pairwise alignment on two clusters of sequences that have already been aligned, or are individual sequences. The tree_file defines which sequences belong to each cluster at each stage of the alignment.

For example, we may have 5 sequences 1,2,3,4,5. At each stage we are aligning two clusters of sequences A and B. The tree file might look like this.

```
The tree (my comments - not in the file)
```

```
1 number of seqs in cluster A
3 seq 3
1 number of seqs in cluster B
4 seq 4 (A and B are now aligned)
1 Number of seqs in next cluster A
1 seq 1
1 Number of seqs in next cluster B
5 seq 5 (new A and B are now aligned)
```

```

2 Number of seqs in next cluster A
3   4 seqs 3 and 4
2
1   5                               seqs 1 and 5 ( now aliged to 3 and 4)
1                                   .
2                                   .
4                                   .
3   4   1   5                       Finally seq 2 is aligned to 3,4,1,5.

```

This tree can be gerated by program ORDER, or might be input from a more sophisticated clustering program.

Format(1x,20i5)
Example: globin_pairs.tree

Note. The sequence numbers identified in the tree_file point to the sequences as stored in the order internally by MULTALIGN. Normally an order_file would be used in conjunction with the tree_file so that similar sequences are clustered together on output. See the documentation on program ORDER for details of producing compatible tree and order files.

17.2.5 block_file

This defines a multiple alignment in vertical format. The print_vertical command produces a file in block_file format.

The minimum requirements for a block_file for N aligned sequences are

```

1. N '>comment line(s)'
2. '* iteration int'
3. 'N or more vertically aligned sequences'
4. '*'

```

1. The comment lines define the sequence identifiers and the number of 'i' characters preceding the first '* iteration int' line define the number of sequences that are defined in the sequence lines.
2. This line specifies the beginning of the alignment to be read. The '*' character specifies the column in which the alignment begins. The 'iteration int' specifier identifies the particular alignment within this block_file. Several alignments may follow each other providing they are identified by a different iteration number (eg. 1,2,3).
3. The alignment is ended by a '*' character which should be in the same column as the '*' character that started the alignment.

Simple example:

This is a block file containing two alternative alignments of three sequences. The comments that I an writing here may appear in the block file, but are ignored by MULTALIGN when the file is read. The only proviso is that no 'greater than' or 'star' characters must be present.

```

>first this is sequence A

```

```

>second this is sequence B
>third This is sequence C
* iteration 1
a
a p
avg
llg
lcr
g
pg
www
s
*
*iteration 2
a
a p
avg
llg
lcr
gpg
www
s
*

```

Examples: See bash.bloc, bash_vg.bloc and globin.bloc.

The block_file may also encode regions to apply variable gap_penalties within. See bash_vg.bloc for example syntax.

17.2.6 Save_lookup format

First line is a comment line - default written by MULTALIGN is the filename. Subsequent lines are all free format position specific residue weights.

Eg.

```

> 1 (weights versus first posn. of block_file)
a 3.0,g 4.0 p,2.0,      (Alanine gly and pro given weights - others score 0)
>2                      (weights for second position of block file)
g 100,
p 100,
> 3
a 33,
.. etc

```

See bash_day.look for a complete example. Note that if a residue is not defined at a position then its score at that position is zero.

17.2.7 Format for Input to program PATT

A rather bizarre but flexible format as generated by the print_horizontal/pattern option in multalign!!

```

#Pattern      !start of pattern description
Nelm, Nchar   !Number of elements in pattern/number of characters/element(10,9)
E1            !Pattern element 1 (eg. AAAGGGCCC) (80a1)
F1            !Flexible Gap range 1 (eg. 0,3)

```

```

"
"
etc
EN          !Nth pattern element - this ends pattern description
#Details    !Details of the first recorded pattern match
ID          !Protein Identifier 10 chars
Title       !Protein title - 80 chars
Len         !Protein Length - 80 chars max
An,Pn       !Sequence No, Pattern alignment Number
Score       !Real number - max 80 chars
#Sequence   !Start of sequence identifier
one letter  code sequence 80a1 format
#Match      !Description of the match follows
1,Pos       !N,Pos - N is the pattern element, Pos is the sequence element
2,Pos       !which it is aligned. If the pattern overhangs, then N is 0.
N,Pos
#Details    !Start of next pattern alignment, or end of file

```

This format file is output by MULTALIGN in response to the
 print_horizontal/pattern=fname command

Note: The An,Pn refer to the An(th) sequence aligned to the pattern,
 and the Pn(th) alignment of the pattern to this sequence - thus
 1,1 refers to the top scoring pattern match with sequence 1,
 1,2 the second best pattern match to seq 1 etc.

17.3 VMS installation guide

All the programs and datafiles should reside in the same directory. This should be logically defined in the command file AMPS.COM. Simply change the assign statements to indicate the directory that you have put AMPS in.

Executing the file AMPS.COM will set up the necessary logical names.

Thereafter, all the above examples as described should work. However, you will have to modify the command files as supplied to agree with the examples shown above.

Not all the programs work under VMS. MULTALIGN and ORDER should be OK. I've no plans to port the other programs to VMS.

17.4 Unix Installation Guide

All files should be read into one directory using the tar utility, then symbolic links set up to a suitable /bin directory for the executable files amult, dayb, sorts, sorter, patt and select. All the example files are contained in the subdirectory /test. For example, assuming the amps programs documentation and example files have been read into /local/progs/amps and that you use a directory /local/bin for storing local executables. Log on a super-user, then type:

```

cd /local/bin ln -s /local/amps/bin/multalign multalign ln -s /local/amps/bin/order order ln -s
/local/amps/bin/dayb dayb ln -s /local/amps/bin/sorter sorter ln -s /local/amps/bin/patt patt ln -s
/local/amps/bin/select select

```

Note that version 2.1 includes executables for Silicon graphics and Sun. These are all in the /bin directory and have the extension .sgi for silicon graphics executables and .sun for sun executables.

17.4.1 Unix usage

NOTE: these notes also apply to other installations, though the directories will be different. This document and the example files hide in /home/nmra/gjb/amps. In order to run the programs you need to add the directory /home/nmra/gjb/bin to the path in your .cshrc file. Once this is done, typing

```
multalign
will run multalign, and typing
order
will run the program order.
```

Any files referred to that are not in your working directory, must be given their FULL PATH NAMES. eg. the file md.mat is in /home/nmra/gjb/amps, so when specifying the matrix, the command is:

```
matrix_file=/home/nmra/gjb/amps/md.mat
and NOT:
matrix_file=ampsdir:md.mat
or similar.
```

Running the program, eg. with the command file bash_look.com is achieved by simply typing

```
multalign < bash_look.com
```

The program may be run in batch using the "at" utility - see the manual page (ie type man at). To use at, you will need to create a simple shell script to run the program, for example, this might look like this:

```
multalign < bash_look.com > bash_look.log
```

or similar.

NOTE: amult program limits in this version are 500 sequences of up to 1200 residues.

Any queries, send e-mail with problem files to gjb@bioch.ox.ac.uk.

18 References

- Needleman, S. B. and Wunsch, C. D. (1970). J. Mol. Biol. 48, 443-453.
- Barton, G. J. and Sternberg, M. J. E. (1987a). Protein Engineering 1, 89-94.
- Barton, G. J. and Sternberg, M. J. E. (1987b). J. Mol. Biol. 198, 327-337.
- Barton, G. J. and Sternberg, M. J. E. (1990). J. Mol. Biol. 212, 389-402.
- Barton, G. J. (1990), Methods. Enzymol., 183, 403-428.
- Zvelebil, M. J., Barton, G. J., Taylor, W. R. and Sternberg, M. J. E. (1987). J. Mol. Biol. 195, 957-961.
- Bashford, D., Chothia, C., and Lesk, A.M. (1987). J. Mol. Biol. (1987).190-216.
- Corter, J. E. (1982). Behaviour Research Methods and Implementation 14,353-354.
- Feng, D. F. and Doolittle, R. F. (1987). J. Mol. Evol. 25, 351-360.

19 Known Bugs

Please send bug reports to gjb@bioch.ox.ac.uk. Include example files. I will try to deal with the problems as they arise.

16th May 1991: Under VAX/VMS the `print_vertical` option is limited to 130 sequences.

13th Sep 1991: In Silicon Graphics version, program `SELECT` does not find four character id's in database - reason is obscure..
THIS BUG NOW FIXED.