UCL
Bioinformatics Unit

THREADER 3
User Guide

Protein Fold Recognition by Threading

Program and documentation by David T. Jones

University College London
Department of Computer Science
Bioinformatics Unit
Gower Street
London
WC1E 6BT
United Kingdom

Tel. +44 020 7679 7982
E-mail. dtj@cs.ucl.ac.uk
WWW. http://bioinf.cs.ucl.ac.uk

Contents

Chapter

1

Getting Started

Installing and running THREADER 3

INSTALLATION

Before you try to install this program, make sure your computer system is suitable. THREADER 3 needs a lot of memory to run properly, and ideally needs a reasonably fast CPU – though you can run it on a slow machine if you have the patience.

The basic Silicon Graphics version requires Irix 6.4 or a later version and will run on R4000, R5000, R8000, R10000 or R12000 series CPUs. R2000/R3000 series CPUs are not supported.

The Sun version requires Solaris 7 or a later version.

The Linux version is built using Red Hat Linux 7, but should work with any comparably recent Linux distribution. If you have problems running the Linux version, try the statically linked version (linux-static).

At least 128 Mb of RAM is required - ideally 256 Mb or more should be installed.

Around 100 Mb of swap space should be free – this is generally not an issue with modern machines.

To install the program and its associated data files, around 300 Mb of disk space is required.

The other thing you need to run this program is patience. If the protein sequence you are trying to thread is very long (say 500-800 residues) then it will take anywhere from 6 to 12 hours of CPU time to match the sequence against the current library of folds - much longer if you are using a particularly slow system.

If you have any problems running the program, check the following:

* Is the THREAD_DIR environment variable set correctly?

* Are the appropriate files located in $THREAD_DIR?

* Are the tdb files correctly located in $THREAD_DIR/tdb or the $TDB_DIR variable set appropriately?

* Is the threader.key file correct and located in $THREAD_DIR?

* Does your machine have enough memory?

* Is your machine running a recent version of the operating system?

RUNNING THREADER 3

In these examples, it will be assumed that the file containing the sequence under investigation is called *test.seq*. Most sequence file formats are supported, but the one we tend to use is compact PIR/OWL format:

```
>P1;TEST
This is our preferred sequence file format, but almost any will do
AIQERAMIILALQISTWPPQLIVMAAPEIPSQQPMSDSFEFGHIKLPEVYSQEWCNMPLQ*
```

The control parameters for THREADER 3 are specified as command-line options, but the defaults will do for many searches. The minimum command-line for running THREADER 3 is as follows:

```
threader test.seq results.out
```

This will fit the sequence in test.seq onto every chain listed in the file *domain.lst* which must be located in the current directory, and will send the results to a file called *results.out*. Logging output from the program can be directed to a file (called *log* in this case) as follows:

```
threader test.seq results.out >& log
```

NOTE that this assumes you are using csh or tcsh as your shell. If you are using another shell such as bash, then the syntax for redirecting to a file will be different – consult the man pages for your shell for more information.

The -p option will output (to standard output) the generated alignments as they are computed:

```
threader -p test.seq results.out >& log
```

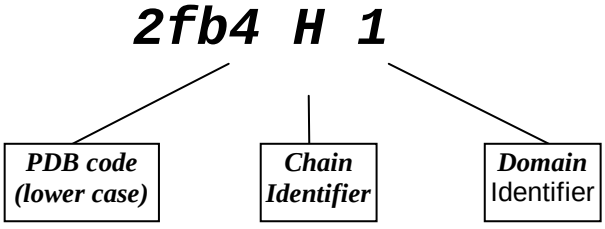
In this case the calculated energies will be summarized in *results.out*, and the alignments (along with other logging information) sent to the *log* file.

The list of protein chains used as a search library is specified as a file called domain.lst by default. This file must be found in the current directory (or THREAD_DIR) else an error will result. An alternative filename can be given, for example:

```
threader -p test.seq results.out mylist.lst >& log
```

This will thread the given sequence through the chains listed in the file mylist.lst. Note the format of this file is different from that used in the original program. THREADER 3 now uses a domain identifier scheme that is compatible with the underlying data in the UCL CATH database (see later). The new identifiers are always 6 characters long e.g. 2fb4H1. This is interpreted as Domain 1 of Chain H of PDB entry 2FB4.

Where no domains are defined, a default domain identifier of '0' is used. For example, the code 1mbd00 refers to the entire PDB entry 1MBD (sperm whale myoglobin) which has a single chain and no structural domains.



USING THE THREADING EXPERT (texp)

In this example, the use of the "Threading Expert" (texp) will be used to make a prediction. Although texp is an experimental utility, it does provide the easiest way to make a prediction using THREADER 3.

To use texp, a secondary structure prediction file is required for the target sequence (see later for details on this). In this case the file is called "test.horiz":

```
# PSIPRED HFORMAT (PSIPRED V2.3 by David Jones)

Conf: 976515778999999976179999999999984776533433312202078889999999
Pred: CCCCCHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCCCCCHHHHHHHHHEECCHHHHHHHHH
  AA: GHIANWQSIDELQDIASDLPRFIHALDELSRRLGLNITPLTADHISLRCHQNATAERWRR
              10        20        30        40        50        60


Conf: 999975898853699448999727741125737999865688788787665035889717
Pred: HHHHHHHHHHHCCCCCEEEEEECCCCCEECCCEEEEEECCCCCCCCCCCCCEEEEEEECC
  AA: GFEQCGELLSENINGRPICLFKLHEPVQVAHWQFSIVELPWPGEKRYPHEGWEHIEIVLP
              70        80        90       100       110       120


Conf: 880788999998753224545651156542689817998378689970676788742059
Pred: CCCHHHHHHHHHHHHHCCCCCCCCEEEEEECCCCCEEEEECCEEEEEECCCHHHHHHCCC
  AA: GDPETLNARALALLSDEGLSLPGISVKTSRLPNPTLAVTDGKTTIKFHPWSIEEIVASEQ
             130       140       150       160       170       180
```

The minimum command-line for running THREADER 3 with this file is as follows:

```
threader test.horiz results.ssout
```

This will fit the sequence and secondary structure in test.seq onto every chain listed in the file *psichain.lst* which must be located in the current directory or the location pointed to by $THREAD_DIR, and will send the results to a file called *results.ssout*.

After the threading run is complete, the output file can then be analysed using texp:

```
texp -s weights.dat results.ssout > results.texp
```

The output from texp should be sorted to find the most likely matches. In this case, the Unix head command is used to give the top 10 matches:

```
sort –n –r results.texp | head -10
```

Chapter

2

Running THREADER 3

Command-line options and interpretation of results

COMMAND-LINE OPTIONS

When running THREADER 3, The following command-line options may be specified:

-d nnn = set max. no. of considered paths (default: 60)

This parameter controls the depth of search for better alignments. The default of 60 results in quick search times at the expense of accurate alignments. The greater the number of alignment paths considered, the more reliable are the results. For short sequences it is reasonable to increase this parameter to say 200 without making the search times overly long. Generally it is sensible to start with the default value of 60 and then move to higher values if nothing is found at this level. If you are aligning a sequence onto just a single structure in order to generate a alignment for modelling then you can of course set this parameter very high (to 1000 say).

-c nnn = set pair potential cutoff distance (default: 10)

This parameter specifies the interatomic distance beyond which the pair potentials are not computed. This parameter should not be changed if the “Threading Expert” program is being used to analyse the output file.

-l nnn = set ungapped segment span (default: 4)

This controls the width of the reference region when calculating energies for lower matrices in the dynamic programming algorithm. As originally described only a single sequence-structure residue equivalence was maintained when optimizing the environment around a particular site (i.e. one residue was fixed to one structural site and the other n-1 equivalences generated by dynamic programming). However, it has been found that improved alignments and faster convergence are achieved if residues either side of the central fixed residue are fixed as well and used in the calculation of the potential environment. This parameter controls the width of this fixed region, and the default of 3 fixes 3 residues either side of the central position (i.e. a window of 3+3+1 = 7 residues). You should not have to adjust this parameter - however, if you do adjust it and find a better value than the default, let me know!

-l nnn = set loop gap penalty weight (default: 0.5)
-s nnn = set s-struct. gap penalty weight (default: 10.0)

These parameters adjust the relative weights given to indels in loop and secondary structure regions. The defaults result in penalising gaps in secondary structural regions 12.5 times more than gaps in loop regions. The defaults have been found to work well, but for your specific application, better values may well be found. N.B. the secondary structure weight is also applied to indels which whilst starting and ending in loop regions would result in

the deletion of entire secondary structural elements.

-x nnn = set gap penalty extension decay (default: 10)

This parameter is a modified length-dependent gap penalty term. For a gap of length p, an additional penalty of l.p/x (l/10 by default) is applied, where l is the loop gap penalty weight above.

-b = optimize solvation energy

Normally when searching for better alignments only the pairwise energy components are considered. In some cases, better alignments result from adding in the solvation terms during the optimization procedure, though this is in a minority of cases.

-p = print alignments

As described above. Final sequence-structure alignments are send to standard output as the search progresses. For the library structure, both the sequence and the associated secondary structure codes are output. The secondary structure codes are as follows: H:Helix, E:Strand, C:Coil.

-pm = print alignments in MODELLER format

As described above. In addition to the alignment printed in the usual format, the alignment is also printed in a format suitable for input into Andrej Sali's MODELLER package. This allows threading alignments to be converted into full atom models with minimal effort.

-v = select Verbose Mode

Sends a lot of semi-debugging junk to standard output as the search progresses. It helps me, but may not be very informative to others.

-j = thread all structures

Normally, THREADER 3 will ignore folds in the library which are either much too short or much too long. This cuts down the search time, but occasionally you might wish to turn this option off so that the target sequence can be threaded against every fold in the library. The –j option ensure that all the folds are considered, but results in a longer search time.

Example:

threader –h 0.5 –p –v test.fasta test.out psichain.lst

Note that unlike previous versions, you can put spaces between the option and the argument if you like. So, for example “-h0.5” would be equivalent to “-h 0.5” in the above command line.

INTERPRETING THE RESULTS

THREADER 3 generates a lot of output (though less than earlier versions) - in the form of a rather unfriendly table of numbers. However, only a few of them are really important in most cases. We make no apologies for dumping great volumes of data out - the different scoring schemes each tell a different side of the story. Matches which score very well in terms of pairwise energy but very badly in terms of solvation energy may be telling you something about the multimeric state of your protein sequence. Fitting haemoglobin onto myoglobin will give a good pairwise energy score, but a poor solvation score because haemoglobin is a tetrameric structure and myoglobin monomeric - the solvent accessibilities for myoglobin will not therefore be an ideal match for those of haemoglobin. The following snippet of output shows the format of the main output file:

```
  -95.29  -155.33  0.32   1.34 -0.34   -83.91 -0.01 -0.01  -36.7   82  92.1  75.9   0 1ay7B0
  -75.59   -69.74 -0.73   6.15 -2.51   -23.26 -2.13 -9.99  -11.0   51  77.6  48.1   0 1b33N0
  -74.59   -74.05 -0.78   0.01  0.26   -74.53 -0.34 -0.34  -84.0   83  91.2  76.9   0 1b6400
  -93.39   -63.68  0.22   0.73 -0.06   -87.18  0.10  0.10  -17.7   60 100.0  55.6   0 1bxyA0
 -112.45   -82.16  1.23   1.81 -0.55   -97.06  0.45  0.45  -83.0   66  93.1  62.0   0 1cc8A0
  -50.57   -93.07 -2.05  -1.46  0.93   -63.03 -0.74 -0.74 -120.1   87  75.0  80.6   0 1chuA3
  -99.76  -147.93  0.55  -3.19  1.70  -126.90  1.49  1.49  -28.5   66  98.5  62.0   0 1ctf00
  -76.86  -176.45 -0.66  -0.33  0.42   -79.70 -0.16 -0.16  -63.9   86  79.6  79.6   0 1dj0A1
  -65.64   -74.75 -1.25   0.11  0.21   -64.68 -0.68 -0.68  -32.1   65  98.5  61.1   0 1eayC0
  -90.50  -103.56  0.06   1.01 -0.19   -81.88 -0.08 -0.08  -92.7   82  81.2  75.9   0 1fj7A0
  -89.52   -86.71  0.01   3.76 -1.43   -57.47 -0.94 -0.94  -79.7   58 100.0  53.7   0 1fxd00
 -110.72  -152.25  1.14  -2.36  1.33  -130.85  1.63  1.63  -77.9   79  94.1  74.1   0 1fxlA2
  -74.94   -61.63 -0.76   3.06 -1.12   -48.86 -1.24 -1.24  -59.1   70  88.8  65.7   0 1jb0C0
 -112.64  -194.80  1.24  -1.13  0.78  -122.27  1.33  1.33  -79.5   69 100.0  64.8   0 1mla02
  -54.25  -115.84 -1.86   0.76 -0.08   -47.80 -1.27 -1.27 -134.5  100  81.5  93.5   0 1mxa02
 -104.20   -93.53  0.79   0.28  0.14  -101.84  0.61  0.61  -78.5   88  90.7  81.5   0 1phzA1
 -101.95  -135.87  0.67  -0.26  0.38  -104.13  0.69  0.69  -40.7   76  91.7  71.3   0 1psdA3
 -117.91  -145.89  1.52  -0.92  0.68  -125.74  1.45  1.45 -105.0   94  91.3  88.0   0 1pysB4
  -88.99   -75.08 -0.02   4.94 -1.96   -46.90 -1.30 -1.30  -67.0   83  94.3  76.9   0 1qm9A2
  -94.38   -52.16  0.27  -0.78  0.62  -100.98  0.58  0.58  -53.8   71 100.0  66.7   0 1qupA2
```

Column 1 is the raw pairwise energy sum for the given threading, the units are in kcal/mol.

Column 2 is the native energy for the template structure (i.e. the energy of the known structure with its own sequence). If the energy of the threaded model (in Column 1) is lower than that of the native structure (Column 2) then it is likely that this is a false match.

Column 3 is the Z-score (-(energy - mean) / standard_deviation) for the pairwise energies.

Column 4 is the raw solvation energy sum.

Column 5 is the Z-score of the solvation energy.

Column 6 is weighted sum of pairwise and solvation energy, weighted by standard deviations as described in the original paper (Jones *et al.*, 1992).

Column 7 is a Z-score of the values in Column 6.

Column 8 is the Z-score of values in Column 7 where models with too little matched structure or with other obvious faults given a score of -9.99. This is the primary field on which predictions should be based.

Column 9 gives the sequence similarity score for the model (using the BLOSUM62 score matrix or sequence profile information in the TDB file where available). A high score here might indicate common ancestry between the target and template structures.

Column 10 gives the number of aligned residues. It is frequently worth looking closely at models which score highest in this column.

Column 11 is the percentage of the structure which has been aligned.

Column 12 is the percentage of the sequence which has been aligned.

Column 13 is the start position for a local domain match (see the section on domain matching). For normal

searches this column always contains zero.

Finally, Column 14 gives the Brookhaven PDB code of the structure plus the chain and domain identifier. For example, 2tmdA1 indicates the 1st domain of the A chain of PDB entry 2TMD.

A useful way to handle this output is to sort the file using the Unix sort utility. For example, the following Unix command will print the top 10 hits, sorted on the basis of the filtered combined energy Z-scores (Column 8), which is the most useful column in general:

```
sort -n -r +7 results.out | head -20
```

Note that the +7 specifies the 8th column of the output file.

You can of course send the whole sorted list to a printer, for example:

```
sort -n -r +7 results.out | lp
```

It is obviously useful to try sorting on different columns to see which protein structures are the best matches by different criteria. A useful second criterion is the number of aligned residues. It is often the case that the correct models are amongst those with the largest number of aligned residues. This can be combined with the sort command above like this:

sort –n –r +7 results.out | head -20 | sort –n –r +9

The following steps are ones we typically use in interpreting the reliability of predictions:

1. Sort the output by the filtered combined energy Z-scores (Column 8) and look at the top Z-scores. For these scores, our experience has shown the following interpretation to be useful:

Z > 4.0	Very significant - probably a correct prediction
Z > 3.5	Significant - good chance of being correct
2.7 < Z < 3.5	Borderline significant - possibly correct
2.0 < Z < 2.7	Poor score - could be right, but needs other confirmation
Z < 2.0	Very poor score - probably there are no suitable folds in the library
Z = -9.99	Too little of the sequence or template fold have been aligned.

Note that these Z-score ranges depend on the number of folds in the fold library. If you use a much smaller library than the default (which has over 700 folds) then the Z-scores will not be useful measures of significance. If you use a larger library then the significance cut-offs should be increased.

There is no substitute for biological knowledge in making a structure prediction. If the structure at rank 3 looks much more biological plausible that the fold at the top of the list then this model should be taken seriously.

Also take careful note of the percentage of sequence/structure matched. If a proposed threading alignment results in only half of a structural folding pattern (e.g. a TIM barrel) being matched, then this prediction is likely to be false. Partial matches to some other folds can be valid, however.

When the '-r' option is used with THREADER 3, a set of randomization tests are performed on each match (see section 3 for more details). This produces an output file with a different format from the normal one. For example:

```
 -122.645  -143.274   2.00    -2.876    -4.313   1.63   2.61  75.7  86.2   0 1prr00
 -133.800  -176.084   1.50    -6.248    -6.218   2.20   2.52  94.2  64.5   0 3rp2A2
 -131.236  -158.455   1.42    -4.857    -3.625   2.35   2.45  94.5  78.9   0 1eta10
  -63.497   -70.730   1.42    -7.938    -6.176   2.32   2.44  90.8  71.7   0 1hfh00
 -191.949  -171.735   3.42    -5.019    -7.250   1.19   2.14  85.1  67.8   0 1igcH1
 -169.914  -208.588   1.33    -5.126    -5.456   1.90   2.14  93.7  68.4   0 1hplA2
.
.
.
```

Column 1 is the pairwise energy for the target sequence on the given structure.

Column 2 is the lowest pairwise energy for the shuffled target sequences on the given structure.

Column 3 gives the Z-score for the energy in Column 1. The mean and standard deviation of the shuffled sequence energies are calculated to arrive at this Z-score.

Columns 4-6 are the same as columns 1-3, but for solvation energies rather than pairwise energies.

Column 7 is a shuffled Z-score for a combination of pairwise and solvation terms, and is the most important scoring column. Significant scores in this column indicate interesting matches.

Column 8 is the percentage of the structure which has been aligned.

Column 9 is the percentage of the sequence which has been aligned.

Column 10 is the start position for a local domain match (see the section on domain matching). For normal searches this column always contains zero.

Finally, Column 11 gives the Brookhaven PDB code of the structure plus the chain and domain identifier.

GENERAL ADVICE

What about non-globular and multi-domain proteins?

Some types of protein can be very problematic. In particular non-globular proteins are not readily identifiable by threading (in particular, very many viral proteins are in this category). If you are trying to thread a very large sequence, say 800 residues, then unless you know where the domain boundaries are, you will not be very successful. Threading *cannot* be reliably used for identifying domain boundaries - sorry! If you do know where the domain boundaries are, then divide your sequence up before trying to thread it, and thread each domain separately. If you do blindly try to predict a very long multidomain sequence as a single chain then be very suspicious of the results, unless you are sure that the matched protein has a similar domain structure. For example, the periplasmic small-molecule binding proteins are two domain structures (two doubly wound parallel $\alpha\beta$ domains), but they all match each other fairly well as they all have identical domain organization (indeed we made a successful bona fide prediction of a protein with a fold similar to these proteins). In contrast, however, something like pyruvate kinase has a number of quite distinct structural domains, and this organization is quite probably unique to pyruvate kinase. If you match pyruvate kinase something without allowing for these distinct domains, then you will probably get bogus results. In general, if you can, divide your sequence into likely domains first.

We do have an experimental domain boundary prediction method available as a Web server which you might find useful. It can be found here:

http://bioinf.cs.ucl.ac.uk/marsden/DomSSEA.html

How do you know which fold corresponds to a particular PDB file?

A good way to find out is to access the UCL fold classification database (which we call CATH - Class Architecture Topology Homology). This database is available via World Wide Web from URL: *http://www.biochem.ucl.ac.uk/bsm/cath*

Chapter

3

New Features for Version 3

Examples using the new facilities in THREADER 3

SO WHAT'S NEW?

In this section, the new features of THREADER 3 will be discussed more fully. Apart from improvements to the basic algorithm and potentials used for threading, THREADER 3 has some new options which help in a number of areas. Many of these new features are experimental, but we've found them very useful.

USING PREDICTED SECONDARY STRUCTURE

Another useful new experimental feature in THREADER 3 is the ability to constrain the threading procedure by inputting predicted secondary structure. THREADER 3 now makes use of PSIPRED secondary structure predictions. PSIPRED is a very accurate secondary structure prediction method which can be accessed either from our Web server or downloaded. See the following URL:

http://www.psipred.net

THREADER 3 can read in the predicted secondary structure and positional reliability information to force the threading alignments generated into agreement with the predicted helices and strands. This feature allows users to combine the best features of potential-based threading methods with the best features of methods which attempt to compare predicted secondary structural patterns with the secondary structural patterns of known structures. Note that the source of the input secondary structure information does not necessarily have to be a prediction method. If you have reliable secondary structure information from, for example, NMR chemical shift experiments, then this can be used just as well as long as the file format is correct. The easiest way to do this is to generate a secondary structure prediction using PSIPRED and then edit the results according to the known experimental information.

In this example, the following sequence was e-mailed to the PSIPRED server:

```
>ArgR N-terminal domain from E.coli
MRSSAKQEELVKAFKALLKEEKFSSQGEIVAALQEQGFDNINQSKVSRMLTKFGAVRTRN
AKMEMVYCLPAELGVPTTS
```

The resulting prediction output looks like this:

```
Conf: 986288999999999986068888899999998634111446788889987321221 58
Pred: CCCHHHHHHHHHHHHHHHHHCCCCCHHHHHHHHHHCCCCCCCHHHHHHHHHHHHHEECCC
  AA: MRSSAKQEELVKAFKALLKEEKFSSQGEIVAALQEQGFDNINQSKVSRMLTKFGAVRTRN
            10        20        30        40        50        60


Conf: 8886789814777888889
Pred: CCCCEEEEECCCCCCCCCC
  AA: AKMEMVYCLPAELGVPTTS
            70
```

IMPORTANT: if you are using the PSIPRED server you should edit the returned e-mail message so that the first line of the file is the first line starting with “Conf:” as shown above. If you are using the standalone PSIPRED software, then the output files (.ss2 or .horiz) files can be recognised by THREADER 3 without any editing.

When THREADER 3 recognizes a PSIPRED input file, it enables an extra command-line option:

-h nnn = set minimum reliability for 2ndary struct. filter (default: 0.5)

Obviously some parts of a secondary structure prediction will be more reliable than others. This might also apply to experimentally derived secondary structural information. The 'h' option allows you to select a cutoff on this reliability. The default is 0.5, on a scale of 0 to 1, which means that any predicted secondary structure that has a reliability lower than 0.5 (marked with 5 in the output file above) will be ignored. This allows THREADER 3 to produce threading alignments that agree with the most confident parts of the secondary structure prediction.

THREADER 3 will attempt to match regions of the secondary structure prediction that are above the reliability threshold to regions in the template structure which have the same type of secondary structure.

NOTE that the additional secondary structural information is not used in calculating the threading energies, but only in the alignment algorithm. Obviously, however, secondary structure constraints will generate different alignments from those obtained without them - the algorithm will ignore lower energy threading alignments in favour of higher energy alignments which are in agreement with the predicted secondary structure.

THREADER 3 will also output the predicted secondary structure in the normal alignment output (-p option). So for the above example, an alignment might look like this:

```
             10        20        30        40        50
---------9999999999-------2219999999999------66666666666666-
-----CCCCHHHHHHHHHHCCCCCCCEEHHHHHHHHHHHHCCCCCCHHHHHHHHHHHHHHC
-----SHPTYSEMIAAAIRAEKSRGGSSRQSIQKYIKSHYKVGHNADLQIKLSIRRLLAA
               |    ||    ||   |                           |
MRSSAKQEELVKAFKALLKEEK---FSSQGEIVAALQEQG-FDNINQSKVSRMLTKFGAV
CCCCHHHHHHHHHHHHHHHHCC---CCCHHHHHHHHHHHC-CCCCCCHHHHHHHHHCCCC
        10        20           30         40        50

   60        70
--1111-------1111-----------
CCEEEECCCCCCCEEEECC---------
GVLKQTKGVGASGSFRLAK---------

RTRNAKME-----MVYCLPAELGVPTTS
ECCCCEEE-----EEEECCCCCCCCCCC
  60            70
```

Note that both the sequence of the template structure *and* the target sequence are now labelled with secondary structure codes, whereas normally just the template sequence is labelled this way.

THE THREADING EXPERT

An experimental new feature in THREADER 3 is called the “Threading Expert”. This is a separate program which attempts to analyse the output from THREADER 3 in order to produce a single number which corresponds to the likelihood of the model being correct. This is done using a neural network which has been trained to identify correct threading matches from incorrect ones.

The default neural network weights are based on the assumption that THREADER was run with default parameters, and with predicted secondary structure for the target. If you change the parameters or omit to use predicted secondary structure, then the Threading Expert will be much less reliable.

SHUFFLING TESTS

One major problem with threading is how to eliminate false positive matches from true positives. A very long practised technique in basic sequence analysis is to compare the match scores for a real sequence with those generated by random sequences of the same length and amino acid composition. Most commonly these random sequences are produced by *shuffling* the original sequence. By using Z-scores calculated by shuffling experiments, false positive matches can be identified, and also the true positive matches which produce the most accurate alignments are also highlighted.

To demonstrate the use of these tests, we will consider a case study. In this study, the target protein is as follows:

```
>P1;GUN1_STRRE
Endo-1,4-beta-glucanase
VEQVRNGTFDTTTDPWWTSNVTAGLSDGRLCADVPGGTTNRWDSAIGQNDITLVKGETYR
FSFHASGIPEGHVVRAVVGLAVSPYDTWQEASPVLTEADGSYSYTFTAPVDTTQGQVAFQ
VGGSTDAWRFCVDDVSLLGGVP*
```

Suppose that, after running THREADER 3 on this sequence, ranking the results by the combined Z-score (column

8) gives the following top-10 matches:

```
1igcH1
1hplA2
3rp2A2
2aaiB1
1dlc02
1brbE1
1f3g00
1eta10
1sriA0
1sdyA0
… etc.
```

In this case one very close structural match for the target sequence is included in the top 10 matches, but not in first place. Also, in this case, the accuracy of the sequence-structure alignment also turns out to be remarkably good. The task in this case is to identify which of these matches are false positives, and which is really the best matching fold. To achieve better discrimination, a set of randomization tests can be carried out. To save computer time it is sensible to limit the rigorous randomization tests to only those matches which looked promising from the initial results (in this example just the top-20 hits), but if time is not an issue then extensive randomizations can be performed on the whole fold library. In this example, the top 20 matches from the initial threading pass are placed in a list file (short.lst in this example):

```
1igcH1
1hplA2
3rp2A2
2aaiB1
1dlc02
1brbE1
1f3g00
.
.
.
```

The format of this temporary file is exactly as previously described for the default domain.lst file, except in this case only 20 domain folds are listed.

To perform the randomizations on these 20 folds, a command like the following is used:

threader -r50 cbdn.seq cbdn.zout short.lst

The "-r50" option indicates that we wish to compute 50 shuffled-sequence threadings for each fold in short.lst. This will take some time to complete, but the results might look something like this:

```
-156.193  -161.142   2.18    -4.585    -4.606   2.15   3.50  86.1  89.5   0 1dlc02
-115.322  -129.319   1.92    -4.391    -2.696   2.20   2.60  73.1  89.5   0 1knb00
-134.981  -160.531   1.95    -2.687    -5.388   1.90   2.52  74.5 100.0   0 1sacA0
-122.645  -143.274   2.00    -2.876    -4.313   1.63   2.41  75.7  86.2   0 1prr00
-133.800  -176.084   1.50    -6.248    -6.218   2.20   2.42  94.2  64.5   0 3rp2A2
-131.236  -158.455   1.42    -4.857    -3.625   2.35   2.45  94.5  78.9   0 1eta10
 -63.497   -70.730   1.42    -7.938    -6.176   2.32   2.34  90.8  71.7   0 1hfh00
-191.949  -171.735   3.42    -5.019    -7.250   1.19   2.14  85.1  67.8   0 1igcH1
-169.914  -208.588   1.33    -5.126    -5.456   1.90   2.14  93.7  68.4   0 1hplA2
-135.036  -167.482   1.24    -3.119    -2.259   2.12   2.09  77.6 100.0   0 1tsrA0
-108.076  -127.304   1.09    -7.779    -7.343   2.19   1.90  87.0  78.9   0 2aaiB1
-117.302  -154.001   1.17    -3.232    -4.404   1.50   1.72  91.5  92.1   0 1sdyA0
-104.070  -138.636   1.10    -4.135    -6.453   1.58   1.69  94.5  68.4   0 1exg00
-125.638  -164.866   2.01    -2.843    -5.617   0.95   1.66  96.8  59.9   0 1cgt02
-119.736  -130.997   1.34    -2.365    -4.963   1.08   1.53  93.8  88.8   0 1tnrA0
-100.863  -134.991   0.78    -7.460    -5.331   3.38   1.52  95.9  76.3   0 1sriA0
-100.520  -128.168   0.75    -5.519    -4.657   2.49   1.47 100.0  61.8   0 1eft03
-103.757  -127.228   0.93    -3.767    -6.217   1.29   1.36  94.7  82.9   0 1bvp12
 -85.509  -132.068   0.48    -6.274    -5.097   2.82   1.15  97.9  62.5   0 1pla00
-112.323  -132.120   1.62    -2.685    -6.396   0.52   1.09  90.6  82.9   0 1prcH2
```

In this example, the results of the shuffling experiments are shown sorted by the values in the 7th column (combined shuffled Z-score). On this basis one match (the correct one!) stands out quite clearly: 1dlc02, with a combined Z-score of 3.5 (which is significant for just 50 shuffles).

USING SEQUENCE SIMILARITY TO IMPROVE ALIGNMENTS

If there is a distant evolutionary relationship between the target sequence and a particular entry in the fold library, then it is often the case that better alignments are produced by taking sequence similarity into account. For example, threading alignments between trypsin-like serine proteases are often improved by also considering amino-acid similarity scores during the alignment step. THREADER 3 allows amino acid similarity to be considered when making alignments by using the following command-line option:

-Snnn = set mutation data matrix weighting (default: 0)

With the default setting of 0, no sequence similarity is weighted into the alignment process, and the alignments are generate purely by threading. With a weighting of 1000, the alignment is completely weighted towards maximizing the sequence similarity, and is therefore merely a sequence alignment and not a threading alignment at all. A weighting of 100 is set to be the point at which there is a balance between threading and sequence alignment - i.e. a 50-50 mix.

LOCATING STRUCTURAL DOMAINS BY THREADING

As mentioned in an earlier section, domains cause a particular problem for threading. People often want to know how they can use threading to locate domain boundaries in a newly characterised sequence. The truthful answer is that threading cannot be relied upon for this purpose - you need to break up your sequence into likely domains before you start threading. Sometimes threading can help identify domain boundaries, however, but it is strongly suggested that you avoid this approach if at all possible as it is unreliable. In addition this feature is poorly tested right now. To locate domains in a sequence the folllowing command-line option should be specified:

-Dn or -Dc = select domain finding mode

When this option is specified, THREADER 3 will restrict threading alignments to the N-terminus (-Dn) of the target sequence, or to the C-terminus (-Dc). Thus for each fold in the fold library, the threading will be anchored either to the start or end of the target sequence. Where the division is made depends on the size of the template structure. So, for example, if the template structure from the fold library is 100 residues long, then THREADER 3 will align this structure to either the first 100 residues of the target sequence, or to the last 100 residues of the target sequence. In this way THREADER 3 will hopefully locate either an N-terminal or C-terminal domain (or both) in your target sequence. Actually slightly more than 100 residues would be considered in the alignment, as an allowance has to be made for insertions in the target sequence.

The position of the start of the domain region is given in the output file (Column 16 in the normal output file format - see Chapter 3). For example, the following snippet of output:

```
  -95.29  -155.33  0.32   1.34 -0.34   -83.91 -0.01 -0.01  -36.7   82  92.1  75.9 120 1ay7B0
  -75.59   -69.74 -0.73   6.15 -2.51   -23.26 -2.13 -9.99  -11.0   51  77.6  48.1 145 1b33N0
.
.
.
```

shows that using the –Dc option, the first match to 1ay7B0 starts at position 120 (underlined in the output) in the target sequence.