Wise2 Documentation (version 2.1.17 beta)

Ewan Birney, Richard Copley
Sanger Centre
Wellcome Trust Genome Campus
Hinxton, Cambridge CB10 1SA,
England.
Email: birney@sanger.ac.uk,richard.copley@embl-heidelberg.de

March 22, 2004

Contents

1 Overview

Wise2 is a package focused on comparisons of biopolymers, commonly DNA sequence and protein sequence. There are many other packages which do this, probably the best known being BLAST package (from NCBI) and the Fasta package (from Bill Pearson). There are other packages, such as the HMMER package (Sean Eddy) or SAM package (UC Santa Cruz) focused on hidden Markov models (HMMs) of biopolymers.

Wise2's particular forte is the comparison of DNA sequence at the level of its protein translation. This comparison allows the simultaneous prediction of say gene structure with homology based alignment. There is currently no other package that I know of that contains this type of algorithm with a full blown gene prediction model and a hidden Markov model of a protein domain.

Wise2 also contains other algorithms, such as the venerable Smith-Waterman algorithm, or more modern ones such as Stephen Altschul's generalised gap penalties, or even experimental ones developed in house, such as dba (see section 6.1). The development of these algorithms is due to the ease of developing such algorithms in the enviroment used by Wise2.

Wise2 has also been written with an eye for reuse and maintainability. Although it is a pure C package you can access its functionality directly in Perl. Parts of the package (or the entire package) can be used by other C or C++ programs without namespace clashes as all externally linked variables have the unqiue identifier Wise2 prepended. Java and CORBA ports are being considered - see 7 the API section

Finally Wise2, although implemented in C makes heavy use of the Dynamite code generating language. Dynamite was written for this project, by Ewan Birney. There is a separate documentation for Dynamite found at http://www.sanger.ac.uk/Software/Dynamit

1.1 Authors

The Wise2 package was principly written by Ewan Birney, who wrote the main genewise and estwise programs. The protein comparison database search program was written by Richard Copley using the underlying Wise2 libraries. Wise2 also uses code from Sean Eddy for reading HMMs and for Extreme value distribution fitting.

However the authorship of Wise2 should be more fairly distributed between the main authors and the wonderful alpha testers on wise-alpha. Special mention goes to Gos Micklem and Niclas Jareborg and for their work at testing and their patience in my coding over the last couple of years. Other notables are (in no apparent order) - Erik Sonnhammer, Doug Rusch, Steve Jones, Ian Korf, Iftach Nachman, George Hartzell and Lars Arvestead. I believe that program writing is a 50-50 partnership between the coders and the testers or developers, and these people have actively helped me make a much better package. The URL for

Wise2 development is http://www.sanger.ac.uk/Software/Wise2/Programming and there is a mailing list to keep people up to date.

Please join us!

2 Introduction for the impatient

It may well be that you want to understand Wise2's functionality now, without bothering with the concepts or the installation instructions. This section is designed for you.

Wise2 has four main executable programs using sequence inputs which are designed to provide access to the main algorithms sensibly. The algorithms you are interested in is *genewise* - compare protein information to genomic DNA and *estwise* - compare protein information to EST/cDNA DNA.

These are the programs which you might use for this.

genewise a single protein vs a single genomic dna sequence

genewisedb a database of proteins vs a database of genomic dna sequences

estwise a single protein vs a single EST/cDNA sequence

estwisedb a database of proteins vs a database of EST/cDNA sequences

If you see error messages like

```
Warning Error
        Could not open human.gf as a genefrequency file
Warning Error
        Could not read a GeneFrequency file in human.gf
...
```

This means that the enviroment variable WISECONFIGDIR has not been set up correctly. You need to find where the distribution was downloaded to (a directory called something like wise2.1.16b) and inside that directory should be the configuration directory wisecfg. You need to setenv WISECONFIGDIR to that directory.

In each of the programs the protein can either be a protein sequence or a protein profile HMM, as made by the HMMER package (both version 1 and version 2 HMMs can be read). Any of the databases can have one entry (in which case more efficient routines are used), and databases of profile HMMs, such as those provided by Pfam, can be used.

The simple running of a protein sequence (drosophila) vs a human genomic sequence, using genewise is given below. The output comes on stdout, which in normal unix notation can be redirected to a file.

```
adnah:[/birney/search]<98>: genewise road.pep hngen.fa
genewise (unreleased release)
This program is freely distributed under a GPL. See source directory
Copyright (c) GRL limited: portions of the code are from separate copyright
```

```
Query protein:       roa1_drome
Comp Matrix:         blosum62.bla
Gap open:            12
Gap extension:       2
Start/End            local
Target Sequence      HSHNRNPA
Strand:              forward
Gene Paras:          human.gf
Codon Table:         codon.table
Subs error:          1e-05
Indel error:         1e-05
Model splice?        model
Model codon bias?    flat
Model intron bias?   tied
Null model           syn
Algorithm            623
Find start end points: [25,1387][346,3962] Score 87719
Recovering alignment: Alignment recoveredExplicit read offone 94%
genewise output
Score 253.10 bits over entire alignment
Scores as bits over a synchronous coding model

Warning: The bits scores is not probablistically correct for single seqs
See WWW help for more info



roa1_drome        88 AQKSRPHKIDGRVVEPKRAVPRQ                       DID
                     A  +RPHK+DGRVVEPKRAV R+                       D
                     AMNARPHKVDGRVVEPKRAVSRE                       DSQ
HSHNRNPA        1867 gaagaccagggagggcaaggtagGTGAGTG  Intron 2   TAGgtc
                     ctacgcaataggttacagctcga<0-----[1936 : 2083]-0>aca
                     tgtagacggtaatgaagatccaa                       tta


roa1_drome       114 SPNAGATVKKLFVGALKDDHDEQSIRDYFQHFGNIVDINIVIDKETGKK
                      P A  TVKK+FVG +K+D +E  +RDYF+ +G I  I I+ D+ +GKK
                     RPGAHLTVKKIFVGGIKEDTEEHHLRDYFEQYGKIEVIEIMTDRGSGKK
HSHNRNPA        2093 acggctagaaatgggaaggaggcccagttgctgaaggagaaagcgagaa
                     gcgcatctaatttggtaaacaaaatgaataaagatattattcaggggaa
                     aatccatgagatttctaactaatcaatttagtaatagtacgtcactcga


roa1_drome       163 RGFAFVEFDDYDPVDKVV                          QKQHQ
```

```
                     RGFAFV FDD+D VDK+V                          QK H
                     RGFAFVTFDDHDSVDKIV          L:I[att]        QKYHT
HSHNRNPA        2240 agtgtgatggcgtggaagAGTAAGTA  Intron 3   TAGTTcatca
                     ggtcttctaaaactaatt <1-----[2295 : 2387]-1>  aaaac
                     gctctactcctccgtgtc                          gactt


roa1_drome       187 LNGKMVDVKKALPKQNDQQGGGGGR
                     +NG   +V+KAL KQ         R
                     VNGHNCEVRKALSKQEMASASSSQR          G:G[ggt]
HSHNRNPA        2405 gagcatggaagctacgagagttacaGGTATGCT  Intron 4
                     tagaagatgactcaaatcgcccgag <1-----[2481 : 2793]
                     gtccctataacgagaggtttaccaa

...truncated
```

The output is as follows

- Parameters of the comparison used (it used default parameters)
- The alignment of a combined homology + gene prediction alignment

The pretty alignment shows the protein sequence on the first line, followed by a line indicating the similarity level of the match followed by 4 lines representing the DNA sequence. The DNA sequence in the exons descending in triplets, each triplet being a codon. The translation of each codon is shown above it. Between the two protein sequences a line indicating the similarity of the match is printed. In introns the DNA sequence is not shown but for the first 7 bases (making the 5' splice site) and the last 3 bases of the 3' splice site. The intervening sequence is indicated in the square brackets. Above each intron, for phase 1 and 2 introns (ones that split a codon) the implied protein to conceptual gene match is displayed, with the codon in square brackets.

Generally the defaults of the options are reasonably sensible, and for the main part you should trust them until you become familar with the package.

The following commands show how to run the other programs in a variety of different modes

2.1 Common running modes

Running modes for genewise (genomic to protein comparisons).

NB, the order of the -options are not important, but the protein file must be before the dna file

```
genewise protein.pep cosmid.dna
```

- compares a protein sequence to a DNA sequence (same as the example above)

```
genewise -hmmer pkinase.hmm cosmid.dna
```

- compares a protein profile HMM to a DNA sequence

```
genewisedb protein.pep human.fa
```

- compares a single protein sequence to a database of DNA sequences

```
genewisedb -hmmer pkinase.hmm human.fa
```

- compares a single protein profile HMM to a database of DNA sequences

```
genewisedb -prodb protein.pep -dnas cosmid.dna
```

- compares a database of protein sequences to a single dna sequence

```
genewisedb -pfam Pfam -dnas cosmid.dna
```

- compares a database of protein profile HMMs to a single dna sequence

```
genewisedb -prodb protein.pep human.fa
```

- compares a database of protein sequences to a database dna sequences - beware, this will take a while!

```
genewisedb -pfam Pfam human.fa
```

- compares a database of protein profile HMMs to a database of single sequences - beware, this will take a while

The estwise (protein to est/cDNA comparisons) have precisely the same running modes. Listed for completeness below

```
estwise protein.pep singleest.fa
```

- compares a protein sequence to a DNA sequence (same as the example above)

```
estwise -hmmer pkinase.hmm singleest.fa
```

- compares a protein profile HMM to a DNA sequence

```
estwisedb protein.pep est.fa
```

- compares a single protein sequence to a database of DNA sequences

```
estwisedb -hmmer pkinase.hmm est.fa
```

- compares a single protein profile HMM to a database of DNA sequences

```
estwisedb -prodb protein.pep -dnas singleest.fa
```

- compares a database of protein sequences to a single dna sequence

```
estwisedb -pfam Pfam -dnas singleest.fa
```

- compares a database of protein profile HMMs to a single dna sequence

```
estwisedb -prodb protein.pep est.fa
```

- compares a database of protein sequences to a database dna sequences - beware, this will take a while!

```
estwisedb -pfam Pfam est.fa
```

- compares a database of protein profile HMMs to a database of single sequences - beware, this will take a while

2.2 Common options to change

There are a number of common options that can be used. Options can be issued anywhere on the command line.

-help help on options

-version show version and build date (useful for bug reporting)

-quiet remove update line on stderr and informational messages

-silent suppress all messages to stderr

-report *number* for database searching, issue a report on stderr every *number* of comparisons (useful to ensure it is actually running)

-trev genewise and estwise - use the reverse strand of the DNA

-both genewise and estwise - use both strands of the DNA

-u position The start point in the DNA sequence for the comparison

-v position The end point in the DNA sequence for the comparison

-init [default/global/local/wing] (see section 4.3.3) For protein sequences the default is to be local (like smith waterman). For protein profile HMMs, the default is read from the HMM - the HMM carries this information internally. The global mode is equivalent to to the ls building option (the default in the HMMer2 package). The local mode is equivalent to to the fs building option (-f) in the HMMer2 package. The wing model is local on the edges and global in the middle.

-gene file change gene model parameters. Currently we have either human (human.gf) or worm (worm.gf)

-genes Output option for genewise algorithms - show an easy to read gene structure report

-trans Output option for genewise algorithms - provide an automatic translation of the predicted gene as a fasta format

-cdna Output option for genewise algorithms - provide an automatic construction of the spliced dna sequence as a fasta format

-ace Output option for genewise algorithms - provide an ACeDB subsequence model output

2.3 Common gripes, Cookbook and FAQ

2.3.1 It hasn't given me a complete gene prediction

The genewise algorithm does not attempt to predict an entire gene, from Met to STOP. It tries to predict regions which are justified with the protein homology and no more.

This does mean you can be confident of the predictions that genewise makes

2.3.2 How can I get rid of the annoying messages on stderr?

Some people like them. use -quiet

2.3.3 It goes far too slow

Well... I have always had the philosophy that if it took you over a month to sequence a gene, then 4 hours in a computer is not an issue. However, in particular for times when people are using genewise simply to confirm that the a gene prediction is correct with respect to a protein sequence (sometimes the notional translation!) it is taking too long. In many cases you will know the rough region to compare the sequence to - if so use the -u and -v options to truncate your DNA at the correct points (the output will remain in the coordinates of the full length sequence).

For database searching there is the option of using SMP boxes efficiently with the pthreads port.

There are also a number of heurisitcs that use the BLAST program to provide the speed. These heuristics are found in the perl/scripts directory, called halfwise and blastwise. The scripts have extensive installation instructions, and I completely expect people to edit them for their system.

There is functionality for providing a heurisitic bound to the space the algorithm explores in the alignment. This is done via the potential gene option in genewise. It is not well tested out.

2.3.4 I have a new cosmid. What do I do?

One thing to do is to use the halfwise script available in the perl/scripts package. Another is to use the blastwise script.

2.3.5 segmentation fault = bottle of champagne

You've found a bug? I am really keen to hear from you. I want to hear about the problems you've got. Each year I award my best tester with a prize. This year (1998/99) it will be a bottle of champagne. Send a mail to birney@sanger.ac.uk for your prize!

2.3.6 Can I modify or use the Wise2 source code?

Of course you can - it is Open Source code, licensed under the Gnu Public Licensed (GPL'd), like emacs or gcc. For more information on this License read the GNULICENSE file in the distribution.

As well as using the source code, you can if you like contribute directly back into the Wise2 source code. Get in contact with me if you would like to do this.

2.3.7 Making a single gene prediction on the basis of a close homolog

This is perhaps the easiest use of genewise. The basic formulation is

```
%genewise protein.fasta dna.fasta
```

To get out computer parsable formats of the gene prediction try -genes or -gff or -ace. To get out the protein translation in one go use -trans

2.3.8 Using non human/worm/fly genomic DNA

At the moment, genewise only has gene frequency files for human and worm sequences. The production of these files are based around somewhat annoying and non portable script. In any case, making a dataset requires alot of effort as it needs to be clean

The consequence of all this is that the species that you are comparing against (eg, hamster) may not have a gene frequency (.gf) file. In which case you basically have two options

- Use a close species - ie, for hamster, use human or rat
- Use -splice flat -intron tied which switches the splice model to "start at GT, finish at AG" with no other information

2.3.9 Working with non spliced (bacterial) genomic DNA

Use genewise with the -alg 333 or -alg 333L options. This has all the outputs of genewise but does not consider introns. The -gene option and -intron, -splice options are all pointless. The only options to worry about is the -subs and -indel for substitution and insertion and deletion errors respectively.

2.3.10 Working with ESTs

Use the estwise/estwisedb programs

2.3.11 Getting out the protein translation

You have three approaches for getting out protein translations

- -pep available on all programs, provides the translations moving over frameshifts and introns
- -trans available on genewise/genewisedb provides the translations across introns but breaks on frameshift errors. This means that the translations can be correctly placed on the genomic DNA provided
- -mul available only on estwisedb when a HMM is used, provides a protein multiple alignment of all the DNA hits derived against the HMM match

2.3.12 Using Pfam

Pfam can be used with the genewisedb or the estwisedb program with the -pfam flag. Usually you want to also use the -dnas (single DNA sequence flag) as well. An example run would be

```
genewisedb -pfam Pfam -dnas myseq.fa
```

If you have set up the HMMER package to work with Pfam using the enviroment variable HMMERDB, Wise2 will also pick that up as well.

2.3.13 Optimising alignment speed

Wise2 assummes you have a rather small amount of memory (20 MBytes). When it is making an alignment, if it cannot make the explicit matrix in that size (being length of query × length of target × state number) it has to move to linear memory (length of query × state number). The linear memory is much slower (it is the one that starts with “Find start end points”).

If you have more memory than 20 Mbytes, then it is really sensible to up the number, using the -kbyte option. For a machine with say 64Mbytes physical memory I would suggest putting an upper limit of 50Mbytes with -kbyte. This does assumme you are not using it for anything else.

You can change the compile time default in basematrix.h if you can’t be bothered to remember to change it every time

2.3.14 Optimising search speed

Make sure you have compiled with optimisation. If you are using the make all from the top level you have.

If you have a large SMP box, you can compile with pthread support. The searches work on SGI/Compaq alpha/Suns. There are some issues about some architecture ports, which I need to expand somewhere in the docs, but first off, just try compiling with pthreads (see section later) and using pthreads in the search.

For real, order-of-magnitude speed ups, you are going to have to use a heuristic stage before the actual database search - in other words, using BLAST. I dislike this, but it is fact of life, and there are two scripts in perl/scripts, halfwise and blastwise, which help you do this. Both scripts use Steve Chervitz excellent perl Blast parser, which is available in bioperl.

- halfwise is for the Pfam search. You need to pick up the halfwise database (done for a specific release of Pfam) from the ftp site.
- blastwise is for post processing blast results. It uses the Wise2 perl port to do this, so you have to go make perl at the top level

halfwise is a pretty sensible, self contained script. blastwise I expect people to modify heavily to get to work as wished on their systems. Please read it, and add in your own heuristics (eg, figuring out start/end points). I am very interested in better heuristics in this area.

3 Installation

Installation is quite easy as long as you are au fait with standard UNIX utilities. You should ftp to ftp.sanger.ac.uk, log in as anonymous and move to pub/birney/wise2. You can then pick up the release - I would pick up the latest numbered in that directory. (NB, if you want to be working in the development release, go to the pub/birney/wise2/alpha directory, but be sure to read the html help at http://www.sanger.ac.uk/Software/Wise2/Programming).

3.1 Building the executables

The release is distributed as a gzipped, tar file. To unzip and untar in a single command you can type

```
%zcat wise2.1.12b.tar.gz | tar -xvf -
```

This will untar into a directory called 'wise2.1.12b' (of course, your version of Wise2 might be different).

Once you have made the tar file, it should build completely cleanly as long as you have an ANSI C compiler. If in doubt, just assumme that it is, but in particular sun users might want to use gcc (gnu cc) as the sun cc compiler installed by default is often non-ANSI. To change the cc compiler you only need to edit the line in the top level makefile called CC = cc to CC = gcc.

To build the package type

```
%cd wise2.1.12b
%make all
%make bin
```

The executable files will now be in wise2.1.12b/bin

I am interested in all compiler errors, and consider most of them to be bugs (which means if you report them you could be on the champagne list!)

3.2 Environment set up

The Wise2 package needs to know where a number of files are (eg, the gene predicition statistics). These files are in the directory called wisecfg/. You will need to setenv WISECONFIGDIR to this directory (you can of course move the directory elsewhere, and set WISECONFIGDIR to it).

3.3 Building with thread support (for SMP machines)

To build with pthread support you must switch on some extra compile time options before you type make all. These are found at the top of the makefile in the top directory, and it is pretty clear from the makefile what to do. See the section 5.5 for information on how to run pthreaded code.

3.4 Building Perl port

To build with Perl support you need to go

```
make perl
```

at the top level. This should build everything correctly. The only problem is if you have a Solaris or *BSD box. If so you need to compile with -fpic or -fPIC depending on your compiler. This needs to go into the top level CFLAGS line. In addition, in the out-of-the box perl distribution for solaris they built it with a different compiler to the one it comes with (idiots!), so the perl generated makefile has the wrong -fpic option. You need to edit that by hand.

4 Concepts and conventions

The algorithms used in Wise2 have a strong theoretical justification, which is useful, though not necessary to understand. For example to understand what most of the options do in the gene model part of genewise you need to understand the algorithm.

4.1 Technical Approach

You can miss this section which describes some of the theoretical background of the work. The algorithms are based around a 'Bayesian' formalism that has been established in Bioinformatics by such people as David Haussler, Gary Churchill, Anders Krogh, Richard Durbin, Sean Eddy and Graeme Mitchinson, as well as many others. In this formalism there is assumed to be a generative model of the process that you are observing, which has probabilities to generate a number of different observations. Deciding whether this model fits a previously unseen piece of data or not is the first decision to make. Given that the data fits, a second question is what actual processes were the most likely to produce the observed data. Both these questions fit naturally into a Bayesian framework where the result is a posterior probability having seen the data.

For people coming from a bioinformatics/biology background where the last paragraph may seem very confusing, it is only because this a different (and well established) field with their own terminology to describe the algorithms. In fact the methods a very close to standard techniques presented in bioinformatics. The generative models that we use are the models that are implied by the standard bioinformatics tools. For example, the Smith-Waterman algorithm implies a process of evolution with certain probabilities for seeing say an Leucine to Valine substitution and certain probabilities for creating and extending a insertion (gap). As you can see you can almost replace the word 'probability' with 'score' to return to the standard method, and mathematically it is almost that easy: the score is related to the log of the probability.

Perhaps a better known example is the relationship between the old profile technology, as developped by Gribskov and Gibson along with others, and its probabilistic partner, profile Hidden Markov Models (profile HMMs). In terms of the actual algorithm these two methods are very similar: it is simply that the profile HMM has a strong probabilistic model underlying it, allowing well established techniques to be used in its generation.

4.2 Introduction to Models in Wise2

Wise2 contains a number of algorithms, each of which are based around one of two biological models.

genewise comparison of a related protein to genomic DNA

estwise comparison of a related protein to cDNA (or ESTs)

This models themselves are built up from two component models, one for how protein residues are matched, and one for the gene prediction process. For the model of protein residues I have taken the established models of profile HMMs. The model of splicing and translation we developed with an eye to biology. It has many of the features of the GenScan model [chris Burge]. The model of translation (for estwise) is simple.

4.3 Model

The main model to understand is the genewise model (called genewise 21:93 for reasons discussed below). It is this model which the other models are based on - for the estwise models, by removing the intron generating part of the models, and for the other genewise algorithms by making approximations to genewise21:93.

A diagramatic representation of genewise21:93 is shown in the file genewise21.ps

The central part of the model is the Match-Insert-Delete trio common to both profile HMMs (such as HMMER models) and the smith waterman model. This trio of states is one model 'position' in the profile HMMs, where each model position contains a Match, Insert and Delete states. This means to interpret the figure of the model in the way the profile HMM models are usually displayed, you have to imagine a series of these states concatonated together. I imagine the model growing as stack of pages out from the figure, each new page being a new position in the profile HMM.

The first addition to the model are the frameshifting transitions, shown in with x4 boxes above them. These occur whenever there is a transition which produces a codon: in effect all transitions that terminate at either match or insert states. There are four frameshifting transitions in each Notice that there are frameshifting transitions from Delete to Match, which is equivalent to saying that a frameshift occurs on the codon just after a run of deletions in the model. It is these sorts of frameshifts that are not well modelled by other algorithms.

The second addition involves the intron emitting states found in the green boxes. Each intron is modelled by having 5 regions, two of which are fixed length. The five regions are

- 5'SS The splice site consensus region at the 5' end of the intron. Fixed length
- The central part of the intron that constitutes the major part of the intron
- The polypyrimidine tract (a region of C/T bias upstream of the 3'SS)
- an optional joining region between the poly-py tract and the 3'SS
- 3'SS The splice site consensus region at the 3' end of the intron. Fixed length

Notice that there is no branch site, because we could not produce a good enough statistical model for it.

This model can be modelled using 3 states, with the fixed length regions being accommodated using transitions which emitted the appropiate length of sequence.

Each of the intron models must be duplicated 3 times to account for the 3 different phases of introns (each phase being a different placement of the intron relative to the codon), so we need to duplicated these 3 states at least 3 times. In addition, if this intron lies in an insert state, ie, the surrounding protein sequence in the exons are being produced by an insert state in the underlying protein profile HMM, so we have to maintain that information across the intron. This means that we need to duplicate the intron states 6 times in total: 3 times for the different phases and twice on top of that for the different protein states this intron could lie in.

4.3.1 Parameterisation of the model

The model presented above seems biological sensible, but how on earth are we going to parameterise it? Are we honestly going to let a user try to juggle the forty odd parameters inherent to this model? Clearly not. The approach we have taken to this is to provide set statistics derived from a maximum likelhood approach from known genes - this requires virtually no training - and then give switches to the user to turn on and off a variety of different parts of the algorithm.

The model is parameterised as probabilities, but actually calculated in log space. If you look in the code you would find that there is alot of switching between the two spaces: these are provided by the functions Probability2Score and Score2Probability (notice that the 'Score' here is very specific to the Wise2 package - you can't put any old score into Score2Probability to get a probability out as it depends on how that Score was converted into Log space).

4.3.2 The protein model

For the emissions of the actually underlying amino acids when we have a profile HMM, we are lucky - we can take the probabilies defined in the HMMer2 models. This is completely natural and means I don't have to worry about deriving probabilities for the profile HMMs

In the case where we have a protein sequence, I somehow have to get to a profile HMM type representation. Thankfully the smith waterman algorithm in terms of architecture is very close to a profile HMM, and so the only problem is mapping the usual scores used in the smith waterman algorithm to probabilites. This is quite hard to do correctly, but I've hacked it by knowing that the blosum62 matrix is given in half bits, in other words using a 2*log2 mapping from probability space to the give scores in the matrix. By reversing this process one can get pretty good emission probability for the amino acids. I now assumme that the gap penalities are *as if* they were written in half bits. A certain amount of normalisation is required to make sure things add to one, and eh voila - one profile HMM from a single sequence.

4.3.3 Start End points

One interesting issue about the protein model is how the start end points work. For proteins it is obvious that for distant homology, it needs to be local - ie can start or finish anywhere in the sequence. For protein HMMs it is less clear. If a HMM really represents a single domain then global start end points are correct. However, many times local start end points are useful.

The HMMer2 models internally carry whether this HMM is has global or local (or indeed any type) of start end policy.

However, the genewise algorithm is quite dependent on the models being global to effectively predict introns in domains, when the looping algorithm (multiple copies of the domain) is present. This is because nearly always in a local HMM, an intron can be better modelled as the end of the domain half way through and the start of a new domain half way through, further down the sequence, thus not predicting the intron. To get clean intron prediction, one needs to go to global mode. However, using global mode forces the start and end point of the model to be really correct, and in some cases (in particular some Pfam models) this makes very incorrect results on the edges of the domain. To combat this another type of start end policy is introduced - wing. This has a local start mode for the first 15 model positions and end mode for the last 15 model positions, but global in the central part of the model.

In the programs one can set four types of start end policy

- default local for protein, and the HMM default for HMMs
- local local
- global global
- wing local on the edges, global in the middle

4.3.4 The gene model

For the emissions of the gene model we had to do more work. What we did was to make a database of known genes, with annotated gene structure. These genes then provided a raw set of counts for particular parts of the gene structure. It is these raw counts which are stored in the .gf files. (we store the raw counts because one might want to do something clever for deriving the probabilities of certain things using these counts. Counts are the basis for the probability derivations, not frequencies).

The only issue here is what to do with the splice sites. We were well aware that the information in the splice sites is considerably more than just the simple position matrix. We chose to use a single branching (biased) decision tree, in which each branch either carried along the main trunk of the tree or ended in a leaf, each leaf representing a consensus build from A,T,G,C or N for any character. This decision tree could be easily constructed by chosing the most common consensus (where N is allowed where a position is better represented by N than any specific residue), and then removing that consensus from the list of observed consensi, and then repeating the process. This also gave us the same basis (counts) for each consensus used in the splice sites.

One additional twist came about in the splice site development. The splice sites overlap between their consensi and the coding sequence region. These overlaps need to be treated correctly: the problem is that probabilistically we have two processes wanting to account for the same DNA bases. This was solved by assumming conditional independence between the two processes. A more formal mathematicall approach can be found in the documented called 'probappendix'.

4.3.5 The NULL model

The probability of the model has to compared to an alternative model (in fact to all alternative models which are possible) to allow proper Bayesian inference. This causes considerable difficulty in these algorithms because from a algorithmical point of view we would probably like to use an alternative model which is a single state, like the random model in profile-HMMs, where we can simply 'log-odd' the scored model, whereas from a biological point of view we probably want to use a full gene predicting alternative model.

In addition we need to account for the fact that the protein HMM or protein homolog probably does not extend over all the gene sequence, nor in fact does the gene have to be the only gene in the DNA sequence. This means that there are very good splice sites/poly-pyrimidine tracts outside of the 'matched' alignment can severely de-rail the alignment.

Basically we are in trouble with the random model parts of this problem.

The solutions is different in the genewise21:93 compared to the genewise 6:23 algorithms

- In 6:23 we force the external match portions of the homology model to be identical to the alternative model, thus cancelling each other out. This is a pretty gross approximation and is sort of equivalent to the intron tie'ing. It makes things algorithmically easier... However this means a) 6:23 is nowhere near a probabilistic model and b) you really have to used a tied intron model in 6:23 otherwise very bad edge effects (final introns being ridiculously long) occur.

- In 21:93 we have a full probabilistic model on each side of the homology segment. This is not reported in the -pretty output but you can see it in the -alb output if you like. Do not trust the gene model outside of the homology segment however. By having these external gene model parts we can use all the gene model features safe in the knowledge that if the homology segments do not justify the match then the external part of the model will soak up the additional intron/py-tract/splice site biases.

However this still does not solve the problem about what to compare it to. There are two approaches to the comparison

flat The homology model is scored against a single state 0.25 emission model. This is effectively 'how likely is this DNA segement has any genes some with this homologous protein/HMM in it' for 21:93. It is, unsurprisingly, a massive 'yes' for nearly all biological DNA, and though a valid number in terms in bayesian inference pretty biologically uninteresing. There is also no decent interpretation of partial scores (ie, scores per domain).

syn For synchronous model pretends that there is an alternative model of a complete gene which is dragged into the coding part of the gene when the homology model is in the coding part. This is not probabilistically valid, but gives better results and interpretable scores for partial regions, ie domain by domain. (in fact, very similar scores to protein sequences). However I'm worried about what I am doing It would be much better to get some mathematically justification for this.

4.4 Algorithms

The algorithms are then based around this central model, but have a variety of features removed from it progressively, either due to biological constraints (bacterial sequences have no introns, so there is no need to model them) or to speed up the the algorithm.

Algorithms are named in two parts, *descriptive-word state-number:transition-number*. The descriptive word indicates the *biological* model. At the moment there are 2 such biological models in the package

genewise comparisons of protein information to genomic DNA

estwise comparisons of protein information to cDNA/bacterial DNA (no introns)

There are many other models being worked on in development

sywise comparisons of genomic DNA to genomic DNA

parawise comparions of cDNA to cDNA

The *state-number:transition-number* is the number of states in the model followed by the number of transitions. GeneWise 21:93 is the most complicated model, with 21 states and 93 transitions. The number of states is directly proportional to the memory usage of the program. The number of transitions is roughly proportional to the CPU time of the algorithm. For comparison the standard smithwaterman algorithm is a 3:7 algorithm (3 states, 7 transitions). These numbers are per compared residue - so as genomic DNA is some 1,000 fold longer than protein sequences on average, there is an additional massive CPU load.

Finally the algorithms can be looping or not. A Looping algorithm is one in which the protein information can be repeated in the DNA target sequence. This could either be due to mutliple copies of the gene in the DNA sequence or multiple copies of a domain in a single gene. Looping algorithms are given a 'L' tag. By default, when you use profile-HMMs you use a looping model

For the genewise family the following algorithms are available.

genewise 21:93 The largest genewise algorithm which also contains a complex flanking model to prevent inappropiate gene predictions

genewise 21:93L The same algorithm with a looping mode. This allows a protein HMM (nearly always a HMM) to match multiple times a DNA sequence. This could be due to multiple domains in a single gene or multiple genes in a DNA sequence with the domain. The algorithm doesn't distinguish between these possibilities.

genewise 6:23 This is a smaller, (and so faster) algorithm. The approximations made compared to genewise 21:93 are that there is no polypyrimidine tract in the intron, and that introns from match states are not distinct from introns in insert states.

A side effect of these approximations is that 6:23 is much more robust with respect to unmasked repeats and strange composition effects found in the DNA sequences.

genewise 6:23L The same algorithm as 6:23 but in looping mode

genewise 4:21 The smallest algorithm in the genewise family, with an additional approximation of not distinguishing between introns of different phases. This has been compiled for short protein sequences only - effectively only profile-HMMs.

For the estwise family the following algorithms are available

estwise 3:33 The largest estwise algorithm, modelling potential insertion or deletions throughout the alignment of the protein information to the DNA sequence.

estwise 3:33L The same algorithm but in looping mode.

estwise 3:12 A slimmer algorithm designed for faster db searching. The algorithm models enough insertions or deletions of DNA bases to 'ride through' a indel region without too much penalty, even if it doesn't model the most correct one.

4.5 Scores

The scoring system for the algorithms, as eluded to earlier is a Bayesian score. This score is related to the probability that model provided in the algorithm exists in the sequence (often called the posterior). Rather than expressing this probability directly I report a log-odds ratio of the likelhoods of the model compared to a random model of DNA sequence. This ratio (often called *bits score* because the log is base 2) should be such that a score of 0 means that the two alternatives *it has this homology* and *it is a random DNA sequence* are equally likely. However there are two features of the scoring scheme that are not worked into the score that means that some extra calculations are required

- The score is reported as a likelhood of the models, and to convert this to a posterior probability you need to factor in the ratio of the prior probabilities for a match. Because you expect a far greater number of sequences to be random than not, this probability of your prior knowledge needs to be worked in. Offhand sensible priors would in the order of probability that there is a match being roughly proportional to the database size.

- The posterior probability should not merely be in favour of the homology model over the random model but also be confident in it. In other words you would want probabilities in the 0.95 or 0.99 range before being confident that this match was correct.

These two features mean that the reported bits score needs to be above some threshold which combines the effect of the prior probabilities and the need to have confidence in the posterior probability. In this field people do not tend to work the threshold out rigorously using the above technique, as in fact, deficiencies in the model mean that you end up choosing some arbitary number for a cutoff. In my experience, the following things hold true: bit scores above 35 nearly always mean that there is something there, bit scores between 25-35 generally are true, and bit scores between 18-25 in some families are true but in other families definitely noise. I don't trust anything with a bit score less than 15 bits for these DNA based searches. For protein-HMM to protein there are a number of cases where very negative bit scores are still 'real' (this is best shown by a classical statistical method, usually given as evalues, which is available from the HMMer2 package), but this doesn't seem to occur in the DNA searches.

I have been thinking about using a classical statistic method on top of the bit score, assumming the distribution is an extreme value distribution (EVD), but for DNA it becomes difficult to know what to do with the problem of different lengths of DNA. As these can be wildly different, it is hard to know precisely how to handle it. Currently a single HMM compared to a DNA database can produce evalues using Sean Eddy's EVD fitting code but, I am not completely confident that I am doing the correct thing. Please use it, but keep in mind that it is an experimental feature.

5 Principle Programs

The main programs are genewise, genewisedb, estwise, estwisedb. These all have basically the same running mode

```
%genewise protein-file dna-file
```

A number of options are common to these programs from the point of view of how they run

-help verbose help of all options

-version show version and compile info

-silent No messages on stderr, whether reports or warnings

-quiet No reports or information messages on stderr

-erroroffstd No warning messages to stderr, but reports are still issued

-errorlog [file] Log warning messages to file (useful for sending to me)

You will probably want to read the 2.1 common modes of usage section as well

5.1 genewise

Genewise compares a protein sequence or a protein profile HMM to a dna sequence

5.1.1 genewise - options: dna/protein

-u start position in dna

-v end position in dna

-trev Compare on the reverse strand

-tfor (default) Compare on the forward strand

-both Both strands

-tabs Report positions as absolute to truncated/reverse sequence

-s start position in protein - has no meaning for HMMs

-t end position in protein - has no meaning for HMMs

-gap [no] default [12] gap penalty to use for protein comparisons. This is used to estimate a probability per gap

-ext [no] default [2] extension penalty to use for protein comparisons. This is used to estimate a probability for an extension of a gap

-matrix default [blosum62.bla] Comparison matrix. Must be in half-bit units (blosum62 is in half bits). This is used to estimate a probability of amino acid comparisons

-hmmer Protein file is HMMer 2 HMM

-hname Use this as the name of the HMM.

-init [default/global/local/wing] (see section 4.3.3) For protein sequences the default is to be local (like smith waterman). For protein profile HMMs, the default is read from the HMM - the HMM carries this information internally. The global mode is equivalent to to the ls building option (the default in the HMMer2 package). The local mode is equivalent to to the fs building option (-f) in the HMMer2 package. The wing model is local on the edges and global in the middle.

5.1.2 genewise - options: gene model

-codon [codon.table] Codon file. The default is for the universal code, but you can supply your own

-gene [human.gf] Gene parameter file. Provide statistics for different gene models. Current human.gf and worm.gf are provided. The statistics are basically too complicated to explain here.

-subs [1e-05] Substitution error rate, ie the assummed probability of base substitutions in the sequencing reaction/assembly that provided the DNA sequence. The substituion error is what dominates the penalty for stop codons - a higher error rate implies a smaller penalty for stop codons

-indel [1e-05] Insertion/deletion error rate, ie the assummed probability of indel events in the sequencing reaction/assembly that provided the DNA sequence. The indel rate is what provides the penalty for frameshift errors. A higher error rate implies a smaller penalty for indels.

-cfreq [model/flat] Using codon bias or not? [default flat] - a reasonably pointless option now, as it only applies when using -syn flat. If codon bias is modelled, then common codons score more than uncommons one for the same amino acid.

-splice [model/flat] Using splice model or GT/AG? [default model] - use the full blown model for splice sites, or a simplistic GT/AG. Generally if you are using a DNA sequence which is from human or worm, then leave this on. If you are using a very different (eg plant) species, switch it off.

-intron [model/tied] Use tied model for introns [default tied] - whether intron base distribution effects the parse. Because varying GC content and/or repeats can seriously drag the algorithm away from correct parses when intron base distribution is used, this is usually switched off.

-null [syn/flat] Random Model as synchronous or flat [default syn] - whether to use a null model which is a simple base distribution (called flat), or imagine that the viterbi path is being compared to a gene based null model that is making all the same gene exon/intron boundaries (synchronous). The latter is basically a hack which demphaises the gene prediction machinery and tries to trust the homology machinery. (not ideal!)

-pg [file] Potential Gene file (heurestic for speeding alignments). The potential gene file should look like

```
pgene # stands for potential gene
ptrans # stands for potential transcript
pexon <start-in-dna> <end-in-dna> <start-in-protein> <end-in-protein>
pexon <start-in-dna> <end-in-dna> <start-in-protein> <end-in-protein>
...
endptrans
<another ptrans if you like>
endpgene
```

When this file is read in, it provides a series of start/end in dna and protein sequences around which is drawn an envelope of possibly alignment area. The alignment is then calculated only in this area

This feature has not been well tested yet. any potential bugs reported in are very useful.

-alg [623/623L/2193/2193L] Algorithm used [default 623/623L] You should read the section on algorithms (4.4). Basically 623 and 623L are cheaper computationally and more robust with respect to repeats etc. 2193 and 2193L are much more expensive, more sensitive to changes in parameters but potentially more accurate.

-kbyte [2000] Max number of kilobytes used in main calculation. Indicates how much memory can be used for the dynamic programming calculation.

5.1.3 genewise - options: output

All output options can be used at the same time. They are separated by the value to -divide option

-pretty show pretty ascii output, as see in Section 2

-pseudo For genes with frameshifts, mark them as pseudo genes

-genes show gene structure - as

```
Gene 1
  Gene 1386 3963
    Exon 1386 1493
    Exon 1789 1935
    Exon 2084 2294
    Exon 2388 2480
    Exon 2794 2868
    Exon 3073 3228
    Exon 3806 3963
//
```

-para show parameters

-sum show summary output. Shows output as

```
Bits   Query         start end Target      start end   idels introns
230.57 roa1_drome     26   347 HSHNRNPA     1386 3963    0    6
```

This is useful for parsing, but probably if you want to do something like that you want to get hold of the API directly.

-cdna show cDNA Show a fasta format of the predicted cDNA sequence

-trans show protein translation Show a fasta format of the predicted protein sequence. Breaks on frameshifts

-pep show predicted peptide. Shows predicted peptide, including frameshifts, which are X's in the proteins

-ace ace file gene structure - ACeDB subsequence model

```
Sequence HSHNRNPA
subsequence HSHNRNPA.1 1386 3963

Sequence HSHNRNPA.1
CDS
CDS_predicted_by genewise 0.00
source_Exons 1 108
source_Exons 404 550
source_Exons 699 909
source_Exons 1003 1095
source_Exons 1409 1483
source_Exons 1688 1843
source_Exons 2421 257
```

-gff Gene Feature Format file - useful for programs which also support GFF

```
HSHNRNPA GeneWise cds_exon 1386 1494 0.00 + 0
HSHNRNPA GeneWise cds_exon 1789 1936 0.00 + 0
HSHNRNPA GeneWise cds_exon 2084 2295 0.00 + 0
```

-gener raw gene structure - a debugging output

-alb show logical AlnBlock alignment - a debugging output

-pal show raw matrix alignment - a debugging output

-block [50] Length of main block in pretty output

-divide [//] divide string for multiple outputs

5.2 genewisedb

genewisedb is the database searching version of genewise. It takes a database of proteins and compares it to a database of dna sequences

5.2.1 genewisedb - search modes

-protein [default] single protein. Protein is a single protein sequence in fasta format

-prodb protein fasta format db. Protein is a database of protein sequences in fasta format

-pfam pfam hmm library. Protein is a database of HMMer2 models as a single file

-pfam2 pfam old style model directory (2.1). Protein is a directory of HMMs with a file called HMMs in it indicating which HMMs there. This is how Pfam databases 2.1 and lower were distributed

-hmmer single hmmer HMM (version 2 compatible). Protein is a single HMM

-dnadb [default] dna fasta database. The DNA sequence is a fasta format file with multiple sequences

-dnas a single dna fasta sequence. The DNA sequence is a single sequence in fasta format

5.2.2 genewisedb - protein comparison options

-gap [12] gap penalty - see genewise option

-ext [2] extension penalty - see genewise option

-matrix [blosum62.bla] Comparison matrix - see genewise option

-hname For single hmms, use this as the name, not filename

5.2.3 genewisedb - gene model options

Many of these options are identical to the genewise options listed above

-init [default/global/local/wing] (see section 4.3.3) For protein sequences the default is to be local (like smith waterman). For protein profile HMMs, the default is read from the HMM - the HMM carries this information internally. The global mode is equivalent to to the ls building option (the default in the HMMer2 package). The local mode is equivalent to to the fs building option (-f) in the HMMer2 package. The wing model is local on the edges and global in the middle.

-codon [codon.table] Codon file -see genewise option

-gene [human.gf] Gene parameter file - see genewise option

-subs [1e-05] Substitution error rate - see genewise option

-indel [1e-05] Insertion/deletion error rate - see genewise option

-cfreq [model/flat] Using codon bias or not? [default flat] - see genewise option

-splice [model/flat] Using splice model or GT/AG? [default model] - see genewise option

-intron [model/tied] Use tied model for introns [default tied] - see genewise option

-null [syn/flat] Random Model as synchronous or flat [default syn] - see genewise option

-alg [421/623/2193/] Algorithm used for searching [default 623] The is the algorithm to use for the database search part of the process. 421 is the cheapest algorithm but can only be used with HMMs or small proteins as it has been compiled for a limited size of query. Looping algorithms (623L and 2193L) are not permitted as it is hard to interpret the results

-aalg [623/623L/2193/2193L] Algorithm used for alignment [default 623/623L] This is the algorithm used for the alignment of the matches. The default for proteins is 623, whereas for HMMs it is the looping model 623L.

-kbyte [2000] Max number of kilobytes used in alignments calculation. Maximum amount of memory allowed in the alignment process.

-cut [20.00] Bits cutoff for reporting in search algorithm. Comparisons scoring greater than this cutoff are aligned.

-ecut [n/a] Evalue cutoff only for searches which can calculate evalues

-aln [50] Max number of alignments (even if above cut). A cutoff for the number of alignments, whatever their bits score.

-nohis Don't show histogram on single protein/hmm vs DNA search. On a single protein (or hmm) vs DNA database search an on-the-fly evalue score is calculated. This disables the production of a histogram

-report [0] Issue a report every x comparisons (default 0 comparisons). Mainly for debugging

5.2.4 genewisedb output - for each comparison

For each alignment made by genewisedb you can output it as a number of different options

-pretty show pretty ascii output, as in genewise

-pseudo For genes with frameshifts, mark them as pseudo genes

-genes show gene structure, as in genewise

-para show parameters, as in genewise

-sum show summary output, as in genewise

-cdna show cDNA, as in genewise

-trans show protein translation, as in genewise

-ace ace file gene structure, as in genewise

-gff Gene Feature Format file, as in genewise

-gener raw gene structure, as in genewise

-alb show logical AlnBlock alignment, as in genewise

-pal show raw matrix alignment, as in genewise

-block [50] Length of main block in pretty output, as in genewise

-divide [//] divide string for multiple outputs, as in genewise

5.2.5 genewisedb output - complete analysis

Each alignment produces a notional gene prediction. At the end of the output, these gene predictions can be displayed together. This only works for -pfam or -prodb and -dnas options, ie a database of protein information vs a single dna sequence

In the future it is hoped that additional options (such as merging consistent gene predictions) will operate before these outptus are made

-ctrans provide all translations

-ccdna provide all cdna

-cgene provide all gene structures

-cace provide all gene structures in ace format

5.3 estwise

Estwise runs very much like genewise with basically a subset of options. For completeness they are all listed below

5.3.1 estwise - options: dna/protein

-u start position in dna

-v end position in dna

-trev reverse complement dna

-tfor use forward strands only

-both [default] do both strands

-tabs Positions reported as absolute to DNA

-s start position in protein

-t end position in protein

-gap [12] gap penalty

-ext [2] extension penalty

-matrix [blosum62.bla] Comparison matrix

-hmmer Protein file is HMMer 1.x file

-hname Name of HMM rather than using the filename

5.3.2 estwise - options: model

-init [default/global/local/wing] (see section 4.3.3) For protein sequences the default is to be local (like smith waterman). For protein profile HMMs, the default is read from the HMM - the HMM carries this information internally. The global mode is equivalent to to the ls building option (the default in the HMMer2 package). The local mode is equivalent to to the fs building option (-f) in the HMMer2 package. The wing model is local on the edges and global in the middle.

-codon [codon.table] Codon file. The default is for the universal code, but you can supply your own

-subs [0.01] Substitution error rate, ie the assummed probability of base substitutions in the sequencing reaction/assembly that provided the DNA sequence. The substituion error is what dominates the penalty for stop codons - a higher error rate implies a smaller penalty for stop codons

-indel [0.01] Insertion/deletion error rate, ie the assummed probability of indel events in the sequencing reaction/assembly that provided the DNA sequence. The indel rate is what provides the penalty for frameshift errors. A higher error rate implies a smaller penalty for indels.

-null [syn/flat] Random Model as synchronous or flat [default syn] whether to use a null model which is a simple base distribution (called flat), or imagine that the viterbi path is being compared to a gene based null model that is making all the same gene exon/intron boundaries (synchronous). The latter is basically a hack which demphaises the placement of frameshifts and tries to trust the homology machinery. (not ideal!)

-alg [333,333L,333F] Algorithm used. 333 is the normal algorithm. 333L is the looping algorithm

-kbyte [2000] Max number of kilobytes used in main calculation

-pretty show pretty ascii output as in genewise

-para show parameters

-sum show summary information as in genewise

-alb show logical AlnBlock alignment, debugging output

-pal show raw matrix alignment, debugging output

-block [50] Length of main block in pretty output - the length of the main text in the pretty output

-divide [//] divide string for multiple outputs, the string used to separate multiple outputs

5.4 estwisedb

estwisedb is the database searching version of the estwise program. Like estwise, it has the same sort of running modes as genewisedb, but with more limited options.

5.4.1 estwisedb - options: running modes

-protein [default] single protein

-prodb protein fasta format db

-pfam pfam hmm library

-pfam2 pfam style model directory (2.1)

-hmmer single hmmer 1.x HMM

-dnadb [default] dna fasta database

-dnas a single dna fasta sequence

5.4.2 estwisedb - options: model

-gap [12] gap penalty

-ext [2] extension penalty

-matrix [blosum62.bla] Comparison matrix

-hname For single hmms, use this as the name, not filename

-codon [codon.table] Codon file

-subs [0.01] Substitution error rate

-indel [0.01] Insertion/deletion error rate

-null [syn/flat] Random Model as synchronous or flat [default syn]

-alg [333/] Algorithm used for searching [default 333]

-aalg [333/333L] Algorithm used for alignment [default 623]

-kbyte [2000] Max number of kilobytes used in alignments calculation

-cut [20.00] Bits cutoff for reporting in search algorithm

-ecut [n/a] Evalue cutoff only for searches which can calculate evalues

-aln [50] Max number of alignments (even if above cut)

-nohis Don't show histogram on single protein/hmm vs DNA search

-report [0] Issue a report every x comparisons (default 0 comparisons)

5.4.3 estwisedb - options: output

-pretty show pretty ascii output

-para show parameters

-sum show summary output

-alb show logical AlnBlock alignment

-pal show raw matrix alignment

-mul produce complete protein multiple alignment from a HMM to DNA db search as a mul format M/A.

-pep show predicted peptide. Shows predicted peptide, including frameshifts, which are X's in the proteins

-block [50] Length of main block in pretty output

-divide [//] divide string for multiple outputs

-help help

-version show version and compile info

-silent No messages on stderr

-quiet No report on stderr

-erroroffstd No warning messages to stderr

-errorlog [file] Log warning messages to file

5.5 Running with pthreads

The two database searching programs, genewisedb and estwisedb can be run with pthread support on SMP boxes. To do so you need to compile the source code with pthread support (it is very easy, see section 3.3). Then the programs need to be run with the additional option `-pthread`. On most machines the executable will pick up the number of available processors automatically and run that number of threads. If you want to override this use the `-pthr_no` option.

6 Other Programs

There are other programs in the wise2 package which are not as well developped as the *wise set of programs. These programs are more an indication of how fast it is to develop algorithms sensibly in the Wise2 environment than anything else.

6.1 dba - Dna Block Aligner

dba - standing for Dna Block Aligner, was developped by Niclas Jareborg, Richard Durbin and Ewan Birney for characterising shared regulatory regions of genomic DNA, either in upstream regions or introns of genes

The idea was that in these regions there would a series of shared motifs, perhaps with one or two insertions or deletions but between motifs there would be any length of sequence.

The subsquent model was a 3 state model which was log-odd'd ratio to a null model of their being no examples of a motif in the two sequences.

6.1.1 dba - options

-match [0.8] match probability

-gap [0.05] gap probability

-blockopen [0.01] block open probability

-umatch [0.99] unmatched gap probability

-nomatchn do not match N to any base

-align show alignment

-params print parameters

-help print this message

6.2 psw - Protein Smith-Waterman and other comparisons

psw is a short and sweet program for calculating smith waterman alginments quickly. It was mainly written as C driver to test the underlying code which is more useful in things like the Perl port.

More recently I added in the generalised gap penalty model of Stephen Altschul, that is known as the *abc* model in Wise2. The abc model is detailed in Proteins 1998 Jul 1, 32 pages 88-96.

6.2.1 psw - options

-g gap penalty (default 12) - gap penalty used for smith waterman

-e ext penatly (default 2) - ext penalty used for smith waterman

-m comp matrix (default blosum62.bla) - comparison matrix used for both smith waterman and the abc model

-abc use the abc model: use Stephen Altschul's 'generalised gap penalty' model (called the abc model in Wise2)

-a a penalty for above (default 120) gap opening penalty in the abc model

-b b penalty for above (default 10) gap extension penalty in the abc model

-c c penalty for above (default 3) unmatched 'gap' region penalty in the abc model

-r show raw output - raw matrix output

-l show label output - label based output

-f show fancy output - pretty output

6.3 pswdb

pswdb - protein smith waterman database searching was written by Richard Copley using the underlying Wise2 libraries

6.3.1 psw - options

-g gap penalty (default 12) - gap penalty used for smith waterman

-e ext penatly (default 2) - ext penalty used for smith waterman

-m comp matrix (default blosum62.bla) - comparison matrix used for both smith waterman and the abc model

-abc use the abc model: use Stephen Altschul's 'generalised gap penalty' model (called the abc model in Wise2)

-a a penalty for above (default 120) gap opening penalty in the abc model

-b b penalty for above (default 10) gap extension penalty in the abc model

-c c penalty for above (default 3) unmatched 'gap' region penalty in the abc model

-max_desc Maximum number of description lines

-max_aln Maximum number of alignments

-ids in alignments, show sequence names, not probe/target

-r show raw output - raw matrix output

-l show label output - label based output

-f show fancy output - pretty output

7 API

This section is really only an introduction to the API. There is another, separate documentation on the API with a complete reference of all the functions etc.

If you end up parsing the programs in the Wise2 package alot, or repeatedly calling them to do something slightly at odds to the way they work, then you should probably be using the API. The API is quite easy to use once you have got used to the number of functions that you can call: all the hard parts of writing a C program, such as the underlying algorithms and memory management are conviently hidden from you.

The API (application programming interface) is a defined layer for you to write programs that use Wise2 functionality. The API has only a subset of the functions available internally to Wise2 (but still it is quite a daunting number). Currently there are two main ways to access the API - firstly using C function calls, and secondly using Perl function calls. In the latter case, the Wise2 code is 'compiled into' perl (in fact dynamically loaded - the unix equivalent of a dll file), meaning that although you call what looks like normal perl functions, it is actually executed by compiled C functions.

The API interface is written in C, but with a very strong object model. This means that the C API can be easily mapped to an object based enviroment. In particular this is taken advantage of in the Perl case, where Perl objects are exported in the Perl space, allowing very idomatic scripts to be written.

The documentation for the API currently lies in the C header files and the Perl .pod files. This is something which I am actively working on at the moment.

7.0.2 Example perl script

This is just a taster for you to see what one can do. For a more detailed examination, look at the API documentation.

```
#!/usr/local/bin/perl

#
# protestwise.pl <protein-seq-fasta> <dna-seq-fasta>\n
# produces on STDOUT a new protein sequence which is the
# DNA sequence 'fixed' by the comparison to the protein sequence.

# in particular frameshift errors get mapped to X

# written by James cuff (james@ebi.ac.uk)
# Hacked by Ewan (birney@sanger.ac.uk). Talk to ewan
# first about the script.


use Wise2;

my $pro_file  = shift; # first argument from @ARGV
my $dna_file  = shift; # second argument @ARGV

if( !defined $dna_file ) {
    die "ProtESTwise.pl <protein-seq-fasta> <dna-seq>\nProduces output of the DNA sequen
}

# read in inputs. Read in first as generic 'Sequence' objects
# and then converted to specific 'Protein' or 'cdna' type
# objects

open(PRO,$pro_file) || die "Could not open $pro_file!";
$seq = &Wise2::Sequence::read_fasta_Sequence(\*PRO);
$pro = &Wise2::Protein::Protein_from_Sequence($seq);

if( $pro == 0 ) {
    # can't interpolate function calls <sigh>
    die sprintf("Could not make protein from sequence %s!",$seq->name());
}

open(DNA,$dna_file) || die "Could not open $pro_file!";
$seq = &Wise2::Sequence::read_fasta_Sequence(\*DNA);
$cdna = &Wise2::cDNA::cDNA_from_Sequence($seq);

if( $cdna == 0 ) {
    # can't interpolate function calls <sigh>
    die sprintf("Could not genomic from sequence %s!",$seq->name());
}

# Read in data structures needed for
# estwise type algorthim
#
```

```
# These will be automatically read from WISECONFIGDIR if necessary.

# this is the indel rate
$cp  = &Wise2::flat_cDNAParser(0.001);

# codon table
$ct  = &Wise2::CodonTable::read_CodonTable_file("codon.table");

#this means we are not using any codon bias
$cm  = &Wise2::flat_CodonMapper($ct);

# this is the substitution error
$cm->sprinkle_errors_over_CodonMapper(0.001);

# random model needed if we are not using syn
$rmd = &Wise2::RandomModelDNA_std();

# means estwise3 algorithm. Not obvious!
$alg = 0;

# sets memory amount for main memory
&Wise2::change_max_BaseMatrix_kbytes(100000); # 10 Megabytes.

# these are for the protein part of the comparison
$comp   = &Wise2::CompMat::read_Blast_file_CompMat("blosum62.bla");
$rm     = &Wise2::default_RandomModel();

# do it!
$alb = &Wise2::AlnBlock_from_Protein_estwise_wrap($pro,$cdna,$cp,$cm,$ct,$comp,-12,-2,0,
$proseq = "";


#
# This is where we get clever!
#
# The for loops across the alignments. The protein sequence is in $alc->alu(0). The
# DNA sequence is in $alc->alu(1). We are interested in codons in the DNA sequence
# and turns those into amino acids. Sequence insertions or deletions become X's
#

for($alc=$alb->start();$alc->at_end() != 1;$alc = $alc->next()) {
    if( $alc->alu(1)->text_label() =~ /^INSERT$/ ) {
next; # skip protein inserts relative to the DNA sequence
# NB different from SEQUENCE_INSERTION.
    }

    if( $alc->alu(1)->text_label() =~ /CODON/ ) {
# get out sequence from $start to $end
# $start and $end are in bio coordinates
$start = $alc->alu(1)->start+1;
$end   = $alc->alu(1)->end+1;
$dnatemp = "";

for($x=$start;$x < $end;$x++){
    $tmp = &Wise2::cDNA::cDNA_seqchar($cdna,$x);
    $dnatemp=$dnatemp.$tmp;
}


$temp = $ct->aminoacid_from_seq($dnatemp);

# if codon has an N, then set the residue to unk X,
# we could be clever about this and work out what
# it is likely to be, but hell...

$temp =~ s/x/X/;

$proseq .= $temp;
    } else {
# deletion or insertion of a base
$proseq .= 'X';
    }


}

# make the new protein sequence and
# dump it to stdout

$namecdna = $cdna->baseseq()->name();
$new = &Wise2::new_Sequence_from_strings($namecdna,$proseq);
$new->write_fasta(STDOUT);
```