CDS-plotcon - a.k.a. MLRGD

(software notes)

Andrew E. Firth
Department of Biochemistry, University of Otago, PO Box 56, Dunedin, New Zealand

Contents

1 Introduction **3**

2 Installation **3**

3 Example **4**

4 Required software and files **5**

5 Input files **5**

6 Output files **6**

7 Error log file **9**

8 Parameters **10**

9 `redo_mlrgd` and `redo_plots` **11**

10 Run time **11**

11 Maximum number/length of sequences **11**

12 Algorithms **12**
- 12.1 Overview . . . 12
- 12.2 Notes . . . 12
- 12.3 Mutation model . . . 13
- 12.4 Substitution matrices . . . 14
  - 12.4.1 Nucleotide substitution matrix . . . 15
  - 12.4.2 Codon usage table . . . 15
  - 12.4.3 Amino acid substitution matrix . . . 15

References **15**

1 Introduction

A powerful technique for locating functional elements in genomes is to look for conserved columns in multiple sequence alignments (Stojanovic et al. 1999; `plotcon`, `EMBOSS` package – Rice et al. 2000; `MultiPipMaker` – Schwartz et al. 2003; Margulies et al. 2003; `VISTA` – Frazer et al. 2004). However it is difficult to use this method to detect additional functional elements within protein-coding sequences (CDSs), since many columns in CDSs show conservation due to constraints on the encoded protein. It is possible to look for conserved columns at four-fold degenerate sites (some, but not all, third nucleotide positions in codons), but this leaves out information from at least two thirds of columns and is more-or-less impossible within overlapping genes (common in viruses). Conserved RNA secondary structures may be found with programmes such as `alidot` (Hofacker et al. 2002) and `RNA-DECODER` (Pedersen et al. 2004), while other features may be detected through database similarity searches. However novel features without significant RNA secondary structure can not be detected using these methods.

The software package `CDS-plotcon` is specifically designed to search for conserved functional elements within CDSs. It uses an average model (§12.3) of the expected mutation patterns within CDSs (incorporating a nucleotide mutation matrix, amino acid substitution matrix, sequence divergence parameter t, mean synonymous:nonsynonymous substitution ratio V and phylogenetic tree; it can handle up to three overlapping CDSs in different read-frames). Using this, it calculates the expected number of mutations across the alignment in each column and compares this with the observed number of mutations. The results are plotted along the genome, and optionally passed through a sliding window (clipped) mean filter (§6).

Particularly conserved regions may indicate non-coding functional elements, new coding ORFs, or more-conserved regions within proteins (e.g. motifs). The software also produces conservation plots for four-fold degenerate sites, that may be used to help distinguish these alternatives. `CDS-plotcon` could also be used in conjunction with complementary programmes (e.g. RNA structure prediction programmes).

As well as running the core conservation-calculating programme, the master script `run_mlrgd` also aligns the input sequences, extracts CDS locations from GENBANK-format files or user-supplied files, calculates a phylogenetic tree, and produces the plots. In `run_mlrgd`, the user may alter many parameters including parameters for fitting t and V, running mean window sizes and clipping levels, whether the genome is circular or not and sequence range to analyse (§8).

The package is particularly useful for analysing virus genomes where (sometimes multiple) CDSs overlapping non-coding conserved features are common and many sequenced genomes with a reasonable range of divergences are often available. In general, a set of viral genomes may be downloaded in GENBANK-format from the NCBI website and fed straight into the package with minimal user input necessary.

2 Installation

The programmes are written in `C++` and should run under any LINUX/UNIX-like environment (e.g. MacOS X11), provided you have a `C++` compiler. First get the file `CDSplotcon.tar.gz`, unpack it and compile the various programmes (replace '`g++`' by an appropriate alternative, e.g. `c++` or `gcc`, if you're using a different `C++` compiler):

```
> gunzip CDSplotcon.tar.gz
> tar xvf CDSplotcon.tar
> cd CDSplotcon
> g++ -o addgaps addgaps.cxx
> g++ -o minmax minmax.cxx
> g++ -o mlrgd mlrgd.cxx
> g++ -o ntadjust ntadjust.cxx
> g++ -o runclipmean runclipmean.cxx
> g++ -o runmean runmean.cxx
```

Then copy these programmes to your `bin` directory.

3 Example

An example set of input sequences (for Hepatitis B virus) is included in the distribution. To run the example, do the following:

```
> cd EXAMPLE1
> ./run_mlrgd hbv
```

(Note that a few of the parameters in `EXAMPLE1/run_mlrgd` differ from those in the default `run_mlrgd` – e.g. the `circular` genome option is switched on.)

The programme produces a number of output files, which will be described in more detail below. The file `hbv.fcm.eps` (Figure 1; postscript image; view with e.g. `ggv hbv.fcm.eps`) shows a number of statistics and conservation plots (running mean) for non-coding positions, 1st, 2nd and 3rd codon positions, four-fold degenerate neutral sites, and all nucleotide positions. There are other tracks for the estimated p-values, the number of contributing sequences in each column (i.e. not gapped in the alignment), annotated CDSs, and any other annotated features (here the annotated features are from Smith et al. 1998, Moolla et al. 2002, Chen et al. 2005, and the NCBI reference sequence NC_003977).

There is another example (Barley Yellow Dwarf Virus), using GENBANK-format sequences, in `EXAMPLE2`:

```
> cd EXAMPLE2
> ./run_mlrgd bydv
```

In general the command is

```
> ./run_mlrgd prefix
```

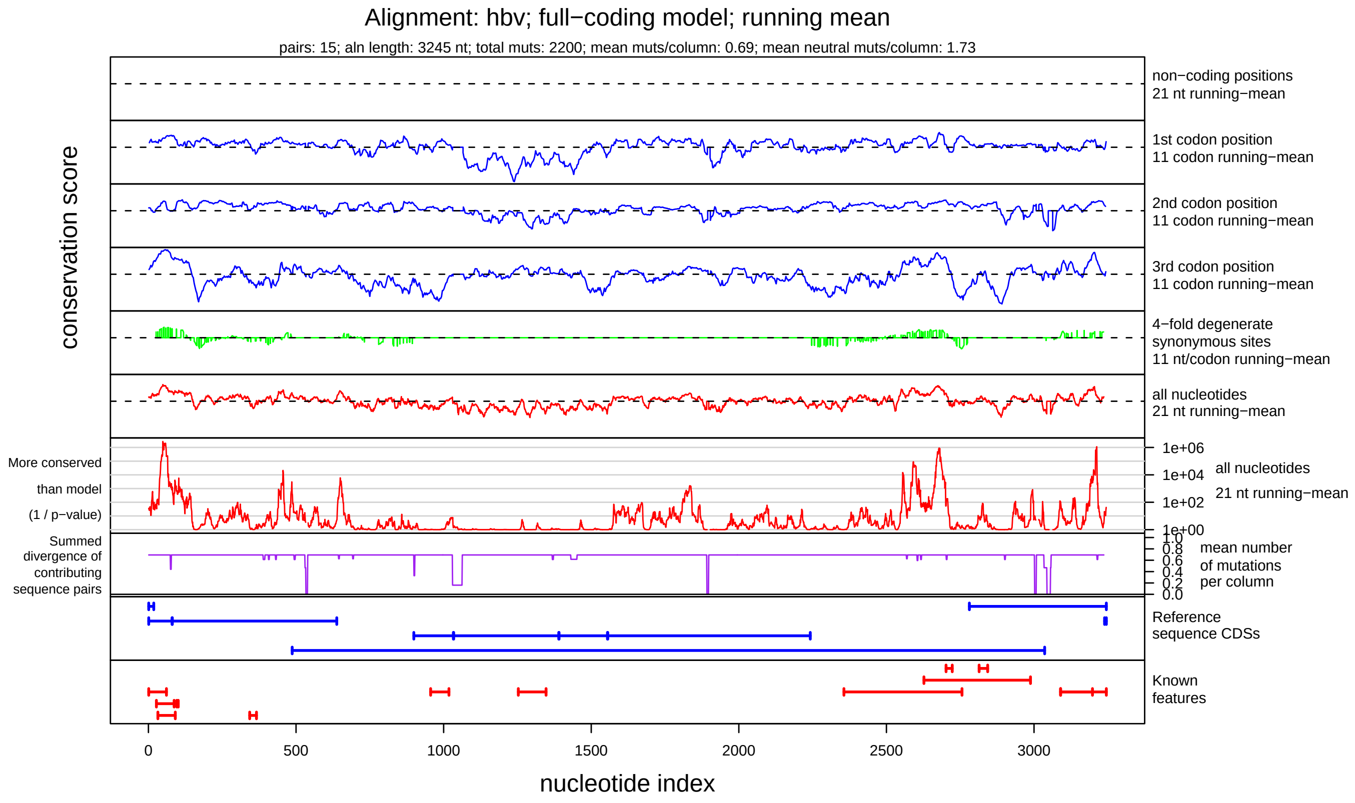

Figure 1: One of the output plots for Hepatitis B Virus example.

4 Required software and files

`run_mlrgd` will die if any of the following software is missing.

`mlrgd`
`runmean`
`runclipmean`
`addgaps`
`minmax`
`ntadjust`
	C++ programmes in the `CDS-plotcon` package.

`run_mlrgd` `mlrgd.R`	Scripts in the `CDS-plotcon` package. These will need to be in the running directory.

`aa2codon.dat`
`aa.dat`
`codon.dat`
`nuc.dat`
	Amino acid, codon and nucleotide matrices.
	These will need to be in the running directory.

`code2aln`	Sequence alignment programme. Obtain from `http://www.tbi.univie.ac.at/~roman/Code2aln/`.

`seqret`
`seqretsplit`
`infoseq`
`degapseq`
	Part of the `EMBOSS` package.
	Obtain from `http://emboss.sourceforge.net/`.

`ednaml` `ednapars`	Phylogeny programmes in the `PHYLIP` package. Repackaged as part of the `EMBASSY` package. Obtain from `http://emboss.sourceforge.net/`.
`R`	Statistics and graphics package. Obtain from `http://www.r-project.org/`.

5 Input files

The following files may be provided as input. Some are optional.

prefix`.seqs`	List of sequence files to use. The first sequence is taken as the reference sequence, *refseq*`.fasta` or *refseq*`.gbk`.
seqname`.fasta` *seqname*`.gbk`	FASTA and/or GENBANK-format sequence files.
seqname`.fasta.orfs` *seqname*`.gbk.orfs`	CDS annotation. Use the corresponding sequence coordinates, not alignment coordinates; the first nt is 1 not 0. (CDS files are only needed for the reference sequence if `refpos = 1`. CDS files are extracted from *seqname*`.gbk` file headers, where available, if `gbkpos = 1`.)
refseq`.fasta.features` or *refseq*`.gbk.features`	Optional list of annotated features to include on plots. Use reference sequence coordinates.
prefix`.pairs`	Optional list of sequence pairs to use. Otherwise the sequence pairs are taken from the phylogenetic tree.

6 Output files

`run_mlrgd` produces the following output.

prefix.`aln`	`code2aln` alignment.
prefix.`ORF.ps` *prefix*.`info.ral`	CDSs used by `code2aln`.
prefix.`tree`	`phylip` tree file.
prefix.`pairs`	Sequence pairs used by `mlrgd`.
prefix.`run_mlrgd`	Copy of the `run_mlrgd` script.
prefix.`errorlog`	Error log file.
prefix.`nc.info` *prefix*.`fc.info`	Some alignment-wide statistics used in the plots. (Note that '`nc`' indicates a non-coding model (annotated CDSs are ignored), while '`fc`' indicates the full-coding model (i.e. up to triple-coding).)
prefix.`nc.dat` *prefix*.`fc.dat`	Log of whole-sequence/region statistics for each pairwise comparison.
prefix.`nc.log` *prefix*.`fc.log`	Log of statistics for each nt in each pairwise comparison.
prefix.`nc.plot` *prefix*.`fc.plot`	Log of statistics for each nt, summed over the phylogenetic tree.
prefix.`ncm.R` *prefix*.`ncc.R` *prefix*.`fcm.R` *prefix*.`fcc.R`	`R` plotting scripts.
prefix.`ncm.eps` *prefix*.`ncc.eps` *prefix*.`fcm.eps` *prefix*.`fcc.eps`	Plots. (Note that '`nc`' indicates a non-coding model, '`fc`' indicates the full-coding model, '`m`' indicates running mean, '`c`' indicates clipped running mean.)

The files *prefix*.`??.dat` contain whole-sequence (or sequence-region) statistics for each pairwise sequence 1 – sequence 2 comparison in *prefix*.`pairs`. Columns are as follows:

1. Best-fitting t (more-or-less evolutionary time).
2. Mean $\sum_i \left[\log(P(N_i^{\mathrm{seq}_1} \to N_i^{\mathrm{seq}_2}))\right]$ per nt, where $N_i^{\mathrm{seq}_1}$ and $N_i^{\mathrm{seq}_2}$ are aligned nucleotides in sequences 1 and 2.
3. Number of nt used in the comparison.
4. Number of nt discarded due to being gaps in one or both sequences.
5. Number of nt discarded due to being 'zero-probability' transitions according to the given substitution matrices `aa.dat` and `codon.dat` (e.g. stop $\leftrightarrow$ non-stop).
6. Number of point mutations (out of the nt used, i.e. Col. 3).
7. Number of neutral or synonymous point mutations (including all non-coding nt).
8. Flag = 1 if problems with t-fitting (outside given range or didn't converge in maximum allowed number of iterations).
9. Number of 4-fold degenerate sites (in sequence 1) with a neutral or null mutation.
10. Number of 4-fold degenerate sites (in sequence 1) with a neutral non-null mutation.
11. Best-fitting V (scaling between nonsynonymous and synonymous substitution acceptabilities).

12. Flag = 1 if problems with V-fitting (outside given range or didn't converge in maximum allowed number of iterations).

Notes:

1. The mutation rate per nt is Col. 6 / Col. 3.
2. Col. 3 + Col. 4 + Col. 5 equals the sequence length in alignment coordinates.

The files *prefix*`.??.log` contain a log of statistics for each nt in each pairwise comparison. Columns are as follows:

1. Sequence pair number.
2. Nucleotide number (in alignment coords).
3. $\log(P(N_i^{\mathrm{seq}_1} \rightarrow N_i^{\mathrm{seq}_2}))$. 9 $\Rightarrow$ gap in both sequences, 8 $\Rightarrow$ gap in sequence 1, 7 $\Rightarrow$ gap in sequence 2, 6 $\Rightarrow$ zero-probability transition, 5 $\Rightarrow$ gap only in reference sequence (when `refpos = 1`).
4. Expected number of mutations (0 if gap in either sequence).
5. Expected number of neutral mutations (0 if gap in either sequence).
6. Observed number of mutations (0 if gap in either sequence).
7. Observed number of neutral mutations (0 if gap in either sequence).

The files *prefix*`.??.plot` contain a log of statistics for each nt, summed over the phylogenetic tree. Running means of these data are used in the plots. Columns are as follows:

1. Nucleotide number (in alignment coords).
2. Expected number of mutations across phylogenetic tree.
3. Observed number of mutations across phylogenetic tree.
4. Number of pairs in which nt is non-coding (using CDS annotation).
5. Number of pairs in which nt is a 1st codon position (using CDS annotation).
6. Number of pairs in which nt is a 2nd codon position (using CDS annotation).
7. Number of pairs in which nt is a 3rd codon position (using CDS annotation).
8. Number of pairs in which nt is a gap (in either sequence) or a 'zero-probability' transition.
9. Standard deviation estimated from expected number of mutations.
10. $\sum \lambda_i$, where λ_i is the mean observed number of mutations per nucleotide for sequence pair i and the sum is over those sequence pairs contributing to the score at that nt position (e.g. not gapped).
11. Expected number of neutral mutations across phylogenetic tree.
12. Observed number of neutral mutations across phylogenetic tree.
13. Expected number of neutral mutations at 4-fold degenerate sites, across phylogenetic tree.
14. Observed number of neutral mutations at 4-fold degenerate sites, across phylogenetic tree.
15. $\sum \lambda_i$ for pairs with 4-fold degenerate neutral sites at each nt.
16. Number of pairs with 4-fold degenerate neutral sites, at each nt.

Notes:

1. Col. 4 + Col. 5 + Col. 6 + Col. 7 + Col. 8 = total number of sequence pairs.
2. Columns 2, 3, 10, 11, 12, 13, 14 and 15 should be multiplied by 0.5 since forward and backward comparisons are done for each sequence pair, and multiplied by 0.5 again since each branch of the tree is crossed with two pairwise comparisons (Figure 2). Column 9 should be multiplied by $\sqrt{0.25} = 0.5$.

The image files *prefix*.`*.eps` contain a variety of plots and statistics. The header lists the alignment name and model, the number of sequence pairs, the alignment length, total number of mutations across the alignment, mean number of mutations per column and the mean number of mutations per column at four-fold degenerate neutral sites. Note that the initial list of sequence pairs covers each branch of the phylogenetic tree twice and, in addition, for each pair both forward (sequence 1 $\rightarrow$ sequence 2) and backwards (sequence 2 $\rightarrow$ sequence 1) comparisons are made. So these scores are divided by four – hence sometimes a fractional number of mutations is listed.

The ten tracks are as follows

1. Conservation in non-coding regions.
2. Conservation in 1st codon positions.
3. Conservation in 2nd codon positions.
4. Conservation in 3rd codon positions.
5. Conservation at non-coding and 4-fold degenerate neutral sites.
6. Conservation for all nucleotides.
7. Significance p-values for track 6 – i.e. the probability that conservation of that magnitude or greater would be observed if the null model were correct. Actually the reciprocal p-values are given on the y-axis scale – e.g. 1000 corresponds to a p-value of 0.001. The scores apply to the running mean scores. The standard deviations for each running mean window are calculated analytically using $\sum_{\text{window}} \sum_{\text{pairs}} p(1-p)$, where the p are $E_k(M)$ values from Equation 5, (§12.3). The p-values are $P(z \geq \text{score/stddev})$, calculated assuming a normal distribution (more-or-less OK, by the Central Limit Theorem).
8. $\sum \lambda_i$, where λ_i is the mean observed number of mutations per nucleotide for sequence pair i and the sum is over those sequence pairs contributing to the score at each nt position. Essentially this is the mean number of mutations per alignment column.
9. The location of CDSs annotated in *refseq*.`fasta.orfs` or *refseq*.`gbk.orfs`.
10. The location of known features annotated in *refseq*.`fasta.features` or *refseq*.`gbk.features` (if any).

The conservation scores in tracks 1–6 are $E_k(M) - O_k(M)$ (expected $-$ observed number of mutations) scores, scaled by $\sum \lambda_i$, where λ_i is the mean observed number of mutations per nucleotide for sequence pair i and the sum is over those sequence pairs contributing to the score at that nucleotide (e.g. not including pairs with gaps at that point). This normalizes regions of the alignment where some sequences are gapped to regions of the alignment where no sequences are gapped. If no pairs contribute at some nucleotide (e.g. if only one sequence is ungapped, or if `refpos = 1` and the reference sequence is gapped) then the track returns to zero. Tracks 7 and 8 may be used to assess the significance of any observed features.

The scores are passed through a running mean filter with (image files *prefix*.`??c.eps`) or without (image files *prefix*.`??m.eps`) clipping. The window size and clipping thresholds are adjustable by the user (use `redo_plots` to redo the plots with different values; see §9). Note that the window skips any gaps, so in track 1, for example, the scores at the end of one non-coding region will be windowed along with the scores at the beginning of the next non-coding region.

Image files are produced both for the full-coding model (*prefix*.`fc?.eps`) specified by all the input CDS files, and for a non-coding model (*prefix*.`nc?.eps`) where all nucleotides are assumed to be non-coding.

A variety of not-really-worth-saving files are moved to the directory `TIDYUP`. Keep these if you might want to use the scripts `redo_mlrgd` or `redo_plots`.

The track for synonymous/neutral sites may look somewhat different from the other tracks – many individual bars rather than a continuous line. This is because the synonymous/neutral sites are scattered within CDSs and the track returns to zero at any gap greater than three nucleotides wide. Note that the sliding window covers 2 × `window2` + 1 adjacent synonymous/neutral sites rather than being a window of size 2 × `window2` + 1 in the alignment coordinates. You can obtain a traditional plot of conservation at neutral sites in this track by setting `fitwhat = 2` or `3`.

Note that a given column may be four-fold degenerate and neutral for some sequence pairs but not for others: four-fold degeneracy depends on the codon in sequence 1. Neutrality depends on the codons in both sequences being synonymous. Hence at each nucleotide, track 5 is scaled by the sum of λ_i values just for those pairs i that contribute, rather than the $\sum \lambda_i$ values in track 8. Tracks 1, 2, 3, 4 and 6 are just scaled by the $\sum \lambda_i$ values in track 8. For tracks 1, 2, 3, 4 this only makes sense if the codon position annotation is the same for all sequence pairs (e.g. if `refpos = 1`).

A combination of track 4 (3rd codon positions) in coding regions – except overlapping CDSs – and track 1 (non-coding positions) in non-coding regions can be useful. This is less susceptible to site-specific variation in the nonsynonymous:synonymous substitution ratios than track 6, but provides denser and more even coverage than track 5. Within track 4, `CDS-plotcon` provides appropriate scaling between 1-, 2-, 3- and 4-fold degenerate positions.

7 Error log file

You should check the error log file *prefix*.`errorlog`. Typical errors include:

```
seqname contains non-ACGTU characters.  Omitting this sequence.
Aborting: sequence 3, unknown nucleotide 'S', at 2345.
Warning: sequence 3, ambiguous nt 'S', at 2345.
```
→ You can set `skip = 0` to allow standard ambiguous nucleotide codes.

```
Fitting number of mutations (87) outside that allowed by given t range. Sequence pair 4.
Iteration on t failed to converge in maxiter iterations. Sequence pair 5.
Fitting number of mutations (987) outside that allowed by given V range. Sequence pair 2.
Iteration on V failed to converge in maxiter iterations. Sequence pair 2.
```
→ You may need to change the parameters `tttmin`, `tttmax`, `tVfit`, `maxiter`, `Vmin`, `Vmax`. Alternatively you may have a bad alignment – perhaps one of your sequences is too divergent from the others. (Note that the `Fitting number of mutations outside that allowed by given V range` will always occur for the `nc` model since, for the `nc` model, all nucleotides are assumed non-coding so the V parameter has no effect.)

```
Warning: seqname.orfs has overlapping same-frame CDSs.
```
→ In this case, regions of the second CDS that overlap the first, in the same frame, are ignored.

```
Warning: gap within coding region not a multiple three nt; sequence pair 3, sequence 1,
sequence coord 1280, alignment coords 1296.
```
→ These error messages are quite common if you have extra CDSs in some sequences but not in others, or if corresponding CDSs are longer in some sequences than in others. In this case there may be local problems with the estimated expected number of mutations, as codons will be picked from the wrong read-frame in sequence 2. Consider discarding more divergent sequences, supplying `*.orfs` files for all sequences, or using the `range1`—`range2` parameters to remove the problematic region.

More serious errors include:

```
Aborting: ORF 3 in seqname.orfs, endpoints outside sequence range.
```
→ Check the `*.orfs` files. Note that these should be in the corresponding sequence coordinates, not alignment coordinates.

```
Warning: number of nucleotides in ORF 2 in seqname.orfs not a multiple of 3.
```
→ The programme will continue to work, but you should check your `*.orfs` files. Note that for

frameshifts, disjoint CDSs and circular genomes, you should use the GENBANK join(2307..3212,1..1623) format in the `*.orfs` files.

8 Parameters

There are a variety of parameters that you can alter by opening the script `run_mlrgd` in a text-editor and editing the `Options` section. A brief description of these are given below (suggested defaults are give by `param = default`).

1. `pairs = 1`: Method for generating the list of sequence pairs to use: 2 → use phylogenetic tree built with `ednapars` (maximum parsimony); 1 → use phylogenetic tree built with `ednaml` (maximum likelihood); 0 or other → use user-input file *prefix*`.pairs`.
2. `refpos = 0`: 0 → use seperate CDS files for each sequence; 1 or other → use reference sequence CDS files for all sequences.
3. `gbkpos = 0`: 0 → get CDS files from *seqname*`.fasta.orfs` (or *seqname*`.gbk.orfs`) files; 1 or other → get CDS files from *seqname*`.gbk` files (for any *seqname*`.fasta` files, the *seqname*`.fasta.orfs` file, if any, will be used).
4. `skip = 0`: 1 or other → skip sequence files containing non-ACGTUacgtu characters (exits if the reference sequence contains non-ACGTUacgtu characters); 0 → read standard ambiguous nt codes as N and ignore for most statistics (ambiguous nt are logged along with the gaps; codons containing ambiguous nt are logged along with the 'zero-probability' transitions).
5. `window1 = 5`: n, where the running mean sliding window size for the 1st, 2nd and 3rd codon positions and for the 4-fold degenerate neutral sites plots is $2n + 1$ nt (or codons).
6. `window2 = 10`: n, where the running mean sliding window size for the all sites and for the non-coding sites plots is $2n + 1$ nt.
7. `circular = 1`: 0 → not a circular genome; 1 or other → circular genome.
8. `clip1 = 0.00`: Upper clipping threshold for clipped running mean: `clip1 = 0.05` means that the upper 5% (i.e. columns with low observed number of mutations relative to expected number of mutations) in each window are clipped before calculating the mean.
9. `clip2 = 0.30`: Lower clipping threshold for clipped running mean: `clip2 = 0.05` means that the lower 5% (i.e. the columns with high observed number of mutations relative to expected number of mutations) in each window are clipped before calculating the mean.
10. `tttmin = 0.0`: Lowest t value to use in t-fitting.
11. `tttmax = 10.0`: Highest t value to use in t-fitting.
12. `tVfit = 0.2`: Required fitting accuracy (total number of mutations between a sequence pair) for t and V.
13. `maxiter = 20`: Maximum number of fitting iterations for t and V.
14. `Vmin = 0.01`: Lowest V value to use in V-fitting (≥ 0.01).
15. `Vmax = 10.0`: Highest V value to use in V-fitting (≤ 10).
16. `Vtype = 1`: 0 → find seperate V value for each sequence pair; 1 or other → find one V value for the alignment.
17. `Vfix = 0`: If > 0, use this V value for all sequence pairs (defaults to 1 if ≤ 0; overridden if `fitwhat = 3`).

18. `fitwhat = 3`: 0 or other → fit t using total number of mutations; 1 → fit t using number of neutral/synonymous mutations; 2 → fit t using number of neutral/synonymous mutations at 4-fold degenerate sites; 3 → as for 2, then adjust V to fit the total number of mutations (i.e. including nonsynonymous mutations). Note: 1—3 are not recommended if the sequence is predominantly double-coding since, in this case, the number of neutral sites is potentially very small.

19. `wholeseq = 1`: Nucleotide range to use for all model-fitting, statistics and plots: 0 → use the region `range1`—`range2`, 1 or other → use the whole sequence.

20. `range1 = 0`: Start nucleotide of region to analyse (if `wholeseq = 0`; otherwise ignored), in reference sequence coordinates.

21. `range2 = 0`: End nucleotide of region to analyse (if `wholeseq = 0`; otherwise ignored), in reference sequence coordinates.

9 `redo_mlrgd` and `redo_plots`

You can use these two scripts to redo the `mlrgd` and plots parts (respectively) of `run_mlrgd`, with different parameters. This is a lot quicker than rerunning all of `run_mlrgd`. Usage is

```
> ./redo_mlrgd prefix tttmin tttmax tVfit maxiter fitwhat refpos circular wholeseq range1
range2 Vfix Vtype Vmin Vmax skip newprefix
```

and

```
> ./redo_plots prefix window1 window2 circular clip1 clip2 newprefix
```

The new output files take the prefix `newprefix`.

You can experiment with the window sizes, and clipping parameters. For example, if you are looking for unusually conserved columns, then you aren't interested in highly variable columns so you may want to set `clip2` quite high (e.g. 50%, provided the window size is sufficiently large) so that such columns don't contaminate conserved columns in the same window. On the other hand, if the typical length of features that you are interested in is 10 nt, then you could set the window size to 10 nt (e.g. `window1` = 5).

10 Run time

The conservation programme `mlrgd` is very quick. On my LINUX PC with a 2.6 GHz Pentium^{R} 4 processor) it takes ∼20–40 s (depending on the parameter options) to run `mlrgd` on the Hepatitis B example alignment. Much more time-consuming is the initial alignment (with `code2aln`) and the phylogenetic tree calculation with `dnaml` (`dnapars` is much quicker and is recommended for alignments of more than about 15 sequences). The time requirements of `code2aln` and `dnaml` are very dependent on the number of sequences. The scripts `redo_mlrgd` and `redo_plots` allow one to rerun the conservation programme and redo the plots with different parameters without having to rerun `code2aln` and `dnaml`.

11 Maximum number/length of sequences

Currently the maximum sequence length (with alignment gaps) is 50000 nt and the maximum number of sequence pairwise comparisons is 100. You can change these limits by editing the '`#define`' lines at the beginning of `*.cxx` and recompiling these programmes.

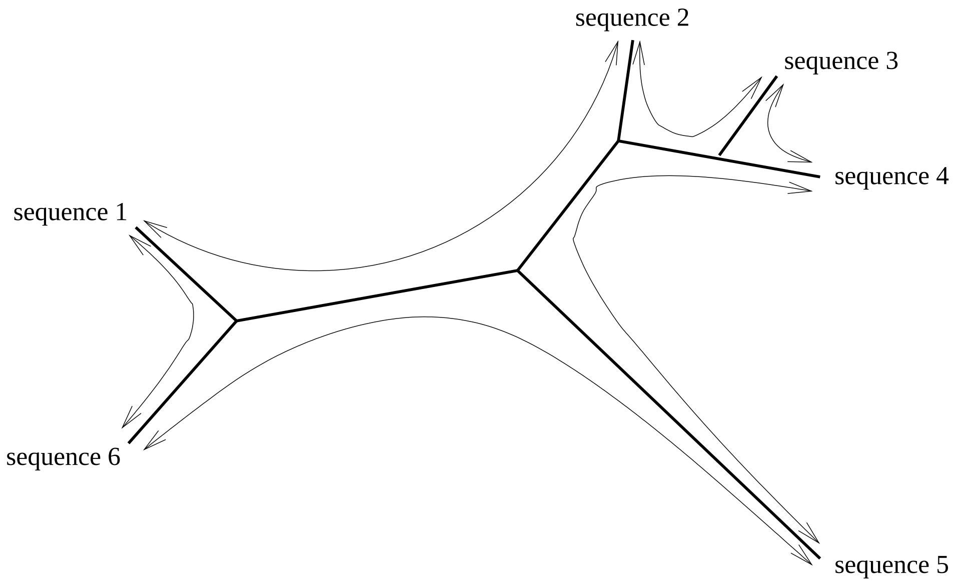

Figure 2: Example phylogenetic tree. For this tree, the sequence pairs used by `mlrgd` would be sequence 1 – sequence 2, sequence 2 – sequence 3, sequence 3 – sequence 4, sequence 4 – sequence 5 and sequence 5 – sequence 6.

12 Algorithms

12.1 Overview

First the sequences are aligned with `code2aln` (Stocsits 2003). This sequence alignment programme tries to keep gaps in groups of three in CDSs and smoothly joins coding regions onto non-coding regions.

Next an unrooted phylogenetic tree is made with `ednaml` (maximum likelihood) or `ednapars` (maximum parsimony). These programmes are part of the `PHYLIP` package (Felsenstein 2004) repackaged into the `EMBASSY` package in `EMBOSS` (Rice et al. 2000). The tree is used to select a list of sequence pairs tracing round the outside of the tree (Figure 2). Conservation scores are calculated with `mlrgd` for each pair and summed over the tree. Note that this set of pairwise comparisons covers each branch of the tree precisely twice – hence no branch is given more weight than another. In general, the set of pairs selected in this way is not unique – since branches of the tree may be flipped into different places without changing the phylogeny. Note that these default trees just use a simple non-coding evolutionary model. However, if desired, the user may input their own list of pairs.

For each sequence pair S_1–S_2, `mlrgd` finds the best-fitting sequence divergence t and, optionally, the best-fitting synonymous:nonsynonymous weighting V (see §12.3). With these t and V values `mlrgd` calculates the expected number of mutations at each nucleotide in S_1 (a number between 0 and 1), and the observed number of mutations (either 0 or 1). These values are summed over all pairs to give the conservation plots.

12.2 Notes

Knowing the correct reading-frame is very important for `mlrgd`'s statistical mutation model. When estimating the probability that a given nucleotide in a CDS will mutate, `mlrgd` needs to know which codon it is a member of in both sequences. In a given pairwise comparison S_1–S_2, either S_1 (`refpos = 0`) or the reference sequence (`refpos = 1`) CDS files are used to define all the reading-frames. If `refpos = 0`, every coding nucleotide in S_1 is therefore assigned a codon position (1st, 2nd or 3rd) and read-direction (forward or reverse). The nucleotides in S_2 are identified purely on the basis of the nucleotides that they align with in S_1. If `refpos = 1`, the nucleotide identification in both S_1 and S_2 comes from their alignment with the reference sequence. In order to maintain the correct reading-frame in the different sequences, it is important that gaps in the alignment occur in groups of three within coding sequences. Hence the use of `code2aln` as the alignment programme. If you have purely coding sequences you can of course use e.g. `CLUSTALW` on the translated (amino acid) sequences and run `mlrgd` on its own (e.g. with the `redo_mlrgd` script).

The sequences used for the CDS annotation (reference sequence or S_1's, depending on `refpos`) mustn't have sequencing-error indels within CDSs, as these will throw `mlrgd` out of frame and cause global problems. However indels are tolerated in the other sequences: `mlrgd` will get a local incorect codon identification, but there'll be no long-range problems. Bad alignments or local (paired) frameshifts (gaps not in threes) will mean that nucleotides within the 'mis-aligned' region may have the wrong codon position identification – leading to wrong codon identifications – but the problems should be local. It is up to the user to check for alignment problems (see *prefix*`.aln` and *prefix*`.errorlog` files). In non-coding regions, of course, gaps not in threes are allowed and don't cause problems.

In general you should use `refpos = 1` (just use the reference sequence CDS annotation) if you only have CDS annotation for one or a few of your sequences, or if you want to ensure that the CDS annotation is identical across the alignment, or if some sequences have less-than-perfect sequencing quality. However, supplying CDS files for all sequences may help `code2aln` (`code2aln` automatically finds long ORFs but may miss ORFs under 300 nt or ORFs without a start codon – e.g. at ribosomal frameshift sites, stop-codon read-through sites, or circular genomes).

For each sequence pair S_1–S_2, `run_mlrgd` runs the forward comparison $S_1 \rightarrow S_2$ and also the backward comparison $S_2 \rightarrow S_1$ so if, for example, there is a CDS annotated in S_1 but not in S_2, each nucleotide in the CDS will get equal weighting as a non-coding nucleotide and as a coding nucleotide in the final output plots.

The software will handle circular genomes as follows. When calculating the mutation probability for coding nucleotides at the ends of the input alignment, it will take into account codons that span the break in the circular genome. Also the running mean plots will use windows that span the break. However, the alignment programme will not reposition the breaks in the input sequences, so you must make sure that the break is in the same place in all of your input sequences.

12.3 Mutation model

In this section we describe the model for non-coding, coding, and multiply-coding regions. We broadly follow the standard approach of modelling sequence evolution as a Markov process (Goldman & Yang 1994; Hein & Stovlbaek 1995; Ewens & Grant 2001). However we do deviate a little (see below) in order to deal with potentially double-coding or triple-coding regions, without having to resort to time-consuming MCMC simulations to estimate probabilities (Pedersen & Jensen 2001). We use more or less the same approach as in Firth & Brown (2005, supplementary material).

Suppose we have an aligned pair of sequences S_1 and S_2. For each nucleotide N_1^k in S_1 we estimate the probability that N_1^k mutates to each of the nucleotides U, C, A, G, as follows. First we define $b(N_1^k \rightarrow i; t, V)$, $i = \mathrm{U}, \mathrm{C}, \mathrm{A}, \mathrm{G}$, by

$$b(N_1^k \rightarrow i; t, V) \quad = \quad \mathbf{P}(N_1^k \rightarrow i; t) \tag{1}$$

for non-coding regions,

$$\begin{aligned} b(N_1^k \rightarrow i; t, V) \quad = \quad & \mathbf{P}(N_1^k \rightarrow i; t) \\ & \times \mathbf{A}(X_1 \rightarrow X_2) \times V^{\mathrm{ns}} \times \mathbf{C}(X_2) \end{aligned} \tag{2}$$

for single-coding regions, and

$$\begin{aligned} b(N_1^k \rightarrow i; t, V) \quad = \quad & \mathbf{P}(N_1^k \rightarrow i; t) \\ & \times \mathbf{A}(X_1 \rightarrow X_2) \times V^{\mathrm{ns}} \times \mathbf{C}(X_2) \\ & \times \mathbf{A}(X_1' \rightarrow X_2') \times V^{\mathrm{ns}} \times \mathbf{C}(X_2') \end{aligned} \tag{3}$$

for double-coding regions, with the obvious extension for triple-coding regions. Here X_1 and X_2 are the original and final amino acids or codons in one read-frame, and X_1' and X_2' are the original and final amino acids or codons in the second read-frame (for double-coding regions), for the nucleotide mutation $N_1^k \rightarrow i$. The mutation is taken in the context of S_2, in the sense that the neighbouring nucleotides for the codons X_2 and X_2' are taken from S_2, while those for X_1 and X_1' are taken from S_1. Also $\mathbf{P}(t) = \exp(\mathbf{Q}t)$; $\mathbf{Q}$, $\mathbf{C}$, $\mathbf{A}$ are the nucleotide, codon and amino acid matrices described in §12.4; V^{ns} is 1 if X_1 and X_2 (X_1' and X_2') are synonymous codons and V^{ns} is a scaling factor V if X_1 and X_2 (X_1' and X_2') are nonsynonymous codons.

The probability that N_1^k mutates to j, after time t, is then given by

$$P(N_1^k \to j; t, V) = \frac{b(N_1^k \to j; t, V)}{\sum_{i=\mathrm{U,C,A,G}} b(N_1^k \to i; t, V)}. \tag{4}$$

Using this model, we can calculated the expected number of non-null mutations M occurring in S_1 after a given time t by

$$E(M) = \sum_{k=1}^{\text{length}} E_k(M) = \sum_{k=1}^{\text{length}} \sum_{j \neq N_1^k} P(N_1^k \to j; t, V). \tag{5}$$

Given the two sequences S_1 and S_2, we adjust t to fit the expected number of mutations $E(M)$ to the observed number of mutations $O(M)$ between S_1 and S_2.

The parameter V may be left at the default value of 1 (usually pretty close to the best-fit value) or fitted seperately for each pair of sequences or one value may be fitted for the whole alignment. In the latter two cases, first t is determined by fitting the observed and expected number of mutations only at non-coding positions and four-fold degenerate sites where the codons in S_1 and S_2 are synonymous. Then V is determined by fitting the observed and expected total number of mutations.

Any transition that is assigned a zero-probability according to the input amino acid and codon matrices (e.g. transitions between stop codons and non-stop codons in the default matrices) is ignored and written to the log file. Any transition involving gaps or ambiguous nucleotide codes is also ignored and written to the log file.

Note that the above methodology involves several approximations. Firstly, in Equation 3, we consider only a single nucleotide and the single codon in each of the primary and secondary read-frames containing that nucleotide. However in double-coding regions, adjacent codons are linked by the overlapping read-frame. Hence mutations within one codon can affect the probabilities for subsequent mutations in neighbouring codons. Secondly, as far as codon and amino acid weightings are concerned, we consider only the start and end points of the unknown mutation pathway connecting an aligned codon pair in S_1 and S_2. It seems reasonable that these simplifications are justified provided S_1 and S_2 are not too divergent (so that mutation pathways are short and inter-codon cross-talk is low in multiply-coding regions). In Firth & Brown (2005) it is shown that this model provides useful results over a wide range of circumstances.

The parameter V models the selection pressure that may vary from one gene to another. In fact the selection pressure will be different for every codon and this is often modelled by site-dependent rates models (Yang et al. 2000). Using a site-dependent model is pointless for our purposes, as it will be impossible to distinguish columns that have enhanced conservation. The BLOSUM amino acid matrix that we use represents the *average* amino acid substitution acceptabilities over all codons. The parameter V allows a global adjustment for selection pressure while still allowing particularly conserved sites to be distinguished from other sites. Such sites may be additional non-coding elements, new coding ORFs, or more-conserved regions within proteins (e.g. motifs). The track for mutations at four-fold degenerate neutral sites may be used to help distinguish between these alternatives, though it will provide information for less than one in three nucleotides within CDSs. The track for 3rd codon positions may also be used and, since it includes 2-fold degenerate sites, will provide information at more positions.

12.4 Substitution matrices

In this section we describe the default input 4×4 nucleotide mutation matrix $\mathbf{Q}$, 64-entry codon usage table $\mathbf{C}$, and 20×20 amino acid substitution matrix $\mathbf{A}$. You can edit these in the files `nuc.dat`, `codon.dat` and `aa.dat` (respectively) if you wish, but generally you won't need to. The genetic code may be altered in the file `aa2codon.dat`. Further details are available from the authors or from Firth & Brown (2005, supplementary material).

12.4.1 Nucleotide substitution matrix

As default, we use a $\kappa = 3$ Kimura (1980) nucleotide matrix, i.e.

$$\mathbf{Q} = \begin{bmatrix} -1.0 & 0.6 & 0.2 & 0.2 \\ 0.6 & -1.0 & 0.2 & 0.2 \\ 0.2 & 0.2 & -1.0 & 0.6 \\ 0.2 & 0.2 & 0.6 & -1.0 \end{bmatrix}, \qquad (6)$$

where the row and column order is U, C, A, G. This corresponds to equal equilibrium nucleotide frequencies π_j and a transition:transversion ratio of $\kappa = 3$.

12.4.2 Codon usage table

As default, we use a null codon usage table (CUT) – i.e. equal codon frequencies. For typical viral genomes, due to the large number of overlapping CDSs and other constrained features, it is not clear that a CUT generated directly from the viral genome will be representative of mutation probabilities. The host species CUT may be more appropriate which, in the case of human viruses, is not strongly biased and so we use a null CUT for simplicity (see also simulations in Firth & Brown 2005). In addition, using a non-null CUT means that four-fold degenerate sites in CDSs are no longer strictly degenerate in terms of substitution probabilities.

More generally a non-null CUT may be incorporated as follows. Suppose we have the codon GGU. Mutations in the 3rd position are synonymous (all code for *gly*) and their relative frequencies are controlled by the nucleotide mutation matrix $\mathbf{Q}$. However we also wish to preserve codon bias as a sequence mutates. Since we are always working from an initial known amino acid, we must use relative (instead of absolute) codon frequencies but each frequency must be multiplied by the degeneracy of the corresponding amino acid, otherwise, for example, ?UG $\rightarrow$ CUG (*leu*) will be a factor of six less probable than ?UG $\rightarrow$ AUG (*met*) simply because there are six codons for *leu* but only one for *met*. In addition, codon usage statistics implicitly include any nucleotide bias and, conversely, any nucleotide bias described by $\mathbf{Q}$ will automatically lead to a codon bias. Hence the nucleotide equilibrium frequencies π_j must be factored out by dividing each codon usage value by $\pi_i.\pi_j.\pi_k$ where i, j, k are the ith, jth, kth nucleotides in the codon. See Firth & Brown (2005) for scripts to produce appropriate CUTs from standard absolute or relative frequency CUTs.

12.4.3 Amino acid substitution matrix

In our model, the probability that a nucleotide mutation occurs at the DNA level and the probability that the mutation is accepted (i.e. is functional) at the protein level are separated into the nucleotide and amino acid matrices. In contrast, the widely used BLOSUM (Henikoff & Henikoff 1992) and PAM (Dayhoff et al. 1978) matrices incorporate both effects into one matrix. In the PAM matrices, the small-t amino acid substitution frequencies are extrapolated to larger t. This is a serious short-coming since, in reality, at small t a mutating sequence is constrained to resemble the original sequence at both the nucleotide and amino acid levels, whereas at large t a mutating sequence is only constrained to resemble the original at the amino acid level. On the other hand, the BLOSUM matrices are calculated, in effect, for a series of t values: BLOSUM100, BLOSUM95, ... BLOSUM35, with the lower indices corresponding to more divergent sequences. By choosing a low-index BLOSUM matrix (viz. BLOSUM40) as our default amino acid distance matrix $\mathbf{A}$, we minimize the effect of the nucleotide mutation constraint relative to the amino acid acceptability constraint.

We use the scaled observed frequencies (Henikoff & Henikoff $\frac{q_{ij}}{e_{ij}}$ values) rather than log odds scores, and treat $\frac{q_{ij}}{e_{ij}} / \frac{q_{ii}}{e_{ii}}$ as the probability of acceptance for the amino acid substitution $X_i \rightarrow X_j$ relative to $X_i \rightarrow X_i$ which is unity. The V parameter (§12.3) scales the off-diagonal terms of $\mathbf{A}$ relative to the diagonal terms, with the default value $V = 1$ giving the original BLOSUM40 matrix. Stop codons are also included, with the acceptabilities for mutations between stops and non-stops set to zero.

References

Chen A., Kao Y. F., Brown C. M., 2005, Translational of the first upstream ORF in the hepatitis B virus pregenomic RNA modulates translation at the core and polymerase initiation codons, *Nucleic*

Acids Res., 33, 1169–1181

Dayhoff M. O., Schwartz R. M., Orcutt B. C., 1978, A model of evolutionary change in proteins, *Atlas of Protein Sequence and Structure*, 5, Suppl. 3, 345–358

Ewens W. J., Grant G. R., 2001, Statistical Methods in Bioinformatics, Springer-Verlag, New York

Felsenstein, J., 2004, PHYLIP (Phylogeny Inference Package) version 3.6, `http://evolution.genetics.washington.edu/phylip.html`

Firth A. E., Brown C. M., 2005, Detecting overlapping coding sequences with pairwise alignments, *Bioinformatics*, 21, 282–92

Frazer K. A., Pachter L., Poliakov A., Rubin E. M., Dubchak I., 2004, VISTA: computational tools for comparative genomics, *Nucleic Acids Res.*, 32, W273–279

Goldman N., Yang Z., 1994, A codon-based model of nucleotide substitution for protein-coding DNA sequences, *Mol. Biol. Evol.*, 11, 725–736

Hein J., Stovlbaek J., 1995, A maximum-likelihood approach to analyzing nonoverlapping and overlapping reading frames, *J. Mol. Evol.*, 40, 181–189

Henikoff S., Henikoff J. G., 1992, Amino acid substitution matrices from protein blocks, *Proc. Natl. Acad. Sci. USA*, 89, 10915–10919

Hofacker I. L., Fekete M., Stadler P. F., 2002, Secondary structure prediction for aligned RNA sequences, *J. Mol. Biol.*, 319, 1059–66

Kimura M., 1980, A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences, *J. Mol. Evol.*, 16, 111–120

Margulies E. H., Blanchette M., Haussler D., Green E. D., NISC Comparative Sequencing Program, 2003, Identification and characterization of multi-species conserved sequences, *Genome Res.*, 13, 2507–2518

Moolla N., Kew M., Arbuthnot P., 2002, Regulatory elements of hepatitis B virus transcription, *J. Viral Hepat.*, 9, 323–331

Pedersen A. K., Jensen J. L., 2001, A dependent-rates model and an MCMC-based methodology for the maximum-likelihood analysis of sequences with overlapping reading frames, *Mol. Biol. Evol.*, 18, 763–776

Pedersen J. S., Meyer I. M., Forsberg R., Simmonds P., Hein J., 2004, A comparative method for finding and folding RNA secondary structures within protein-coding regions, *Nucleic Acids Res.*, 32, 4925–4936

Rice P., Longden I., Bleasby A., 2000, EMBOSS: the European molecular biology open software suite, *Trends Genet.*, 16, 276–277

Schwartz S., Elnitski L., Li M., Weirauch M., Riemer C., Smit A., Green E. D., Hardison R. C., Miller W., NISC Comparative Sequencing Program, 2003, MultiPipMaker and supporting tools: Alignments and analysis of multiple genomic DNA sequences, *Nucleic Acids Res.*, 31, 3518–3524

Smith G. J. 3rd, Donello J. E., Luck R., Steger G., Hope T. J., 1998, The hepatitis B virus post-transcriptional regulatory element contains two conserved RNA stem-loops which are required for function, *Nucleic Acids Res.*, 26, 4818–4827

Stocsits R. R., Nucleic Acid Sequence Alignments of Partly Coding Regions, Dissertation, 2003

Stojanovic N., Florea L., Riemer C., Gumucio D., Slightom J., Goodman M., Miller W., Hardison R., 1999, Comparison of five methods for finding conserved sequences in multiple alignments of gene regulatory regions, *Nucleic Acids Res.*, 27, 3899–3910

Yang Z., Nielsen R., Goldman N., Pedersen A.-M. K., 2000, Codon-Substitution models for heterogeneous selection pressure at amino acid sites, *Genetics*, 155, 431–449