
# Homologue Alignment Quality, Establishment of Subfamilies and Ancestor Construction

Richard J. Edwards (2005)

# 1: Introduction

Software manuals are boring: boring to write and probably even more boring to read. I have therefore tried to keep this one concise. However, given (a) my propensity to waffle, (b) the fact that I am a biologist and not a computer scientist, and (c) my lack of experience in writing manuals, there is a good chance that the pleiotropic effect of this is a lack of clarity and/or coherence. For this I apologise, and encourage anyone out there to send in errata and/or suggested improvements.

The fundamentals should be covered under the sections **2.1: Running HAQESAC**, **2.2: Input**, and **2.3: Output**. Further information can be found in **3: HAQESAC Algorithm** and/or at the website (http://www.bioinformatics.rcsi.ie/~redwards/haqesac/). Gluttons for punishment can get even more details in the **Appendices**.

Like the software itself, this manual is a 'work in progress' to some degree. If the version you are now reading does not make sense, then it may be worth checking the website to see if a more recent version is available, as indicated by the **Version** section of the manual. Good luck.

Rich Edwards, 2005.

## *1.1: Version*

This manual is designed to accompany HAQESAC version 1.5.

The manual was last edited on 12 September 2006.

## *1.2: Copyright, License and Warranty*



## *1.3: Using this Manual*

As much as possible, I shall try to make a clear distinction between explanatory text (this) and text to be typed at the command-prompt etc. Command prompt text will be `written in Courier New` to make the distinction clearer. Program options, also called 'command-line parameters', will be **`written in bold Courier New`** (and coloured **`red`** for fixed portions or **`dark red`** for user-defined portions, such as file names etc.). Command-line examples will be given in (purple) *`italicised Courier New`*. Optional parameters will (where I remember) be `[in square brackets]`. Names of files supplied with HAQESAC will be marked in text by (dark yellow) **bold Times New Roman**. Please let me know if you find any confusing formatting errors!

## 1.4: Getting Help

Much of the information here is also contained in the HAQESAC website (http://www.bioinformatics.rcsi.ie/~redwards/haqesac/) and the documentation of the Python modules themselves. A full list of command-line parameters can be printed to screen using the **help** option, with short descriptions for each one.

```
python haqesac.py help
```

If none of the above are of help, then please e-mail me (richard.edwards@ucd.ie) whatever question you have. If it is the results of an error message, then please send me that and/or the log file (see **Output**) too. Usually, it will be a problem with the input files (possibly formatting) but there are probably still a few bugs in there somewhere too.

## 1.5: Why use HAQESAC?

HAQESAC is a tool designed for the semi-automated processing of datasets of homologous proteins. The primary focus is making a trustworthy global (whole sequence) alignment based around a query sequence, with a corresponding phylogeny with high bootstrap support for key branches. In the standard scenario, these branches divide the tree into several related protein subfamilies in a larger protein family. Finally, HAQESAC can use the GASP algorithm (Edwards and Shields 2004) to construct predicted ancestral sequences.

Many biological studies make use of multiple sequence alignments and phylogenetic trees of closely-related protein sequences in order to make inferences about the function or evolution of the proteins of interest. Several resources now exist for downloading pre-computed datasets of protein families (e.g. HoVerGen/HoBactGen (Perriere et al. 2000), TAED (Liberles et al. 2001), PANTHER (Thomas et al. 2003)) or domain alignments (e.g. PFam (Bateman et al. 2004)). While these are extremely useful for high-throughput automated analyses for identifying biological trends, they are of less use when one is interested in a specific protein. Because such resources must aim to be useful for as many different analyses as possible, they generally suffer by not being optimized for any given analysis. Even if one is happy with the automated rules used to generate these data, the sequences and/or databases of interest may not be included in the database at all.

An alternative is to take a protein of key interest and use search tools to query sequence databases to find homologous proteins from which to infer evolutionary information. This too has its problems, however. Querying multiple databases can lead to unwelcome redundancy and duplicity within the returned dataset. At the same time, most commonly used search techniques – such as BLAST (Altschul et al. 1990), or downloading proteins with a shared domain – are based on finding regions of local similarity and the proteins found do not necessarily align well with each other. (This is especially true when two proteins that both share some local similarity to the query share no homology with each other.) The task of processing such a dataset – removing redundant sequences, constructing and alignment and removing badly aligned sequences, and constructing a well-supported phylogeny – can be daunting and time-consuming, even to the experienced biologist.

HAQESAC is a semi-automated tool designed to take some of the pain out of processing an initial dataset consisting of potential homologues (detected by BLAST, for example, although HAQESAC can take totally unrelated sequences as input) into a high quality dataset that is useful for further analyses. By default, the goal is to construct a dataset consisting of the query protein and those proteins that align well along as much of the query as possible, attempting to maximize the number of well-aligned residues in the dataset that overlap with the query (or a selected region of interest). Redundant and otherwise unwanted sequences are removed on the basis of sequence identity and/or phylogenetic information, Badly aligned sequences are discarded and regions of bad alignment are marked. Where desired,

HAQESAC can then be used to divide sequences into groups using an interactive menu system and predict ancestral sequences.

These goals are achieved by a performing a series of **Homologue Alignment Quality (HAQ)** cleanup steps, interactive **Establishment of Subfamilies (ES)** and finally, if desired, **Ancestor Construction (AC)** to generate predicted ancestral amino acid sequences. By default, HAQESAC will perform all three operations. However, it is possible to turn one or more off and only, for example, only reject individually badly aligned sequences, or only discard redundant sequences, if desired.

HAQESAC is flexible enough for a number of dataset cleanup exercises where the goal is an alignment of closely related sequences. It is not designed, however, for alignments of distant homologues. It may still have a use in these circumstances but there are probably better tools available elsewhere that make use of structural data etc.

## *1.6: Installation*

HAQESAC is distributed as a number of open source Python modules. It should therefore work on any system with Python installed without any extra setup required – simply copy the relevant files to your computer and run the program (see **2.1: Running HAQESAC**, below.)

If you do not have Python, you can download it free from **www.python.org** at http://www.python.org/download/. The modules are written in **Python 2.4**. The Python website has good information about how to download and install Python but if you have any problems, please get in touch and I will help if I can.

### 1.6.1: Files Required for HAQESAC

The following files are required for HAQESAC to run correctly. All these files should have been provided in the download zip file. The Python Modules are open source and may be changed if desired, although please give me credit for any useful bits you pillage. I cannot accept any responsibility if you make changes and HAQESAC stops working, however! The additional files may all be replaced with other files in the correct format. These files are described later in this manual and/or in the Appendix.

**Python Modules (*.py):** **haqesac, rje, rje_ancseq, rje_blast, rje_haq, rje_seq, rje_tree, rje_pam, rje_sequence, rje_tree_group, rje_uniprot**

**Additional Files:** **jones.pam, aaprop.txt**

### 1.6.2: Programs Used by HAQESAC

In addition to the python modules listed above, HAQESAC makes use of the following published programs. These are freely available for downloading and installing. It is recommended that the user downloads and installs these programs according to the instructions given on the appropriate website. Depending on the options selected, not all these programs may be used in a given HAQESAC run.

**ALIGN:** This is part of the Fasta package (Pearson 1994, 2000) and can be downloaded from the University of Virginia: http://fasta.bioch.virginia.edu/. Make sure that align is part of the download. For some reason it seems to have been dropped from later packages. You may need to install an earlier package first (e.g. 2.1) and then a later package.

**BLAST:** BLAST (Altschul, et al. 1990) is a very well-known and widely used homology search tool and is freely available for download from NCBI at: http://www.ncbi.nlm.nih.gov/blast/download.shtml.

**CLUSTALW:** ClustalW (Higgins and Sharp 1988, Thompson et al. 1994) is an old stalwart for bioinformatics and is freely available from EMBL: ftp://ftp-igbmc.u-strasbg.fr/pub/ClustalW/. Note that CLUSTALW is used as an alignment backup for MUSCLE (below) and to draw trees. See **3.8: Replacing Components with Other Programs** for details of how to incorporate other tree-drawing packages.

**MUSCLE:** MUSCLE (Edgar 2004) is a newer multiple alignment program available from http://www.drive5.com/muscle.

**PHYLIP:** PHYLIP (Phylogeny Inference Package) (Felsenstein 2005) is an alternative package used for inferring phylogenies using a variety of different methods and is freely available from: http://evolution.genetics.washington.edu/phylip.html. Although the trees generated using PHYLIP are probably better than those made with ClustalW, it is also slower. See **3.5.1: Tree Drawing with ClustalW or PHYLIP** for more details.

### 1.6.3: Setting up the INI File

It is recommended that a file named **rje.ini** or **haqesac.ini** is made and placed in the same directory as the **haqesac.py** program. This file should contain the paths to the above programs:

```
blastpath=PATH
fastapath=PATH
clustalw=PATH
maketree=PATH
muscle=PATH
phylip=PATH
```

Note that for BLAST, FASTA and PHYLIP, the `PATH` is the directory in which programs can be found, while for ClustalW and MUSCLE the actual program commands themselves must be included. This is to make it easier to replace these programs with alternatives. (See **3.8: Replacing Components with Other Programs**.) See the included **haqesaq.ini** file for an example. If running in windows, it is also advisable to add the `win32=T` command to the ***.ini** file.

**NB.** For `PATH` variables, directories should be separated by a forward slash (`/`). If paths contain spaces, they should be enclosed in double quotes: `path="example path"`. It is recommended that paths do not contain spaces as function cannot be guaranteed if they do.

### 1.6.4: Reducing Memory Requirements

HAQESAC can theoretically be run on datasets of unlimited size but in reality, depending on the platform used, your computer will probably crash if the dataset is too big. There are a couple of parameters that can be used to reduce the computational burden, however.

You can reduce the strain on your computer using the `blastcut=X` option, which will limit the number of sequences put into the alignment program to the top `X` BLAST hits. If you are only really after orthologues, then `blastcut=100` would not be too detrimental. This will only help if your computer dies after the initial cleanup stage.

The `cwcut=X` option can also be used to make your computer use ClustalW rather than MUSCLE for its alignments. The `X` in this case is the total number of amino acids in the dataset. Below `X`, MUSCLE is used. Above `X`, ClustalW is used. Setting this quite low (`cwcut=1000`) will mean that ClustalW is used nearly all the time.

# 2: Fundamentals

## 2.1: Running HAQESAC

### 2.1.1: The Basics

If you have python and the other relevant files installed on your system (see **1.6: Installation**), you should be able to run HAQESAC directly from the command line in the form:

```
python haqesac.py seqin=FILENAME
```

For the example provided in the distribution:

```
python haqesac.py seqin=haqesac_eg.fas
```

If running in Windows, you can just double-click the **haqesac.py** file and enter a sequence file name and command-line options when prompted. If running from the command-line and the named file does not exist, or contains no sequences, then this same prompt will be given (unless **i=-1** as described below).

**Note:** If filenames contain spaces, they must be enclosed in double quotes, e.g.

```
python haqesac.py seqin="haqesac example with spaces.fas"
```

HAQESAC produces quite a lot of files. I recommend that you create a separate directory for each query sequence and run HAQESAC from this directory, e.g.:

```
mdkir haqesac_eg
cd haqesac_eg
python ../haqesac.py seqin=haqesac_eg.fas
```

### 2.1.2: Interactivity and Verbosity settings

By default, HAQESAC will run through to most bits without any user-interaction and then pause to ask questions where required. For more interaction with the program as it runs, use the argument '**i=1**':

```
python haqesac.py seqin=haqesac_eg.fas i=1
```

Both the level of interactivity and the amount printed to screen can be altered, using the interactivity [**i=X**] and verbosity [**v=X**] command-line options, respectively, where **X** is the level from none (-1) to lots (2+). Although in theory **i=-1** and **v=-1** will ask for nothing and show nothing, there is a good chance that some print statements will have escaped in these early versions of the program. Please report any irritations.

**Note:** The Interactivity and Verbosity settings are still under development. Please send suggestions of things you would like more or less prompts for.

### 2.1.3: Other Options

At first, you will probably want to run HAQESAC with its default parameters. If you want to change them, there are a number of parameters that can be set by the user and other options. These are described in **5.1: Appendix I: Command-line Options** and in the relevant sections of the manual. These may be given after the run command, as above, or loaded from a ***.ini** file (see **5.1.3: INI Files** details).

There are essentially three types of command-line option:

1. Those that require a value (numerical or text), **option=X**. Those that require a filename as the value will be witten: **option=FILE**
2. True/False (On/Off) options, **option=T/F**. For these options:
    a. **option=F** and **option=F**alse are the same and turn the option off
    b. **option**, **option=T** and **option=True** are the same and turn the option on
3. List options. These are like the value options but have multiple values, separated by commas: **option=X,Y**. Where **..** is used, the number elements is optional, e.g. **option=X,Y,..,Z** could take **option=X** or **option=A,B,C,D**. Where **option=LIST** is used, the number of elements is optional and **LIST** could actually be the name of a file containing the list of elements.

### 2.1.4: Picking up where you left off

This section will be expanded in time but there are two ways to pick up where you left off:

1. Simply start HAQESAC again with the appropriate input file. If it has already gone through the clean-up process then use **cleanup=F** to turn this off.
2. To backtrack to an earlier stage, use one of the output files from SAQ or PAQ and map the 'real' sequences back onto them with the **mapseq=FILE** command and the backup file from the initial cleanup.
   *E.g.* *seqin=example.saqx.1.fas mapseq=example.fas.bak* or *seqin=example.saqx.1.fas mapseq=example.clean.fas*.

## *2.2: Input*

### 2.2.1: Input sequences

The recommended main input for HAQESAC is a fasta file of protein sequences using the **seqin=FILE** option. This may be aligned or unaligned and the sequences may be related or not – it is the job of HAQESAC to cleanup the input dataset to an alignment of related sequences. For details on how this is done, see the **3: HAQSEAC Algorithm** section below. Fasta format is commonly used in bioinformatics applications and a variety of subformats will be recognised by HAQESAC. To get the most out of the program, one of the set fasta formats from common sequence databases should be used. These are listed in more detail in **3.2: Preliminary Dataset Cleanup**. The basics are that descriptions should be on one line that starts '>' followed by one or more lines containing the sequence. The first word in each description should be unique. *e.g.*

```
>Seq1 And its description
SEQUENCE-ONE-GOES-HERE
>Seq2
---GAPS--ARE--ALLOWED-
AS-ARE-MULTIPLE-LINES
```

Most databases and homology search results *etc.* can be downloaded in fasta format.

Alternatively, HAQESAC may be given the following formats:

- UniProt download in Swiss-Prot format (**seqin=FILE**)
- ClustalW alignment format (**seqin=FILE**)
- Phylip format (**seqin=FILE**)

- A list of sequences names (**seqin=FILE**) to be extracted from a BLAST-formatted sequence database (**fasdb=FILE**) using fastacmd.

These additional inputs are not fully supported yet and may have bugs. Output (below) is always fasta.

To restrict input to a list of accession numbers in the given file, use the (**acclist=LIST**), where **LIST** could be a filename or a comma-separated list of accession numbers (`acclist=acc1,acc2,acc3` etc.).

### 2.2.2: PAM Matrix

The PAM matrix contains amino acid substitution probabilities and is used by the GASP algorithm (Edwards and Shields 2004), which is used to generate ancestral sequences. A basic PAM matrix (Jones et al. 1992) is available with the HAQESAC program in the file **jones.pam** (and is known as JTT, I believe). Alternative files can be given using the **pamfile=FILE** option. The important part of this file is the top section, which has the single letter amino acid codes on the first line, followed by the PAM1 matrix, where each subsequent line consists of an amino acid code and the probability of that aa being substituted by each other aa, in the order given in the first line:

```
A   R   N   D   C   Q   E   G   H   I   L   K   M   F   P   S   T   W   Y   V
Ala 0.98754 0.00030 0.00023 0.00042 0.00011 0.00023 0.00065 …
Arg 0.00044 0.98974 0.00019 0.00008 0.00022 0.00125 0.00018 …
Asn 0.00042 0.00023 0.98720 0.00269 0.00007 0.00035 0.00036 …
…
Val 0.00226 0.00009 0.00007 0.00016 0.00012 0.00008 0.00027 …
```

Note that the amino acids must be in the same order in both columns and rows. See http://www.bioinformatics.rcsi.ie/~redwards/gasp/ for more details.

### 2.2.3: HAQ Matrix

HAQESAC can also take an input file in the same format as the PAM Matrix (**2.2.2: PAM Matrix**) that contains scores for pairwise amino acid comparisons in SAQ and PAQ (see **3: HAQESAC Algorithm** for details). By default, these values are 1 for self comparisons (e.g. A vs A, or L vs L) and 0 for non-self (e.g. A vs L, or C vs D). Uploading a file with alternative values using **haqmatrix=FILE** allows for different scoring matrices based on amino acid property similarities etc.

## *2.3: Output*

The main output for HAQESAC is an alignment, a tree and ancestral sequence predictions. Details for some of the outputs are given below, along with a full summary of all output files. See the HAQESAC and GASP websites for more information.

### 2.3.1: Output Alignment

The output alignment is in FASTA format (see **2.2.1: Input Sequences**) with descriptions on one line followed by one line containing the sequence. All sequences should be of the same length. The first word in each description should be unique. Freely available programs such as BioEdit (Hall 2001) can be used to view and edit alignments. Some sequence names may have been changed relative to the input data (see **3.2.6: The gnspacc=T/F Option**).

### 2.3.2: Additional Sequence Files

In addition to the 'standard' fasta file (***.fas**), HAQESAC will output alignments with 'bad' residues replaced by Xs. The number of files produced and their content is dependent on

the 'Data Level' as determined by the **d=X** option. See **2.3.7: Summary of HAQESAC Output** for details.

HAQESAC also outputs an unaligned sequence file with gaps removed. This is given in two formats: fasta format, and "scansite parallel format". The latter consists of sequence accession numbers and sequences all on one line, separated by a space:

```
Q92633 MAAISTSIPVISQPQFTAMNEPQCFYNESIAFFYNRSGKH…
AAK07693 QCYYNESIAFFYNRSGKYLATEWNTVSKLVMGLGITVC…
…
```

This format is useful for certain downstream applications, such as motif prediction using scansite (Obenauer et al. 2003).

### 2.3.3: Output trees

HAQESAC will output a bootstrapped tree in the general Newick Standard format, suitable for viewing with programs such as Tamura's TreeExplorer (Tamura 2001):

`(Seq1:dis1,Seq2:dis2)Bootstrap:dis3` (formatting added for clarity)

More information on Newick Format can be found at the PHYLIP homepage and GASP website.

In addition, HAQESAC will output the same tree in raw text format (see **2.3.5: Ancestral Sequence Data**).

### 2.3.4: Sequence Group Data

If saved, sequence groups are written to a simple text file containing the unique sequence IDs for the defined groups:

```
Group 1: edg7
edg7_HUMAN/AAF91291
EDG7_RAT/Q8K5E0
EDG7_MOUSE/Q9EQ31

Group 2: edg2
EDG2_HUMAN/Q92633
edg2_CAVPO/AAK07693
EDG2_SHEEP/P46628
EDG2_BOVIN/Q28031
EDG2_MOUSE/Q61130
LP11_XENLA/Q9PU17
```

Group names (edg7 and edg2 in this example) are optional.

### 2.3.5: Ancestral Sequence Data

In addition to the extant sequences, HAQESAC will output predicted ancestral sequences in GASP Fasta format, with ancestral nodes distributed vertically to match the text tree.

```
>06 LP11_XENLA/Q9PU17 Lysophosphatidic acid receptor LPA1…
MASLSEFVSEPISMMSQTSAASESQCYYNETIAFFYNRSGKYLATEWN-AVSKLVMGLGITVC…
>16 Node 16 (6,14,17)
MA--AAFTSSPVISQPQFTAMNEQQCYYNESIAFFYNRSGKYLATEWN-TVSKLVMGLGITVC…
```

```
>05 EDG2_MOUSE/Q61130 Lysophosphatidic acid receptor Edg-2…
MA--AASTSSPVISQPQFTAMNEQQCFYNESIAFFYNRSGKYLATEWN-TVSKLVMGLGITVC…
>14 Node 14 (5,12,16)
MA--AASTSSPVISQPQFTAMNEQQCFYNESIAFFYNRSGKYLATEWN-TVSKLVMGLGITVC…
>01 EDG2_HUMAN/Q92633 Lysophosphatidic acid receptor Edg-2…
MA--AISTSIPVISQPQFTAMNEPQCFYNESIAFFYNRSGKHLATEWN-TVSKLVMGLGITVC…
>11 Node 11 (1,2,12)
MA--AISTSIPVISQPQFTAMNEPQCFYNESIAFFYNRSGKYLATEWN-TVSKLVMGLGITVC…
>02 edg2_CAVPO/AAK07693 endothelial differentiation…
----------------------QCYYNESIAFFYNRSGKYLATEWN-TVSKLVMGLGITVC…
>12 Node 12 (11,10,14)
MA--AASTSSPVISQPQFTAMNEPQCFYNESIAFFYNRSGKYLATEWN-TVSKLVMGLGITVC…

… [truncated]
```

In this example, Node 14 is linked to sequence 5, (descendant) node 12 and (ancestral) node 16 as can be seen with the corresponding portion of the tree:

```
           +---+ 6: LP11_XENLA/Q9PU17 {0.060000}
           |
    #======+ Node 16 (6,14,17) [1000] {0.130000}
    |      |
    |      | +-+ 5: EDG2_MOUSE/Q61130 {0.010000}
    |      | |
    |      +-+ Node 14 (5,12,16) [1000] {0.020000}
    |        |
    |        |   +-+ 1: EDG2_HUMAN/Q92633 {0.010000}
    |        |   |
    |        | +-+ Node 11 (1,2,12) [235] {0.010000}
    |        | | |
    |        | | +-+ 2: edg2_CAVPO/AAK07693 {0.010000}
    |        | |
    |        +-+ Node 12 (11,10,14) [441] {0.010000}
    |          |
  [truncated]
```

Post-duplication nodes are marked with # and post-duplication branches with =. See the GASP website and Manual for more information (Edwards 2004, Edwards and Shields 2004).

### 2.3.6: Temporary HAQESAC Output

The following files may be generated during HAQESAC runs but should generally be deleted upon completion if the program exits correctly. If any of these files consistently remain after apparently error-free execution of HAQESAC then please let me know. Otherwise, they can (probably) be safely deleted. As always, however, if in doubt, make a backup first.

| File(s) | Description | Stages |
|---|---|---|
| *.fas.phr, *.fas.pin, *.fas.psd, *.fas.psi, *.fas.psq | BLAST database files. | Cleanup |
| *.tmp.fas | Temporary fasta files for ALIGN | Cleanup |
| *.dnd | Guide tree for alignment | Cleanup, HAQ |
| *.aln | ClustalW alignment file | Cleanup, HAQ |
| *_cw.phb | ClustalW Tree | ES |

### 2.3.7: Summary of HAQESAC Output

The main output files from HAQESAC are summarised in this table. There are additional optional outputs at the Establishment of Subfamilies stage but these are not covered here.

| File | Description | Type | Stages | Data Level |
|---|---|---|---|---|
| *.log | Activity log for run of program. Contains summary information, reasons for sequence removal etc. and any error messages. | Text | All | 0 |
| *.fas | The output alignment from HAQESAC. During HAQESAC this will sometimes be unaligned. | Fasta | Cleanup, HAQ, ES | 0 |
| *.fas.bak | Backup file produced pre-cleanup. (Unless `backup=F`.) | Fasta (text) | Cleanup | 1 |
| *.clean.fas | File produced after cleanup but before HAQ. (Unless `cleanup=F`.) | Fasta | Cleanup | 1 |
| *.haq.fas | Post HAQ alignment file with bad regions of alignment replaced with Xs. | Fasta | HAQ | 1 |
| *.saq.N.fas | Post SAQ alignment file with bad regions of alignment replaced with Xs. N represents the number of the SAQ cycle producing the alignment. (**NB.** If HAQESAC is killed and restarted, this numbering will restart and overwrite old files.) | Fasta | HAQ | 2 |
| *.saqx.N.fas | Post SAQ alignment file with bad residues of individual sequences replaced with Xs. N is the SAQ cycle number (see above). | Fasta | HAQ | 3 |
| *.paq.N.fas | Post PAQ alignment file with bad regions of alignment replaced with Xs. N is the PAQ cycle number (see above). | Fasta | HAQ | 2 |
| *.nsf | Phylogenetic Tree. | NSF | ES | 0 |
| *.tree.txt | ASCII text version of the tree, with gene duplications marked. | Text | ES | 0 |
| *.grp | Sequence Group information. | Text | ES | 0 |
| *.anc.fas | Predicted ancestral sequences. (See GASP documentation for details.) | Fasta | AC | 0 |
| *.anc.nsf | Tree produced by GASP with branch lengths replaced by ML PAM distances between nodes. | NSF | AC | |
| *.anc.txt | ASCII text version of the tree produced by GASP. | Text | AC | 0 |
| *.degap.fas | Output sequences with gaps removed. | Fasta | End | 1 |
| *.scanseq | Output sequences in correct format to upload to Scansite parallel server. | Special | End | 1 |

**File:** File name. **Description:** File description. **Type:** Type/Format of file. **Stages:** Stages of HAQESAC that generate file. **Data Level:** Data level setting (`d=X`) required to generate file.

# 3: HAQESAC Algorithm

The basics of the HAQESAC algorithm are also covered at the HAQESAC website: http://www.bioinformatics.rcsi.ie/~redwards/haqesac/. Please report any discrepancies that you discover between this account and the web version, as the latter was written for a previous (Perl) version of HAQESAC, before a number of improvements and extra features were added. The website does have a useful diagrammatic representation of the HAQ process, which is worth looking at while reading the appropriate sections below (**3.3: Single Sequence Alignment Quality** and **3.3.1: Pairwise Alignment Quality**).

**Note:** HAQESAC is intended to be run with a focal Query sequence. All descriptions of the algorithm will therefore be written with this in mind. It is possible to run HAQESAC without a query sequence but the program is not optimised for this. (See **3.7: 'No Query' Mode**.)

## *3.1: An overview of HAQESAC*

An overview of HAQESAC is given in **Figure 1**. HAQESAC is a tool designed for processing a dataset of potential homologues into a trustworthy global alignment and well bootstrap-supported phylogeny, with predicted ancestral sequences (optional). An initial dataset consisting of homologues (detected by BLAST, for example, see **4.1: Building a Dataset Using BLAST**) is processed into a dataset consisting of the query protein and those proteins that align well along the entire length of the query. Where desired, sequences will be divided groups of orthologues (same protein in different species, separated by speciation) and paralogous subfamilies (different proteins separated by gene duplications).

Individual sequences are therefore screened fulfil two criteria:

1. They must be homologous to (and alignable with) the query sequence of interest.
2. They must be a member of a subfamily within the gene family to which the query sequence belongs.

If you are not interested in subfamily groupings then parameter values can be set such that the second screening is downweighted or removed. (See **4.2: Example Parameter Settings**.)

These goals are achieved by a performing a series of **Homologue Alignment Quality (HAQ)** cleanup steps, interactive **Establishment of Subfamilies (ES)** and finally, if desired, **Ancestor Construction (AC)** to generate predicted ancestral amino acid sequences. These stages are summarised below and then described in more detail in the appropriate sections. By default, HAQESAC will perform all three operations. However, it is possible to turn one or more off and only, for example, reject individually badly aligned sequences, if desired. This is controlled by `cleanup=T/F`, `haq=T/F`, `es=T/F` and `ac=T/F`, where `T` is on (True) and `F` is off (False). HAQESAC is designed for use with a specific query in mind (identified using `query=X`) but can also be used in 'No Query' mode (`noquery=T`), with no query sequence for SAQ, and a random Query for PAQ if none is given (see **3.7: 'No Query' Mode**). To focus on a specific portion of the Query, rather than the entire length, use `qregion=X,Y`, where `X` and `Y` are the beginning and end of the region of interest, as defined by positions within the query. This allows focussing analysis on a given domain, for example.

By default, HAQESAC will realign sequences, and throw out badly aligned sequences based on residues of identity only. If you have already spent time manually honing your alignment, or cleaning out unwanted sequences, you may not want this to happen. `keep=T` will keep all sequences in the input dataset unless they are manually removed during sequence grouping, while `usealn=T` will keep the input alignment.

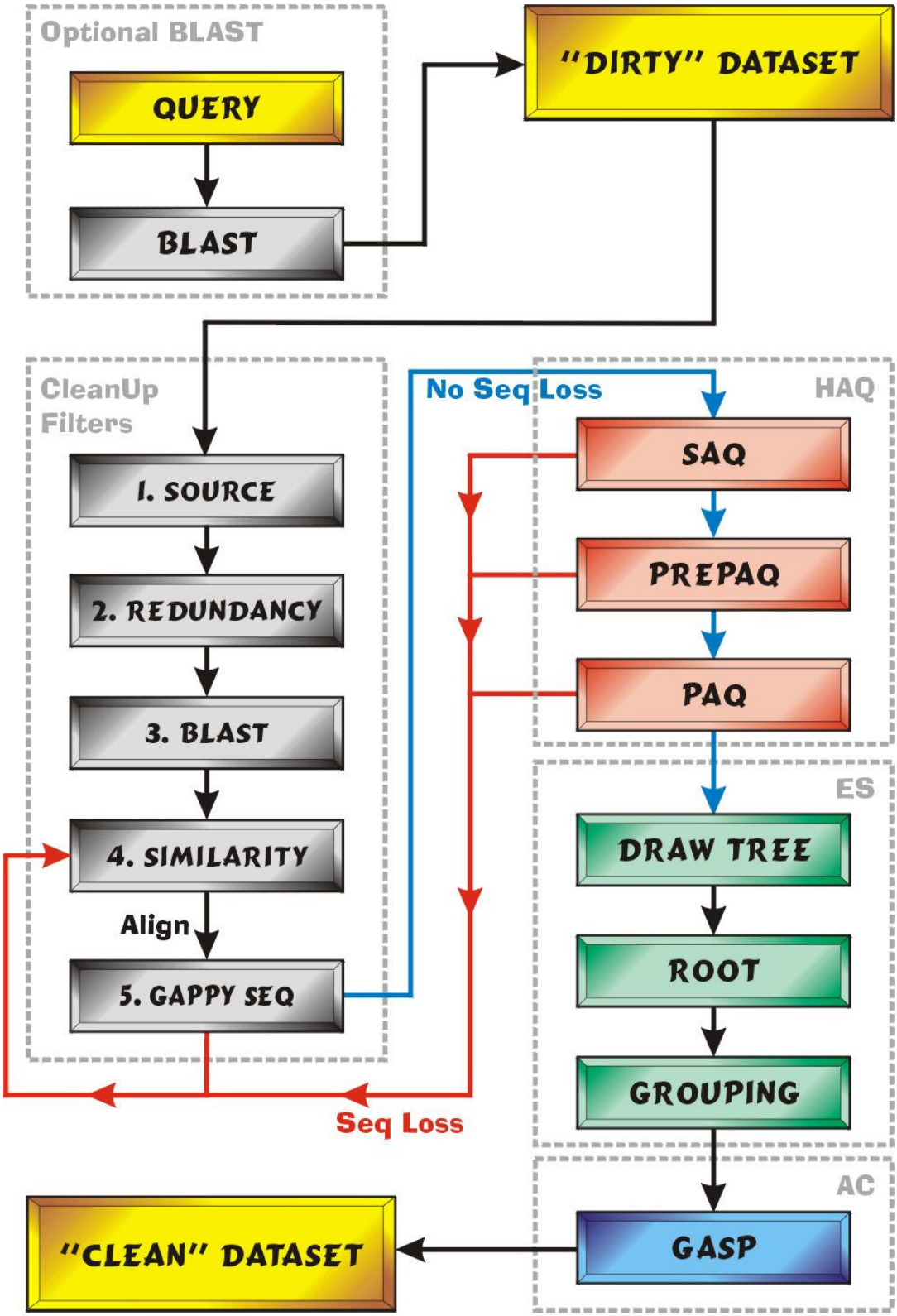


**Figure 1. Overview of HAQESAC Data Processsing.** An initial "dirty" dataset first undergoes a series of filtering stages before the main Homologue Alignment Quality (HAQ) cleanup. If sequences are removed during any stage then the cleanup process will cycle (red arrows) until no more are lost before moving on to the next stage (blue arrows). Following data cleanup, HAQESAC can be used to draw and root a tree, establish sequence groups and perform Gapped Ancestral Sequence Prediction (GASP).

### 3.1.1: Homologue Alignment Quality (HAQ)

The first stage of data cleanup is to remove rogue sequences that either do not fit in the gene family at all or are too distantly related to the query protein for a decent alignment that can be used for useful further analysis. This is achieved firstly by a simple BLAST e-values and pairwise percentage identity cut-offs, followed by a more complex procedure of removing sequences for whom the overall alignability is poor. During this procedure, sequences that have too many gaps are also removed as too many gapped residues can cause problems for downstream evolutionary analyses. Further screening is achieved based on phylogenetic information.

### 3.1.2: Establishment of Subfamilies (ES)

Once the dataset has been 'cleaned up' (and, indeed, during processing), HAQESAC can be used to assign sequences to subgroups or subfamilies, if such information is needed for downstream analyses. The **Establishment of Subfamilies** phase is also very useful for identifying and removing any undesirable sequences that were not removed in earlier clean-up stages, including splice variants, polymorphic sequence variants and in-paralogues. (Note that for some analyses, it is desirable to keep these sequences and HAQESAC will allow this.)

### 3.1.3: Ancestor Construction (AC)

The final step that HAQESAC is able to perform is ancestral sequence prediction using the GASP (Gapped Ancestral Sequence Prediction) algorithm (Edwards and Shields 2004).

### 3.1.4: General HAQESAC Options

| Option | Description | Default |
|---|---|---|
| `qregion=X,Y` | Concentrate on the region of the query from (and including) residue X to residue Y | [0,-1] |
| `noquery=T/F` | No Query for SAQ, Random Query for PAQ (else `query=X` or first sequence) | [False] |
| `keep=T/F` | Keep all sequences (`saqkl=0`, `paqkl=0`) | [False] |
| `usealn=T/F` | Use current alignment (do not realign, `degap=F`) | [False] |
| `cwcut=X` | Total number of residues above which to use ClustalW for alignment, not MUSCLE | [1e5] |
| `haqmatrix=FILE` | File of AA vs AA scores used in SAQ and PAQ. (See **2.2.3: HAQ Matrix** for details.) | [None] |
| `cleanup=T/F` | Pre-HAQESAC dataset cleanup | [True] |
| `haq=T/F` | Homologue Alignment Quality | [True] |
| `es=T/F` | Establishment of Subfamilies | [True] |
| `ac=T/F` | Ancestor Construction (GASP) | [True] |

## 3.2: Preliminary Dataset Cleanup

The aim of the preliminary dataset cleanup is to accelerate the cleanup process by filtering out sequences that are likely to be filtered by later stages of HAQESAC. This includes the removal of sequences with low levels of homology to the Query (and, optionally, other proteins) as well as the removal of redundant sequences. This process itself comprises of four main parts:

1. Sequence/Database filters
2. Redundancy checking
3. BLAST filtering of distant sequences
4. Pair-wise Percentage Identity filter
5. Gappy sequence filter

These are described in more detail below. 4 and 5 are repeated until no more sequences are removed.

### 3.2.1: Sequence/Database Filters

Before HAQESAC even gets to work, there are a number of sequence filters that can be applied to the dataset. By default, a copy of the initial input data is saved in ***.fas.bak** (overwriting any existing file of the same name). Use the **backup=F** option to turn this off.

Sequences can be filtered according to several options. **filterseq=FILE** will filter out any sequences that are present in a given file. This may be a HAQESAC log file, a Fasta file, or simply a list of names. **filterspec=FILE** will do the same thing for a file of unwanted species names or codes, while **unkspec=F** will removes any sequences of unknown species. **keepdesc=FILE** performs the opposite function and only keeps sequences with 1+ of text listed in given file in their description line. Finally, **dbonly=T** restricts input to sequence belonging to recognised databases. These databases, and their hierarchy for redundant sequence removal, is set by the **dblist=X,Y,..,Z** option, where **X,Y,..,Z** constitutes a list of databases in order of preference (good to bad).

In addition to these filters based on sequence details, input may be filtered according to sequence length using the **minlen=X** and **maxlen=X** options.

### 3.2.2: Redundancy Checking

By default, sequences of the same species that are at least 99% identical are removed. Annotated fragments are removed in preference to full length sequences. Then sequences are kept according to the database hierarchy (set by the **dblist=X,Y,..,Z** option) and, within databases, longer sequences are preferentially kept over shorter ones. The **seqnr=F** option will turn off this redundancy check sequence, while **specnr=F** will search for redundancy across sequences from different species. The percentage identity cut-off is set with **nrid=X**.

Further to this redundancy check, if exactly the same sequence may be present several times due to concatenating datasets etc. then the **accnr=T** option can be used to check for redundant Accession Numbers/Names upon loading the sequences.

### 3.2.3: BLAST filtering of distant sequences

Following the redundancy check, a BLAST search is performed using the query versus the input dataset. (This does not occur if **noquery=T**.) Only sequences that are hit by the query with an e-value of $10^{-4}$ (or **blaste=X**) are kept. This can also be used to reduce the

number of sequences to the best X hits, using **blastcut=X**. To turn off this BLAST filter, set **blastcut=-1**. The Complexity Filter is on by default but can be disabled with **blastf**=F. If GABLAMO pairwise percentage similarity/identity filtering (below) is used, the Complexity Filter will be switched off.

### 3.2.4: Pairwise Percentage Similarity/Identity filter

Following the BLAST cut-off, sequences are compared in a pairwise fashion to screen out any that are not similar enough with a simple measure. By default, the GABLAMO (Global Alignment from BLAST Local Alignment Matrix (Ordered)) method is used (Edwards & Davey, unpublished). As the name suggests, this constructs global alignment statistics from individual local BLAST alignments. The best-scoring alignment is taken first and positions of identity and similarity mapped back on to both Query and Hit proteins. Then each other alignment is taken and, provided the start and end positions in Query and Hit do not contradict a previous (better scoring) alignment, its positions of identity and similarity are likewise mapped. This is continued until all local alignments have been processed. The total identity, similarity and coverage (total number of residues included in mapped alignments) is then calculated by counting how many positions in Query and Hit have been mapped from local alignments.

To filter according to coverage – the percentage of the sequence that is covered by (ordered) local BLAST alignments – use the **qcover=X** option. By default this is set at 60.0, meaning that at least 60% of the query must align with the protein *or* at least 60% of the protein must align with the query. In addition, sequences can be constrained to have a certain degree of similarity within the aligned regions. This is set by the **qryid=X** option and is a percentage of the entire sequence length, not just the aligned regions. Furthermore, because the primary focus is to group sequences into well-aligned subfamilies, each sequence can be constrained to have a minimum global similarity with another sequence of a different species, as controlled by the **pairid=X** option. Where singletons are desired, this second filter should be turned off using **pairid=0**.

To use identity rather than similarity, set **gabsim=F**. Likewise, to use ALIGN (Pearson 1994, 2000) instead of GABLAMO, set **gablam=F**.

### 3.2.5: Gappy Sequence Filter

Following Percentage Identity cut-offs, the dataset is aligned and very gappy sequences are eliminated. Each sequence is compared to its closest neighbour in the multiple sequence alignment (defined by global percentage identity) and if it is over 50% gaps then it is rejected. (Residues that are gaps in both sequences are ignored.) This threshold is controlled by the **maxgap=X** option.

### 3.2.6: The gnspacc=T/F Option

The **gnspacc=T** command reformats the names of the input sequences from something like:

```
>ENSP000000223233 blah blah blah
```

to

```
>ens_HUMAN__ENSP000000223233 blah blah blah
```

It is not necessary, and an be switched off with **gnspacc=F**, but does make it much easier to work out what species each sequence is from or, at least, whether they are the same species.

The first word of the description always becomes:

>**GN_SP__ACC**, where:

- **GN** is a gene or database identifier. (If you select 'New Gene' while reviewing a group, this is the bit you change. It can be very helpful in trees of multiple subfamilies but keep it short and match SwissProt where possible, e.g. DBOH in DBOH_MOUSE)

- **SP** is a species code. This is generally the Uniprot species code or the best guess the program can make (namely the first three letters of the genus and the first two letters of the species).

- **AC** is the accession number.

This is done because the first word must be unique for each sequence and, for clustalW, the first 30 characters must be unique. It always should be but if you get an error relating to the names, this may be one of the potential causes.

### 3.2.7: Sequence/Dataset Filter Options

| Option | Description | Default |
|---|---|---|
| **gnspacc=T/F** | Convert sequences into gene_SPECIES__AccNum format wherever possible. | [True] |
| **backup=T/F** | Backup initial fasta file. Overwrites existing *.fas.bak. | [True] |
| **accnr=T/F** | Check for redundant Accession Numbers/Names on loading sequences. | [False] |
| **mapseq=FILE** | Maps sequences from FILE to sequences of same name | [None] |
| **filterseq=FILE** | Filters out sequences in given file (Log, Fasta or list of names) | [None] |
| **filterspec=FILE** | Filters out sequences according to species (codes) listed in given file | [None] |
| **unkspec=T/F** | Sequences of unknown species are allowed | [True] |
| **dblist=X,Y,..,Z** | List of databases in order of preference (good to bad) | [sprot,ipi,uniprot,trembl,ens_known,ens_novel,ens_scan] |
| **dbonly=T/F** | Only allow sequences from listed databases | [False] |
| **keepdesc=FILE** | Only keeps sequences with 1+ of text listed in given file in sequence description | [None] |
| **minlen=X** | Minimum length of sequences | [0] |
| **maxlen=X** | Maximum length of sequences (<=0 = No maximum) | [0] |
| **seqnr=T/F** | Make sequence Non-Redundant | [True] |
| **specnr=T/F** | Non-Redundancy within same species only | [True] |
| **nrid=X** | %Identity cut-off for Non-Redundancy | [99.0] |

| Option | Description | Default |
|---|---|---|
| blastcut=X | Maximum number of sequences to have in dataset (BLAST query against NR dataset.) | [0] |
| blaste=X | E-Value cut-off for BLAST searches (BLAST -e X) | [1e-4] |
| blastf=T/F | Complexity Filter (BLAST -F X) | [True] |
| qcover=X | Min %Coverage vs Query | [60.0] |
| gablam=T/F | Use Ordered BLAST alignments rather than ALIGN for qryid and pairid cut-offs | [True] |
| gabsim=T/F | Use %Similarity not %Identity for %ID cut-offs | [True] |
| pairid=X | %ID cut-off for any pair of sequences | [0.0] |
| qryid=X | %ID vs Query cut-off | [40.0] |
| maxgap=X | Maximum proportion of sequence that may be gaps compared to nearest neighbour (<=0 = No maximum) | [0.5] |

## 3.3: Single Sequence Alignment Quality (SAQ)

Because subfamilies should have at least three members for analyses, the next step is to remove those sequences that do not align well along most of their length with at least two other sequences. (This number is controlled by **saqc=X** and **haqmatrix=FILE** as described below.) This further filters out rogues and can be used to identify regions of the alignment that are questionable (**Figure 2**).

First, each sequence is compared with all others and it is determined which residues are "well-aligned". Blocks of well- and poorly-aligned residues are then identified and marked:

1. Each residue in each sequence is compared with all other sequences and given a score as determined by the amino acids at that position in the two sequences. By default, identities are given a score of 1, while differences are given a score of 0. (Alternative values can be used with the **haqmatrix=FILE** option.) If the total of these scores exceeds the cut-off (**saqc=X**) then that residue is considered to be "well-aligned" and the residue is given a score of 1 for that sequence. If not, the residue is given a score of 0. This is done for all residues in all sequences. For default settings, a "well aligned" residue must have at least two other sequences with the same amino acid at that position. **Note.** Gaps are treated asymmetrically in this process (**Figure 3**). For the focal sequence, gaps are skipped and ignored, while gaps in the sequence it is being compared to are treated as mismatches and given a score of 0. This enables the flanks around an indel to be marked as well-aligned, even if the indel itself is not.
2. Each sequence is then taken and well aligned blocks identified. These consist of 4 well-aligned residues in a maximum of a 10 aa block. (These cut-offs can be changed with **saqm=X** and **saqb=X**.) Each well-aligned residue is taken in turn and then the sequence moved along until four well-aligned residues have been counted. If the first and last of the four are up to 10 aa apart, the whole block is marked as being aligned. This is done for the whole sequence. Any regions not marked as aligned are therefore deemed to be badly aligned and the relevant sequence portions replace by Xs. These are remembered for later use in **PAQ**.

Steps 1 and 2 are repeated until no new residues are replaced with an X. (An X residue cannot be used to establish a residue in another sequence as being "well-aligned".)

3. Unless in "No Query" mode (**noquery=T**), any residues that have an X in the Query sequence are replaced with Xs in all the other sequences.
4. Each sequence is now considered in turn and either enforces its badly-aligned regions onto all other sequences or is removed. For each sequence, two things are calculated:
    a. The number of good residues that will be lost if the badly-aligned regions of that sequence were enforced on the whole alignment, (i.e. how many residues in other sequences will be changed to an X if the sequence of interest's bad regions are marked out.), and
    b. The number of good residues that will be lost if the sequence is removed. (i.e. the number of residues in the sequence not marked with an X.)

    **saqkl=X** and **saqks=X** are used to upweight these relevant measure by multiplying **a** and **b**, respectively. By default, if **a** is 3 times more than **b** (**saqkl=1** and **saqks=3**), the sequence is removed. Otherwise, all sequences have the relevant areas replaced with Xs.

5. Step 4 is then repeated until all sequences are dealt with.

If any sequences were removed, the remaining sequences have their Xs are replaced with the correct amino acids and are re-aligned, then the SAQ process is repeated. This is done until no new sequences are removed.

If you want more control over the removal of sequences during SAQ, then the **mansaq=T** option allows you to review each decision and manually over-ride them if desired. Accepting the default choices will give the same result as automated removal. The sequences are saved to ***.mansaq.fas**, with Xs in bad residues, for manual review during this process. This file is deleted again upon completion of the SAQ Cycle.

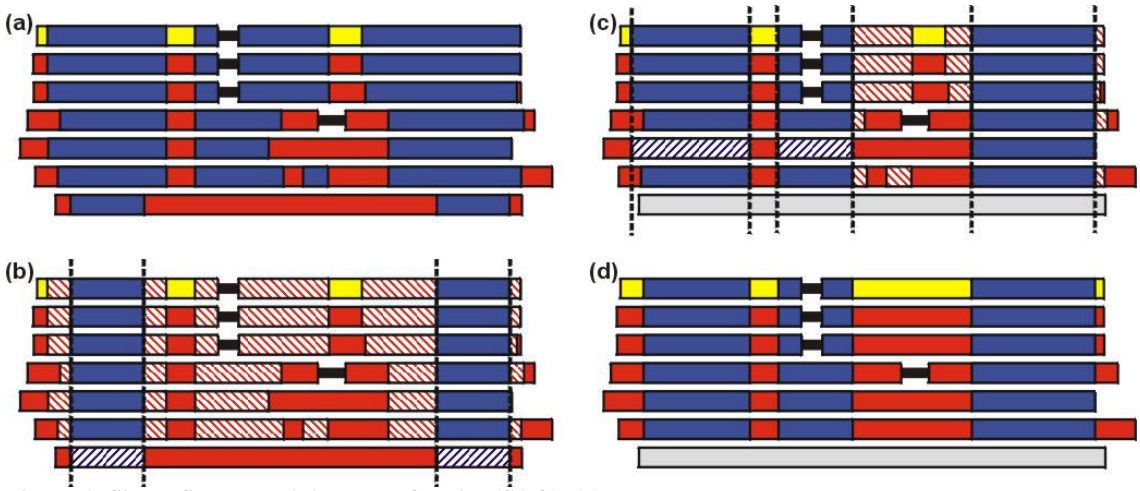


**Figure 2. Single Sequence Alignment Quality (SAQ). (a)** A 'Query' sequence (yellow) and six aligned homologues (red) (terminal "gaps" not shown) are compared to identify well-aligned blocks (blue). **(b)** For each sequence, the number of good residues lost if the badly-aligned regions of that sequence were enforced on the whole alignment (red hatching) is compared to the number of good residues that will be lost if the sequence is removed (blue hatching). **(c)** If keeping the sequence would remove too many good residues compared to losing it, the sequence is removed, otherwise **(d)** the well-aligned regions in other sequences are trimmed accordingly.

### 3.3.1: Single Sequence Alignment Quality (SAQ) Options

| Option | Description | Default |
|---|---|---|
| **saq=T/F** | Single Sequence AQ | [True] |
| **saqc=X** | Min score for a residue in SAQ (Default matrix: no. seqs to share residue). | [2] |
| **saqb=X** | SAQ Block length. | [10] |
| **saqm=X** | No. residues to match in SAQ Block. | [7] |
| **saqks=X** | Relative Weighting of keeping Sequences in SAQ. | [3] |
| **saqkl=X** | Relative Weighting of keeping Length in SAQ. | [1] |
| **mansaq=T/F** | Manual over-ride of sequence rejection decisions in SAQ. | [False] |

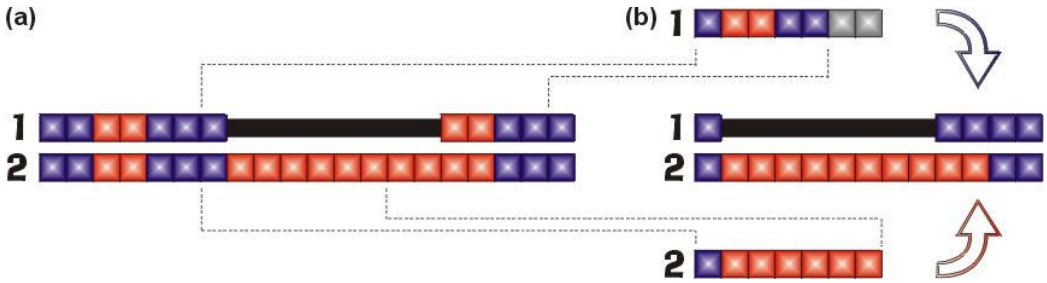


**Figure 3. Asymmetric Indel Treatment. (a)** Gaps are skipped when comparing sequence 1 to 2 but not the other way round. As a result, in a pairwise comparison, the region of the indel is marked a bad (red) in sequence 2, even though flanking regions are well-aligned (blue). **(b)** Gaps are ignored when identifying well-aligned blocks. In this example, three well-aligned residues must lie in a block of up to seven residues. Starting from the same residue flanking the indel, sequence 1 skips the gap and meets the conditions for good alignment, whereas sequence 2 does not.

## *3.4: Pairwise Alignment Quality (PAQ)*

Once the dataset is a stable size from **SAQ**, it undergoes a further alignment quality test. This is to try and identify regions of the alignment where sequences are well aligned with orthologues but these paralogous subfamilies are badly aligned with each other. By default, this is preceded by tree-based cleanup (see **3.5: Establishment of Subfamilies (ES)**) but this can be disabled using `prepaq=F`.

As its name suggests, PAQ (**Figure 4**) is comprised of pairwise combinations of sequences:

1. Each sequence is compared is a pairwise fashion with each other sequence. Similar to SAQ, blocks of well-aligned residues are identified, where the total score of compared amino acids exceeds a certain threshold for a given block size. Each residue is therefore marked as well- or badly-aligned for each pairwise comparison of sequences. By default, identities score 1 and different amino acids score 0, with 3+ identical amino acids needed in a block of upto seven residues to be scored as "well aligned". The scoring matrix can be altered using `haqmatrix=FILE` (see **2.2.3: HAQ Matrix**), while the score threshold and block size are controlled by `paqm=X` and `paqb=X`, respectively. Residues marked as badly aligned for individual sequences during **SAQ** Step 2 are still marked as badly aligned. **Note.** Gaps are treated asymmetrically in this process. For A vs B, gaps in A are skipped and ignored, while gaps in B (that correspond to residues in A) are treated as mismatches and given a score of 0. The comparison is made in both directions, so when B is compared back to A, gaps in B will be ignored and gaps in A are potential mismatches. This enables the flanks around an indel to be marked as well-aligned, even if the indel itself is not (**Figure 3**).
2. Each residue is taken in turn and, starting with the Query, links are made through well-aligned pairs of sequences for that residue to as many sequences as possible:
    a. All sequences except the query are marked as 'bad' for that residue.
    b. Sequences which are 'well-aligned' for that residue in their pairwise comparison with the query are marked as 'good'.
    c. Step (b) is repeated comparing sequences with all 'good' sequences until no new sequences are marked as 'good'.
    d. Once no more 'good' sequences are found, all the 'bad' sequences have the relevant residue replaced with an 'X'.

3. Because PAQ is quite conservative, an additional step includes blocks that may be well-aligned but have diverged. This is done by filling in contiguous blocks of sequence between 'well-aligned' anchors. Each block of 'bad' residues is considered in turn. If it contains no *internal* gaps in any sequence in the MSA (terminal gaps are ignored to allow for the inclusion of sequence fragments), and is flanked by well-aligned regions, then it is considered to be well-aligned. This can be switched off with the **anchors=F** option.
4. Unless in "No Query" mode (**noquery=T**), any residues that have an X in the Query sequence are replaced with Xs in all the other sequences.
5. As for **SAQ**, each sequence is now considered in turn and either enforces its badly-aligned regions onto all other sequences or is removed. For each sequence, two things are calculated:
    a. The number of good residues that will be lost if the badly-aligned regions of that sequence were enforced on the whole alignment, (i.e. how many residues in other sequences will be changed to an X if the sequence of interest's bad regions are marked out.), and
    b. The number of good residues that will be lost if the sequence is removed. (i.e. the number of residues in the sequence not marked with an X.)

    **paqkl=X** and **paqks=X** are used to upweight these relevant measure by multiplying **a** and **b**, respectively. By default, if **a** is 3 times more than **b** (**paqkl=1** and **paqks=3**), the sequence is removed. Otherwise, all sequences have the relevant areas replaced with Xs.
6. Step 5 is then repeated until all sequences are dealt with.

If any sequences are removed, both SAQ **and** PAQ are repeated. This continues until no more sequences are removed. Badly aligned regions - marked with 'X's - are then stored in a file for later use and the sequences restored to their original amino acids at each position. The original sequences are then used to draw a tree and **Establish Subfamilies**.

If you want more control over the removal of sequences during PAQ, then the **manpaq=T** option allows you to review each decision and manually over-ride them if desired. Accepting the default choices will give the same result as automated removal. The sequences are saved to ***.manpaq.fas**, with Xs in bad residues, for manual review during this process. This file is deleted again upon completion of the PAQ Cycle.

### 3.4.1: Pairwise Alignment Quality (PAQ) Options

| Option | Description | Default |
|---|---|---|
| **prepaq=T/F** | PrePAQ tree grouping | [True] |
| **paq=T/F** | Pairwise AQ | [True] |
| **paqb=X** | PAQ Block length. | [7] |
| **paqm=X** | Min score in PAQ Block (Default: No. residues to match). | [3] |
| **paqks=X** | Relative Weighting of keeping Sequences in PAQ. | [3] |
| **paqkl=X** | Relative Weighting of keeping Length in PAQ. | [1] |
| **manpaq=T/F** | Manual over-ride of sequence rejection decisions. | [False] |
| **anchors=T/F** | Use conserved 'anchors' to extend well-aligned regions in PAQ | [True] |

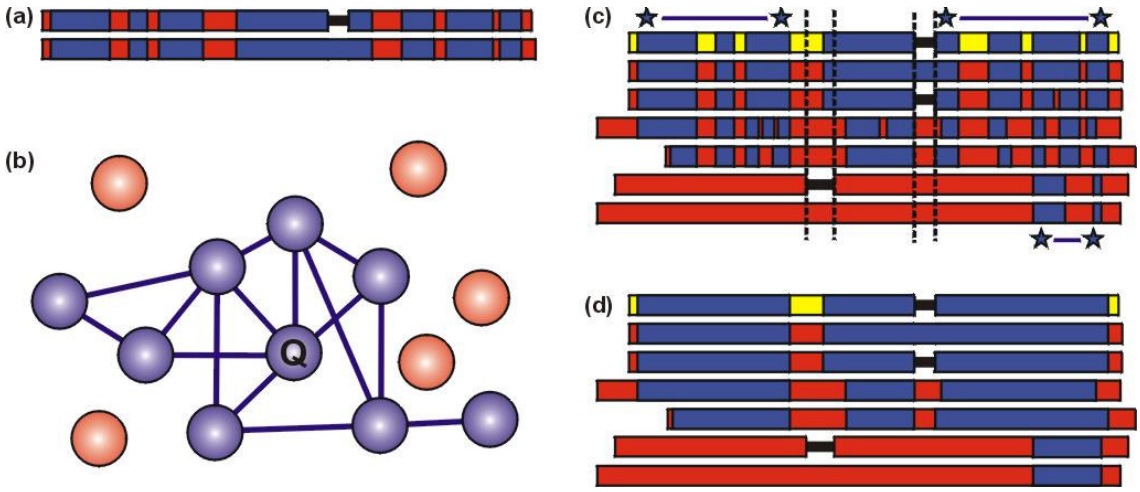


**Figure 4. Pairwise Alignment Quality. (a)** Sequences are compared in a pair-wise fashion and blocks of well-aligned residues marked (blue; badly-aligned residues in red). **(b)** Each residue is taken in turn and a network made of the query (Q) and any sequences that link to the query via well-aligned pair-wise comparisons (blue). **(c)** This produces blocks of good (blue) and bad (red) residues. In situations where blocks of bad sequence are flanked by blocks of good sequence (blue stars), these are extended across ungapped regions of the alignment to account for local divergence in an otherwise conserved region. This is done individually for each sequence **(d)** before sequence removal and marking of bad regions as in SAQ.

## *3.5: Establishment of Subfamilies (ES)*

It is important for certain downstream evolutionary analyses, such as functional specificity predictions, to establish subfamily relationships for the sequences. An important part of this procedure is the correct determination of representative sequences from species that have several possible sequences for a given paralogue. This is particularly common where multiple databases have been used to collect sequences and a given species may have, for example, a SwissProt entry and an EnsEMBL peptide, or two different EnsEMBL gene predictions.

Establishment of Subfamilies is semi-automated in a three step process:

1. ClustalW or PHYLIP is used to generate a bootstrapped Tree (see **3.5.1: Tree Drawing with ClustalW or PHYLIP**). This is rooted as determined by the user.
2. Sequences are grouped according to user parameters and/or manual decisions
3. Sequence groups are manually reviewed to remove unwanted sequence variants

### 3.5.1: Tree Drawing with ClustalW or PHYLIP

By default, HAQESAC will use ClustalW to generate a bootstrapped Neighbour-Joining (NJ) tree. The main Establishment of Subfamilies uses 1000 bootstraps and corrects branch lengths for multiple hits. A reduced version is used to edit the dataset prior to PAQ, however, (unless `prepaq=F`), which uses 100 bootstraps and does not correct branch lengths for multiple hits.

Alternatively, a program from PHYLIP may be selected using the `maketree=X` option. Acceptable values for `X` are:

- `clustalw` = ClustalW NJ method
- `neighbor` = PHYLIP NJ method

- **upgma** = PHYLIP UPGMA (neighbor) method

- **fitch** = PHYLIP Fitch method

- **kitsch** = PHYLIP Kitsch (clock) method

- **protpars** = PHYLIP MP method

- **proml** = PHYLIP ML method

- **PATH** = A path to a different tree program/script can be given. This should accept ClustalW parameters.

If using one of the PHYLIP programs, HAQESAC will, by default, randomise input sequence order. Otherwise, default settings are used. HAQESAC can feed PHYLIP additional options in a file, as described in PHYLIP documentation, using **phyoptions=FILE**. If using neighbor, upgma, fitch or kitch, additional options for PROTDIST can be fed to PHYLIP using **protdist=FILE**. There is no need to include the 'Y' option in these files to run the programs as these will be added by HAQESAC.

HAQESAC will use PHYLIP to draw 'true bootstrapped' trees, rather than using the consense program in PHYLIP. This is done by creating random alignment datasets of the same length (with replacement), running these through the same programs as the "real" data, and bootstrapping each branch using these trees within HAQESAC itself.

**NB.** Using PHYLIP may give more accurate trees than ClustalW but they are considerably slower to generate. In most situations, it is probably best to use ClustalW for the main data cleanup and then run HAQESAC again with **cleanup=F haq=F maketree=X** to draw a more accurate PHYLIP tree at the end.

### 3.5.2: Tree Rooting

HAQESAC provides five rooting options. By default, midpoint rooting is used (**root=mid**), where the root is placed halfway between the most distant tips. (This rooting is sensitive to long branch attraction but tip sequences with very long branches are likely to be filtered out by the HAQ cleanup.) Other options are manually placing the root using the Root Menu described below (**root=man**), randomly rooting with (**root=ranwt**) or without (**root=ran**) weighting for branch lengths (not recommended), or rooting from a file of outgroup sequences (**root=FILE**). If a file is given, HAQESAC will place the root on the branch that includes all sequences in the file in one half of the tree, while the other half contains as many sequences not in the file as possible.

Manual rooting gives a menu that, in addition to the above automatic rootings, has three manual rooting options:

```
<0> Keep current rooting.
---
<1> Midpoint Root.
<2> Root from Outgroup File.
<3> Make Outgroup File.
<4> Root on Branch.
<5> Manual rooting using text Tree (with editing options).
<6> Random Root.
<7> Random Root (branch-length weighted).
---
<8> Unroot.
```

These options are as follows:

0. Keep Current Rooting. Proceed onto next stage of ES with current rooting. (If tree is rooted.)
1. Midpoint Rooting. As **root=mid**, above.
2. Root From Outgroup. As **root=FILE**, above.
3. Make Outgroup File. Selects sequences to form outgroup file for future rooting.
4. Root on branch. This option goes through each branch, showing the two clades that the tree would be divided into if the branch was used for the root. Select <R> to root on that branch, <N> to move to the next branch, <P> to return to the previous branch, <I> to skip to internal branches, and <Q> to quit back to the Rooting Menu.
5. Manual rooting using Text Tree. This will show a text tree on screen with options to edit and/or root the tree (see **3.5.3: Editing the Tree**).
6. Random root. Places the root on a random branch with equal rooting probabilities for each branch (**root=ran**).
7. Random root (branch length weighted). Places the root on a random branch with rooting probabilities proportional to branch length (**root=ranwt**).
8. Unroot. Clears current rooting.

### 3.5.3: Editing the Tree

This shows a text representation of the tree and some editing options. Note that this can be unwieldy for large trees. When rooting the tree, the following options are given:

```
Display: <C>ollapse, <E>xpand, <F>lip, <N>ame Clade, <Z>oom,
<O>ptions, <D>escriptions, <S>ave to file, <Q>uit

Edit: <B>ranch, <T>erminal node, <P>rune, <R>oot on Branch,
<U>nroot, Root <M>enu
```

In addition to these options, you can enter a node number. This will Collapse or Expand that node as appropriate (see below).

**Display Options:**

**Collapse**. Collapse a clade to a single node on the displayed tree.

**Expand.** Reverses Collapse.

**Flip**. Vertically flips a clade around the chosen node.

**Name** Clade. Gives a name to a collapsed node.

**Zoom**. Zooms the displayed tree in on a particular clade. The chosen node becomes the 'root' for on screen display only. This is useful for large trees. Giving node '0' will zoom back out to the whole tree.

**Options**. Gives options for changing the way the tree is displayed.

**Descriptions**. Toggles whether description part of sequence name is displayed.

**Save to file**. Saves the current text tree to a text file.

**Quit**. Returns to Root Menu.

**Edit Options:**

`Branch`. Allows you to edit branch attriubutes (length, bootstrap etc.)

`Terminal node`. Allows you to edit sequence details (name, species etc.)

`Prune`. Permanently removes and node and its descendant clade.

`Root on Branch`. Roots the tree on the selected branch.

`Unroot`. Unroots the tree.

`Root Menu`. Return to the Root Menu.

### 3.5.4: Sequence Grouping Overview

The Establishment of Subfamilies (sequence grouping) phase of the program has two goals:

1. Removal of sequence variants (if unwanted). These may be redundancies left by using multiple databases that have not been cleaned up using the program's redundancy filter, or splice variants or 'in-paralogs' (the results of genome-specific duplications).
2. Subdividing the sequences into groups for e.g. specificity analysis or orthologous groups for evolutionary conservation analysis etc.

By default, the program aims to retain related subfamilies that have at least three members and a single representative sequence from each species. If this is not what you want, then you will need to change the settings (`mfs=X`, see **3.5.5: Important Grouping Options**).

The key to the grouping is to look at the tree and the species codes (see **3.2.6: The gnspacc=T/F Option**). The first time around, you are generally better to manually group sequences as the presence of variants and in-paralogs etc. will muck up automatic grouping. After that, then try to group by duplication. (The `group=dup` option can be used in the command-line but over-ridden the first time by editing the groups manually from the Grouping Menu (**3.5.7: The Grouping Menu**, below).

### 3.5.5: Important Grouping Options

There are four main options that affect how grouping is directed:

1. **FAMILY SIZE.** (`mfs=X`) This is the minimum number of sequences there should be in each family. For specificity analysis, or solid evolutionary inference, this is probably 3 (the default). For a more relaxed conservation analysis, set `mfs=2`. `mfs=1` allows groups of '1' but this is really the same as the using the Orphans setting (below) and is therefore not recommended.
2. **FAMILY NUMBER.** (`fam=X`) This is the minimum number of families that HAQESAC will accept as OK. For specificity analysis, this is 2. By default, this is 0 and so no grouping is necessary (though it can still be useful to get rid of variants etc.).
3. **ORPHANS.** (`orphans=T/F`) Orphans are sequences not in any groups. For specificity and conservation within orthologues, these sequences are useless and can be dumped. For more general evolutionary studies, they may be important and can be retained (the default). If you trust the input sequences, then I would suggest keeping Orphans. If, however, there are a lot of potentially spurious sequences (e.g. EnsEMBL genscans) then orphans should probably be dumped as anything only present in one species is more likely to be noise. If in doubt, leave `orphans=T` and manually remove them at the group review stage (see **3.5.10: Reviewing Sequence Groups**).

4. **VARIANTS.** (`allowvar=T/F`) This option dictates whether groups may contain multiple sequences from the same family. If you are confident that the dataset you are searching contains no redundancy and you want to retain splice variants and in-paralogs then set `allowvar=T`. If, however, you are compiling a dataset from several complimentary databases and you only want the best representative sequence for each species, set `allowvar=F`.

### 3.5.6: Grouping Methods

In addition to the Grouping 'Rules' above, there are several different ways of grouping the sequences using the `group=X` option (or the Grouping Menu):

| Method | Command | Description |
|---|---|---|
| Manual | `group=man` | This is done interactively with a tree, selecting the node number that is at the base of the group to group or ungroup sequences. See **3.5.8: Manual Group Editing/Selection** for details. |
| Duplication | `group=dup` | The program tries to automatically group sequences into orthologous groups according to the options that are set. By default, all duplication nodes will be considered. If the `groupspec=X` option is used then only duplications of the relevant species will be used. See **3.5.9: Duplication Grouping** for details. |
| Query | `group=qry` | Duplication Grouping only considering duplications of the Query sequence species |
| One Group | `group=one` | All sequences in a single group |
| No Groups | `group=None` | No sequences are grouped. (case sensitive) |
| Load Groups | `group=FILE` | Load groups from `FILE`. (See **2.3: Output**.) |

### 3.5.7: The Grouping Menu

The Grouping Menu allows interactive control and review of the sequence grouping procedure. The Menu first summarises the grouping 'Rules', which are set by command-line parameters:

```
#*# Grouping Options #*#
[Bootstrap cut-off: 800 (0.8)]          (bootcut=X)
[Min Family Size: 3]                    (mfs=X)
[Min Family Number: 0]                  (fam=X)
[Group by Query Species: False (None)] (groupspec=X or group=qry)
[Allow Sequence Variants: False]        (allowvar=T/F)
[Allow Orphan Sequences: True]          (orphans=T/F)
```

Next, a summary of the current grouping selection is given, with the number of groups and the number of orphan sequences.

```
#*# Grouping Summary #*#
Currently X groups. (Y Orphans)
```

If there are any groups, they will be listed along with their size and the sequences they contain.

After these summaries is the menu itself:

```
<K>eep current grouping. (Current Grouping OK)
---
<O>ptions (Change Grouping 'rules').
---
<L>oad Grouping.
<D>uplication Grouping. (All Duplications)
<M>anual Grouping.
<N>o Grouping (MinFamNum=0).
<A>ll sequences in one group.
<E>dit Grouping (Manual)
---
<R>eview Groups
<G>roup in SeqList (Reorder)
<P>urge Orphans Sequences (not in a group)
<S>ave Groups
---
<Q>uit
```

| Choice | Option | Description |
|---|---|---|
| **K** or **ENTER** | Keep Current Grouping | If the current grouping breaks any of the rules, then *** **WARNING: Current Grouping breaks 1+ rules.** *** will be displayed instead of **(Current Grouping OK)**. Breaking the rules includes: having insufficient groups (**fam=X**), having a group with insufficient sequences (**mfs=X**), having a group without sufficient bootstrap support (**bootcut=X**), having species variants in a group when **allowvar=F**, having orphan sequences when **orphans=F**. Groups may still be accepted when rules are broken but HAQESAC will take you into the **Review Groups** to sort out (or manually accept) any problem groupings. Upon keeping groups, you will be given the option to save the groups to a file (See **2.3: Output**.). |
| **O** | Options | This allows you to change the displayed Grouping Options ('rules'). If you change the options using the menu you will have to re-group the sequences for them to have an effect on automated grouping. |
| **L** | Load Grouping | Loads grouping from file. (See **2.3: Output**.) |
| **D** | Duplication Grouping | If **groupspec=X** is used, **(All Duplications)** will be replaced with **(X Duplications)**, where X is the species code. See **3.5.9: Duplication Grouping** for details. |
| **M** | Manual Grouping | Clears all groups and instigates manual grouping. See **3.5.8: Manual Group Editing/Selection** for details. |
| **N** | No Grouping | Clears all grouping. |
| **A** | All sequences in one group | Exactly. |
| **E** | Edit Grouping | This is like Manual Grouping except that the current groups are not cleared. This allows you to keep the ones you like and change the ones you don't. |

| Choice | Option | Description |
|---|---|---|
| R | Review Groups | See **3.5.10: Reviewing Sequence Groups**. |
| G | Group in SeqList | Reorders the sequences in the file to match the grouping, i.e. sequences in the same group will be put next to each other. |
| P | Purge Orphans Sequences | Deletes any sequences not in a group. |
| S | Save Groups | Saves groups to a file (see **2.3: Output**). Group names can be given if desired (see **3.5.8: Manual Group Editing/Selection**). |
| Q | Quit | Quit the Grouping Menu. Depending on the circumstances, this may behave like **Keep Current Grouping** or, if rules are broken, take you to the **Review Groups**. |

### 3.5.8: Manual Group Editing / Selection

This is the essentially the same as the manual tree editing for rooting (see **3.5.3: Editing the Tree**) with two slight changes:

1. The Root options should not be given
2. `<C>ollapse` and `<E>xpand` have become `<C>ollapse/Group` and `<E>xpand/Ungroup`.

Now, collapsing a node will create a sequence group at that node, which can be named if desired. Expanding a node will remove the grouping at that node. Note that groups cannot overlap, so any groups formed at nodes deep in the tree will automatically remove any descendant groupings.

### 3.5.9: Duplication Grouping

Duplication grouping is an automated attempt to group sequences into groups of orthologous proteins, i.e. the same protein in different species. Duplication grouping is executed as follows:

1. Duplication nodes are identified using the species information of the input sequences. If both descendant branches from a node have descendant sequences of the same species, then that node is identified as a duplication node. If `groupspec=X` or `group=qry` are used, then only the relevant species are considered.
2. Each node is considered and assigned as a grouping node if:
    a. It is a duplication node
    b. The bootstrap value of the ancestral branch is high enough (`bootcut=X`)
    c. The number of descendant sequences is sufficient (`mfs=X`)
3. Each grouping node is reconsidered and removed if any of its descendants are also valid groups. This prevents an ancient duplication node wiping out all the more recent duplication groupings.

**Note.** The presence of sequence variants and branches with poor bootstrap support can lead to strange results from automated duplication grouping. It is therefore highly recommended to manually edit the groups the first time round.

### 3.5.10: Reviewing Sequence Groups

The first and last times around during **ES**, and if any rules are broken, you will also have to review each group and decide which variants to keep (if there are multiple sequences from the same species).

The Review Groups stage is broken down into stages:

1. The group is displayed as a fragment of a text tree. This is so that you can use phylogenetic relationships to guide your decisions regarding sequence inclusion or exclusion. If you decide that the grouping is bad after all, then retain all variants and return to the Grouping Menu (see below).
2. A group 'master' sequence will be chosen. This is the Query sequence if one is given, or a sequence of the same species. If there is no query, or no sequence in the group from the same species, then the master will be selected as the sequence with the highest mean %ID (based on the multiple sequence alignment (MSA)) to the other sequences in the group. Each sequence is then summarised, relative to the Group Master for three criteria, which are displayed as percentage values with the actual number of amino acids in brackets afterwards:

```
SeqName: X.XX% ID (X aa); X.XX% Gaps (X aa); X.XX% Extra (X aa); vs
GroupMaster
```

   a. **% ID.** This is the percentage of the variant sequence that is identical (in the MSA) in the group 'master'.
   b. **% Gaps.** This is the percentage of the overlap between the two sequences that is a gap in the variant and a residue in the group master. (Gaps in both are ignored.)
   c. **% Extra.** This is the percentage of the variant that overlaps with a gap in the Group Master.

3. If the group master is itself a sequence variant, then the "best" sequence of that species must be selected as the group master. By default, HAQESAC will suggest the sequence with the most identical residues (X aa), as determined by the MSA, compared to the Query sequence. Selecting `0` or simply hitting **`ENTER`** will choose this sequence and the other variants will be removed. Alternatively, you can keep all sequence variants but this sequence will remain as group 'master' for the next stage.
4. Once the group 'master' is chosen, any other sequences from species with more than one variant in the group will be compared in a similar fashion. This time, however, HAQESAC will suggest the sequence with the highest identity to the group master. This is to maximise the chance of getting the same splice variant, for example, across different species. Again, selecting `0` or simply hitting **`ENTER`** will choose this sequence and the other variants will be removed. Alternatively, you can keep all sequence variants.

HAQESAC will suggest what it thinks is the best sequence based on certain rules but it is important at this stage to think also about what you are trying to achieve by removing variants of the same species and not to simply accept the suggested sequence without looking. You may choose to keep a SwissProt sequence, for example, even though it has slightly lower %ID to the group 'master' than a trEMBL or ensEMBL one.

Once all the variants are dealt with, there are a number of options presented:

```
Subfamily Options:

<P>revious, <T>ree Edit, <G>ene Edit, <U>ngroup, <D>elete Group,
Group <M>enu, <Q>uit Review, <N>ext, <F>inish Review
```

Note that not all of these options are displayed every time. (The `<P>revious` option for example, will only be there when there *is* a previous group.)

| Choice | Option | Description |
|---|---|---|
| `P` | Previous | Return to previous group. |
| `T` | Tree Edit | Edit details of the group using the Tree Edit functions. See **3.5.3: Editing the Tree**. |
| `G` | Gene Edit | This allows you to give a new 'Gene' annotation to the sequences. See **3.2.6: The gnspacc=T/F Option** for details. This will not change the 'Gene' part of SwissProt IDs. |
| `U` | Ungroup | This removes this group, converting the sequences to orphans. |
| `D` | Delete Group | This removes the group and deletes all the sequences in it from the dataset. |
| `M` | Group Menu | Returns to the Group Menu. (See **3.5.7: The Grouping Menu**, above.) |
| `Q` | Quit Review | This will return to the Group Menu or proceed to the next stage of HAQESAC, depending on what initiated the review. |
| `N` or `ENTER` | Next | This will proceed to the next group if there is one. |
| `F` or `ENTER` | Finish Review | This will finish the review, if this is the last group. |

Upon quitting or finishing the review, you are given the choice to enter a sentence for the log file, which may be useful if you wish to annotate any abnormal decisions made. There is also the option to save a backup of the current alignment and tree.

### 3.5.11: Establishment of Subfamilies (and PrePAQ) Options

| Option | Description | Default |
|---|---|---|
| **es=T/F** | Establishment of Subfamilies | [True] |
| **root=X** | Rooting of tree, where **X** is:<br>- **mid** = midpoint root tree.<br>- **ran** = random branch.<br>- **ranwt** = random branch, weighted by branch lengths.<br>- **man** = always ask for rooting options (unless i<0).<br>- **FILE** = with seqs in **FILE** as outgroup. (Any option other than above) | [mid] |
| **bootcut=X** | cut-off percentage of tree bootstraps for grouping. | |
| **mfs=X** | minimum family size | [3] |
| **fam=X** | minimum number of families (If 0, no subfam grouping) | [0] |
| **orphan=T/F** | Whether orphans sequences (not in subfam) allowed. | [True] |
| **allowvar=T/F** | Allow variants of same species within a group. | [False] |
| **groupspec=X** | Species for duplication grouping | [None] |
| **group=X** | Grouping of tree, where **X** is<br>- **man** = manual grouping (unless i<0).<br>- **dup** = duplication (all species unless groupspec specified).<br>- **qry** = duplication with species of Query sequence (or Sequence 1) of treeseq.<br>- **one** = all sequences in one group.<br>- **None** = no group (case sensitive).<br>- **FILE** = load groups from file | [None] |
| **maketree=X** | Program for making tree:<br>- **clustalw** = ClustalW NJ method<br>- **neighbor** = PHYLIP NJ method<br>- **upgma** = PHYLIP UPGMA (neighbor) method<br>- **fitch** = PHYLIP Fitch method<br>- **kitsch** = PHYLIP Kitsch (clock) method<br>- **protpars** = PHYLIP MP method<br>- **proml** = PHYLIP ML method<br>- **PATH** = Alternatively, a path to a different tree program/script can be given. This should accept ClustalW parameters. | [clustalw] |
| **phyoptions=FILE** | File containing extra Phylip tree-making options ('batch running') to use. | [None] |
| **protdist=FILE** | File containing extra Phylip PROTDIST options ('batch running') to use | [None] |

## 3.6: Ancestral Sequence Reconstruction (AC)

Ancestor Construction is performed using the Gapped Ancestral Sequence Prediction (GASP) algorithm (Edwards and Shields 2004). Details of this can be found in the GASP Manual and at the website: http://www.bioinformatics.rcsi.ie/~redwards/gasp/

### 3.6.1: Ancestor Construction (GASP) Options

| Option | Description | Default |
|---|---|---|
| **ac=T/F** | Ancestor Construction (GASP) | [True] |
| **pamfile=FILE** | Sets PAM1 input file | [jones.pam] |
| **pammax=X** | Initial maximum PAM matrix to generate | [100] |
| **pamcut=X** | Absolute maximum PAM matrix | [1000] |
| **fixpam=X** | PAM distance fixed to **X** | [100]. |
| **rarecut=X** | Rare aa cut-off | [0.05]. |
| **fixup=T/F** | Fix AAs on way up (keep probabilities) | [True]. |
| **fixdown=T/F** | Fix AAs on initial pass down tree | [False]. |
| **ordered=T/F** | Order ancestral sequence output by node number | [False]. |
| **pamtree=T/F** | Calculate and output ancestral tree with PAM distances | [True]. |
| **desconly=T/F** | Limits ancestral AAs to those found in descendants | [True]. |
| **xpass=X** | How many extra passes to make down & up tree after initial GASP | [1]. |

## 3.7: 'No Query' Mode

As can be seen from the way the algorithms are constructed, HAQESAC is designed to work with a focal Query protein. If there is no such protein, then this can be disabled using the **noquery=T** option. Because the query is the focus, anything badly aligned in the query will be mapped onto the other sequences before other comparisons are made, which will include regions outside of the query. The **noquery=T** option also disables this. To disable this and keep a designated query for PAQ, the **query=X** and **noquery=T** options may be combined without problem. (To truly have no designated query, make sure there is no **query=X** parameter.

## 3.8: Replacing Components with other Programs

The most important functions performed by the external programs called by HAQESAC are alignment and tree-drawing. This section lists some ways to incorporate alternative programs for these functions into HAQESAC. I am always interested to add more functionality, so if there is a program you would like to use instead of those listed, then please contact me and I may be able to add them in a more controlled fashion than below.

### 3.8.1: Alignment programs

By default, HAQESAC will use MUSCLE (Edgar 2004) for alignments as I have found this to be both fast and accurate. There can be problems with memory allocation for larger datasets and so and ClustalW (Higgins and Sharp 1988, Thompson, et al. 1994) is used for large datasets above a certain total number of residues (as determined by the **cwcut=X** parameter). Either of these programs can be replaced, however, by another program that uses the same command-line format used in HAQESAC to call the programs.

For MUSCLE, the system call is:

`muscle` **-in INFILE -out OUTFILE**, where **INFILE** and **OUTFILE** are both fasta format.

The path to MUSCLE can be changed to redirect to another program using the **muscle=PATH** option.

For ClustalW, the system call is:

`clustalw` **INFILE**, where **INFILE** is in fasta format (***.fas**) and the output file (***.aln**) is in ClustalW align format.

The path to ClustalW can be changed to redirect to another program using the **clustalw=PATH** option.

### 3.8.2: Tree-drawing programs

HAQESAC uses the Neighbour-joining method implemented in ClustalW for drawing trees. Although this is not the most accurate phylogeny construction algorithm around, it is fast and efficient and reasonable for trees of closely-related sequences with high bootstrap support, such as those HAQESAC was designed to build and work with. Again, this program can be replaced with another using the **maketree=PATH** option. The system call used by HAQESAC is:

`clustalw` **-infile=INFILE -bootstrap=X -seed=X [-kimura]** for UNIX, or

`clustalw` **INFILE -bootstrap=X -seed=X [-kimura]** for Windows, where **INFILE** is in fasta format (***.fas**) and the output file (***.phb**) is in bootstrapped Phylip format (I think).

It *should* work to have a program output a Newick Standard Format tree as ***.nsf** but I have not tested that.

### 3.8.3: Wrapper scripts

If the chosen program does not accept the same input/output commands/formats then a wrapper script should be written. It is suggested to use Perl or Python for this. Although I cannot promise help in every suggestion, you are welcome to e-mail me for help with this and I will see what I can do.

### 3.8.4: Incorporating Other Programs into the Python Code

If you are feeling brave, you can actually edit the Python modules themselves. The key methods for this are **rje_seq.muscleAln()**, **rje_seq.clustalAln()** and **rje_tree.makeTree()**. Obviously, I cannot promise to give technical support for any changes that are made but, if you know what you are doing, you should be OK and I will help where I can.

# 4: HAQESAC and Evolutionary Analyses

## 4.1: Building a Dataset using BLAST

When time allows, this section will be expanded to give some general advice but for now it is assumed that the user has some experience in generating datasets of putative homologues, using BLAST or some other tool. If such a dataset has not been generated, however, and the file of sequences to search is local on the machine, HAQESAC can be used to generate the starting dataset using the following parameters. For this, you will need to supply a query sequence and a search database.

The search database is given using the **qblast=FILE** option. This will be BLAST formatted with formatdb if it is not already. The query may be identified in one of two ways. If it is one of the sequences in the qblast database, then the unique identifier can be used to extract it using fastacmd with the **qfastacmd=X** option. Alternatively, a separate file containing the sequence may be given with **qseqfile=FILE** and the query identified using **query=X** as normal. (Note that if **qfastacmd=X** is used to extract the query from the **qblast** database, **query=X** may also be needed for the rest of HAQESAC, taking the **gnspacc** setting into account (see **3.2.6: The gnspacc=T/F Option**))

The query will then be used to BLAST against the database. Default options are e=$1e^{-7}$, 1000 hits reported, complexity filter on. See the documentation of **rje_blast.py** for details of how to change these options.

## 4.2: Example Parameter Settings

This section will list some example uses of HAQESAC and the suggested parameter settings. This is a work in progress and will be added to in response to user questions. Here are some general pointers, though.

### 4.2.1: Orthologous datasets

Where the key is to generate sets of orthologues from multiple and/or redundant databases, it is recommended to set: *mfs=2 fam=1 orphans=T allowvar=F*

### 4.2.2: Large datasets

The SAQ and PAQ parameters are typically set to deal with (or, at least, end up with) quite small datasets of around a dozen proteins. If your dataset consists of a lot of closely related proteins, you may find that you lose a lot of proteins from SAQ and PAQ because the default saqks and paqks settings are too small. It may be a good idea to start off with these settings quite high (10+) and then, once you have reduced the dataset somewhat, run HAQESAC again with the smaller settings. (Alternatively, leave the settings high and run with **mansaq=T** and/or **manpaq=T**.)

### 4.2.3: Non-redundant datasets

If you are confident that your starting data is non-redundant and you want to eliminate sequences based on bad alignment only, set: *seqnr=F, allowvar=T*

# 5: Appendices

## 5.1: Appendix I: Command-line Options

### 5.1.1: How to Use this Section

This section lists all the Command-line options than may be of use when using HAQESAC. Note that different options are associated with different modules. These are indicated by the name of the module given (**in brackets**). Default values are given [in square brackets]. Not all the options for a given module are listed here but can be found by printing the `__doc__` attribute of the module at a Python prompt, or using the **`help`** option:

*`print haqesac.__doc__`* (in Python)

*`python haqesac.py help`* (commandline)

Note that the **`help`** option of HAQESAC will list all the options for all the relevant modules.

This section has not been completed. For now, the listing provided as part of the module documentation is given. This shall be expanded with time. (Hopefully soon.) In the meantime, please contact me if you want any further details of a specific option and/or advice as to when (not) to use it.

### 5.1.2: Option Types

There are essentially three types of command-line option:

1. Those that require a value (numerical or text), **`option=X`**. Those that require a filename as the value will be witten: **`option=FILE`**
2. True/False (On/Off) options, **`option=T/F`**. For these options:
    a. **`option=F`** and **`option=F`**alse are the same and turn the option off
    b. **`option`**, **`option=T`** and **`option=True`** are the same and turn the option on
3. List options. These are like the value options but have multiple values, separated by commas: **`option=X,Y`**. Where **`..`** is used, the number elements is optional, e.g. **`option=X,Y,..,Z`** could take **`option=X`** or **`option=A,B,C,D`**. Where **`option=LIST`** is used, the number of elements is optional and **`LIST`** could actually be the name of a file containing the list of elements.

### 5.1.3: INI Files

As well as feeding commands in on the command-line, any options listed can also be save in a plain text file and called using the option **`ini=FILE`**. Automatically, the program will read in any options from the file **haqesac.ini** and **rje.ini**, if present.

### 5.1.4: Option Precedence

Later options will supersede earlier ones if they are mutually exclusive. Options from an ini file will be inserted into the list at the point the ini file is called. (At the start for **haqesac.ini**.) This means that ini file options can be over-ruled, e.g.

*`haqesac.py ini=eg.ini i=1`* will supersede any interactivity setting in **eg.ini** with **`i=1`**.

*`haqesac.py i=1 ini=eg.ini`* will use any interactivity setting in **eg.ini** and over-rule **`i=1`**.

### 5.1.5: Command-line Options

| Option | Description | Default | Module |
|---|---|---|---|
| | **General Dataset Input/Output** | | |
| Seqin=FILE | Loads sequences from FILE | [None] | rje_seq |
| Query=X | Selects query sequence by name (or part of name, e.g. Accession Number) | [None] | rje_seq |
| basefile=X | Basic 'root' for all files X.* | ['root' of seqin=FILE] | rje_seq |
| v=X | Sets verbosity (-1 for silent) | [0] | rje |
| i=X | Sets interactivity (-1 for full auto) | [0] | rje |
| d=X | Data output level (0-3) | [1] | haqesac |
| log=FILE | Redirect log to FILE | [haqesac.log or basefile.log] | rje |
| newlog=T/F | Create new log file. | [False] | rje |
| | **Pre-HAQESAC Data selection** | | |
| qseqfile=FILE | Sequence file from which to extract query (query=X) and peform BLAST | [None] | haqesac |
| qblast=FILE | BLAST database against which to BLAST query (['blaste=1e-7','blastv=1000,blastb=0'] see rje_blast.py options) | [None] | haqesac |
| qfastacmd=X | Extract query X from qblast database using fastacmd (may also need query=X) | [None] | haqesac |
| | **Pre-HAQESAC Sequence Filtering** | | |
| gnspacc=T/F | Convert sequences into gene_SPECIES__AccNum format wherever possible. | [True] | rje_seq |
| backup=T/F | Whether to backup initial fasta file. Will overwrite existing *.fas.bak. | [True] | haqesac |
| accnr=T/F | Check for redundant Accession Numbers/Names on loading sequences. | [False] | rje_seq |
| filterseq=FILE | Filters out sequences in given file (Log, Fasta or list of names) | [None] | rje_seq |
| filterspec=FILE | Filters out sequences according to species (codes) listed in given file | [None] | rje_seq |
| unkspec=T/F | Whether sequences of unknown species are allowed | [True] | rje_seq |
| dblist=X,Y,..,Z | List of databases in order of preference (good to bad) | | rje_seq |
| dbonly=T/F | Whether to only allow sequences from listed databases | [False] | rje_seq |

| Option | Description | Default | Module |
|---|---|---|---|
| keepdesc=FILE | Only keeps sequences with 1+ of text listed in given file in sequence description | [None] | rje_seq |
| minlen=X | Minimum length of sequences | [0] | rje_seq |
| maxlen=X | Maximum length of sequences (<=0 = No maximum) | [0] | rje_seq |
| | **Pre-HAQ Data Cleanup** | | |
| cleanup=T/F | Initial data cleanup | [True] | haqesac |
| seqnr=T/F | Make sequence Non-Redundant | [True] | rje_seq |
| specnr=T/F | Non-Redundancy within same species only | [True] | rje_seq |
| nrid=X | %Identity cut-off for Non-Redundancy | [99.0] | rje_seq |
| blastcut=X | Maximum number of sequences to have in dataset (BLAST query against NR dataset.) | [0] | haqesac |
| blaste=X | E-Value cut-off for BLAST searches (BLAST -e X) | [1e-4] | rje_blast |
| blastv=X | No. of one-line hits (BLAST -v X) | [500] | rje_blast |
| blastf=T/F | Complexity Filter (BLAST -F X) | [True] | rje_blast |
| qcover=X | Min %Coverage vs Query | [60.0] | haqesac |
| gablam=T/F | Use Ordered BLAST alignments rather than ALIGN for qryid and pairid cut-offs | [True] | haqesac |
| gabsim=T/F | Use %Similarity not %Identity for %ID cut-offs | [True] | haqesac |
| pairid=X | Fasta Alignment ID cut-off for any pair of sequences | [40.0] | haqesac |
| qryid=X | Fasta Alignment ID with Query cut-off | [20.0] | haqesac |
| maxgap=X | Maximum proportion of sequence that may be gaps compared to nearest neighbour (<=0 = No maximum) | [0.5] | rje_seq |
| | **General HAQ Options** | | |
| qregion=X,Y | Concentrate on the region of the query from (and including) residue X to residue Y | [0,-1] | haqesac |
| haq=T/F | Homologue Alignment Quality | [True] | haqesac |
| noquery=T/F | No Query for SAQ, Random Query for PAQ (else query=X or first sequence) | [False] | haqesac |
| keep=T/F | Keep all sequences (saqkl=0, paqkl=0) | [False] | haqesac |
| usealn=T/F | Use current alignment (do not realign, degap=F) | [False] | haqesac |
| cwcut=X | Total number of residues above which to use ClustalW for alignment, not MUSCLE | [1e5] | haqesac |

| Option | Description | Default | Module |
|---|---|---|---|
| haqmatrix=FILE | File of AA vs AA scores used in SAQ and PAQ. | [None] | rje_haq |
| | **Single Sequence Alignment Quality (SAQ)** | | |
| saq=T/F | Single Sequence AQ | [True] | haqesac |
| saqc=X | Min score for a residue in SAQ (Default matrix: no. seqs to share residue). | [2] | rje_haq |
| saqb=X | SAQ Block length. | [10] | rje_haq |
| saqm=X | No. residues to match in SAQ Block. | [7] | rje_haq |
| saqks=X | Relative Weighting of keeping Sequences in SAQ. | [3] | rje_haq |
| saqkl=X | Relative Weighting of keeping Length in SAQ. | [1] | rje_haq |
| mansaq=T/F | Manual over-ride of sequence rejection decisions in SAQ. | [False] | rje_haq |
| | **Pairwise Alignment Quality (PAQ)** | | |
| prepaq=T/F | PrePAQ tree grouping | [True] | haqesac |
| paq=T/F | Pairwise AQ | [True] | rje_haq |
| paqb=X | PAQ Block length. | [7] | rje_haq |
| paqm=X | Min score in PAQ Block (Deafult matrix: No. residues to match). | [3] | rje_haq |
| paqks=X | Relative Weighting of keeping Sequences in PAQ. | [3] | rje_haq |
| paqkl=X | Relative Weighting of keeping Length in PAQ. | [1] | rje_haq |
| manpaq=T/F | Manual over-ride of sequence rejection decisions in PAQ. | [False] | rje_haq |
| anchors=T/F | Whether to use conserved 'anchors' to extend well-aligned regions in PAQ | [True] | rje_haq |
| | **Establishment of Subfamilies (and PrePAQ)** | | |
| es=T/F | Establishment of Subfamilies | [True] | haqesac |
| root=X | Rooting of tree, where X is:<br>- mid = midpoint root tree.<br>- ran = random branch.<br>- ranwt = random branch, weighted by branch lengths.<br>- man = always ask for rooting options (unless i<0).<br>- FILE = with seqs in FILE as outgroup.<br>(Any option other than above) | [mid] | rje_tree |

| Option | Description | Default | Module |
|---|---|---|---|
| bootcut=X | cut-off percentage of tree bootstraps for grouping. | | rje_tree |
| mfs=X | minimum family size | [3] | rje_tree |
| fam=X | minimum number of families (If 0, no subfam grouping) | [0] | rje_tree |
| orphan=T/F | Whether orphans sequences (not in subfam) allowed. | [True] | rje_tree |
| allowvar=T/F | Allow variants of same species within a group. | [False] | rje_tree |
| groupspec=X | Species for duplication grouping | [None] | rje_tree |
| group=X | Grouping of tree, where **X** is<br>- **man** = manual grouping (unless i<0).<br>- **dup** = duplication (all species unless groupspec specified).<br>- **qry** = duplication with species of Query sequence (or Sequence 1) of treeseq.<br>- **one** = all sequences in one group.<br>- **None** = no group (case sensitive).<br>- **FILE** = load groups from file | [None] | rje_tree |
| maketree=X | Program for making tree:<br>- **clustalw** = ClustalW NJ method<br>- **neighbor** = PHYLIP NJ method<br>- **upgma** = PHYLIP UPGMA (neighbor) method<br>- **fitch** = PHYLIP Fitch method<br>- **kitsch** = PHYLIP Kitsch (clock) method<br>- **protpars** = PHYLIP MP method<br>- **proml** = PHYLIP ML method<br>- **PATH** = Alternatively, a path to a different tree program/script can be given. This should accept ClustalW parameters. | [clustalw] | rje_tree |
| phyoptions=FILE | File containing extra Phylip tree-making options ('batch running') to use. | [None] | rje_tree |
| protdist=FILE | File containing extra Phylip PROTDIST options ('batch running') to use | [None] | rje_tree |
| | **Ancestor Construction (GASP)** | | |
| ac=T/F | Ancestor Construction (GASP) | [True] | haqesac |
| pamfile=FILE | Sets PAM1 input file | [jones.pam] | rje_pam |
| pammax=X | Initial maximum PAM matrix to generate | [100] | rje_pam |
| pamcut=X | Absolute maximum PAM matrix | [1000] | rje_pam |
| fixpam=X | PAM distance fixed to **X** | [100]. | rje_ancseq |
| rarecut=X | Rare aa cut-off | [0.05]. | rje_ancseq |

| Option | Description | Default | Module |
|---|---|---|---|
| fixup=T/F | Fix AAs on way up (keep probabilities) | [True]. | rje_ancseq |
| fixdown=T/F | Fix AAs on initial pass down tree | [False]. | rje_ancseq |
| ordered=T/F | Order ancestral sequence output by node number | [False]. | rje_ancseq |
| pamtree=T/F | Calculate and output ancestral tree with PAM distances | [True]. | rje_ancseq |
| desconly=T/F | Limits ancestral AAs to those found in descendants | [True]. | rje_ancseq |
| xpass=X | How many extra passes to make down & up tree after initial GASP | [1]. | rje_ancseq |
| | **System Info** | | |
| blastpath=PATH | Path to BLAST programs * Use forward slashes (/) | ['c:/bioware/blast/'] | rje_blast |
| fastapath=PATH | Path to FASTA programs * Use forward slashes (/) | ['c:/bioware/fasta/'] | rje_seq |
| clustalw=PATH | Path to CLUSTALW program * Use forward slashes (/) | ['c:/bioware/clustalw.exe'] | rje_seq |
| muscle=PATH | Path to MUSCLE * Use forward slashes (/) | ['c:/bioware/muscle.exe'] | rje_seq |
| phylip=PATH | Path to PHYLIP programs * Use forward slashes (/) | ['c:/bioware/phylip3.65/exe/'] | rje_tree |
| win32=T/F | Run in Win32 Mode | [False] | rje |

## 5.2: Appendix II: Distributed Python Modules

This appendix is also incomplete and is liable to go out of date. Please look at the documentation within the modules themselves for more details. (And, if you're brave, look at the code!)

| Module | Description | Classes |
|---|---|---|
| haqesac | Master module for HAQESAC program. Controls main thread and objects. | HAQESAC |
| rje | General module containing Classes used by all my scripts plus a number of miscellaneous methods. | RJE_Object_Shell, RJE_Object, Info, Out, Log |
| rje_ancseq | This module contains the objects and methods for ancestral sequence prediction. Currently, only GASP (Edwards & Shields 2004) is implemented. Other methods may be incorporated in the future. | Gasp, GaspNode |
| rje_blast | Performs BLAST searches and loads results into objects. | BLASTRun, BLASTSearch, BLASTHit, PWAln |
| rje_haq | Performs actual HAQ (SAQ and PAQ) algorithms. | HAQ |
| rje_seq | Contains Classes and methods for sets of DNA and protein sequences. (Currently only protein sequences supported.) | SeqList, Sequence, DisMatrix |
| rje_tree | Reads in, edits and outputs phylogenetic trees. Executes duplication and subfamily determination. | Tree, Node, Branch |
| rje_pam | This module handles functions associated with PAM matrices. A PAM1 matrix is read from the given input file and multiplied by itself to give PAM matrices corresponding to greater evolutionary distance. (PAM1 equates to one amino acid substitution per 100aa of sequence.) | PamCtrl, PAM |
| rje_sequence | Contains the Sequence class used by rje_seq.py | Sequence |
| rje_tree_group | Contains all the Grouping Methods for rje_tree.py | - |

## 5.3: Appendix III: Log Files

This part of the manual has not yet been written. Sorry! Contact the author if you have questions about the log files.

## 5.4: Appendix IV: Troubleshooting

Currently, this is a small section as I have not had enough feedback to have FAQs, or anything like that. Here is a list of things that I think MAY cause problems to the unwary:

- Giving file names with spaces without enclosing them in "double quotes". (Otherwise only the first word will be taken as the filename.) It is recommended to use paths and filenames without spaces if possible.
- Incorrect formatting of input files (see **2.2: Input**).
- Disabling some of the pre-HAQESAC sequence/name filters (see **3.2.1: Sequence/Database Filters**)
- I'm not sure when but there is a possibility of problems if running in Windows without the **`win32`** option, especially when calling accessory applications that need different system calls.
- You can reduce the strain on your computer using the **`blastcut=X`** option, which will limit the number of sequences put into the alignment program to the top **`X`** BLAST hits. If you are only really after orthologues, then **`blastcut=100`** would not be too detrimental. This will only help if your computer dies after the initial cleanup stage.
- The **`cwcut=X`** option can also be used to make your computer use ClustalW rather than MUSCLE for its alignments. The **`X`** in this case is the total number of amino acids in the dataset. Below **`X`**, MUSCLE is used. Above **`X`**, ClustalW is used. Setting this quite low (**`cwcut=1000`**) will mean that ClustalW is used nearly all the time.
- Check sequence names. The first word must be unique for each sequence and, for clustalW, the first 30 characters must be unique. Special characters (other than – and _) in the first word may give some programs problems.
- The program (and its default values) has been designed empirically on particular datasets. Every dataset is different and so the settings may not be ideal. If you find that too many sequences are rejected or, conversely, when you look at the alignment, too many badly-aligned sequences are retained, try changing some of the parameters and running the program again.
- Some command-line options are incompatible with particular parts of the program and will be set automatically. If certain filters etc. are not used where expected, this may be the cause. Please contact me if this happens and is confusing.

## 5.5: Appendix V: Glossary

Please e-mail with any terms used in the manual that you do not understand, and I will endeavour to add them to the glossary. This is not, however, designed to be a comprehensive encyclopaedia but rather a starting point for finding more information. Where possible, useful links will be suggested but a quick search on Google normally does the job nicely.

- **ML.** Maximum Likelihood
- **PAM.** Point Accepted Mutation
- **NSF.** Newick Standard Format.
- **MSA.** Multiple Sequence Alignment.

## 5.6: Appendix VI: References

Altschul, S.F., Gish, W., Miller, W., Myers, E.W. and Lipman, D.J. (1990) Basic local alignment search tool, *J Mol Biol*, **215**, 403-410.

Bateman, A., Coin, L., Durbin, R., Finn, R.D., Hollich, V., Griffiths-Jones, S., Khanna, A., Marshall, M., Moxon, S., Sonnhammer, E.L., Studholme, D.J., Yeats, C. and Eddy, S.R. (2004) The PFAM protein families database, *Nucleic Acids Res*, **32**, D138-142.

Edgar, R.C. (2004) MUSCLE: a multiple sequence alignment method with reduced time and space complexity, *BMC Bioinformatics*, **5**, 113.

Edwards, R.J. (2004) GASP: Gapped Ancestral Sequence Prediction. *http://www.bioinformatics.rcsi/~redwards/gasp/*

Edwards, R.J. and Shields, D.C. (2004) GASP: Gapped Ancestral Sequence Prediction for proteins, *BMC Bioinformatics*, **5**, 123.

Felsenstein, J. (2005) PHYLIP (Phylogeny Inference Package) version 3.6., *Distributed by the author. Department of Genome Sciences, University of Washington, Seattle.*,

Hall, T. (2001) BioEdit Sequence Alignment Editor for Windows 95/98/NT. *http://www.mbio.ncsu.edu/BioEdit/bioedit.html*

Higgins, D.G. and Sharp, P.M. (1988) CLUSTAL: a package for performing multiple sequence alignment on a microcomputer, *Gene*, **73**, 237-244.

Jones, D.T., Taylor, W.R. and Thornton, J.M. (1992) The rapid generation of mutation data matrices from protein sequences, *Comput Appl Biosci*, **8**, 275-282.

Liberles, D.A., Schreiber, D.R., Govindarajan, S., Chamberlin, S.G. and Benner, S.A. (2001) The adaptive evolution database (TAED), *Genome Biol*, **2**, RESEARCH0028 Epub 2001 Jul 0024.

Obenauer, J.C., Cantley, L.C. and Yaffe, M.B. (2003) Scansite 2.0: Proteome-wide prediction of cell signaling interactions using short sequence motifs, *Nucleic Acids Res*, **31**, 3635-3641.

Pearson, W.R. (1994) Using the FASTA program to search protein and DNA sequence databases, *Methods Mol Biol*, **24**, 307-331.

Pearson, W.R. (2000) Flexible sequence similarity searching with the FASTA3 program package, *Methods Mol Biol*, **132**, 185-219.

Perriere, G., Duret, L. and Gouy, M. (2000) HOBACGEN: database system for comparative genomics in bacteria, *Genome Res*, **10**, 379-385.

Tamura, K. (2001) TreeExplorer manual. *http://evolgen.biol.metro-u.ac.jp/TE/TE_man.html*

Thomas, P.D., Campbell, M.J., Kejariwal, A., Mi, H., Karlak, B., Daverman, R., Diemer, K., Muruganujan, A. and Narechania, A. (2003) PANTHER: a library of protein families and subfamilies indexed by function, *Genome Res*, **13**, 2129-2141.

Thompson, J.D., Higgins, D.G. and Gibson, T.J. (1994) CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice, *Nucleic Acids Res*, **22**, 4673-4680.