GOPHER: Generation of Orthologous Proteins from High-throughput Estimation of Relationships

Richard J. Edwards © 2006.

Contents

Figures

Tables

1. Introduction

This manual gives an overview of GOPHER, a utility for the high throughput generation of orthologous sequence alignments for specific sets of query sequences. The fundamentals are covered in Chapter 2, Fundamentals, including input and output details. Later sections give more details on how the methods work and statistics are generated. General details about Command-line options can be found in the **RJE Appendices** document included with this download. Details of command-line options specific to GOPHER can be found in the distributed **readme.html** file and online at **bioware.soton.ac.uk**.

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. Options may have been added over the past few weeks and not all found their way into the manual yet. Check the readme on the website for up-to-date options *etc*. In particular, default values for options are subject to change and should be checked in the readme.

Good luck.

Rich Edwards, 2012.

1.1. Version

This manual is designed to accompany GOPHER version 3.0. **NB.** Much of the text is currently from a (much) older version and is in the process of being updated. Please contact the author for clarification and/or advice.

The manual was last edited on 30 November 2012.

1.2. Copyright, License and Warranty

GOPHER is Copyright © 2012 Richard J. Edwards.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

The GNU General Public License should have been supplied with the GOPHER program and is also available at www.gnu.org.

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`** for user-defined portions, such as file names etc.). Command-line examples will be given in (grey) *`italicised Courier New`*. Optional parameters will (if I remember) be `[`in square brackets`]`. Names of files will be marked in **coloured normal text**.

1.4. Why use GOPHER?

The rapid and automated generation of orthologous protein datasets is useful for many bioinformatics applications. True orthologue identification requires the use of phylogenetic information, in which outgroup sequences are used to identify the closest clustering homologues in the different species. Phylogenetic analysis can also identify "in-paralogues" – lineage-specific gene duplications – which can hinder compilation of orthologues as they are both true orthologues to the same protein in a different species. For high-throughput analyses, this can be impractical. Identifying suitable outgroups is not a trivial task and phylogenetic inference has its own weaknesses and biases. As a result, pairwise sequence comparisons are routinely used in its place.

One common approach is to use BLAST (Altschul et al. 1990) scores or e-values. For example, orthologues are often identified using the "Mutual Best Hit" (MBH) approach. Under this model, human sequence A and mouse sequence B are only considered orthologues if A has the best score when B is BLASTed against the human genome, and B has the best score when A is BLASTed against the mouse genome. MBH has severe flaws, however. In-paralogues in one species will be missed, with only one identified as an orthologue, while in-paralogues in both the species being compared may disrupt the MBH assignment totally. Using BLAST for this purpose introduces even more problems. BLAST scores are notoriously sensitivity to sequence length and multi-domain proteins. If multiple regions of one protein are homologous to the same region of the other then the BLAST score will be artificially amplified. At the other end of the scale, unrelated proteins will generate BLAST hits if they both contain similar low complexity regions, such as poly-glutamine repeats or leucine-rich regions. This can be avoided using the Complexity Filter but at the risk of artificially decreasing the BLAST scores of closely related sequences with low complexity regions, which are of most interest for orthologue assignment.

GOPHER attempts to combine the high speed of BLAST MBH-based methods with the accuracy obtained by phylogenetic inference. By considering the relative sequence similarities of a query sequence, different putative orthologues, and paralogous sequences, GOPHER identifies those proteins that, assuming an approximately constant rate of evolution within the protein family, should be true orthologues of the query sequence. Details are given in Chapter 3.

1.5. Getting Help

Much of the information here is also contained in 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 gopher.py help
```

General details about Command-line options can be found in the **RJE Appendices** document included with this download. Details of command-line options specific to GOPHER can be found in the distributed **readme.html** file and online at **bioware.soton.ac.uk**.

If still stuck, then please e-mail me (seqsuite@gmail.com) whatever question you have. If it is the results of an error message, then please send me that and/or the log file (see Chapter 2) too.

1.5.1. Something Missing?

As much as possible, the important parts of the software are described in detail in this manual. If something is not covered, it is generally not very important and/or still under development, and can therefore be safely ignored. If, however, curiosity gets the better of you, and/or you think that something important is missing (or badly explained), please contact me.

1.6. Citing GOPHER

Until published, please cite the SLiMFinder webserver paper: Davey, Edwards & Shields (2007), Nucleic Acids Res. 35(Web Server issue):W455-9. [PMID: 17576682] {Davey, 2007 #83}.

1.7. Availability and Local Installation

This software should work on any system with Python installed. With the exception of any external programs listed below (1.7.1), all the required files should have been provided in the downloaded zip file. Further information can be found at http://bioware.soton.ac.uk/ and the accompanying **RJE Appendices** document.

1.7.1. Programs Used by GOPHER

In addition to the python modules listed above, GOPHER 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.

BLAST: BLAST (Altschul et al. 1990) 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 a backup for MUSCLE (below).

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

Other Alignment Programs. GOPHER can make use of other alignment programs including MAFFT, PROBCONS and Clustal Omega. Details will appear here, or contact the author. See Commandline Options for more details.

2. Fundamentals

2.1. Running GOPHER

2.1.1. The Basics

If you have python installed on your system, you should be able to run GOPHER directly from the command line in the form:

python gopher.py `gopher=FILENAME orthdb=FILENAME`

IMPORTANT: If filenames contain spaces, they should be enclosed in double quotes: `gopher="example file"`. That said, it is recommended that files do not contain spaces as function cannot be guaranteed if they do.

2.1.2. Forking

If using a multiple processor machine in UNIX, GOPHER can fork out multiple processes to increase processing speed using the `forks=X` option. Each query sequence is processed by a different fork.

IMPORTANT: Forking has been implemented for UNIX only. (Python does not support forking in Windows.) If running in windows, use the `win32=T`, `forks=0` or `noforks=T` options.

2.1.3. Options

Command-line options are suggested in the following sections. General details about Command-line options can be found in the **RJE Appendices** document included with this download. Details of command-line options specific to GOPHER can be found in the distributed **readme.txt** and **readme.html** files. These may be given after the run command, as above, or loaded from one or more ***.ini** files (see **RJE Appendices** for details).

2.1.4. Interactivity and Verbosity settings

By default, GOPHER will run through to completion without any user-interaction if given all the options it needs. For more interaction with the program as it runs, use the argument '`i=1`'

```
python gopher.py gopher=gopher_eg.fas orthdb=metazoa.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. There is also the possibility that accessory programs may print things to the screen beyond the control of GOPHER.

Please report any irritations and suggestions for changes to what is printed at different verbosity levels.

2.1.5. Running in Windows

If running in Windows, you can just double-click the **gopher.py** file provided that there is a **gopher.ini** file with all the commands (see 2.1.6). GOPHER is not currently implemented to be menu driven. It is recommended to use the `win32=T` option. (Place this command in a file called **gopher.ini**.)

2.1.6. Setting up the INI File

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

```
blastpath=PATH

clustalw=COMMAND

muscle=COMMAND
```

Note that the first is a path to the BLAST programs, while for ClustalW and MUSCLE the actual program commands themselves must be included. Other alignment programs are also available. This is to make it easier to replace these programs with alternatives. (See **Appendix 3.3** of the **RJE Appendices** document.)

It is also advisable to set the default alignment program of choice (`alnprog=X`) and file organisation options (2.3.1) in the INI file.

See the included **gopher.ini** file for an example or contact the author for advice. 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.

2.2. Input

2.2.1. Input sequences

The main input for GOPHER is two fasta files of protein sequences:

1. The query sequences (`gopher=FILE`)
2. The search database (`orthdb=FILE`)

To get the most out of the program, one of the set fasta formats from common sequence databases should be used. The included manual for the **rje_seq.py** sequence manipulation module has more details on formats and reformatting/filtering of input sequences.

IMPORTANT: Accession numbers in the query sequence file must be **unique** and **must be in the search database** also. The first word for each sequence name in the search database must also be unique (as for any BLAST database to function correctly.)

2.2.2. Removing 'Variant'/In-Paralogue Queries

If searching with a redundant query dataset, the `allqry=F` option can be used to only retain the "best" sequence of any "In-Paralogues" (lineage-specific duplicates), splice variants etc. (see **3The GOPHER Algorithm**). Where such variants exist, the "best" one will be selected according the source database and, where the database is the same, the longest sequence will be retained. (The database hierarchy can be set using the `dblist=X,Y,..,Z` option, where `X,Y,..,Z` constitutes a list of databases in order of preference (good to bad). See the manual for RJE_SEQ for details.)

It is recommended to use a non-redundant database for the query input, such as IPI (Kersey et al. 2004), and leave the `allqry=T/F` option set to True. Many different formats can be converted into an optimal format for use with GOPHER using the supplied **rje_seq.py** program. Contact the author if you need assistance.

2.2.3. Limiting Orthologues by Species/Database

In addition to controlling the returned orthologues by restricting the sequences in the search database, a number of filters can be applied to the orthologous dataset returned, including species and database filters, using the `goodX=LIST` (retention) and `badX=LIST` (exclusion) options, where `X` is one of the following:

- acc = list of accession numbers
- seq = list of sequence names
- spec = list of species codes
- db = list of source databases [sprot,ipi,uniprot,trembl,ens_known,ens_novel,ens_scan]
- desc = list of terms that, at least one of which must be in description line

The `LIST` element can be a list `X,Y,..,Z` or a file containing the relevant terms. E.g. *`goodspec=HUMAN,MOUSE,RAT baddesc=bad_desc.txt`* would only retain sequences annotated as having the species codes HUMAN, MOUSE or RAT and would exclude any of these sequences that have any of the terms from the file **bad_desc.txt** in their description lines. These filters are described in more detail in the manual for RJE_SEQ.

2.3. Output

The main output for GOPHER is series of directories containing output files from the different stages of the GOPHER Run. (See section 2.3.1 for more details on directory organisation.)

1. BLAST (BLAST/)
2. Orthology (ORTH/)
3. Alignment (ALN/)
4. (Optional) paralogue alignment (PARALN/)

Within each directory, all files are named in the form **AccNum.***, where **AccNum** is the accession number of the query. Main output is the alignment of orthologues, **ALN/*.orthaln.fas**. A full list of output files is given in Table 1. **NB.** These files need to be revised following changes during the upgrade to GOPHER Version 3.0.

Table 1. Main GOPHER output files.

File	Description	Directory
*.blast	The main BLAST results file from the initial BLAST against the search database (one-line hits only)	BLAST
*.blast.id	The fastacmd IDs of the hits from the initial BLAST	BLAST
*.qry	The query sequence	BLAST
*.para.fas	Paralogues in fasta format	ORTH
*.minsim.fas	All hits exceeding minimum similarity, in fasta format	ORTH
*.orth.fas	Putative Orthologues in fasta format	ORTH
*.orthaln.fas	Alignment of orthologues	ALN
*.paraln.fas	Alignment of paralogues in fasta format	PARALN

2.3.1. Organisation of output files and directories (Version 3.0)

GOPHER Version 3.0 has improved directory organisation for multi-species and multi-orthdb GOPHER runs on the same system, in line with the bioware.ucd.ie webserver. Additional data cleanup has been added too. (NB. Experimental Sticky GOPHER runs will not work with Organise=T.) This version features four levels of directories:

1. The output directory is now set at the highest level by `gopherdir=PATH`.
2. If `organise=T` (default) then a subdirectory will be created within this directory named after the `orthdb=FILE` input (path and extension stripped).
3. Each species will have its own subdirectory within this in turn. By default, these are named using the UniProt species codes, which are read from the input sequences. To use TaxIDs, the `unispec=FILE` option must be used. This should point to a file that has one line per UniProt species code, starting with the species code and ending with `:TaxID:` where TaxID is a number. The file should be sorted by species code.
4. Within each species directory, the subdirectories listed above (2.3) will be created. ALN/ and PARALN/ directories are now prefixed with the alignment program (`alnprog=X`), *e.g.* aligning with MUSCLE will create alignments in muscleALN/. This allows multiple runs with different alignment tools that will not conflict. **NOTE:** Phylogenetic tree construction does not currently differentiate alignment programs or phylogenetic methods.

Further details will be added here shortly. In the meantime, contact the author with any questions.

2.3.2. Renaming key output files

To enable multiple runs with different orthology prediction methods, the standard "orth" file identifiers can be replaced using **orthid=X**. *E.g.* *orthid=mbh* will generate **ORTH/*.mbh.fas** and **ALN/*.mbhaln.fas**.

2.3.3. Log Files

The GOPHER log file records information that may help subsequent interpretation of results or identify problems. Probably it's most useful content is any error messages generated. By default the log file is **gopher.log** but this can be changed with the **log=FILE** option. Logs will be appended unless the **newlog** option is used.

2.4. Commandline Options

The key commandline options are given in Table 2 and the appropriate sections. A full list of commandline options can be found in the **readme** file, online at **bioware.soton.ac.uk** or by running:

```
python gopher.py help
```

Table 2. Key GOPHER Commandline Options.

Option	Description	Default	Module
	General Dataset Input/Output		
gopher=FILE	Loads sequences from FILE	[None]	**gopher**
orthdb=FILE	BLAST database for homologue searching	[None]	**gopher**
allqry=T/F	Whether all query sequences are to have alignments etc. (If False, the 'best' in-paralogues only are used.)	[True]	**gopher**
startfrom=X	Accession Number / ID to start from.	[None]	**gopher**
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	[gopher.log]	**rje**
newlog=T/F	Create new log file.	[False]	**rje**
	GOPHER		
postdup=T/F	Restrict orthologues to within the query species' post-duplication clade	[False]	**gopher**
minsim=X	Minimum %similarity of Query for each "orthologue".	[40.0]	**gopher**
gablamo=X	Measure to use for minimum %, where **X** can be **Sim** (similarity), **ID** (identity) or **Len** (coverage).	[Sim]	**gopher**

simfocus=X	Style of MinSim used, where **X** can be: **query** = %query must > minsim;
	hit = %hit must > minsim;
	either = %query > minsim OR %hit > minsim;
	both = %query > minsim AND %hit > minsim.
		[query]	**gopher**

paralign=T/F	Whether to produce paralogue alignments (>minsim) in PARALN/.	[True]	**gopher**
parasplice=T/F	Whether to allow splice variants in paralogue alignments (where identified).	[False]	**gopher**

orthblast	Run to blasting versus orthdb (Stage 1).		gopher
orthfas	Run to output of orthologues (Stages 2-6).		gopher
orthalign	Run to alignment of orthologues (Stage 7).	Default	gopher
repair=T/F	Repair mode - replace previous files if date mismatches or files missing. (Skip missing files if False)	[True]	gopher
force=T/F	Whether to force execution at current level even if results are new enough.	[False]	gopher
fullforce=T/F	Whether to force current and previous execution even if results are new enough.	[False]	gopher
ignoredate=T/F	Ignores the age of files and only replaces if missing.	[False]	gopher
savespace=X	Save space by deleting intermediate blast files during orthfas.	[True]	gopher
	Sequence Filters		
goodX=LIST	Only keeps sequences meeting the requirement of LIST (X,Y,..,Z or a FILE which contains a list). ➢ goodacc = list of accession numbers ➢ goodseq = list of sequence names ➢ goodspec = list of species codes ➢ gooddb = list of source databases ➢ gooddesc = list of terms that, at least one of which must be in description line	[None]	rje_seq
badX=LIST	As goodX but excludes rather than retains sequences	[None]	rje_seq
accnr=T/F	Check for redundant Accession Numbers/Names on loading sequences.	[False]	rje_seq
seqnr=T/F	Make sequence Non-Redundant	[False]	rje_seq
specnr=T/F	Non-Redundancy within same species only	[False]	rje_seq
nrid=X	%Identity (GABLAM) cut-off for Non-Redundancy	[100.0]	rje_seq
nrsim=X	%Similarity (GABLAM) cut-off for Non-Redundancy	[100.0]	rje_seq
autofilter=T/F	Whether to apply sequence filters upon loading	[True]	rje_seq

Option	**Description**	**Default**	**Module**
	System Info		
blastpath=PATH	Path to BLAST programs * Use forward slashes (/)	['c:/bioware/blast/']	rje_blast
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
win32=T/F	Run in Win32 Mode	[False]	rje

memsaver=T/F	Run in "Memory Saver" mode	[False]	**rje**
	Forking		
forks=X	Number of forks	[0]	**rje**
killforks=X	Number of seconds of inactivity before killing forks	[3600]	**rje**
noforks=T/F	Option to over-ride and cancel forking	[False]	**rje**

NB. This table needs to be updated to reflect changes in Version 3.0.

3. The GOPHER Algorithm

3.1. Algorithm Overview

An overview of GOPHER is given in **Figure 1**. The default protocol is as follows:

1. BLAST the sequence database (see **3.3Recommended BLAST Database**), reporting the first 1000 hits with a BLAST threshold of e=10^{-4} and the complexity filter on.
2. Repeat the BLAST of the query against the initially detected homologues with a very relaxed threshold of e=10 and without the complexity filter, and use the GABLAMO algorithm (see **3.1.1GABLAMO Sequence Similarity**) to filter out any sequence with less than 40% global percentage similarity with the query, i.e. at least 40% of residues in the query must be similar to each potential orthologue, and keep only the most similar sequence to the query for each species. The minimum level of similarity can be changes using the `minsim=X` option. To use percentage identity or coverage instead of similarity, use the `gablamo=X` option, where `X` can be `Sim` (similarity), `ID` (identity) or `Len` (coverage).

   By default, the focus of the similarity score is the query, *i.e.* the GABLAMO percentage similarity, identity or coverage of the query sequence is used. GABLAMO is asymmetric (see **3.1.1GABLAMO Sequence Similarity**) and so the percentage similarity, identity or coverage of the homologue may not meet the threshold even if the query does. To change the focus of the similarity measure, use the `simfocus=X` option, where `X` can be:

   a. `query` (default): %query must > minsim. (Best if query is ultimate focus and maximises closeness of returned orthologues)

   b. `hit`: %hit must > minsim. (Best if lots of sequence fragments are in searchdb and should be retained)

   c. `either`:= %query > minsim OR %hit > minsim. (Best if both above conditions are true)

   d. `both`: %query > minsim AND %hit > minsim. (Gets most similar sequences in terms of length but can be too stringent.)
3. Filter out chimp sequences as they are too closely related too their human orthologues and can disrupt the filtering of splice variants (below).
4. Each paralogue (homologous proteins in the same species) is identified and used to query the remaining orthologues as in (2), generating GABLAMO percentage similarity statistics.
5. Filter out in-paralogues (species-specific duplicates) and splice variants, as these will share the same set of "true orthologues" as the query. These are identified as those paralogues which:

   a. Have a greater percentage similarity to the query than any sequence of a different species.

   b. The query has a greater percentage similarity to the paralogue than any sequence of a different species.
6. Putative orthologues are then identified (**Figure 2**). To be considered an orthologue:

   a. If the orthologue is within the same clade as the query, i.e. diverged from the query more recently than the closest paralogue:

      i. The query must have a higher global percentage similarity with the sequence than any other sequence of same species.

      ii. The sequence must have a higher global percentage similarity with the query than with the closest paralogue.

      iii. The query must have a higher global percentage similarity with the sequence than with the closest paralogue.

   b. If the orthologue is outside of the most recent post-duplication clade for the query, it must not be within the post-duplication clade for another paralogue. For such an orthologue, if the query protein is not the closest protein of the query species:

i. The query must have a higher global percentage similarity to the putative orthologue's closest paralogue than to the putative orthologue.

ii. The putative orthologue's closest paralogue must have a higher global percentage similarity to the query than to the putative orthologue.

If the `postdup=T` option is used, these orthologues will be ignored.

7. The remaining orthologues are then aligned using MUSCLE (Edgar 2004).
8. If `paralign=T` (default), the paralogues will also be aligned using MUSCLE (Edgar 2004). If the `parasplice=T` option is used, these will include IPI splice variants that have been filtered out as in-paralogues.

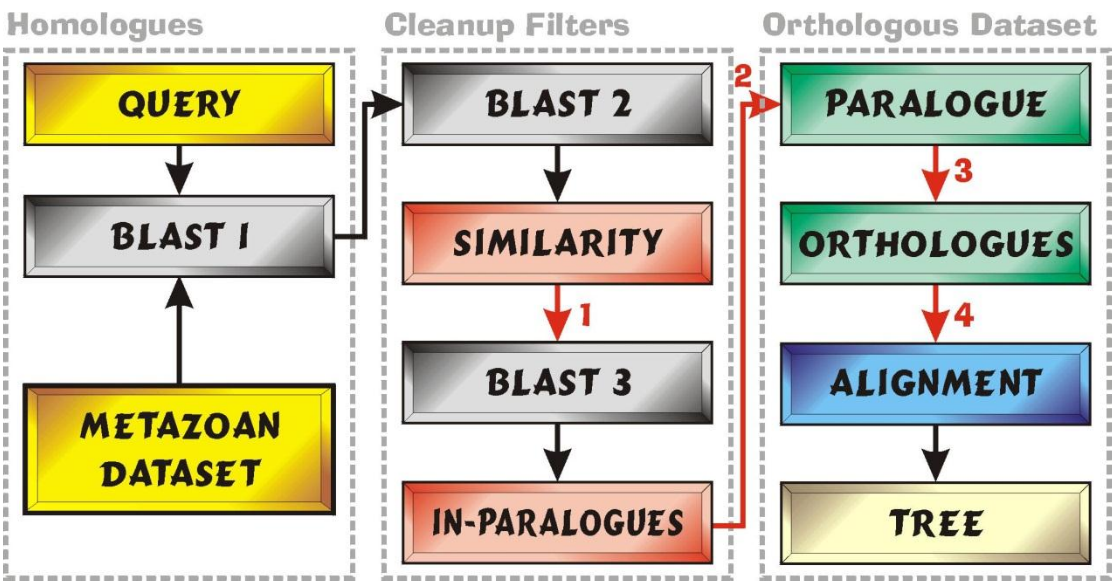
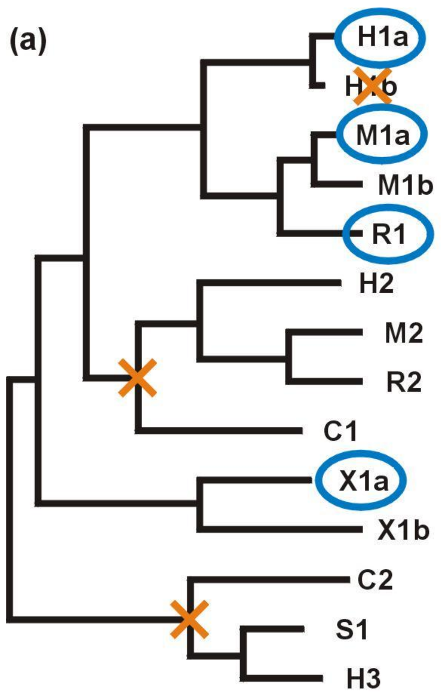


Figure 1. Overview of GOPHER process.

Potential orthologues are removed at each of the four stages marked by red arrows.

(a)



(b)

Seq	Human	Best	Orth
H1b	H1a	-	No
M1a	H1a*	Yes	Yes
R1	H1a	Yes	Yes
M1b	H1a*	No	Yes
C1	H1b	Yes	No
H2	H1a	-	No
M2	H2	No	No
R2	H2	No	No
X1a	H2	Yes	Yes
X1b	H2	No	Yes
S1	H3	Yes	No
H3	H1a	-	No
C2	H3	No	No

Figure 2. Example Orthologue Identification with GOPHER.

(a) Example selection of orthologous proteins for human protein H1a, circled in blue. Inparalogue H1b is removed. Mouse sequences M1a and M1b are both orthologues of H1a but M1b is arbitrarily removed in favour of the closer M1a. C1 is the closest chicken sequence but is closer to H2. Similarly, sheep sequence S1 is part of the H3 clade. X1a is considered an orthologue, even though H2 is the closest human sequence to it, because H2 is closer to H1a than either is to X1a, while H3 is further from both H1a and X1a. H, human; M, mouse; R, rat; C, chicken; S, sheep; X Xenopus; **(b)** Table summarising relationships. Seq, sequence ID from (a); Human, closest human sequence; Best, whether closest sequence of species to H1a; Orth, whether a true orthologue of H1a.

3.1.1. Controlling Which Parts of GOPHER Run

By default, GOPHER will be run to conclusion on all query sequences. To run GOPHER up to a certain point only, use the **`orthblast`**, **`orthfas`** and **`orthalign`** options. To skip part of the input file and start at a particular sequence (e.g. if picking up from killing the program for any reason), use the **`startfrom=X`** option, where **`X`** is an accession number or sequence ID. When GOPHER is running at a given level of execution (**`orthblast`**, **`orthfas`** or **`orthalign`**) and the files from a previous level are missing, they will be generated. To turn this off, and only run the selected level, use **`repair=F`**. E.g. to only align those query datasets for which orthologues have already been identified, use the commands *`orthalign repair=F`*.

If GOPHER is run and some results files already exist, then those parts of GOPHER will be skipped. By default, these files will only be used if they are newer than the files needed to generate them. To override the time/date check and use existing results files of any age, use the **`ignoredate=T`** option. To force GOPHER to regenerate results for the given level of execution (**`orthblast`**, **`orthfas`** or **`orthalign`**), use the **`force=T`** option. To force GOPHER to regenerate results for all levels, use the **`fullforce=T`** option.

3.1.2. Saving disk space

A full GOPHER run on a large query dataset will use up quite a lot of disk space. To reduce this somewhat, GOPHER will delete some of the intermediate files during execution. To keep these extra intermediates, use the **`savespace=F`** option.

3.2. GABLAMO Sequence Similarity

Although fast to generate, using BLAST scores or e-values alone for measuring relative sequence similarity introduces many problems. BLAST scores are notoriously sensitivity to sequence length and multi-domain proteins. If multiple regions of one protein are homologous to the same region of the other then the BLAST score will be artificially amplified. At the other end of the scale, unrelated proteins will generate BLAST hits if they both contain similar low complexity regions, such as poly-glutamine repeats or leucine-rich regions. This can be avoided using the Complexity Filter but at the risk of artificially decreasing the BLAST scores of closely related sequences with low complexity regions, which are of most interest for orthologue assignment.

To avoid reducing the scores associated with highly similar sequences, one alternative is to leave the Complexity Filter off and use a stricter BLAST score or e-value cut-off to filter out weak homologues. The appropriate threshold can be hard to determine, however, as neither BLAST score nor e-value translate into the degree of similarity in a particularly intuitive fashion. A more intuitive measure is the percentage sequence identity or similarity between two sequences, which can be calculated using pairwise sequence alignment programs, such as ALIGN (Pearson 2000). The disadvantage is that performing multiple pairwise sequence alignments is slow compared to a single BLAST search, which can have serious run-time issues with large genomic analyses. Furthermore, pairwise sequence alignment has well documented drawbacks (Rosenberg 2005). The most obvious is that the program will enforce an alignment, whether the sequences have reasonable similarity or not. As a result, when default settings are used, even random sequences will return a percentage identity of 20% or more for the shorter sequence. Furthermore, if one is interested in the overall similarity of two sequences incorporating domain duplications or rearrangements then pairwise alignment will fail.

GOPHER makes use the GABLAMO (Global Alignment from BLAST Local Alignment Matrix (Ordered)) algorithm (Edwards and Davey 2006), which returns a set of informative and intuitive pairwise sequence similarity statistics using the results from a basic BLAST search. GABLAMO compiles local sequences alignments generated by BLAST and returns ordered sequence coverage, identity and similarity statistics independently for query and hit sequences (**Figure 3**).

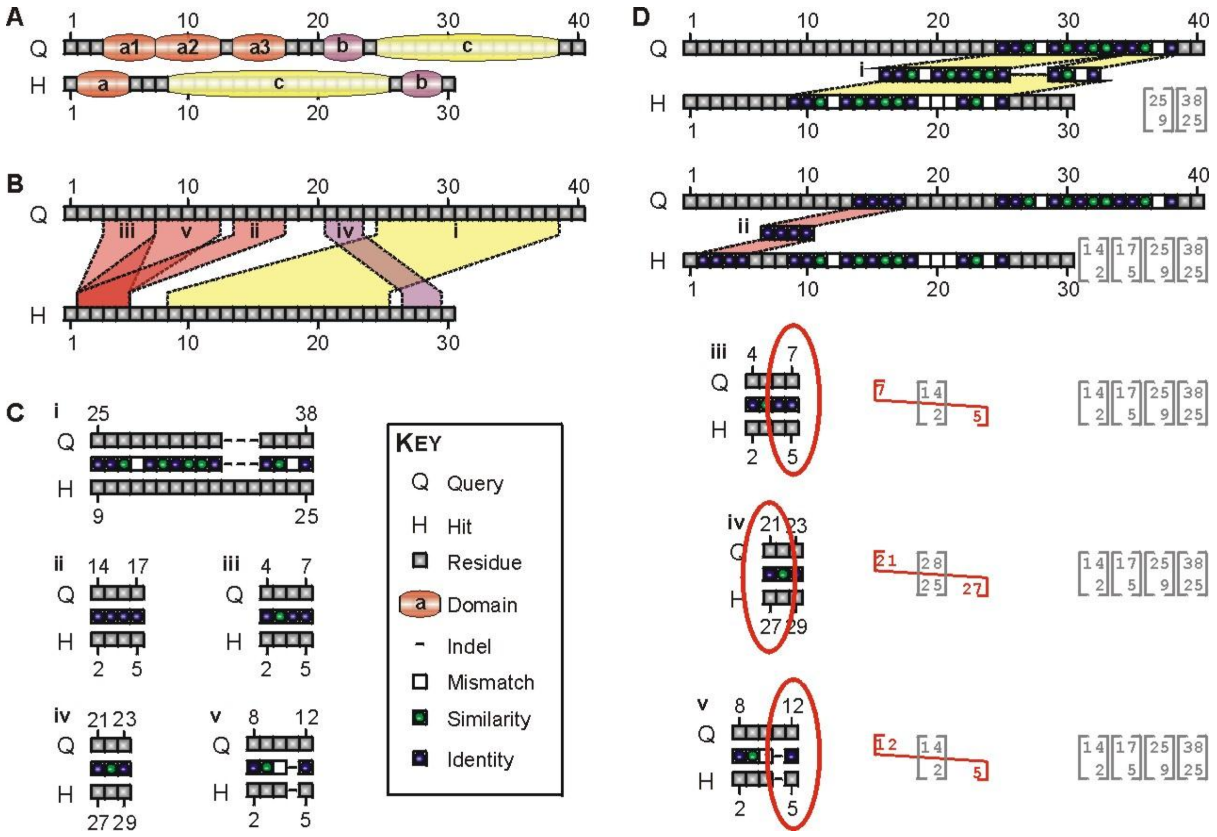


Figure 3. Overview of GABLAMO (Global Alignment from BLAST Local Alignment Modules (Ordered)) Percentage Identity/Similarity calculations method.

(A) Each Query-Hit pair from the BLAST search is compared in a pair-wise fashion. For simplicity, this example has used artificially short sequences and "Domains", which for this purpose are regions of local homology. ***(B)*** BLAST detects five regions of local homology, labelled i-v in order of score, producing ***(C)*** five local alignments between regions of query and hit. ***(D)*** Each local alignment is taken in order of BLAST score and mismatches, similarities and identities transposed on to the Query and

Hit, assuming that residue has not already been assigned a better status. Pairs of Query and Hit positions are also stored. If, as in iii, an alignment would split an existing pair of Query-Hit positions, it is judged to be in conflict with the linear ordering of previous alignments and is not added.

3.3. Recommended BLAST Database

Please contact the author for advice.

4. Appendices

4.1. Troubleshooting & FAQ

There are currently no specific Troubleshooting issues arising with GOPHER. Please see general items in the **RJE Appendices** document and contact me if you experience any problems not covered.

4.2. Abbreviations and Glossary

The following terms and abbreviations are used in this manual:

- **GO.** Gene Ontology. (See www.geneontology.org.)
- **PPI**. Protein-protein interaction.

4.3. References

- Amanchy R, Periaswamy B, Mathivanan S, Reddy R, Tattikota SG & Pandey A (2007). "A curated compendium of phosphorylation motifs." *Nat Biotechnol.* **25**(3): 285-6.
- Balla S, Thapar V, Verma S, Luong T, Faghri T, Huang CH, Rajasekaran S, del Campo JJ, Shinn JH, Mohler WA, Maciejewski MW, Gryk MR, Piccirillo B, Schiller SR & Schiller MR (2006). "Minimotif miner: A tool for investigating protein function." *Nat Methods.* **3**(3): 175-7.
- Davey NE, Shields DC & Edwards RJ (2006). "Slimdisc: Short, linear motif discovery, correcting for common evolutionary descent." *Nucleic Acids Res.* **34**(12): 3546-54.
- Davey NE, Edwards RJ & Shields DC (2007). "The slimdisc server: Short, linear motif discovery in proteins." *Nucleic Acids Res* **35**(Web Server issue): W455-9.
- Edwards RJ, Davey NE & Shields DC (2007). "Slimfinder: A probabilistic method for identifying over-represented, convergently evolved, short linear motifs in proteins." *PLoS ONE* **2**(10): e967.
- Neduva V, Linding R, Su-Angrand I, Stark A, de Masi F, Gibson TJ, Lewis J, Serrano L & Russell RB (2005). "Systematic discovery of new recognition peptides mediating protein interaction networks." *PLoS Biol.* **3**(12): e405.
- Neduva V & Russell RB (2005). "Linear motifs: Evolutionary interaction switches." *FEBS Lett.* **579**(15): 3342-5 Epub 2005 Apr 18.
- Neduva V & Russell RB (2006). "Dilimot: Discovery of linear motifs in proteins." *Nucleic Acids Res.* **34**(Web Server issue): W350-5.
- Puntervoll P, Linding R, Gemund C, Chabanis-Davidson S, Mattingsdal M, Cameron S, Martin DM, Ausiello G, Brannetti B, Costantini A, Ferre F, Maselli V, Via A, Cesareni G, Diella F, Superti-Furga G, Wyrwicz L, Ramu C, McGuigan C, Gudavalli R, Letunic I, Bork P, Rychlewski L, Kuster B, Helmer-Citterich M, Hunter WN, Aasland R & Gibson TJ (2003). "Elm server: A new resource for investigating short functional sites in modular eukaryotic proteins." *Nucleic Acids Res* **31**(13): 3625-30.