GFam Documentation

Release 1.1

Tamás Nepusz, Rajkumar Sasidharan

August 04, 2012

CONTENTS

gene
family
annotation &
maintenance

This is the documentation of `gfam`, a Python module to aid the automatic annotation of gene families based on consensus domain architecture, written by Tamás Nepusz.

GFam has no official publication yet, but there will soon be one. You are kindly asked to cite the webpage of GFam until an official GFam publication comes out:

> **GFam: a platform for automatic annotation of gene families**
> Tamás Nepusz, Rajkumar Sasidharan, David Swarbreck, Eva Huala and
> Alberto Paccanaro. <http://github.com/ntamas/gfam>

GFam can generally be considered stable. We have used it successfully to annotate the whole genome of *Arabidopsis thaliana* and *Arabidopsis lyrata*. Please email the author if you discover any bugs, or feel free to submit a bug report on GitHub.

If you are interested in the latest and greatest (but maybe unstable) version of GFam, you can get it from GitHub - just click on the **Download source** button on the GitHub page.

CHAPTER

ONE

INTRODUCTION

1.1 What is GFam?

GFam (or `gfam`) is a Python module to aid the automatic annotation of gene families based on consensus domain architectures. `gfam` started out as a collection of loosely coupled Python scripts that process the output of `iprscan` (a tool to obtain domain assignments of individual genes from InterPro) and conduct some analyses using BLAST to detect novel, previously uncharacterised domains. The original domains and the detected novel domain candidates are then used to create a consensus domain assignment for each gene sequence. Genes are then finally assigned to families based on their domain architectures. Finally, the tool derives functional labels for families based on the Gene Ontology and an assignment between InterPro domains and Gene Ontology terms. Optionally, a Gene Ontology overrepresentation analysis can also be conducted on the GO annotations of individual domains in the same sequence to reinforce the functional labels.

1.2 Requirements

You will need the following tools to run `gfam`:

- Python 2.5 or later. Python 3 is not supported yet. `gfam` was also tested with Jython 2.5.1.
- NCBI BLAST; in particular, the `formatdb` and `blastall` tools from the legacy C-based BLAST distribution. You can still use `gfam` wit the newer, C++-based BLAST if you have the `legacy_blast.pl` wrapper script in the BLAST folder.

The latest release of SciPy is recommended, but not necessary. `gfam` uses SciPy for calculating the logarithm of the gamma function in the overrepresentation analysis routines, but it falls back to a (somewhat slower) Python implementation if SciPy is not installed.

1.3 For the impatient

`gfam` is driven by a master configuration file named `gfam.cfg`. A sample configuration file is given in the distribution. The sample file works fine for the gene sequences of *Arabidopsis thaliana*; for other species, you might have to tweak some of the parameters, and you will surely have to modify the paths to the data files. The configuration file is documented and mostly self-explanatory.

If you do not have a configuration file for some reason, or you want to generate a new one from scratch, you can ask `gfam` to do it:

```
$ bin/gfam init
```

This will create a default configuration file named gfam.cfg (if it does not exist already) and lists the configuration options you have to set in the file before starting GFam.

If the configuration file is well in order, you can launch gfam by typing:

```
$ bin/gfam
```

This will run the whole gfam analysis pipeline using the configuration specified in gfam.cfg. If your configuration file is named otherwise, you can run it by typing:

```
$ bin/gfam -c my_config.cfg
```

The results will be put into whatever work directory you specified in the configuration file. By default, this is named work. See *Output files* for more details on what will be calculated and where you can find them.

1.4 Questions, comments

If you have a question or a comment about gfam or you think you have found a bug, feel free to contact me using the email address given in the header of this document.

CHAPTER

TWO

RUNNING GFAM

2.1 Input files

The input files can be grouped into three large groups: *data files*, *mapping files* and the *configuration file*. Data files contain the actual input data that is specific to a given organism. Mapping files usually map between IDs of different data sources (for instance, from InterPro domain IDs to Gene Ontology terms) or IDs to human-readable descriptions. The *configuration file* tells GFam where to find the data files and the mapping files. When one wants to process a new organism with GFam, it is therefore usually enough to replace the paths of the data files in the configuration only, as the mapping files can be re-used for multiple analyses.

GFam requires the following data files:

Sequence file This file must contain all the sequences that are being analysed and annotated by GFam.

Domain assignment file This file is produced by running `iprscan`, the command-line variant of InterProScan and it assigns sections of each segment in the sequence file to known domains in InterPro. The sequence IDs in this file must be identical to the ones in the sequence file; if not, one can specify a regular expression in the *configuration file* to extract the sequence ID from the FASTA defline.

Besides the data files, the following mapping files are also needed:

InterPro – GO mapping This file maps InterPro IDs to their corresponding GO terms, and it can be obtained from <http://www.geneontology.org/external2go/interpro2go>.

Mapping of domain IDs to human-readable names Fairly self-explanatory; a tab-separated flat file with two columns, the first being the domain ID and the second being the corresponding human-readable name. It is advisable to construct a file which contains at least the InterPro, Pfam, SMART and Superfamily IDs as these are the most common (and many Pfam, SMART and Superfamily IDs do not have corresponding InterPro IDs yet). If you want to create such a mapping file easily, please refer to *Updating the mapping of IDs to human-readable names*.

Parent-child relationships of InterPro terms This file contains the parent/child relationships between InterPro accession numbers to indicate family/subfamily relationships. This file is used to map each InterPro subfamily ID to the corresponding family ID, and it can be obtained from EBI.

The Gene Ontology This file contains the Gene Ontology in OBO format, and it is required only for the label assignment and overrepresentation analysis steps. The latest version of the file can be obtained from the homepage of the Gene Ontology project.

GFam accepts uncompressed files or files compressed with `gzip` or `bzip2` for both the data and the mapping files. Compressed files will be decompressed on-the-fly in memory when needed.

2.2 The configuration file

The default configuration file of GFam is called gfam.cfg, but you can specify an alternative configuration file name on the command line using the -c switch. A sample configuration file is included in the GFam distribution; however, you can always generate a new one by running the following command:

```
$ bin/gfam init
```

This will generate a file named gfam.cfg in the current directory and list the configuration keys you have to modify before starting your analyses.

The configuration file consists of sections, led by a [section] header and followed by name=value entries. Lines beginning with # or ; are ignored and used to provide comments. Lines containing whitespace characters only are also ignored. For more details about the configuration file format, please refer to the ConfigParser module in the documentation of Python.

The full list of supported configuration keys and their default values is as follows:

2.3 Output files

GFam produces four output files in the output folder specified in the configuration file. These files are as follows:

2.3.1 domain_architectures.tab

A simple tab-separated flat file that contains the inferred domain architecture for each sequence in a simple, summarised format. The file is sorted in a way such that more frequent domain architectures are placed at the top. Sequences having the same domain architecture are sorted according to their IDs.

The file has six columns. The first column is the ID of the sequence (e.g., AT1G09650.1), the second is the sequence length (e.g, 382). The third column contains a summary of the domain architecture of the sequence, where domains are ordered according to the starting position, and consecutive domain IDs are separated by semicolons (e.g., IPR022364;IPR017451). The InterPro domain ID is used whenever possible. Novel domains identified by GFam are denoted by NOVELxxxxx, where xxxxx is a five-digit identifier. The fourth column contains the frequency of this domain architecture (i.e. the number of sequences that have the same domain architecture). The fifth column is the same as the third, but the exact starting and ending positions of the domain are also added in parentheses after the domain ID (e.g., IPR022364(9-57);IPR017451(112-357)). The sixth column contains the concatenated human-readable descriptions of the domains (for instance, F-box domain, Skp2-like;F-box associated interaction domain).

2.3.2 domain_architecture_details.txt

This file is the human-readable variant of domain_architectures.tab (which is more suitable for machine parsing). It contains blocks separated by two newline characters; each block corresponds to a sequence and has the following format:

```
AT1G09650.1
    Primary assignment source: HMMTigr
    Number of data sources used: 2
    Data sources: superfamily, HMMTigr
    Coverage: 0.772
    Coverage w/o novel domains: 0.772
       9-  57: SSF81383 (superfamily, stage: 2) (InterPro ID: IPR022364)
```

```
              F-box domain, Skp2-like
    112- 357: TIGR01640 (HMMTigr, stage: 1) (InterPro ID: IPR017451)
              F-box associated interaction domain
```

The first line of each block is unindentend and contains the sequence ID. The remaining lines are indented by at least four spaces. The second line contains the name of the InterPro data source that was used to come up with the primary assignment in *step 2 of the pipeline* (see more details later in *Steps of the GFam pipeline*), followed by the number of data sources used to construct the final assignment, and of course the data sources themselves. The fifth and sixth lines contain the fraction of positions in the sequence that are covered by at least one domain; the fifth line takes into account novel domains (`NOVELxxxxx`), while the sixth line does not. The remaining lines list the domains themselves along with the data source they came from and the stage in which they were selected. For more details about the stages, see *Steps of the GFam pipeline*.

2.3.3 `assigned_labels.txt`

TODO

2.3.4 `overrepresentation_analysis.txt`

This file contains the results of the Gene Ontology overrepresentation analysis for the domain architecture of each sequence. Note that since the results of the overrepresentation analysis depend only on the domain architecture, the results of sequences having the same domain architecture will be completely identical.

The file consists of blocks separated by two newlines, and each block corresponds to one sequence. Each block has the following format:

```
AT1G61040.1
  0.0009: GO:0016570 (histone modification)
  0.0009: GO:0016569 (covalent chromatin modification)
  0.0024: GO:0016568 (chromatin modification)
  0.0036: GO:0006325 (chromatin organization)
  0.0049: GO:0051276 (chromosome organization)
  0.0055: GO:0006352 (transcription initiation)
  0.0095: GO:0006461 (protein complex assembly)
  0.0109: GO:0065003 (macromolecular complex assembly)
  0.0111: GO:0006996 (organelle organization)
  0.0126: GO:0043933 (macromolecular complex subunit organization)
```

In each block, the first number is the p-value obtained from the overrepresentation analysis, the second column is the GO ID. The name corresponding to the GO label is contained in parentheses. Blocks containing a sequence ID only represent sequences with no significant overrepresented GO labels in their domain architecture.

2.4 Command line options

GFam is started by the master script in `bin/` as follows:

```
$ bin/gfam
```

The exact command line syntax is `bin/gfam [options] [command]`, where `command` is one of the following:

`init` Generates a configuration file for GFam from scratch. The name of the configuration file will be `gfam.cfg` by default, but you can change it with the `-c` switch. GFam will refuse to overwrite existing configuration files. Example:

```
$ bin/gfam -c a_lyrata.cfg init
```

run Runs the whole GFam pipeline. This is the default command.

clean Removes the temporary directory used to store the intermediate results. The name of the temporary directory is determined by the `folder.work` configuration option in the `configuration file`.

> **Warning:** If the output directory is the same as the temporary directory (`folder.work` is equal to `folder.output` in the configuration), the `clean` command will also delete the final results from the output folder!

The default configuration file used is always `gfam.cfg`, but it can be overridden with the `-c` switch. For example, the following command will clean the work directory specified in `a_lyrata.cfg`:

```
$ bin/gfam -c a_lyrata.cfg clean
```

The following extra command line switches are also available:

-h, --help shows a help message and then exits

-c FILE, --config-file=FILE specifies the name of the configuration `FILE`

-v, --verbose enables verbose logging

-d, --debug shows debug messages as well

-f, --force forces the recalculation of the results of intermediary steps in the GFam pipeline even when GFam thinks everything is up-to-date.

Besides the master script, there are scripts for re-running individual steps of the GFam pipeline. These scripts are separate Python modules in `gfam/scripts` and they correspond to the *steps of the GFam pipeline*. It is unlikely that you will have to run them by hand, but if you do, you have to supply the necessary input on the standard input stream of the scripts. For instance, if you want to do some custom filtering on a BLAST tabular result file, you can use `gfam/scripts/blast_filter.py` as follows:

```
$ python -m gfam.scripts.blast_filter -e 1e-5 <input.blast
```

This will filter `input.blast` and remove all entries with an E-value larger than 10^{-5}. The result will be written to the standard output.

You can get a summary of the usage of each script in `gfam/scripts` as follows:

```
$ python -m gfam.scripts.blast_filter --help
```

Of course replace `blast_filter` with the name of the script you are interested in. The default values of the command line switches of these scripts come from the *configuration file*, and they also support `-c` to change the name of the configuration file.

In 99.9999% of the cases, you will only have to do `bin/gfam init` to create a new configuration file, `bin/gfam` to run the pipeline and `bin/gfam clean` to clean up the results.

CHAPTER

THREE

STEPS OF THE GFAM PIPELINE

The GFam pipeline consists of multiple steps. In this section, we will describe what input files does the GFam pipeline operate on, how the steps are executed in order one by one and what output files are produced in the end. First, a short overview of the whole process will be given, followed by a more detailed description of each step.

3.1 Overview of a GFam analysis

GFam infers annotations for sequences by first finding a consensus domain architecture for each step, then collecting Gene Ontology terms for each domain in a given domain architecture, and selecting a few more specific ones. Optionally, a Gene Ontology overrepresentation analysis can also be performed on the terms to determine whether some GO terms occur more frequently in a given domain architecture than expected by random chance. Out of these three steps, the calculation of the consensus domain architecture is the most complicated one, as GFam has to account for not only the known domain assignments from InterPro, but also for the possible existence of novel, previously uncharacterised domains. The whole pipeline can be broken to 8+1 steps as follows:

1. Extracting valid gene IDs from the sequence file.
2. Determining a preliminary domain architecture for each sequence by considering known domains from the domain assignment file only.
3. Finding the unassigned regions of each sequence; i.e. the regions that are not assigned to any domain in the preliminary domain architecture.
4. Running an all-against-all BLAST comparison of the unassigned sequence fragments and filtering BLAST results to determine which fragments may correspond to the same novel domain. Such filtering is based primarily on E-values and alignment lengths. At this point, we obtain a graph on the sequence fragments where two fragments are connected if they passed the BLAST filter.
5. Calculating the Jaccard similarity of the sequence fragments based on the connection patterns and removing those connections which have a low Jaccard similarity.
6. Finding the connected components of the remaining graph. Each connected component will correspond to a tentative novel domain.
7. Calculating the consensus domain architecture by merging the preliminary domain architecture with the newly detected novel domains.
8. Selecting a functional label for each of the domain architectures based on a mapping between InterPro domains and Gene Ontology terms.
9. Conducting a Gene Ontology overrepresentation analysis on each of the sequences and their domain architectures to derive the final annotations.

These steps will be described more in detail in the next few subsections.

3.2 Step 1 – Extracting valid gene IDs

In this step, the input sequence file is read once and the gene IDs are extracted from the FASTA deflines. The gene ID is assumed to be the first word of the defline. If the deflines in the original FASTA file follow some other format, one can supply a regular expression in the *configuration file* that can be used to extract the actual ID from the first word of the defline.

3.3 Step 2 – Preliminary domain architecture

This step processes the domain assignment file and tries to determine a preliminary domain architecture for each sequence. A preliminary domain architecture considers known domains from InterPro only. Domain architectures for each sequence are determined in isolation, so the domain architecture of one sequence has no effect on another.

For each sequence, we first collect the set of domain assignments from the domain assignment file. Each assignment has a data source (e.g., HMMPfam, Superfamily, HMMSmart and so on), a domain ID according to the schema of the source, the starting and ending indices of the domain in the amino acid chain, an optional InterPro ID to which the domain ID is mapped, and an optional E-value. First, the list is filtered based on E-values, where one might apply different E-value thresholds for different data sources. This leads to a list of trusted domain assignments that are not likely to be artifacts. After that, GFam performs multiple passes on the list of trusted domain assignments, starting with a subset focused on more reliable data sources. Less reliable data sources join in the later stages, and it is possible that some data sources are not considered at all.

During the first pass, one single data source that is giving the highest coverage of the sequence is selected from the most reliable data sources. This data source will be referred to as the *primary data source*, and the domains of the primary data source will be called the *primary assignment*. After the first pass, the primary assignment will be extended by domains from other data sources in a greedy manner using the following rules:

1. Larger domains from other data sources will be considered first. (In other words, the remaining assignments not included already in the primary assignment are sorted by length in descending order).
2. Domains are considered one by one for addition to the primary assignment.
3. If a domain is the exact duplicate of some other domain already added (in the sense that it starts and ends at the same amino acid index), the domain is excluded from further consideration.
4. If a domain to be added overlaps with an already added domain from another data source, the domain is excluded from further consideration.
5. If a domain to be added is inserted *completely* into another domain from the same data source, it is added to the primary assignment and the process continues with the next domain from step 2. Note that the opposite cannot happen as we consider domains in decreasing order of their sizes.
6. If a domain to be added overlaps partially with an already added domain from the same data source, the size of the overlap decides what to do. Overlaps smaller than a given threshold are allowed, the domain will be added and the process continues from step 2. Otherwise, the domain is excluded from further consideration and the process continues from step 2 until there are no more domains left in the current stage.

We call this five-step procedure the *expansion* of a primary assignment. Remember, GFam works in multiple stages; the first stage creates the primary assignment with a limited set of trusted data sources, the second stage expands the primary assignment with an extended set of data sources, and there might be a third or fourth stage and so on with even more extended sets of data sources. For *Arabidopsis thaliana* and *Arabidopsis lyrata*, we found the following strategy to be successful:

1. Assignments from HAMAP, PatternScan, FPrintScan, Seg and Coil are thrown away completely for the following reasons:

- HAMAP may not be a suitable resource for eukaryotic family annotation as it is geared towards completely sequenced microbial proteome sets and provides manually curated microbial protein families in UniProtKB/Swiss-Prot [1]. For *Arabidopsis thaliana*, there were only 133 domains annotated by HAMAP and all domains had E-values larger than 0.001.
- PatternScan and FPrintScan [2] are resources for identifying motifs in a sequence and are not very helpful in understanding larger evolutionary units or domains. The match size ranges between 3 and 103 amino acids for PatternScan and between 4 and 30 amino acids for FPrintScan.
- Seg and Coil were ignored as these define regions of low compositional complexity and coiled coils, respectively, and are not particularly informative in the context of defining gene families.

2. An E-value threshold of 10^{-3} is applied to the remaining data sources, except for Superfamily, HMMPanther, Gene3D and HMMPIR which are taken into account without any thresholding.

   The threshold of 10^{-3} was chosen based on the following observation. There are 3,816 domain assignments from HMMPfam with a E-value larger than 0.1, 1,625 assignments with an E-value between 0.1 and 0.01 and 1,650 assignments with an E-value between 0.01 and 0.001. We looked at the type of domains that had an E-value between 0.1 and 0.01 and 0.01 and 0.001. We noticed that at least 80% of the domains are some kind of repeat domains (PPR, Kelch, LLR, TPR etc) or short protein motifs (different types of zinc fingers, EF-hand, HLH etc). It is reasonable to believe that at an E-value less than 0.001, the majority of the domains are likely to be spurious matches due to the sequence nature (low-complex and short) of these domains. We decided to consider domains from HMMPfam that had an E-value of 0.001 or smaller. We may miss but only a handful of real domains if we choose 0.001 as our E-value threshold. However, we would like to point out that the threshold is not hard-wired into GFam, rather it is a parameter than can be tuned for each assignment source to suit the users' needs.

3. GFam performs three passes on the list of domain assignments obtained up to now. The first and second passes do not consider HMMPanther and Gene3D assignments as they tend to split the sequence too much. The third stage considers all the data sources.

4. The maximum overlap allowed between two domains of the same source (excluding complete insertions which are always accepted) is 30 amino acids. This was based on the distribution of domain overlap lengths for the different resources.

The stages and the E-value thresholds are configurable in the *configuration file*.

3.4 Step 3 – Finding unassigned sequence fragments

This step begins the exploration for novel, previously uncharacterised domains among the sequence fragments left uncovered by the preliminary assignment that we calculated in *step 2*. We improvised on the method described by Haas *et al* [3] to identify novel domains. The step iterates over each sequence and extract the fragments that are not covered by any of the domains in the preliminary domain assignment. Sequences or fragments that are too short are thrown away, the remaining fragments are written in FASTA format into an intermediary file. The sequence and fragment length thresholds are configurable. For the analysis of *A.thaliana* and *A.lyrata* sequences, the minimum fragment length is set to 75 amino acids.

[1] Lima T, Auchincloss AH, Coudert E, Keller G, Michoud K, Rivoire C, Bulliard V, de Castro E, Lachaize C, Baratin D, Phan I, Bougueleret L and Bairoch A. HAMAP: a database of completely sequenced microbial proteome sets and manually curated microbial protein families in UniProtKB/Swiss-Prot. *Nucl Acids Res* **37**(Database):D471-D478, 2009.

[2] Scordis P, Flower DR and Attwood TK. FingerPRINTScan: intelligent searching of the PRINTS motif database. *Bioinformatics* **15**(10):799-806, 1999.

[3] Haas BJ, Wortman JR, Ronning CM, Hannick LI, Smith RK Jr, Maiti R, Chan AP, Yu C, Farzad M, Wu D, White O, Town CD. Complete reannotation of the *Arabidopsis* genome: methods, tools, protocols and the final release. *BMC Biol* **3**:7, 2005.

3.5 Step 4 – All-against-all BLAST comparison and filtering

This step uses the external `NCBI BLAST` executables (namely `formatdb` and `blastall`) to determine pairwise similarity scores between the unassigned sequence fragments. First, a database is created from all sequence fragments using `formatdb` in a temporary folder, then a BLAST query is run on the database with the same set of unassigned fragments using `blastall -p blastp`. Matches with a sequence percent identity or an alignment length less than a given threshold are thrown away, so are matches with an E-value larger than a given threshold. The user may choose between using unnormalised alignment lengths or normalised alignment lengths with various normalisation methods (normalising with the length of the smaller, the larger, the query or the hit sequence).

For *A.thaliana* and *A.lyrata*, the following settings were used:

- Minimum sequence identity: 45%
- Minimum normalised alignment length: 0.7 (normalisation done by the length of the query sequence)
- Maximum E-value: 10^{-3}

3.6 Step 5 – Calculation of Jaccard similarity

After the fourth step, we have essentially obtained a graph representation of similarity relations between unassigned sequence fragments. In this graph representation, each sequence fragment is a node, and two fragments are connected by an edge if they passed the BLAST filter in *step 4*. We will be looking for tightly connected regions in this graph in order to identify sequence fragments that potentially contain the same novel domain. It is a reasonable assumption that if two sequences contain the same novel domain, their neighbour sets in the similarity graph should be very similar. Jaccard similarity is a way of quantifying similarity between nodes in a graph by looking at their neighbour sets. Let *i* and *j* denote two nodes in a graph and let Γ_i denote the set consisting of *i* itself and *i*'s neighbours in the graph. The Jaccard similarity of *i* and *j* is then defined as follows:

$$\sigma_{ij} = \frac{|\Gamma_i \cap \Gamma_j|}{|\Gamma_i \cup \Gamma_j|}$$

where $|\ldots|$ denotes the size of a set. We calculate the Jaccard similarity of each connected pairs of nodes and keep those which have a Jaccard similarity larger than 0.66. This corresponds to keeping pairs where roughly 2/3 of their neighbours are shared. The Jaccard similarity threshold can be adjusted in the *configuration file*.

3.7 Step 6 – Identification of novel domains

Having obtained the graph filtered by Jaccard similarity in *step 5*, we detect the connected regions of this graph by performing a simple connected component analysis. In other words, sequence fragments corresponding to the same connected component of the filtered graph are assumed to belong to the same novel domain. Note that these novel domains should be treated with care, as some may belong to those that were already characterised in the original input domain assignment file but were filtered in *step 2*.

Novel domains are given temporary IDs consisting of the string `NOVEL` and a five-digit numerical identifier; for instance, `NOVEL00042` is the 42nd novel domain found during this process. Components containing less than four sequence fragments are not considered novel domains. The size threshold of connected components can be adjusted in the *configuration file*.

3.8 Step 7 – Consensus domain architecture

This step determines the final consensus domain architecture for each sequence by starting out from the preliminary domain architecture obtained in *step 2* and extending it with the novel domains found for the given sequence. The consensus domain architectures are written into two files, one containing a simpler flat-file representation of the consensus architectures suitable for further processing, while the other containing a detailed domain architecture description with InterPro IDs and human-readable descriptions for each domain in each sequence. This latter file also lists the primary data source for the sequence, the coverage of the sequence with and without novel domains, and also the number of the stage in which each domain was selected into the consensus assignment.

3.9 Step 8 – Functional label assignment

This step tries to assign a functional label to every sequence by looking at the list of its domains and collecting the corresponding Gene Ontology terms using a mapping file that assigns Gene Ontology terms to InterPro IDs. Such a file can be obtained from the InterPro2GO project. For each sequence, the collected Gene Ontology terms are filtered such that only those terms are kept which are either leaf terms (i.e. they have no descendants in the GO tree) or none of their descendants are included in the set of collected terms. These terms are then written in decreasing order of specificity to an output file, where specificity is assessed by the number of domains a given term is assigned to in the InterPro2GO mapping file; terms assigned to a smaller number of domains are considered more specific.

3.10 Step 9 – Overrepresentation analysis

This optional step conducts a Gene Ontology overrepresentation analysis on the domain architecture of the sequences given in the input file. For each sequence, we find the Gene Ontology terms corresponding to each of the domains in the consensus domain architecture of the sequence, and check each term using a hypergeometric test to determine whether it is overrepresented within the annotations of the sequence domains or not.

During the overrepresentation analysis, *multiple* hypergeometric tests are performed to determine the significantly overrepresented terms for a *single* sequence. GFam lets the user account for the effects of multiple hypothesis testing by correcting the p-values either by controlling the family-wise error rate (FWER) using the Bonferroni or Sidák methods, or by controlling the false discovery rate (FDR) using the Benjamini-Hochberg method.

The result of the overrepresentation analysis is saved into a human-readable text file that lists the overrepresented Gene Ontology terms in increasing order of p-values for each sequence.

CHAPTER

FOUR

SUPPLEMENTARY SCRIPTS

The scripts described in this chapter are not parts of the main GFam pipeline, but they provide useful extra functionality nevertheless. These scripts are found in the `bin` subdirectory of GFam, and they can be run separately from the command line, provided that GFam itself is on the Python path. Since the current directory is always on the Python path, it is best to run these scripts from the root directory of GFam.

4.1 Plotting descriptive statistics of the input file

The GFam pipeline has several parameters (e.g., E-value and domain length thresholds) with sensible default values, but in order to achieve the best results on a given dataset, these parameters can be adapted to the properties of the input data if necessary. Such decisions are made by humans after inspecting the distribution of E-values and domain lengths, and the distribution of overlap sizes between different domains in the InterPro domain assignment file. GFam can readily generate these plots using Matplotlib, a plotting library for Python. Matplotlib is available as a package in all major Linux distributions, and the project provides an installer for Microsoft Windows and Mac OS X.

The script can be invoked as follows (assuming that the configuration file is named `gfam.cfg`):

```
$ bin/plot.py -c gfam.cfg figurename
```

where *figurename* is the name of the figure to be plotted. You may also save figures to an output file:

```
$ bin/plot.py -c gfam.cfg -o *outfile*.pdf figurename1 figurename2 ...
```

The supported output formats include PDF, PNG, JPG and SVG, provided that the corresponding Matplotlib backends are installed. ASCII art representations of the histograms may also be printed if the extension of the output file is `.txt`.

To get a list of the supported figure names, specify `list` in place of the figure name:

```
$ bin/plot.py -c gfam.cfg list
```

The supported figures are as follows:

`evalue_distribution` Plots the count or relative frequency of domains with a given log E-value, sorted by different data sources in the InterPro input file, using a bin for each integer log E-value.

`length_distribution` Plots the count or relative frequency of domains with a given length, sorted by different data sources in the InterPro input file, using 25 bins up to a length of at most 750. The rightmost column contains all domains with length greater than 750.

`overlap_distribution` Plots the count or relative frequency of overlap length between all pairs of domains that overlap by at least one residue and have the same data source. The plots are sorted by data sources and use a bin width of 5.

4.1.1 Command line options

-a FILE, --assignment-file=FILE If you don't have a GFam configuration file or you want to run the script on a different InterPro assignment file (not the one specified in the configuration file), you may specify the name of the InterPro file directly using this switch. In this case, `-c` is not needed.

--cumulative Plot cumulative distributions (if that makes sense for the selected plot).

-o FILE, --output=FILE Specify the name of the file to save the plots to. The desired format of the file is inferred from its extension. Supported formats: PNG, JPG, SVG and PDF (assuming that the required Matplotlib backends are installed). You may also use a `.txt` extension here, which turns on `--text-mode` automatically.

--relative Plot relative frequencies instead of absolute counts on the Y axis (if that makes sense for the selected plot), and use a line chart instead of a bar chart.

--survival Plot survival distributions (if that makes sense for the selected plot), and use a line chart instead of a bar chart.

-t, --text-mode Print an ASCII art representation of each histogram. This option is useful if you are sitting at a non-graphical terminal (e.g., an ssh shell) or if you want to dump the histograms to a text file that you can analyze later. This option is turned on automatically if the extension of the output file is `.txt`.

4.2 Updating the mapping of IDs to human-readable names

GFam relies on an external tab-separated flat file to map domain IDs to human-readable descriptions when producing the final output. Such a file should contain at least the InterPro, Pfam, SMART and Superfamily IDs. The GFam distribution contains a script that can download the mappings automatically from known sources on the Internet. The script can be invoked as follows:

```
$ bin/download_names.py >data/names.dat
```

This will download the InterPro, Pfam, SMART and Superfamily IDs from the Internet and prepare the appropriate name mapping file in `data/names.dat`. If you wish to put it elsewhere, simply specify a different output file name. If you omit the trailing `>data/names.dat` part, the mapping will be written into the standard output. You can also compress the mapping file on-the-fly using `gzip` or `bzip2` and use the compressed file directly in the configuration file as GFam will uncompress it when needed. The following command constructs a compressed name mapping file:

```
$ bin/download_names.py | gzip -9 >data/names.dat.gz
```

Note that the script relies on the following locations to download data:

- <ftp://ftp.ebi.ac.uk/pub/databases/interpro/names.dat> for the InterPro name mapping
- <http://pfam.sanger.ac.uk/families?output=text> for the Pfam name mapping
- <http://smart.embl-heidelberg.de/smart/descriptions.pl> for the SMART name mapping
- <http://scop.mrc-lmb.cam.ac.uk/scop/parse/> for the SCOP description files (named `dir.des.scop.txt_X.XX`, where `X.XX` stands for the SCOP version number). It also relies on the most recent version of the SCOP description file being linked from the above page. The script will simply scan the links of the above page to determine what is the most recent version of SCOP. If the version number cannot be determined, the script will silently skip downloading the SCOP IDs.

CHAPTER

FIVE

API DOCUMENTATION

5.1 `gfam` – The main module

5.2 `gfam.assignment` – Routines related to sequence-domain annotations

5.3 `gfam.blast` – Handling BLAST file formats and utilities

5.4 `gfam.compat` – Compatibility classes for Python 2.5

5.5 `gfam.config` – Configuration file handling

5.6 `gfam.enum` – A simple enumeration class

5.7 `gfam.fasta` – FASTA parser and emitter

5.8 `gfam.go` – Handling the Gene Ontology

5.8.1 `gfam.go.obo` – Parsing OBO ontology files

5.8.2 `gfam.go.overrepresentation` – Overrepresentation analysis

5.9 `gfam.interpro` – Handling InterPro-related files

5.10 `gfam.modula` – Modular calculation framework

Modula is a modular calculation framework for Python that allows you to define tasks that depend on input files and on each others. Modula will figure out which tasks have to be executed in which order in order to calculate the final results – this is done by a simple depth first search on the task dependency graph.

Modula started out as a separate project, and you don't have to know its internals in order to use the GFam API. The only reason why it has been placed as a submodule of GFam is to avoid foricng users to install Modula separately. The

Modula API is not documented here as it is not an internal part of GFam. Modula is used only by the GFam master script (see `gfam.scripts.master`) to execute the calculation steps in the proper order.

5.11 `gfam.sequence` – Simple sequence and sequence record classes

5.12 `gfam.scripts` – Command line scripts

Each step in the GFam pipeline is implemented in a separate submodule of `gfam.scripts`. These submodules contain only a single class per submodule, derived from `gfam.scripts.CommandLineApp`. The submodules are invoked automatically in the right order by a master script in `gfam.scripts.master`. In general, you only have to invoke the master script and it will do the rest for you, but some of the steps might be useful on their own, so they can be invoked independently from the command line as:

```
$ python -m gfam.scripts.modulename
```

where *modulename* is the name of the submodule to be executed. You can get usage information for each submodule by typing:

```
$ python -m gfam.scripts.modulename --help
```

5.13 `gfam.utils` – Utility classes and functions

CHAPTER

SIX

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*