gFunc Documentation

Release 0.1

Augustine (Gus) Dunn

January 11, 2013

CONTENTS

Contents:

CHAPTER

ONE

GFUNC TUTORIAL

Here be the learing of the majic.

CHAPTER

TWO

PROJECT SUMMARY

2.1 Goals

- Galaxy integration
- Graph based integration of multiple OMICs data streams
- Rank/cull gene names based on multidimentional association with user defined weigh structure.

2.2 Lessons Learned

- None so far!

CHAPTER THREE

GFUNC AUTO-GENERATED CODE DOCUMENTATION

3.1 gfunc

Root directory and main package namespace.

3.2 parsers

Code supporting parsing of supported raw data files.

3.2.1 base.py

Code defining base parser class: TODO.

class `gfunc.parsers.base.`**`GFuncParserBase`**

TODO: define once I have settled on common needs.

3.2.2 Cufflinks.py

Code supporting parsing of Cufflinks type raw data files.

class `gfunc.parsers.Cufflinks.`**`CDiffFpkmTrackerParser`**(*cuffdiff_fpkm_path*, *species*, *cuffdiff_exp_path=None*, *name_col='nearest_ref_id'*, *combine_transcripts=True*, *tx_2_gene=None*)

Class to accept CuffDiff FKPM data table and init/update the relevant gFuncNode Objects.

`__init__`(*cuffdiff_fpkm_path*, *species*, *cuffdiff_exp_path=None*, *name_col='nearest_ref_id'*, *combine_transcripts=True*, *tx_2_gene=None*)

Test doc for init'ing CuffDiff parser.

`resgister_nodes_and_edges`(*node_dict*, *edge_dict*, *graph*)

Parses each row from the CuffDiff FKPM data table and either adds the data to the relevant GFuncNode in node_dict or creates one and adds it to that then registers it in node_dict.

`gfunc.parsers.Cufflinks.`**`am_i_sigDiff`**(*xloc_number*, *expDiffTable_dict*, *q_thresh*) for lines in cuffdiff_fpkm_table:

Return `True` if:

at least one of current line's XLOC_xxxx pair tests in `expDiffTable_dict` has *q_val* <= `q_thresh`

Else Return `False`

`gfunc.parsers.Cufflinks.`**`build_expDiffTable_dict`**(*expDiffTable_path*)

Build isoformExpDiffTable_dict:

Keys: XLOC_xxxxx

Values: namedtuple-ified rows with same XLOC_xxxxx

`gfunc.parsers.Cufflinks.`**`transfer_nearestRefgeneSymbol_from_isoform_to_gene_tracking`**(*isoform_f* *gene_fpk*

TODO: doc

3.2.3 edge_lists.py

Code supporting parsing of lists into edge connections.

class `gfunc.parsers.edge_lists.`**`OneToOneOrthoListParser`**(*list_path='`*, *divergence_info=None*, *relation_type='one_to_one_ortholog'*)

Class to accept list of rows when each item/node_name in the row should have edges to all other items in the row and init the relevant gFuncNode/gFuncEdges Objects. Each Column should have a header that is supported by gfunc.fileIO.tableFile2namedTuple representing the species of the nodeName in that column (Anopheles_gambiae).

`__init__`(*list_path='`*, *divergence_info=None*, *relation_type='one_to_one_ortholog'*)

TODO: doc for init'ing.

`resgister_nodes_and_edges`(*node_dict*, *edge_dict*, *graph*)

Iterates through every row in list_path ensuring that a GFuncNode exists for each nodeName and is registered. GFuncNodes are initialized with basic info (name,species) if it doesnt already exist. Then GFuncEdge objects are registered/initialized for nodeName combinations in each row while setting <relation_type> data_type to 'True' for each edge.

`gfunc.parsers.edge_lists.`**`combine_multiple_one2one_tables`**(*path_list*, *species_prefixes=['AGAP', 'AAEL', 'CPIJ']*)

Builds a set of linked dicts with:

Keys: GeneName

Values: Link to other GeneName keys that are supossed one2one orthologs

`gfunc.parsers.edge_lists.`**`follow_all_links`**(*graph*, *node*)

Return all nodes in connected subgraph wrt node.

3.2.4 ETE.py

Code supporting parsing phyloXML files using the ETE2 package.

class `gfunc.parsers.ETE.`**`PhyloXMLParser`**(*phyloXML_path='`*, *species=[ ]*, *pickle_path=None*)

Class to accept PhyloXML files or directories and init the relevant gFuncNode/gFuncEdges Objects.

RIGHT NOW: only used for branch_length

__init__(*phyloXML_path=''*, *species=[ ]*, *pickle_path=None*)
Test doc for init'ing.

get_distance(*leaf1*, *leaf2*)
For two leaf objs in a common tree, returns the branch length that separates them.

get_species(*leaf*)
Returns the species scientific name of a leaf obj.

resgister_nodes_and_edges(*node_dict*, *edge_dict*, *graph*)
Iterates through every leaf in every tree in self.trees ensuring that a GFuncNode exists for each leaf and is registered. GFuncNodes are initialized with basic info (name,species) if it doesnt already exist. Then GFuncEdge objects are registered/initialized for leaf combinations in each tree while setting 'branch_length' data for each edge.

gfunc.parsers.ETE.**load_phyloXMLs**(*path*, *species=None*, *pickle_path=None*)
Loads at least one phyloXML file and returns a Phyloxml() project containing at least one phyloXML tree. If path is a directory, all subdirectories are scrubed for xml files too.

If species!=None, prunes trees to leaves that are of the supplied scientific species names (can save a LOT of memory if the number of trees is large).

path = str() species = list()

gfunc.parsers.ETE.**prune_trees_by_species**(*ete2_tree_list*, *species_list*)
XXXX

3.2.5 GTF.py

Code supporting parsing/indexing of GTF/GFF data files.

NOTE: The "business" end of this code uses (read: depends on) the gtf_to_genes module:

Metadata-Version: 1.0 Name: gtf-to-genes Version: 1.07 Summary: Fast GTF parser Home-page: http://code.google.com/p/gtf-to-genes/ Author: Leo Goodstadt Author-email: gtf_to_genes@llew.org.uk License: MIT

3.2.6 JASPAR.py

Code supporting reading and writing of basic PSSMs representing TFBS profiles; especially in JASPAR format.

class gfunc.parsers.JASPAR.**BasicTFBSParser**(*tfbs_path*)
Class to accept TFBS profile data table from 'find_motifs.py' and init/update the relevant gFuncNode Objects.

__init__(*tfbs_path*)
Test doc for init'ing TFBS parser.

resgister_nodes_and_edges(*node_dict*, *edge_dict*, *graph*)
Parses each row from the TFBS data table and either adds the data to the relevant GFuncNode in node_dict or creates one and adds it to that then registers it in node_dict.

class gfunc.parsers.JASPAR.**ParseJasparMatrixOnly**(*filePath*)
Returns a record-by-record motif parser for JASPAR matric_only.txt files analogous to file.readline().

Example:

```
>MA0001.1 AGL3
A  [ 0  3 79 40 66 48 65 11 65  0 ]
C  [94 75  4  3  1  2  5  2  3  3 ]
G  [ 1  0  3  4  1  0  5  3 28 88 ]
```

```
T  [ 2 19 11 50 29 47 22 81  1  6 ]
>MA0002.1 RUNX1
A  [10 12  4  1  2  2  0  0  0  8 13 ]
C  [ 2  2  7  1  0  8  0  0  1  2  2 ]
G  [ 3  1  1  0 23  0 26 26  0  0  4 ]
T  [11 11 14 24  1 16  0  0 25 16  7 ]
```

__init__(*filePath*)
Returns a record-by-record motif parser analogous to file.readline(). Exmpl: parser.next() Its ALSO an iterator so "for rec in parser" works too!

next()
Reads in next element, parses, and does minimal verification. *RETURNS:* tuple: (seqName,seqStr)

to_dict()
Returns a single OrderedDict populated with the motifRecs contained in self._file.

3.2.7 MAST.py

Code supporting parsing of MAST type raw data files.

3.3 analysis_classes.py

Code defining controller classes for supported analysis types.

class `gfunc.analysis_classes.`**BranchLength**(*poll_me=False*)
TODO: doc

__init__(*poll_me=False*)
TODO: doc

measure_relation(*gfunc_edge*)
TODO: doc

class `gfunc.analysis_classes.`**ExpressionSimilarity**(*poll_me=False*)
TODO: doc

__init__(*poll_me=False*)
TODO: doc

class `gfunc.analysis_classes.`**Metric**(*poll_me=False*)
TODO: doc

__init__(*poll_me=False*)
TODO: doc

mean(*greater_than=0*)
Returns the mean of the encountered values greater than the value provided.

measure_relation(*gfunc_edge*)
TODO: doc

median(*greater_than=0*)
Returns the median of the encountered values greater than the value provided.

class `gfunc.analysis_classes.`**PhyloExpnCorrelationIndex**(*poll_me=False*)
TODO: doc

__init__(*poll_me=False*)
TODO: doc

class `gfunc.analysis_classes.`**RelationsHandler**(*list_of_metrics*)
Relations are metrics that characterize how an edge's 2 connected nodes relate given some type of relationship (branch length, expression profile similarity, etc).

__init__(*list_of_metrics*)
TODO: Doc

get_vote_types()
TODO: Doc

measure_relations(*edge_dict*)

```
For each gfunc_edge in edge_dict:
    iterates through metrics:
        calculates & stores result in gfunc_edge and metric_handler
```

class `gfunc.analysis_classes.`**TFBSSimilarity**(*poll_me=False*)
TODO: doc

__init__(*poll_me=False*)

class `gfunc.analysis_classes.`**VoteHandler**(*graph*)
VoteHandlers determine how well the current GFuncNode's neighborhood agrees with it regarding each type of relation being used. The result is a weighted mean, weighted by a strength (or trustability metric) between each neighbor and the current node (exp: branch length, p-value, etc). If no weight relationship is specified, the result is a standard mean (weights are equal).

__init__(*graph*)
TODO: Doc

set_vote_types(*vote_types*, *weight_by=None*)
Recieves and stores as LIST 'Metric.relation_metric' string for each Metric class that needs a vote taken.

weight_by: one or none of the gfunc_edge.data key strings to use to weight the votes of each node's neighbor edge metrics. (weight_by=None results in equal weights)

take_votes(*node_list*, *poll_func=None*)
TODO: Doc

3.4 clustering.py

Code supporting efforts to optimize and automate external clustering libraries for gFunc purposes.

`gfunc.clustering.`**plot_centers_and_points**(*data*, *clusters*, *means*, *truth=None*)

class `gfunc.data_classes.`**Bunch**(**args*, ***kwds*)
A dict like class to facilitate setting and access to tree-like data.

__init__(**args*, ***kwds*)

class `gfunc.data_classes.`**GFuncEdge**(*node1*, *node2*)
TODO: Doc

__init__(*node1*, *node2*)
TODO: Doc

set_data (*data*, *data_type*)
TODO: Doc

class `gfunc.data_classes.`**GFuncNode** (*name*, *species*, *graph*, *is_target=False*, *debug=False*)
TODO: Doc

__init__ (*name*, *species*, *graph*, *is_target=False*, *debug=False*)
TODO: Doc

get_copy ()
Returns a deep copy of the node.

get_sub_scores (*target_node*, *graph*)

set_data (*data*, *data_type*)
TODO: Doc

total_votes ()

`gfunc.data_classes.`**bunchify** (*dict_tree*)
TODO: doc

3.5 ensembl_data.py

Code supporting automated retrieval/mirroring/processing of ensembl data files and directories.

WARNING: To some extent the functionality of this code is dependant on the current formating of Ensembl's internal directory structure and how it responts to `urllib2.urlopen(url)`.

class `gfunc.ensembl_data.`**DataGrabber** (*base_url*, *species*, *data_types*, *base_local_path*, *verbose=False*)
Class to manage conecting to ensembl-based ftp data dumps and retrieving them.

__init__ (*base_url*, *species*, *data_types*, *base_local_path*, *verbose=False*)
Initiate DataGrabber object for conecting to ensembl-based ftp data dumps.

settings ()

RETURNS:

- dict of current settings for ensembl access.

transfer_data (*unzip=False*)

GIVEN:

- sufficently initiated `self` instance

DOES:

- [x] decides which method to use to get the data
- [x] creates local directory if needed based on base_local_path
- [x] initiates data transfer
- [?] complains if it detects incomplete transfer
- [?] if unzip evaluates to True, recursively unzip any files with extentions suggesting they are compressed.

RETURNS:

- None

`gfunc.ensembl_data.`**`check_files`**(*dir_path*, *cksum_data*)

GIVEN:

- `dir_path` = path to a directory containing files to be validated
- `cksum_data` = parsed contents of the CHECKSUMS file for this directory

DOES:

- Compares files in the directory with checksums in the CHECKSUMS file and scores them as PASS/FAIL
- Documents and classifies discrepancies between files listed in CHECKSUMS file vs files actually in the directory: classifies them as NOT_IN_CHECKSUMS or NOT_IN_DIR.

RETURNS:

- `results` = list of strings

`gfunc.ensembl_data.`**`validate_downloads`**(*base_dir*)

GIVEN:

- `base_dir` = top-level directory containing files to be validated.

DOES:

- Decends into `base_dir` and records paths to all lower-level files.
- Uses the CHECKSUMS files it finds to set up external system calls to `sum` for each file listed in the direcory's CHECKSUMS file.
- **A file named VALIDATIONS is created in each directory listing each file in the directory and one of the followin**
  - PASS = passed checksum match
  - FAIL = failed checksum match
  - NOT_IN_CHECKSUMS = file found in directory but NOT listed in the CHECKSUMS file
  - NOT_IN_DIR = file listed in CHECKSUMS file, but not found in the directory.

RETURNS:

- `results` = a summary of all VALIDATIONS files (list of strings)

`gfunc.ensembl_data.`**`web_ls`**(*url*)

GIVEN:

- `url` = the url of an ensembl-based ftp:// target

DOES:

- reads the url data and extracts file/directory names
- stores info in a dict named `contents`; keyed by 'dirs' or 'files' and pointing to lists of respective urls.

RETURNS:

- `contents` dictionary.

`gfunc.ensembl_data.`**`web_walk`**(*base_url*)

GIVEN:

- `base_url` = the url of an ensembl-based ftp:// target directory

DOES:

- recursively stores directories and file urls (uses `web_ls`), continues to follow directories until all file urls have been collected below `base_url`.

RETURNS:

- `file_urls` = list of file url strings under `base_url`.

3.6 errors.py

Code defining custom base error classes to provide a foundation for graceful error handling.

exception `gfunc.errors.`**`GFuncError`**
Base class for exceptions in the gFunc package.

exception `gfunc.errors.`**`SystemCallError`**(*errno*, *strerror*, *filename=None*)
Error raised when a problem occurs while attempting to run an external system call.

Attributes:

`errno` – return code from system call
`filename` – file in volved if any
`strerror` – error msg

`__init__`(*errno*, *strerror*, *filename=None*)

exception `gfunc.errors.`**`UnsatisfiedDependencyError`**
Exception raised when gFunc can not find a suitable option to satisfy an external dependency.

3.7 externals.py

Code supporting running external system processes.

`gfunc.externals.`**`mkdirp`**(*path*)
Create new dir while creating any parent dirs in the path as needed.

`gfunc.externals.`**`runExternalApp`**(*progName*, *argStr*)
Convenience func to handle calling and monitoring output of external programs.

3.8 fdr.py

Code supporting easy generation of emprical FDR for results.

`gfunc.fdr.`**`shuffle_dict`**(*original_dict*, *shuffle_count*)

GIVEN:

- `original_dict`
- `shuffle_count`

RETURNS:

- generator that yields randomly shuffled key/value pairs as a new dict `shuffle_count` times.

3.9 fileIO.py

Code supporting reading and writing from files not related to specific parsers.

`gfunc.fileIO.`**`tableFile2namedTuple`**(*tablePath*, *sep='\t'*, *headers=None*)
: Returns namedTuple from table file using first row fields as col headers or a list supplied by user.

`gfunc.fileIO.`**`walk_dirs_for_fileName`**(*dir_path*, *pattern='*.xml'*)
: Recursively collects file paths in a dir and subdirs.

3.10 graphTools.py

Code supporting building and querying the graphs.

class `gfunc.graphTools.`**`GraphBuilder`**(*parsers*)

`__init__`(*parsers*)

`map_registries_to_graph`(*nodes=True*, *edges=True*)
: Iterates through each registry creating graph nodes and edges. Returns GraphHandler.

`populate_registries`()
: Iterates through provided parsers and calls the parsers 'resgister_nodes_and_edges' method to populate/update the relevant registries (node/edge_dict).

class `gfunc.graphTools.`**`GraphHandler`**(*node_dict*, *edge_dict*, *graph*)
: TODO: Doc

`__init__`(*node_dict*, *edge_dict*, *graph*)
: TODO: Doc

`clone_node_as_target`(*node_name*)
: TODO: Doc

`install_metric_handlers`(*rel_hndler*, *vote_hndlr*)
: TODO: Doc

`install_target`()
: TODO: Doc

`measure_relations`()
: Cues RelationsHandler to do its thing after pasing it self.edge_dict.

`take_votes`(*node_list*, *poll_func=None*)
: Cues VoteHandler to do its thing after pasing it a list of specific GFuncNode objects.

Example: >>> node_list = [node for node in node_dict.itervalues() if node.species == 'Anopheles gambie'].

3.11 maths.py

Code supporting specialized calculations for gfunc.

`gfunc.maths.`**`bayesian_score`**(*c*, *m*, *n*, *scores*, *scale_mod=1*)

BS = ((c * m) + sum([x for x in scores])) / (n + c)

Where:

`n`: number of votes for THIS item

`C`: median number of votes for all items that got at least 1 vote (weighting or dampening factor)

`m`: median UNweighted score for all items that got at least 1 vote

`gfunc.maths.`**`weight_d_for_ptci`**(*d_i*, *d_min*, *d_max*, *w_min=1.0*, *w_max=1.1*)

Scaling function to transform ‘d’ onto a weight-spectrum to either punish or reward the final ptci score based on the phylogenetic distance between the two current species as it relates to the range of phylogenetic distances in the data set.

Default weight scale is no change for the shortest distance (return 1.0) to a 10% reward for the longest distance (return 1.1).

3.12 motifs.py

Code supporting the searching, recording, and analysis of sequence motifs for gFunc.

`gfunc.motifs.`**`load_MOODS_result`**(*in_path*)

GIVEN:

- `in_path`: path to store results

DOES:

- loads `processed_moods_result_dict` from binary pickle to `in_path`.

RETURNS:

- un-pickled MOODS result object.

`gfunc.motifs.`**`motif_profiles_weighted_by_score`**(*processed_moods_result_dict*)

GIVEN:

- processed_moods_result_dict: output from (def `process_MOODS_results()`)

DOES:

- iterates through `processed_moods_result_dict` and calculates a motif presence score (mps) for each motifName:seqName pair by summing the all positive `site_scores` for motifName in seqName.
- mps are recorded in a `pandas.DataFrame` with columns = motifName and indexs = seqName.

RETURNS:

- mps_table: `pandas.DataFrame` constructed as above.

`gfunc.motifs.`**`process_MOODS_results`**(*moods_result_dict*, *motif_names*)

GIVEN:

- `moods_result_dict`: orderedDict ???is this true??? of moods_results_tuples keyed by seqName/geneName
- `motif_names`: correctly ordered motif names (Motifs.motifs.keys())

DOES:

- converts `moods_result_dict` into a three key'd multi-level dict as follows: `processed_moods_result_dict[SeqName][motifName][location] = score`

RETURNS:

- `processed_moods_result_dict`

`gfunc.motifs.`**`save_MOODS_result`**`(`*moods_hits*, *out_path*`)`

GIVEN:

- `moods_hits`: non-processed result from `Motif.scan_seq()` or `Motif.scan_seqDict()`.
- `out_path`: path to store results

DOES:

- stores `moods_hits` as binary pickle to `out_path`.

RETURNS:

- `None`

3.13 stats.py

Code supporting calculations of statistical probabilities for gfunc.

`gfunc.stats.`**`basic_bootstrap_est`**`(`*vec*, *reps=1000*`)`

GIVEN:

- `vec` = vector of sample values
- `reps` = number of resampling reps

DOES:

- Resample w/ replacement `reps` times and record the medians
- Calculate stdv of resampled medians which should approach the actual SE as `reps` approaches `inf`.
- Calculate the 95% CI bounds.

RETURNS:

- tuple([*median of resampled medians*, *SE est*, *loBound*, *hiBound*])

`gfunc.stats.`**`benjHochFDR`**`(`*table*, *pValColumn=-1*`)`

GIVEN:

- `table`: 2D list(*hypothesis*,*p-value*) hypothesis could = *geneName* tested for enrichment
- `pValColumn`: integer of column index containing the *p-value*.

DOES:

- Calculates the Benjamini-Hochberg adjusted *p-values*

RETURNS:

- a new version of `table` with an extra column added to the end representing the BH corrected *p-values*

`gfunc.stats.`**`binComb`**`(`*n*, *k*`)`

GIVEN:

- `n`

- `k`

DOES:

- Computes `n` *choose* `k`.

RETURNS:

- The number of ways `k` objects can be sampled from a population of size `n`.

`gfunc.stats.`**`binomialPval`**(*n*, *k*, *p*)

RETURNS:

- exact binomial P-value.

`n` = number of trials
`k` = number of successes
`p` = probability of a success

P(k succeses in n trials) = choose(n,k) (p^k) ((1-p)^(n-k))

`gfunc.stats.`**`binomialPval_gte`**(*n*, *k*, *p*)

RETURNS:

- binomial *p-value* of `k` or greater successes in `n` trials with probability of success for each trial `p`.

`n` = number of trials
`k` = number of successes
`p` = probability of a success

sum(choose(n,k) (p^k) ((1-p)^(n-k))) as `k` goes from `k` to `n`

`gfunc.stats.`**`cumHypergeoP`**(*n*, *i*, *m*, *N*)

Calculates the cumulative hypergeometric *p-value* for variables:

`n` = number of positives in population
`i` = number of positives in sample
`m` = number of negatives in population
`N` = sample size

P(i) = sum([as i->N] (choose(n,i)choose(m,N-i))/choose(n+m,N))

For more details -> http://mathworld.wolfram.com/HypergeometricDistribution.html

`gfunc.stats.`**`hypergeoP`**(*n*, *i*, *m*, *N*)

Calculates the non-cumulative hypergeometric *p-value* for variables:

`n` = number of positives in population
`i` = number of positives in sample
`m` = number of negatives in population
`N` = sample size

P(x=i) = (choose(n,i)choose(m,N-i))/choose(n+m,N)

For more details -> http://mathworld.wolfram.com/HypergeometricDistribution.html

3.14 galaxy_tools

Code supporting the integration of gFunc with the Galaxy paradigm.

So far this is empty!

CHAPTER

FOUR

INDICES AND TABLES

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

PYTHON MODULE INDEX

INDEX