GOcats Documentation

Release 1.2.1

Eugene Hinderer

Jun 16, 2023

Contents:

GOcats

GOcats is an Open Biomedical Ontology (OBO) parser and categorizing utility–currently specialized for the Gene Ontology (GO)–which can help scientists interpret large-scale experimental results by organizing redundant and highly-specific annotations into customizable, biologically-relevant concept categories. Concept subgraphs are defined by lists of keywords created by the user.

Currently, the *GOcats* package can be used to:

- Create subgraphs of GO which each represent a user-specified concept.

- Map specific, or fine-grained, GO terms in a Gene Annotation File (GAF) to an arbitrary number of concept categories.

- Remap ancestor Gene Ontology term relationships and the gene annotations with a set of user defined relationships.

- Explore the Gene Ontology graph within a Python interpreter.

1.1 Citation

Please cite the following papers when using GOcats:

Hinderer EW, Moseley NHB. GOcats: A tool for categorizing Gene Ontology into subgraphs of user-defined concepts. PLoS One. 2020;15(6):1-29.

Hinderer EW, Flight RM, Dubey R, Macleod JN, Moseley HNB. Advances in Gene Ontology utilization improve statistical power of annotation enrichment. PLoS One. 2019;14(8):1-20.

1.2 Installation

GOcats runs under Python 3.4+ and is available through python3-pip. Install via pip or clone the git repo and install the following dependencies and you are ready to go!

1.2.1 Install on Linux

Pip installation

Dependencies should be automatically installed using this method. It is strongly recommended that you install with this method. .. code:: bash

> pip3 install gocats

GitHub Package installation

Make sure you have git installed:

```
cd ~/
git clone https://github.com/MoseleyBioinformaticsLab/GOcats.git
```

Dependencies

GOcats requires the following Python libraries:

- docopt for creating the gocats command-line interface.
- JSONPickle for saving Python objects in a JSON serializable form and outputting to a file.

To install dependencies manually:

```
pip3 install docopt
pip3 install jsonpickle
```

1.2.2 Install on Windows

GOcats can also be installed on windows through pip.

1.3 Quickstart

For instructions on how to format your keyword list and advanced argument usage, consult the tutorial, guide, and API documentation at readthedocs.

Subgraphs can be created from the command line.

```
python3 -m gocats create_subgraphs /path_to_ontology_file ~/GOcats/gocats/exampledata/
→examplecategories.csv ~/Output --supergraph_namespace=cellular_component --subgraph_
→namespace=cellular_component --output_termlist
```

Mapping files can be found in the output directory:

- GC_content_mapping.json_pickle # A python dictionary with category-defining GO terms as keys and a list of all subgraph contents as values.
- GC_id_mapping.json_pickle # A python dictionary with every GO term of the specified namespace as keys and a list of category root terms as values.

GAF mappings can also be made from the command line:

```
python3 -m gocats categorize_dataset YOUR_GAF.goa YOUR_OUTPUT_DIRECTORY/GC_id_mapping.
↪json_pickle YOUR_OUTPUT_DIRECTORY MAPPED_DATASET_NAME.goa
```

Gene to GO Term remappings with consideration of `has_part` relationships can created from the command line:

```
python3 -m gocats remap_goterms /path_to_ontology_file.obo /path_to_gaf.goa ancestors_
↪output.json namespace_output.json --allowed_relationships=is_a,part_of,has_part --
↪identifier_column=1
```

Gene to GO terms will be in JSON format in `ancestor_output.json`, and new GO term to namespace in `namespace_output.json`.

1.4 License

Made available under the terms of The Clear BSD License. See full license in LICENSE.

The Clear BSD License

Copyright (c) 2017, Eugene W. Hinderer III, Hunter N.B. Moseley All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted (subject to the limitations in the disclaimer below) provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

1.5 Authors

- **Eugene W. Hinderer III** - ehinderer

- **Hunter N.B. Moseley** - hunter-moseley

1.5.1 The GOcats API Reference

The following are located in /GOcats/gocats.

The Gene Ontology Categories Suite (GOcats)

This module provides methods for the creation of directed acyclic concept subgraphs of Gene Ontology, along with methods for evaluating those subgraphs.

`gocats.gocats.`**`build_graph`**(*args*)
> **Not yet implemented**

> Try build_graph_interpreter to create a GO graph object to explore within a Python interpreter.

`gocats.gocats.`**`build_graph_interpreter`**(*database_file*, *supergraph_namespace=None*, *allowed_relationships=None*, *relationship_directionality='gocats'*)
> Creates a graph object of GO, which can be traversed and queried within a Python interpreter.

> **Parameters**

> - **`database_file`** (*file_handle*) – Ontology database file.

> - **`supergraph_namespace`** (*str*) – Optional - Filter graph to a sub-ontology namespace.

> - **`allowed_relationships`** (*list*) – Optional - Filter graph to use only those relationships listed.

> - **`relationship_directionality`** – Optional - Any string other than 'gocats' will retain all original GO relationship directionalities. Defaults to reverseing has_part direction.

> **Returns** A Graph object of the ontology provided.

> **Return type** class

`gocats.gocats.`**`categorize_dataset`**(*dataset_file*, *term_mapping*, *output_directory*, *mapped_dataset_filename*, *dataset_type='GAF'*, *entity_col=0*, *go_col=1*, *retain_unmapped_annotations=False*)
> Reads in a Gene Annotation File (GAF) and maps the annotations contained therein to the categories organized by GOcats or other methods. Outputs a mapped GAF and a list of unmapped genes in the specified output directory.

> **Parameters**

> - **`dataset_file`** – A file containing gene annotations.

> - **`term_mapping`** – A dictionary mapping category-defining ontology terms to their subgraph children terms. May be produced by GOcats or another method.

> - **`output_directory`** – The directory where the output file will be stored.

> - **`mapped_dataset_filename`** – The desired name of the mapped GAF.

> - **`dataset_type`** – Enter file type for dataset [GAF|TSV|CSV]. Defaults to "GAF".

> - **`entity_col`** – If CSV or TSV file type, indicate which column the entity IDs are listed. Defaults to 0.

> - **`go_col`** – If CSV or TSV file type, indicate which column the GO IDs are listed. Defaults to 1.

> - **`retain_unmapped_annotations`** – If specified, annotations that are not mapped to a concept are copied into the mapped dataset output file with its original annotation.

> **Returns** None

> **Return type** None

gocats.gocats.**create_subgraphs**(*database_file, keyword_file, output_directory, super-graph_namespace=None, subgraph_namespace=None, supergraph_relationships=['is_a', 'part_of', 'has_part'], subgraph_relationships=['is_a', 'part_of', 'has_part'], map_supersets=False, output_termlist=False, go_basic_scoping=False, network_table_name=None, test=False*)

Creates a graph object of an ontology, processed into `gocats.dag.OboGraph` or to an object that inherits from `gocats.dag.OboGraph`, and then extracts subgraphs which represent concepts that are defined by a list of provided keywords. Each subgraph is processed into `gocats.subdag.SubGraph`.

> **Parameters**
>
> - **database_file** – Ontology database file.
>
> - **keyword_file** – A CSV file with two columns: column 1 naming categories, and column 2 listing search strings (no quotation marks, separated by semicolons).
>
> - **output_directory** – The directory where results are stored.
>
> - **supergraph_namespace** – a supergraph sub-ontology to filter e.g. cellular_component, optional
>
> - **subgraph_namespace** – a subgraph sub-ontology to filter e.g. cellular_component, optional
>
> - **supergraph_relationships** – a list of relationships to limit in the supergraph e.g. ['is_a', 'part_of'], optional
>
> - **subgraph_relationships** – a list of relationships to limit in subgraphs e.g. ['is_a', 'part_of'], optional
>
> - **map_supersets** – whether to allow subgraphs to subsume other subgraphs, logical, optional
>
> - **output_termlist** – whether to create a translation of ontology terms to their names to improve interpretability of dev test results, logical, optional
>
> - **go-basic-scoping** – whether to create a GO graph similar to go-basic with only scoping-type relationships (is_a and part_of), logical, optional
>
> - **network_table_name** – whether to make a specific name for the network table produced from the subgraphs (defaults to NetworkTable.csv)
>
> **Returns** None
>
> **Return type** `None`

gocats.gocats.**find_category_subsets**(*subgraph_collection*)

Finds subgraphs which are subsets of other subgraphs to remove redundancy, when specified.

> **Parameters** **subgraph_collection** – A dictionary of subgraph objects (keys: subgraph name, values: subgraph object).
>
> **Returns** A dictionary relating which subgraph objects are subsets of other subgraphs (keys: subset subgraph, values: superset subgraphs).
>
> **Return type** `dict`

gocats.gocats.**json_format_graph**(*graph_object, graph_identifier*)

Creates a dictionary representing the edges in the graph and formats it in such a way that it can be encoded into JSON for comparing the graph objects between versions of GOcats.

gocats.gocats.**remap_goterms**(*go_database*, *goa_gaf*, *ancestor_filename*, *namespace_filename*, *allowed_relationships*, *identifier_column*)

Reads in a Gene Ontology relationship file, and a Gene Annotation File (GAF), and follows the GOcats rules for allowed term-to-term relationships. Generates as output a new GAF, and a new term to ontology namespace mapping.

> **Parameters**
>
> - **go_database** – the gene ontology dataset
> - **goa_gaf** – the gene annotation file
> - **ancestor_filename** – the output file containing new gene to ontology mappings
> - **namespace_filename** – the output file containing the term to ontology mappings
> - **allowed_relationships** – what term to term relationships will be considered (is_a,part_of,has_part)
> - **identifier_column** – which column is being used for the gene identifiers (1)
>
> **Returns** None
>
> **Return type** None

Directed Acyclic Graph (DAG)

Contains necessary objects for creating a Directed Acyclic Graph (DAG) object to represent Open Biomedical Ontologies (OBO).

class gocats.dag.**OboGraph**(*namespace_filter=None*, *allowed_relationships=None*)

A pythonic graph of a generic Open Biomedical Ontology (OBO) directed acyclic graph (DAG).

__init__(*namespace_filter=None*, *allowed_relationships=None*)

OboGraph initializer. Leave *namespace_filter* and *allowed_relationship* as None to create the entire ontology graph. Otherwise, provide filters to limit what information is pulled into the graph.

> **Parameters**
>
> - **namespace_filter** (*str*) – Specify the namespace of a sub-ontology namespace, if one is available for the ontology.
> - **allowed_relationships** (*list*) – Specify a list of relationships to utilize in the graph, other relationships will be ignored.

orphans

property defining a set of nodes in the graph which have no parents. When the graph is modified, calls *_update_graph()* to repopulate the sets of orphan and leaf nodes.

> **Returns** Set of 'orphan' *gocats.dag.AbstractNode* objects.
>
> **Return type** set

leaves

property defining a set of nodes in the graph which have no children. When the graph is modified, calls *_update_graph()* to repopulate the sets of orphan and leaf nodes.

> **Returns** Set of 'leaf' *gocats.dag.AbstractNode* objects.
>
> **Return type** set

valid_node(*node*)

Defines condition of a valid node. Node is valid if it is not obsolete and is contained within the given ontology namespace constraint.

> > **Parameters node** – A *gocats.dag.AbstractNode* object
>
> > **Returns** True if node is valid, False otherwise
>
> > **Return type** `True` or `False`

valid_edge(*edge*)
> Defines condition of a valid edge. Edge is valid if it is within the list of allowed edges and connects two nodes that are both contained in the graph in question.
>
> > **Parameters edge** – A *gocats.dag.AbstractEdge* object
>
> > **Returns** True if node is valid, False otherwise
>
> > **Return type** `True` or `False`

_update_graph()
> Repopulates graph orphans and leaves sets.
>
> > **Returns** None
>
> > **Return type** `None`

add_node(*node*)
> Adds a node object to the graph, adds an object pointer to the vocabulary index to reference nodes to every word in the node name and definition. Sets modification state to `True`.
>
> > **Parameters node** – A *gocats.dag.AbstractNode* object.
>
> > **Returns** None
>
> > **Return type** `None`

remove_node(*node*)
> Removes a node from the graph and deletes node references from all entries in the vocabulary index. Sets modification state to `True`.
>
> > **Parameters node** – A *gocats.dag.AbstractNode* object.
>
> > **Returns** None
>
> > **Return type** `None`

add_edge(*edge*)
> Adds an edge object to the graph, and counts the edge relationship type. Sets modification state to `True`.
>
> > **Parameters edge** – A *gocats.dag.AbstractEdge* object.
>
> > **Returns** None
>
> > **Return type** `None`

remove_edge(*edge*)
> Removes an edge object from the graph, and removes references to that edge from the node objects involved. Sets modification state to `True`.
>
> > **Parameters edge** – A *gocats.dag.AbstractEdge* object.
>
> > **Returns** None
>
> > **Return type** `None`

add_relationship(*relationship*)
> Adds a *gocats.dag.AbstractRelationship* object to the graph's relationship index, referenced by that relationships ID. Sets modification state to `True`.
>
> > **Parameters relationship** – A *gocats.dag.AbstractRelationship* object.

Returns None

Return type `None`

instantiate_valid_edges()

Add all edge references to their respective nodes and vice versa if both nodes of the edge are in the graph. This is carried out by `AbstractEdge.connect_nodes()`. Also adds `gocats.dag.AbstractRelationship` object reference to each edge. If both nodes are not in the graph, the edge is deleted from the graph. Sets modification state to `True`.

Returns None

Return type `None`

node_depth(*sample_node*)

Returns an integer representing how many nodes are between the given node and the root node of the graph (depth level).

Parameters **sample_node** – A `gocats.dag.AbstractNode` object.

Returns Depth level.

Return type `int`

filter_nodes(*search_string_list*)

Returns a list of node objects that contain vocabulary matching the keywords provided in the search string list. Nodes are selected by searching through the vocablary index.

Parameters **search_string_list** – A `list` of search strings provided in the keyword_file provided to `gocats.gocats.create_subgraphs()`.

Returns A list of `gocats.dag.AbstractNode` objects.

Return type `list`

filter_edges(*filtered_nodes*)

Returns a list of edges in the graph that connect the nodes provided in the filtered nodes list.

Parameters **filtered_nodes** – List of filtered nodes provided by `filter_nodes()`.

Returns A list of `gocats.dag.AbstractEdge` objects.

Return type `list`

nodes_between(*start_node*, *end_node*)

Returns a set of nodes that occur along all paths between the start node and the end node. If no paths exist, an empty set is returned.

Parameters

- **start_node** – `gocats.dag.AbstractNode` object to start the paths.

- **end_node** – `gocats.dag.AbstractNode` object to end the paths.

Returns A set of `gocats.dag.AbstractNode` objects if there is at least one path between the parameters, an empty set otherwise.

Return type `set`

__weakref__

list of weak references to the object (if defined)

class gocats.dag.**AbstractNode**

A node containing all basic properties of an OBO node. The parsing object, `gocats.ontologyparser.OboParser` currently has direct access to data members (id, name, definition, namespace, edges, and obsolete) so that information from the database file can be added to the object.

`__init__`()
> *AbstractNode* initializer

descendants
> `property` defining a set of nodes in the graph that are recursively reverse of a node with a scoping-type relationship. When the node is modified, calls *`gocats.dag.AbstractNode._update_node()`* to repopulate the sets of descendants and ancestors. This represents a "lazy" evaluation of node descendants.
>
> > **Returns** Set of *`gocats.dag.AbstractNode`* objects
> >
> > **Return type** `set`

ancestors
> `property` defining a set of nodes in the graph that are recursively forward of a node with a scoping-type relationship. When the node is modified, calls *`gocats.dag.AbstractNode._update_node()`* to repopulate the sets of descendants and ancestors. This represents a "lazy" evaluation of node ancestors.
>
> > **Returns** Set of *`gocats.dag.AbstractNode`* objects
> >
> > **Return type** `set`

_update_node()
> Repopulates ancestor and descendant sets for a node. Sets modification state to `True`.
>
> > **Returns** None
> >
> > **Return type** `None`

add_edge(*edge*, *allowed_relationships*)
> Adds a given *`gocats.dag.AbstractEdge`* to a each *`gocats.dag.AbstractNode`* objects that the edge connects. If there is a filter for the types of relationships allowed, edges with non-allowed relationship types are not processed. Sets modification state to `True`.
>
> > **Returns** None
> >
> > **Return type** `None`

remove_edge(*edge*)
> Removes a given *`gocats.dag.AbstractEdge`* the *`gocats.dag.AbstractNode`* object. Also removes parent or child node references that the edge referenced. Sets modification state to `True`.
>
> > **Returns** None
> >
> > **Return type** `None`

_update_descendants()
> Used for the lazy evaluation of graph descendants of the current *`gocats.dag.AbstractNode`* object. Creates internal `set` variable, descendant_set. Iterates through node children until the bottom of the graph is reached. The descendant_set is a set of all nodes across all paths encountered from the current node.
>
> > **Returns** None
> >
> > **Return type** `None`

_update_ancestors()
> Used for the lazy evaluation of graph ancestors of the current *`gocats.dag.AbstractNode`* object. Creates internal `set` variable, ancestors_set. Iterates through node parents until the top of the graph is reached. The ancestors_set is a set of all nodes across all paths encountered from the current node.
>
> > **Returns** None
> >
> > **Return type** `None`

`__weakref__`
> list of weak references to the object (if defined)

---

class gocats.dag.**AbstractEdge**(*node1_id*, *node2_id*, *relationship_id*, *node_pair=None*)

An OBO edge which links two ontology term nodes and contains a relationship type describing now the two nodes are related.

__init__(*node1_id*, *node2_id*, *relationship_id*, *node_pair=None*)

AbstractEdge initializer. Node pair refers to a `tuple` of `gocats.dag.AbstractNode` objects that are connected by the edge. Defaults to `None` and is later populated.

> **Parameters**
>
> - **node1_id** (`str`) – The ID of the first term referenced from the ontology file's relationship line.
>
> - **node2_id** (`str`) – The ID of the second term referenced from the ontology file's relationship line.
>
> - **relationship_id** (`str`) – The ID of the relationship in the ontology file's relationship line.
>
> - **node_pair** (`tuple`) – Default-`None`, provide a `tuple` containing two `gocats.dag.AbstractNode` objects if they are already created and able to be referenced.

json_edge

`property` which returns a tuple where position 0 is a unique string representation of the edge made by combining the ID of the reverse node and the id of the forward nodes and where position 1 is a list of two node IDs: the reverse and forward node.

> **Returns** `tuple` of a unique `AbstractEdge` ID and a list of that edge object's reverse and forward node IDs, respectively. Returns an empty :py:obj:str at a position for which there are no forward or reverse nodes in the graph.
>
> **Return type** `tuple`

parent_id

`property` defining the ID of the node forward of the current `gocats.dag.AbstractEdge` object.

> **Returns** `str` ID of the forward node in the node_pair associated with the edge if the edge's relationship is assigned, `None` otherwise.
>
> **Return type** `str` or `None`

child_id

`property` defining the ID of the node reverse of the current `gocats.dag.AbstractEdge` object.

> **Returns** `str` ID of the reverse node in the node_pair associated with the edge if the edge's relationship is assigned, `None` otherwise.
>
> **Return type** `str` or `None`

forward_node

`property` defining the `gocats.dag.AbstractNode` object forward of the current `gocats.dag.AbstractEdge` object.

> **Returns** `gocats.dag.AbstractNode` object of the forward node in the node_pair associated with the edge if the edge's relationship is assigned, the node_pair is assigned, and the type of relationship is instantiated by `gocats.dag.DirectionalRelationship` `None` otherwise.
>
> **Return type** `gocats.dag.AbstractNode` or `None`

reverse_node

`property` defining the `gocats.dag.AbstractNode` object reverse of the current `gocats.dag.AbstractEdge` object.

> > **Returns** *gocats.dag.AbstractNode* object of the reverse node in the node_pair associated with the edge if the edge's relationship is assigned, the node_pair is assigned, and the type of relationship is instantiated by *gocats.dag.DirectionalRelationship* None otherwise.
> >
> > **Return type** *gocats.dag.AbstractNode* or None

> **parent_node**
> > *property* defining the *gocats.dag.AbstractNode* object forward of the current *gocats.dag. AbstractEdge* object. This designation will be unique to scoping-type relationships, although this is **not yet specified**.
> >
> > > **Returns** *gocats.dag.AbstractNode* object of the forward node in the node_pair associated with the edge if the edge's relationship is assigned, the node_pair is assigned, and the type of relationship is instantiated by *gocats.dag.DirectionalRelationship* None otherwise.
> > >
> > > **Return type** *gocats.dag.AbstractNode* or None

> **child_node**
> > *property* defining the *gocats.dag.AbstractNode* object reverse of the current *gocats.dag. AbstractEdge* object. This designation will be unique to scoping-type relationships, although this is **not yet specified**.
> >
> > > **Returns** *gocats.dag.AbstractNode* object of the reverse node in the node_pair associated with the edge if the edge's relationship is assigned, the node_pair is assigned, and the type of relationship is instantiated by *gocats.dag.DirectionalRelationship* None otherwise.
> > >
> > > **Return type** *gocats.dag.AbstractNode* or None

actor_node
> **not yet implemented**
>
> > **Returns** None
> >
> > **Return type** None

recipient_node
> **not yet implemented**
>
> > **Returns** None
> >
> > **Return type** None

ordinal_prior_node
> **not yet implemented**
>
> > **Returns** None
> >
> > **Return type** None

ordinal_post_node
> **not yet implemented**
>
> > **Returns** None
> >
> > **Return type** None

other_node
> **not yet implemented**
>
> > **Returns** None
> >
> > **Return type** None

connect_nodes(*node_pair*, *allowed_relationships*)
    Adds the current edge object to the `gocats.dag.AbstractNode` objects that are connected by the edge. Populates the node_pair with `gocats.dag.AbstractNode` objects.

    Returns None

    Return type `None`

__weakref__
    list of weak references to the object (if defined)

class `gocats.dag.`**AbstractRelationship**
    A relationship as defined by a [typedef] stanza in an OBO ontology and augmented by GOcats to better interpret semantic correspondence.

    __init__()
        AbstractRelationship initializer.

    __weakref__
        list of weak references to the object (if defined)

class `gocats.dag.`**DirectionalRelationship**
    A singly-directional relationship edge connecting two nodes in the graph. The two nodes are designated 'forward' and 'reverse.' The 'forward' node semantically succeeds the 'reverse' node in a way that depends on the context of the type of relationship describing the edge to which it is applied.

    __init__()
        DirectionalRelationship initializer.

    forward(*pair*)
        Returns the forward node in a node pair that semantically succeeds the other and is independent of the directionality of the edge. Default position is the second position [1].

        Parameters **pair** (`tuple`) – A pair of `gocats.dag.AbstractNode` objects.

        Returns The forward `gocats.dag.AbstractNode` object as determined by the pre-defined semantic directionality of the relationship.

    reverse(*pair*)
        Returns the reverse node in a node pair that semantically precedes the other and is independent of the directionality of the edge. Default position is the second position [1].

        Parameters **pair** (`tuple`) – A pair of `gocats.dag.AbstractNode` objects.

        Returns The reverse `gocats.dag.AbstractNode` object as determined by the pre-defined semantic directionality of the relationship.

class `gocats.dag.`**NonDirectionalRelationship**
    A non-directional relationship whose edge directionality is either non-existent or semantically irrelevant.

    __init__()
        NonDirectionalRelationship initializer.

Gene Ontology Directed Acylic Graph (GODAG)

Defines a Gene Ontology-specific graph which may have special properties when compared to other OBO formatted ontologies.

class `gocats.godag.`**GoGraph**(*namespace_filter=None*, *allowed_relationships=None*)
    A Gene-Ontology-specific graph. GO-specific idiosyncrasies go here.

__init__(*namespace_filter=None*, *allowed_relationships=None*)
    GoGraph initializer. Inherits and specializes properties from *gocats.dag.OboGraph*.

        Parameters

- **namespace_filter** (*str*) – Specify the namespace of a sub-ontology namespace, if one is available for the ontology.

- **allowed_relationships** (*list*) – Specify a list of relationships to utilize in the graph, other relationships will be ignored.

class gocats.godag.**GoGraphNode**
    Extends AbstractNode to include GO relevant information.

    __init__()
        GoGraphNode initializer. Inherits all properties from *gocats.dag.AbstractNode*.

Directed Acyclic Subgraph (SubDAG)

A subgraph object of an OBOGraph object.

class gocats.subdag.**SubGraph**(*super_graph*, *namespace_filter=None*, *allowed_relationships=None*)
    A subgraph of a provided supergraph with node contents.

    __init__(*super_graph*, *namespace_filter=None*, *allowed_relationships=None*)
        SubGraph initializer. Creates a subgraph object of :class:'gocats.dag.OboGraph. Leave *namespace_filter* and *allowed_relationship* as None to create the entire ontology graph. Otherwise, provide filters to limit what information is pulled into the subgraph.

        Parameters

- **super_graph** (*obj*) – A supergraph object i.e. *gocats.godag.GoGraph*.

- **namespace_filter** (*str*) – Specify the namespace of a sub-ontology namespace, if one is available for the ontology.

- **allowed_relationships** (*list*) – Specify a list of relationships to utilize in the graph, other relationships will be ignored.

    root_id_mapping
        Property describing a mapping dict that relates every ontology term ID of subgraphs in *gocats.dag.OboGraph* to a list of root *gocats.subdag.CategoryNode* IDs.

        Returns dict of *gocats.subdag.SubGraphNode* IDs mapped to a list of root *gocats.subdag.CategoryNode* IDs.

        Return type dict

    root_node_mapping
        Property describing a mapping dict that relates every ontology *gocats.subdag.SubGraphNode* object of subgraphs in *gocats.subdag.SubGraph* to a list of root *gocats.subdag.CategoryNode* objects.

        Returns dict of *gocats.subdag.SubGraphNode* objects mapped to a list of root *gocats.subdag.CategoryNode* objects.

        Return type dict

    content_mapping
        Property describing a mapping dict that relates every root *gocats.subdag.CategoryNode* IDs of subgraphs in a *gocats.subdag.SubGraph* to a list of their subgraph nodes' IDs.

> **Returns** `dict` of `gocats.dag.AbstractNode` IDs mapped to a list' of
> :class:`gocats.dag.AbstractNode IDs.
>
> **Return type** `dict`

subnode(*super_node*)

Defines a `gocats.subdag.SubGraph` node object. Calls `add_node()` to convert a supergraph node into a `gocats.subdag.SubGraphNode` and add this node to the subgraph.

> **Parameters super_node** – A node object from the supergraph i.e. `gocats.godag.GoGraphNode`.
>
> **Returns** A `gocats.subdag.SubGraphNode` object.
>
> **Return type** class

add_node(*super_node*)

Converts a supergraph node into a `gocats.subdag.SubGraphNode` and adds this node to the subgraph. Sets modification state to `True`.

> **Parameters super_node** (*obj*) – A node object from the supergraph i.e. `gocats.godag.GoGraphNode`.
>
> **Returns** None
>
> **Return type** `None`

connect_subnodes()

Analogous to `gocats.dag.instantiate_valid_edges()` and `gocats.dag.AbstractEdge.connect_nodes()`. Updates child and parent node sets for each `gocats.subdag.SubGraphNode` in the `gocats.subdag.SubGraph`. Adds edge object references to nodes and node object references to edges. Counts instances of relationship IDs and sets modification state to `True`.

> **Returns** None
>
> **Return type** `None`

greedily_extend_subgraph()

Extends a seeded subgraph to include all supergraph descendants of the nodes. Searches through the supergraph to add new SubGraphNode objects.

> **Returns** None
>
> **Return type** `None`

conservatively_extend_subgraph()

Not currently in use.* Needs to be updated to handle CategoryNode.

Extends a seeded subgraph to include only nodes in the supergraph that occur along paths between nodes in the subgraph. Searches through the supergraph to add new node objects.

> **Returns** None
>
> **Return type** `None`

remove_orphan_paths()

Not currently in use. Needs to be updated ot handle CategoryNode.

Removes nodes and their descendants from the subgraph which do not root to the category-representative node.

> **Returns** None
>
> **Return type** `None`

static find_representative_nodes (*subgraph*, *search_string_list*)
Compiles a list candidate *gocats.subdag.SubGraphNode* objects from the *gocats.subdag.SubGraph* object based on a list of search strings matching strings in the names of the nodes (using regular expressions). Returns a list containing a single candidate node with the highest number of descendants when possible, returns the sole node if the subgraph only contains one node, returns a list of all seeded nodes when choosing candidates is impossible, or aborts if the subgraph is empty.

> Parameters
>> • **subgraph** – A *gocats.subdag.SubGraph* object.
>>
>> • **search_string_list** – A list of search term str entries.
>
> Returns A list of one or more candidate term gocats.subgraph.SubGraphNode chosen as the subgraph's representative ontology term(s).

static from_filtered_graph (*super_graph*, *subgraph_name*, *keyword_list*, *namespace_filter=None*, *allowed_relationships=None*, *extension='greedy'*)
Staticmethod for extracting a subgraph from the supergraph by selecting nodes that contain vocabulary in the supplied keyword list. Leave *namespace_filter* and *allowed_relationship* as None to create the entire ontology graph. Otherwise, provide filters to limit what information is pulled into the subgraph. Graph *extension* variable defaults to 'greedy' which calls *greedily_extend_subgraph()* to add nodes to the subgraph after instantiation. Conversely, 'conservative' may be used to call *conservatively_extend_subgraph()* for this function.

> Parameters
>> • **super_graph** (*obj*) – A supergraph object i.e. *gocats.godag.GoGraph*.
>>
>> • **subgraph_name** (*str*) – The name of the subgraph being created; will be used as the id of the *gocats.subdag.CategoryNode*.
>>
>> • **keyword_list** – A list of str entries used to query the supergraph for concepts to be extracted into subgraphs.
>>
>> • **namespace_filter** (*str*) – Specify the namespace of a sub-ontology namespace, if one is available for the ontology.
>>
>> • **allowed_relationships** (*list*) – Specify a list of relationships to utilize in the graph, other relationships will be ignored.
>>
>> • **extension** (*str*) – Specify 'greedy' or 'conservative' to determine how subgraphs will be extended after creation (defaults to greedy).
>
> Returns A *gocats.subdag.SubGraph* object.

class gocats.subdag.**SubGraphNode** (*super_node=None*, *allowed_relationships=None*)
An instance of a node within a subgraph of an OBO ontology (supergraph)

__init__ (*super_node=None*, *allowed_relationships=None*)
SubGraphNode initializer. Inherits from *gocats.dag.AbstractNode* and contains a reference to the supergraph node it represents e.g. *gocats.godag.GoGraphNode*.

> Parameters
>> • **super_node** – A node from the *supergraph*.
>>
>> • **allowed_relationships** – **Not currently used** Used to specify a list of allowable relationships evaluated between nodes.

super_edges

`property` describing the set of edges referenced in the supergraph node, filtered to only those
edges with nodes in the subgraph node.

> **Returns** A set of `gocats.subgraph.SubGraphNode` edges that were copied from the
> supergraph node.
>
> **Return type** `set`

id
> `property` describing the ID of the supernode
>
> > **Returns** The ID of a supernode e.g. *`gocats.godag.GoGraphNode`*
> >
> > **Return type** `str`

name
> `property` describing the name of the supernode
>
> > **Returns** The name of a supernode e.g. *`gocats.godag.GoGraphNode`*
> >
> > **Return type** `str`

definition
> `property` describing the definition of the supernode
>
> > **Returns** The definition of a supernode e.g. *`gocats.godag.GoGraphNode`*
> >
> > **Return type** `str`

namespace
> `property` describing the namespace of the supernode
>
> > **Returns** A namespace of a supernode e.g. *`gocats.godag.GoGraphNode`*
> >
> > **Return type** `str`

obsolete
> `property` describing whether or not supernode is marked as obsolete.
>
> > **Returns** `True` or `False`

update_parents(*parent_set*)
> Updates the parent_node_set with a set of new parents provided. Sets modification state to `True`.
>
> > **Parameters** `parent_set` – A set of parent nodes to be added to this objects parent_node set.
> >
> > **Returns** None
> >
> > **Return type** `None`

update_children(*child_set*)
> Updates the child_node_set with a set of new children provided. Sets modification state to `True`.
>
> > **Parameters** `child_set` – A set of child nodes to be added to this objects child_node set.
> >
> > **Returns** None
> >
> > **Return type** `None`

class gocats.subdag.**CategoryNode**(*category_name*, *representative_node_list*, *namespace_filter=None*)
A special node added to the subgraph which contains all representative nodes identified and serves as the single
representative of the subgraph which represents a concept.

> **__init__**(*category_name*, *representative_node_list*, *namespace_filter=None*)
> *AbstractNode* initializer

Ontology Parser

A parser which reads ontologies in the OBO format and calls appropriate graph objects to store information in a graph representation. Separate parsing classes within this module operate on distinct ontologies in the OBO Foundry to handle any subtle differences among ontologies.

class gocats.ontologyparser.**OboParser**

A scaffolding for parsing OBO formatted ontologies. Contains regular expressions for the basic stanzas and information pertinent for creating a graph object of an ontology.

> **__init__**()
>
> *OboParser* initializer. Contains Regular Expressions for identifying crucial information from OBO formatted ontologies.

> **__weakref__**
>
> list of weak references to the object (if defined)

class gocats.ontologyparser.**GoParser**(*database_file*, *go_graph*, *relationship_directionality='gocats'*)

An ontology parser specific to Gene Ontology

> **__init__**(*database_file*, *go_graph*, *relationship_directionality='gocats'*)
>
> *GoParser* initializer. Parses a Gene Ontology database file and adds properties found therein to a godag. GoGraph object. **Importantly:** includes descriptions of semantic directionality of all GO relationships. :param file_handle database_file: Specify the location of a Gene Ontology .obo file. :param go_graph: *gocats.godag.GoGraph* object. :return: None :rtype: None

> **parse**()
>
> Parses the ontology database file and accesses the ontology graph object to add information found in the database. Once all information is added, this function calls the graph's instantiate_valid_edges function to connect all nodes in the graph by their edges.
>
> > **Returns** None
> >
> > **Return type** None

Tools

Functions for handling some file input and output and reformatting tasks in GOcats.

gocats.tools.**json_save**(*obj*, *filename*)

Takes a Python object, converts it into a JSON serializable object (if it is not already), and saves it to a file that is specified.

> **Parameters**
> - **obj** – A Python obj.
> - **filename** (*file_handle*) – A path to output the resulting JSON file.

gocats.tools.**jsonpickle_save**(*obj*, *filename*)

Takes a Python object, converts it into a JsonPickle string, and writes it out to a file.

> **Parameters**
> - **obj** – A Python obj
> - **filename** (*file_handle*) – A path to output the resulting JsonPickle file.

gocats.tools.**jsonpickle_load**(*filename*)

Takes a JsonPickle file and loads in the JsonPickle object into a Python object.

        Parameters filename (*file_handle*) – A path to a JsonPickle file.

`gocats.tools.`**`list_to_file`**(*filename*, *data*)

    Makes a text document from a `list` of data, with each line of the document being one item from the list and outputs the document into a file.

        Parameters

- **filename** (*file_handle*) – A path to the output file.

- **data** – A Python `list`.

`gocats.tools.`**`write_out_gaf`**(*data*, *filename*)

    Writes out an object representing a Gene Annotation File (GAF) to a file.

        Parameters

- **data** (*list*) – A `list` object representing a GAF. Each item in the list represents a row.

- **filename** (*file_handle*) – A path and name for the GAF.

`gocats.tools.`**`parse_gaf`**(*filename*)

    Converts a Gene Annotation File (GAF) into a `list` object where every item is a row from the GAF.

        Parameters filename (*file_handle*) – Specify the location of the GAF.

        Returns A list representing the GAF.

        Return type `list`

1.5.2 User Guide

Description

`GOcats` is an Open Biomedical Ontology (OBO) parser and categorizing utility–currently specialized for the Gene Ontology (GO)–which can help scientists interpret large-scale experimental results by organizing redundant and highly- specific annotations into customizable, biologically-relevant concept categories. Concept subgraphs are defined by lists of keywords created by the user.

Currently, the *GOcats* package can be used to:

- Create subgraphs of GO which each represent a user-specified concept.

- Map specific, or fine-grained, GO terms in a Gene Annotation File (GAF) to an arbitrary number of concept categories.

- Reorganize GO terms based on allowed term-term relationships, and re-create the gene to all direct and ancestor GO terms.

- Explore the Gene Ontology graph within a Python interpreter.

Installation

GOcats runs under Python 3.4+ and is available through python3-pip. Install via pip or clone the git repo and install the following dependencies and you are ready to go!

Install on Linux

Pip installation (method 1)

Dependencies should automatically be installed using this method. It is strongly recommended that you install with this method.

```
pip3 install gocats
```

GitHub Package installation (method 2)

Make sure you have git installed:

```
cd ~/
git clone https://github.com/MoseleyBioinformaticsLab/GOcats.git
```

Dependencies

GOcats requires the following Python libraries:

- docopt for creating the `gocats` command-line interface.
- JSONPickle for saving Python objects in a JSON serializable form and outputting to a file.

To install dependencies manually:

```
pip3 install docopt
pip3 install jsonpickle
```

Install on Windows

Windows version not yet available. Sorry about that.

Basic usage

To see command line arguments and options, navigate to the project directory and run the –help option:

```
cd ~/GOcats
python3 -m gocats --help
```

`gocats` can be used in the following ways:

- To extract subgraphs of Gene Ontology that represent user-defined concepts and create mappings between high level concepts and their subgraph content terms.

    1. Create a CSV file, where column 1 is the name of the concept category (this can be anything) and column 2 is a list of keywords/phrases delineating that concept (separated by semicolons). See *The GOcats Tutorial* for more information.

    2. Download a Gene Ontology database obo file

3. To create mappings, run the GOcats command, *gocats.gocats.* *create_subgraphs()*. If you installed by cloning the repository from GitHub, first navigate to the GOcats project directory or add the directory to the PYTHONPATH.

```
python3 -m gocats create_subdags <ontology_database_file>
↪<keyword_file> <output_directory>
```

4. Mappings can be found in your specified <output_directory>:

   – GC_content_mapping.json_pickle # A python dictionary with category-defining GO terms as keys and a list of all subgraph contents as values.

   – GC_id_mapping.json_pickle # A python dictionary with every GO term of the specified namespace as keys and a list of category root terms as values.

- To map gene annotations in a Gene Annotation File (GAF) to a set of user-defined categories.

   1. Create mapping files as defined in the previous section.

   2. Run the *gocats.gocats.categorize_dataset()* to map terms to their categories:

```
# NOTE: Use the GC_id_mapping.jsonpickle file.
python3 -m gocats categorize_dataset <GAF_file> <term_mapping_
↪file> <output_directory> <mapped_gaf_filename>
```

   3. The output GAF will have the specified <mapped_gaf_filename> in the <output_directory>

- To reorganize parent - child Gene Ontology terms relationships and the gene annotations with a set of user defined relationships.

This has been shown to increase statistical power in GO enrichment calculations (see Hinderer).

   1. Download a Gene Ontology database obo file.

   2. Download a Gene Ontology gene annotation format gaf file.

   3. Run the *gocats.gocats.remap_goterms()* to generate new gene to annotation relationships:

```
python3 -m gocats remap_goterms <go_database> <goa_gaf>
↪<ancestor_filename> <namespace_filename> [--allowed_
↪relationships=<relationships> --identifier_column=<column>]
```

   4. --allowed_relationships should be a comma separated string: is_a, part_of,has_part

   5. The output <ancestor_filename> will be in JSON format, with genes as the keys, and annotated GO terms as the set.

- Within the Python interpreter to explore the Gene Ontology graph (advanced usage, see *The GOcats Tutorial* for more information).

   1. If you've installed GOcats via pip, importing should work as expected. Otherwise, navigate to the Git project directory, open a Python 3.4+ interpreter, and import GOcats:

```
>>> from gocats import gocats as gc
```

   2. Create the graph object using *gocats.gocats.* *build_graph_interpreter()*:

```
>>> # May filter to GO sub-ontology or to a set of relationships.
>>> my_graph = gc.build_graph_interpreter("path_to_database_file")

You may now access all properties of the Gene Ontology graph␣
↪object. Here are a couple of examples:
```

```
>>> # See the descendants of a term node, GO:0006306
>>> descendant_set = my_graph.id_index['GO:0006306'].descendants
>>> [node.name for node in descendant_set]
>>> # Access all graph leaf nodes
>>> leaf_nodes  = my_graph.leaves
>>> [node.name for node in leaf_nodes]
```

1.5.3 The GOcats Tutorial

Currently, *GOcats* **can be used to:**

- Create subgraphs of the Gene Ontology (GO) which each represent a user-specified concept.
- Map specific, or fine-grained, GO terms in a Gene Annotation File (GAF) to an arbitrary number of concept categories.
- Remap ancestor Gene Ontology term relationships and the gene annotations with a set of user defined relationships.
- Explore the Gene Ontology graph within a Python interpreter.

In this document, each use case will be explained in-depth.

Using GOcats to create subgraphs representing user-specified concepts

Before starting, it is important to decide what concepts you as the user wish to extract from the Gene Ontology. You may have an investigation that is focused on concepts like "DNA repair" or "autophagy," or you may simply be interested in enumerating many arbitrary categories and seeing how ontology terms are shared between concepts. As an example to use in this tutorial, let's consider a goal of extracting subgraphs that represent some typical subcellular locations of a eukaryotic cell.

Create a keyword file

The phrase "keyword file" might be slightly misleading because GOcats does not only handle keywords, but also **short phrases** that may be used to define a concept. Therefore, both may be used in combination in the keyword CSV file.

The CSV file is formatted as so:

- Each row represents a separate concept.
- Column 1 is the name of the concept (this is for reference and will not be used to parse GO).
- **Column 2 is a list of keywords or short phrases used to describe the concept in question.**
  - Each item in column 2 is separated by a semicolon (;) with no whitespace around the semicolon.

Here is an example of what the file contents should look like (do not include the header row in the actual file):

Concept	Keywords/phrases
mitochondria	mitochondria;mitochondrial;mitochondrion
nucleus	nucleus;nuclei;nuclear
lysosome	lysosome;lysosomal;lysosomes
vesicle	vesicle;vesicles
er	endoplasmic;sarcoplasmic;reticulum
golgi	golgi; golgi apparatus
extracellular	extracellular;secreted
cytosol	cytosol;cytosolic
cytoplasm	cytoplasm;cytoplasmic
cell membrane	plasma;plasma membrane
cytoskeleton	cytoskeleton;cytoskeletal

We'll imagine this file is located in the home directory and is called `cell_locations.csv`.

Download the Gene Ontology .obo file

The `go.obo` file is available here: http://www.geneontology.org/page/download-ontology. Be sure to download the .obo-formatted version. All releases of GO in this format as of Jan 2015 have been verified to be compatible with *GOcats*. We'll assume this database file is located in the home directory and is called `go.obo`.

Extract subgraphs and create concept mappings

This is where *GOcats* does the heavy lifting. We'll assume *GOcats* was already installed via pip or the repository was already cloned into the home directory (refer to *User Guide* for instructions on how to install *GOcats*). We can now use Python to run the `gocats.gocats.create_subgraphs()` function. We can also specify that we only want to parse the *cellular_component* sub-ontology of GO (the `supergraph_namespace`), since we are only interested in concepts of this type. Although it is redundant, we can also play it safe and limit subgraph creation to only consider terms listed in *cellular_component* as well (the `subgraph_namespace`). Run the following if you hav installed via pip (if running from the Git repository navigate to the *GOcats* directory or add this directory to your PYTHONPATH beforehand).

```
python3 -m gocats create_subgraphs ~/go.obo ~/cell_locations.csv ~/cell_
↪locations_output --supergraph_namespace=cellular_component --subgraph_
↪namespace=cellular_component
```

The results will be output to `~/cell_locations_output`.

Let's look at the output files

In the output directory (i.e. `~/cell_locations_output`) we can see several files. The following table describes what can be found in each:

File Name	Description
GC_content_mapping.json	JSON version of Python dictionary (keys: concept root nodes, values: list of subgraph term nodes).
GC_content_mapping.json_pickle	Same as above, but a JSONPickle version of the dictionary.
GC_id_mapping.json	JSON version of Python dictionary (keys: subgraph term nodes, values: list of concept roots).
GC_id_mapping.json_pickle	Same as above, but a JSONPickle version of the dictionary.
id_translation.json_pickle	A JSONPickle version of a Python dictionary mapping GO IDs to the name of the term.
NetworkTable.csv	A csv version of id_translation for visualizing in Cytoscape (best results with –map_supersets)
subgraph_report.txt	A summary of the subgraphs extracted for mapping. See below for more details.

We can look in subgraph_report.txt to get an overview of what our subgraphs contain, how they were constructed, and how they compare to the overall GO graph.

subgraph_report.txt

The first few lines give an overview of the subgraphs and supergraph (which is the full GO graph, unless a supergraph_namespace filter was used). In our example case, the supergraph is the *cellular_component* ontology of GO.

In each divided section, the first line indicates the subgraph name (the one provided from column 1 in the keyword file) . The following describes the meaning of the values in each section:

- **Subgraph relationships**: the prevalence of relationship types in the subgraph.

- **Seeded size**: how many GO terms were initially filtered from GO with the keyword list.

- **Representative node**: the name of the GO term chosen as the root for that concept's subgraph.

- **Nodes added**: the number of GO terms added when extending the seeded subgraph to descendants not captured by the initial search.

- **Non-subgraph hits (orphans)**: GO terms that were captured by the keyword search, but do not belong to the subgraph.

- **Total nodes**: the total number of GO terms in the subgraph.

Loading mapping files programmatically (optional)

While *GOcats* can use the mapping files described in the previous section to map terms in a GAF, it may also be useful to load them into your own scripts for use. Since the mappings are saved in JSON and JSONPickle formats, it is relatively simple to load them in programmatically:

```
>>># Loading a JSON file
>>>import json
>>>with open('path_to_json_file', 'r') as json_file:
>>>    json_str = json_file.read()
>>>    json_obj = json.loads(json_str)
>>>my_mapping = json_obj

>>># Loading a JSONPickle file
>>>import jsonpickle
>>>with open('path_to_jsonpickle_file', 'r') as jsonpickle_file:
```

```
>>>    jsonpickle_str = jsonpickle_file.read()
>>>    jsonpickle_obj = jsonpickle.decode(jsonpickle_str, keys=True)
>>>my_mapping = jsonpickle_obj
```

Using GOcats to map specific gene annotations in a GAF to custom categories

With mapping files produced from the previous steps, it is possible to create a GAF with annotations mapped to the categories, or concepts, that we define. Let's consider our current *cell_locations* example and imagine that we have some gene set containing annotations in a GAF called `dataset_GAF.goa` in the home directory. To map these annotations, use the *gocats.gocats.categorize_dataset()* function. Again, this should work from any location if you've installed via pip, otherwise navigate to the GOcats directory or add this directory to your PYTHONPATH and run the following:

```
# Note that you need to use the GC_id_mapping.json_pickle file for this step
python3 -m gocats categorize_dataset ~/datasetGAF.goa ~/cell_locations_output/GC_id_
↪mapping.json_pickle ~/mapped_dataset mapped_GAF.goa
```

Here, we named the output directory `~/mapped_dataset` and we named the mapped GAF `mapped_GAF.goa`. The mapped gaf and a list of unmapped genes will be stored in the output directory.

Using GOcats to remap ancestor Gene Ontology term relationships and the gene annotations with a set of user defined relationships

As noted in the last two examples, *GOcats* can consider *has_part* relationships properly, in addition to the *is_a* and *part_of* relationships normally used for generating gene annotations to ancestor GO terms. We have previously shown that doing this can improve the statistical power of GO term enrichment (see Hinderer). In this case, we need a Gene Ontology obo file, as well as a gene annotation format gaf file.

```
python3 -m gocats remap_goterms ~/go.obo ~/goa_human.gaf ~/ancestors.json ~/namespace.
↪json --allowed_relationships=is_a,part_of,has_part --identifier_column=1
```

The output in `ancestors.json` will be a JSON list, where each gene is the name of a JSON vector of annotated GO terms. `namespace.json` provides the new namespace for each GO term. In contrast to the API in Python, the `--allowed_relationships` takes a comma separated list of relationships to use. In the GAF files, there will often be two identifiers, the database identifier (Uniprot) for human, and gene symbol. `--identifier_column` allows the user to select to use the database (1) or gene symbol (2) as the identifier in the output.

Exploring Gene Ontology graph in a Python interpreter or in your own Python project

If you've installed GOcats via pip, importing should work as expected. Otherwise, navigate to the Git project directory, open a Python 3.4+ interpreter, and import GOcats:

```
>>> import gocats
```

Next, create the graph object using *gocats.gocats.build_graph_interpreter()*. Since we have been looking at the *cellular_component* sub-ontology in this example, we can specify that we only want to look at that part of the graph with the supergraph_namespace option. Additionally we can filter the relationship types using the `allowed_relationships` option (only *is_a*, *has_part*, and *part_of* exist in cellular_component, so this is just for demonstration):

```
>>> # May filter to GO sub-ontology or to a set of relationships.
>>> my_graph = gocats.gocats.build_graph_interpreter("~/go.obo", supergraph_namespace=
→"cellular_component", allowed_relationships=["is_a", "has_part", "part_of"])
>>> full_graph = gocats.gocats.build_graph_interpreter("~/go.obo")
```

The filtered graph (`my_graph`) and the full GO graph (`full_graph`) can now be explored.

The graph object contains an `id_index` which allows one to access node objects by GO IDs like so:

```
>>>my_node = my_graph.id_index['GO:0004567']
```

It also contains a `node_list` and an `edge_list`.

Edges and nodes in the graph are objects themselves.

```
>>>print(my_node.name)
```

Here is a list of some important graph, node, and edge data members and properties:

Graph

- node_list: list of **node** objects in the graph.
- edge_list: list of **edge** objects in the graph.
- id_index: dictionary of node IDs that point to their respective **node** objects.
- vocab_index: dictionary listing every word used in the gene ontology, pointing to **node** objects those words can be found in.
- relationship_index: dictionary of relationships in the supergraph, pointing to their respective relationship objects.
- root_nodes: a set of root nodes of the supergraph.
- orphans: a set of nodes which have no parents.
- leaves: a set of nodes which have no children.

Node

- id
- name
- definition
- namespace
- edges: a set of **edges** that connect the node.
- parent_node_set
- child_node_set
- descendants: a set of recursive graph children.
- ancestors: a set of recursive graph parents.

Edge

- node_pair_id: tuple of IDs of the **nodes** connected by the edge.
- node_pair: a tuple of the **node objects** connected by the edge.
- relationship_id: the ID of the relationship type (i.e. the name of the relationship).

- relationship: the relationship object used to describe the edge
- parent_id
- parent_node
- child_id
- child_node
- forward_node: see *The GOcats API Reference*
- reverse_node: see *The GOcats API Reference*

Plotting subgraphs in Cytoscape for visualization

Coming soon!

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

Index

Symbols