Data-Driven Ontology Toolkit Documentation

Release 1.0

Michael Ku Yu

Jun 21, 2019

Contents

The Data-Driven Ontology Toolkit (DDOT) facilitates the inference, analysis, and visualization of biological hierarchies using a data structure called an ontology.

- Open-source Python package under MIT license. Supports Python 2.7 or >=3.6.
- The HiView web application visualizes hierarchical structure and the biological evidence for that structure.

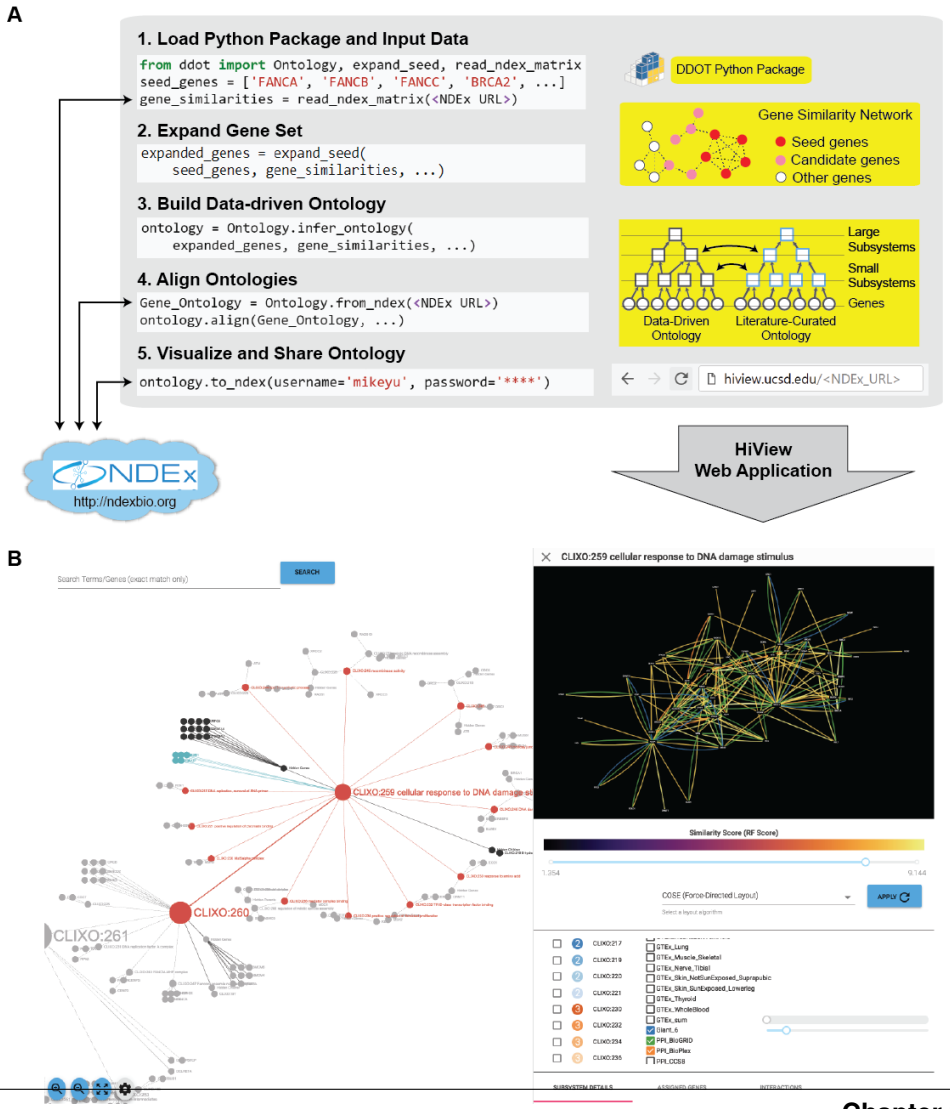
CHAPTER 1

Getting started

1. Install the DDOT Python package using the instructions on the github page.
2. Go through the tutorial.
3. Browse a list of Python functions in *Ontology Class* and *Utility Functions*.
4. Post questions or issues to the Google Groups forum.

CHAPTER 2

Features



1. **Build Data-Driven Ontology:** Given a set of genes and a gene similarity network, hierarchically cluster the genes to infer cellular subsystems using the CliXO algorithm (Kramer et al. Bioinformatics 2014). The resulting hierarchy of subsystems defines a data-driven ontology.
2. **Align Ontologies:** Annotate a data-driven ontology by aligning it to a curated ontology such as the Gene Ontology (GO). For instance, if a data-driven subsystem contains a similar set of genes as the GO term for DNA repair, then annotate this subsystem as being involved in DNA repair. Data-driven subsystems with no such matches represent new molecular mechanisms.
3. **Visualize Hierarchical Structure:** Browse the full hierarchical structure of a data-driven ontology, including the network of gene similarities used to infer it, in a web application called the Hierarchical Viewer (HiView, http://hiview.ucsd.edu)
4. **Examine ontology structure:** For each subsystem, retrieve its hierarchical connections (genes, child and descendant subsystems, parent and ancestral subsystems) and the subnetwork of gene similarities that supports the subsystem's existence. For each gene, retrieve its set of subsystems.
5. **Modify ontology structure:** Reduce the size of an ontology by removing a set of subsystems or genes. Randomize connections between genes and subsystems to create new ontologies representing a null model for statistical tests.
6. **Flatten ontology structure:** Instead of inferring an ontology from a gene similarity network, perform the reverse process of inferring a gene similarity network from an ontology. In particular, the similarity between two genes is calculated as the size of the smallest common subsystem, known as the Resnik score.
7. **Expand Gene Set:** Given a set of genes as a "seed set" and a gene similarity network, identify an expanded set of genes that are highly similar to the seed set. This function can broaden the scope of a data-driven ontology beyond genes that are already well known.
8. **Map genotypes to the ontology:** Given a set of mutations comprising a genotype, propagate the impact of these mutations to the subsystems containing these genes in the ontology. In particular, the impact on a subsystem is estimated by the number of its genes that have been mutated. These subsystem activities, which we have called an "ontotype", enables more accurate and interpretable predictions of phenotype from genotype (Yu et al. Cell Systems 2016).
9. **Load curated ontologies:** Parse Open Biomedical Ontologies (OBO) and gene-association file (GAF) formats that are typically used to describe curated ontologies like GO.
10. **Interface with Network Data Exchange:** Ontologies and networks can be stored and retrieved online at NDEx (http://ndexbio.org). We encourage use of NDEx to make DDOT-based software pipelines more reproducible and shareeable with others.
11. **Interface with other tools and Python libraries:** DDOT can readily interface with other desktop applications, such as Cytoscape, and other programming libraries in Python, such as the Pandas, NetworkX, igraph, and matplotlib.

CHAPTER 3

How to cite

If you use DDOT for your research, please cite

Yu MK, Ma J, Ono K, Zheng F, Fong S, Gary A, Chen J, Demchak B, Pratt D, Ideker T. “A swiss-army knife for hierarchical modeling of biological systems.” (in preparation)

CHAPTER 4

References

1. Yu MK, Kramer M, Dutkowski J, Srivas R, Licon K, Kreisberg JF, Ng CT, Krogan N, Sharan R, Ideker T. "Translation of genotype to phenotype by a hierarchy of cell subsystems". *Cell Systems*, 2(2), pp.77-88. 2016.
2. Kramer M, Dutkowski J, Yu M, Bafna V, Ideker T. "Inferring gene ontologies from pairwise similarity data." *Bioinformatics*, 30(12), pp.i34-i42. 2014.
3. Kramer MH, Farre JC, Mitra, K, Yu MK, Ono K, Demchak B, Licon K, Flagg M, Balakrishnan R, Cherry JM, Subramani S, Ideker T. "Active Interaction Mapping Reveals the Hierarchical Organization of Autophagy". *Molecular Cell*, 65(4), pp.761-774. 2017.
4. Dutkowski J, Ono K, Kramer M, Yu M, Pratt D, Demchak B, Ideker T. "NeXO Web: the NeXO ontology database and visualization platform." *Nucleic Acids Research*, 42(D1), pp.D1269-D1274. 2013.
5. Dutkowski J, Kramer M, Surma MA, Balakrishnan R, Cherry JM, Krogan NJ, Ideker T. "A gene ontology inferred from molecular networks." *Nature Biotechnology*, 31(1). 2013.

4.1 Ontology Class

class ddot.**Ontology**(*hierarchy*, *mapping*, *edge_attr=None*, *node_attr=None*, *parent_child=False*, *add_root_name=None*, *propagate=None*, *ignore_orphan_terms=False*, *verbose=True*, ***kwargs*)

A Python representation for constructing, analyzing, and manipulating the hierarchical structure of ontologies.

An Ontology object contains the following attributes for representing the hierarchical structure. Do not directly modify these attributes.

Parameters

- **genes** (*list*) – Names of genes
- **terms** (*list*) – Names of terms
- **gene_2_term** (*dict*) – gene_2_term[<gene>] –> list of terms connected to <gene>. Terms are represented as their 0-based index in self.terms.

- **term_2_gene** (*dict*) – term_2_gene[<term>] –> list of genes connected to <term>. Genes are represented as their 0-based index in self.genes.
- **child_2_parent** (*dict*) – child_2_parent[<child>] –> list of the parent terms of <child>
- **parent_2_child** (*dict*) – parent_2_child[<parent>] –> list of the children terms of <parent>
- **term_sizes** (*list*) – A list of every term's size, i.e. the number of unique genes that it and its descendant terms contain. This list has the same order as self.terms. It holds that for every i,

  term_sizes[i] = len(self.term_2_gene[self.terms[i]])

4.1.1 Read/write Ontology objects

classmethod Ontology.**from_table**(*table*, *parent=0*, *child=1*, *is_mapping=None*, *mapping=None*, *mapping_parent=0*, *mapping_child=1*, *header=0*, *propagate=False*, *verbose=False*, *clixo_format=False*, *clear_default_attr=True*, ***kwargs*)

Ontology.**to_table**(*output=None*, *term_2_term=True*, *gene_2_term=True*, *edge_attr=False*, *header=True*, *parent_child=True*, *clixo_format=False*)

Convert Ontology to a table representation. Return a pandas.DataFrame and, optionally, write it to a file as a tab-delimited file.

Parameters

- **output** (*filepath or file-like*) – File to write table. If None, then only return a pandas.DataFrame
- **term_2_term** (*bool*) – Include (child term, parent term) pairs
- **gene_2_term** (*bool*) – Include (gene, term) pairs
- **edge_attr** (*array-like or bool*) – List of extra edge attributes to include. If True, then include all attributes. If False, then don't include any attribute.
- **header** (*bool*) – If True (default), then write the column names as the first row of the table.
- **parent_child** (*bool*) – If True, then the first column is the parent term and the second column is the child term or gene. If False, then the columns are reversed.
- **clixo_format** (*bool*) – If True, the table is the same format used the CLIXO C++ implementation. In particular, the table has three columns:

  Column 1) Parent Term Column 2) Child Term or Gene Column 3) The string "gene" if the row is a

  gene-term mapping, otherwise the string "default".

Returns Contains at least three columns: (1) "Parent", (2) "Child", and (3) "EdgeType".

Return type pandas.DataFrame

classmethod Ontology.**read_pickle**(*file*, *compression='infer'*)

Loads an Ontology object from a pickled state.

Ontology.**to_pickle**(*file*, *compression='infer'*)

Saves Ontology object with the Python pickle protocol.

Ontology.**to_ndex**(*ndex_user*, *ndex_pass*, *ndex_server=None*, *name=None*, *description=None*, *network=None*, *main_feature=None*, *subnet_max_term_size=None*, *visible_term_attr=None*, *layout='bubble'*, *propagate='reverse'*, *style=None*, *node_alias='Original_Name'*, *term_2_uuid=None*, *visibility='PUBLIC'*, *verbose=False*)

Upload an Ontology object to NDEx. The Ontology can be preformatted in several ways including

1. Set a name and description of the Ontology
2. Upload a supporting gene-gene subnetwork for every term in the Ontology
3. Propagate gene-term annotations
4. Layout the nodes.
5. Apply a visual style, e.g. specifying node and edge colors

Parameters

- **name** (*str*) – Name of Ontology
- **description** (*str*) – Description of Ontology
- **layout** (*str*) – The name of the layout algorithm for laying out the Ontology as a graph. Node positions are stored in the node attributes 'x_pos' and 'y_pos'. If None, then do not perform a layout.
- **style** (*ndex.networkn.NdexGraph*) – The Cytoscape.js visual style on NDEx. Represented using CX and stored in an NdexGraph.
- **network** (*pandas.Dataframe*) – Dataframe describing gene-gene network from which to create subnetworks for every term. To be passed to Ontology.upload_subnets_ndex().
- **features** (*list of str*) – Columns in the gene-gene network to upload. To be passed to Ontology.upload_subnets_ndex().
- **ndex_server** (*str*) – URL of NDEx server
- **ndex_user** (*str*) – NDEx username
- **ndex_pass** (*str*) – NDEx password
- **public** (*bool*) – Whether to make the Ontology public on NDEx
- **node_alias** (*str*) –
- **visibility** (*str*) –

Returns

Return type ndex.networkn.NdexGraph

classmethod Ontology.**from_ndex**(*ndex_uuid*, *ndex_user=None*, *ndex_pass=None*, *ndex_server=None*, *edgetype_attr=None*, *edgetype_value=None*)

Reads an Ontology stored on NDEx. Gene and terms are distinguished according by an edge attribute.

Parameters

- **ndex_uuid** (*str*) – NDEx UUID of ontology
- **edgetype_attr** (*str*) – Name of the edge attribute that distinguishes a (gene, term) pair from a (child term, parent term) pair
- **gene_value** (*str*) – Value of the edge attribute for (gene, term) pairs

Returns

Return type ddot.Ontology.Ontology

`Ontology.`**`to_cx`**(*output=None*, *name=None*, *description=None*, *term_2_uuid=None*, *spanning_tree=True*, *layout='bubble'*, *style=None*)

Formats an Ontology object into a CX file format

Parameters

- **`output`** (*str*) – Filename or file-like object to write CX file. If None, then CX is returned as a JSON object, but not written to a file.
- **`name`** (*str*) – Name of Ontology, as would appear if uploaded to NDEx.
- **`description`** (*str*) – Description of Ontology, as would appear if uploaded to NDEx.
- **`term_2_uuid`** (*list*) – A dictionary mapping a term to a NDEx UUID of a gene-gene subnetwork of genes in that term. the UUID will be stored in the node attribute 'ndex:internallink'. If uploaded to NDEx, then this attribute will provide a hyperlink to the gene-gene subnetwork when the term is clicked upon on the NDEx page for this ontology.

  This dictionary can be created using Ontology.upload_subnets_ndex(). Default: no dictionary.
- **`layout`** (*str*) – Layout the genes and terms in this Ontology. Stored in the node attributes 'x_pos' and 'y_pos'. If None, then do not perform a layout.

Returns

Return type CX representation as a JSON-like dictionary

`Ontology.`**`to_graphml`**(*output*, *layout='bubble'*, *spanning_tree=True*)

Writes an Ontology object in graphml format.

Parameters

- **`output`** (*str*) – Filename or file-like object to write CX file. If None, then CX is returned as a JSON object, but not written to a file.
- **`layout`** (*str*) – Layout the genes and terms in this Ontology. Stored in the node attributes 'x_pos' and 'y_pos'. If None, then do not perform a layout.

4.1.2 NetworkX and igraph

`Ontology.`**`to_networkx`**(*layout='bubble'*, *spanning_tree=True*, *layout_params=None*, *verbose=False*)

Converts Ontology into a NetworkX object.

Parameters

- **`node_attr`** (*pandas.DataFrame*) – Meta-data about genes and terms that will be included as node attributes in the NetworkX object.
- **`edge_attr`** (*pandas.DataFrame*) – Meta-data about connections among genes and terms that will be included as edge attributes in the NetworkX object.
- **`spanning_tree`** (*bool*) – If True, then identify a spanning tree of the DAG. include an edge attribute "Is_Tree_Edge" that indicates
- **`layout`** (*str*) – The name of the layout algorithm for laying out the Ontology as a graph. Node positions are astored in the node attributes 'x_pos' and 'y_pos'. If None, then do not perform a layout.

Returns

Return type nx.DiGraph

classmethod Ontology.**from_networkx**(*G*, *edgetype_attr=None*, *edgetype_value=None*, *clear_default_attr=True*)

Converts a NetworkX object to an Ontology object. Gene and terms are distinguished by an edge attribute.

Parameters

- **G** (*nx.DiGraph*) –
- **edgetype_attr** (*str*) – Name of the edge attribute that distinguishes a (gene, term) pair from a (child term, parent term) pair
- **edgetype_value** (*str*) – Value of the edge attribute for (gene, term) pairs
- **clear_default_attr** (*bool*) – If True (default), then remove the node and edge attributes that are created in a NetworkX graph using Ontology.to_networkx() or Ontology.to_ndex(). These attributes include 'Label', 'Size', 'NodeType', and 'EdgeType'. These attributes were created to make the NetworkX graph be an equivalent representation of an Ontology object; however, they are no longer necessary after reconstrcting the Ontology object.

Returns

Return type ddot.Ontology.Ontology

classmethod Ontology.**from_igraph**(*G*, *edgetype_attr=None*, *edgetype_value=None*, *verbose=False*)

Converts a igraph Graph object to an Ontology object. Gene and terms are distinguished by an edge attribute.

Parameters

- **G** (*igraph.Graph*) –
- **edgetype_attr** (*str*) – Name of the edge attribute that distinguishes a (gene, term) pair from a (child term, parent term) pair
- **edgetype_value** (*str*) – Value of the edge attribute for (gene, term) pairs

Returns

Return type ddot.Ontology.Ontology

Ontology.**to_igraph**(*include_genes=True*, *spanning_tree=False*)

Convert Ontology to an igraph.Graph object. Gene and term names are stored in the 'name' vertex attribute of the igraph object.

Parameters

- **include_genes** (*bool*) – Include genes as vertices in the igraph object.
- **spanning_tree** (*bool*) – If True, then identify a spanning tree of the DAG. include an edge attribute "Is_Tree_Edge" that indicates

Returns

Return type igraph.Graph

4.1.3 Inspecting structure

Ontology.**connected**(*descendants=None*, *ancestors=None*, *sparse=False*)

Calculate which genes or terms are descendants of other genes or terms.

Parameters

- **descendants** (*list*) – A list of genes and/or terms. Default: A list of all genes followed by a list of all terms, in the same order as *self.genes* and *self.terms*.
- **ancestors** (*list*) – A list of genes and/or terms. Default: Same as the `descendants` parameter.
- **sparse** (*bool*) – If True, return a scipy.sparse matrix. If False (default), return a NumPy array.

Returns **d** – A descendants-by-ancestors matrix. `d[i,j]` is 1 if term i is a descendant of term j, and 0 otherwise. Note that `d[i,i]==1` and `d[root,i]==0`, for every i.

Return type np.ndarray or scipy.sparse.matrix

`Ontology.`**get_best_ancestors**(*node_order=None*, *verbose=False*, *include_genes=True*)

Compute the 'best' ancestor for every pair of terms. 'Best' is specified by a ranking of terms. For example, if terms are ranked by size, from smallest to largest, then the smallest common ancestor is calculated.

Parameters

- **node_order** (*list*) – A list of terms, ordered by their rank with the 'best' term at the beginning.
- **include_genes** (*bool*) –

Returns

- **ancestors** (*np.ndarray*) – ancestors[a,b] = the best common ancestor of terms a and b, represented as a 0-based index of self.terms
- **nodes** (*list*) – List of the row and column names. Rows and columns are the same.

`Ontology.`**topological_sorting**(*top_down=True*, *include_genes=False*)

Perform a topological sorting.

top_down :

If True, then ancestral nodes (e.g. the root nodes) come before descendants in the sorting. If False, then reverse the sorting

4.1.4 Manipulating structure

`Ontology.`**unfold**(*duplicate=None*, *genes_only=False*, *levels=None*, *tree_edges=None*)

Traverses the ontology from the root to the leaves while duplicating nodes during the traversal to create a tree representation.

Traverse the ontology from the root nodes to the leaves in a breadth-first manner. Each time a node is traversed, then create a duplicate of it

Parameters

- **duplicate** (*list*) – Nodes to duplicate for unfolding. Default: all genes and terms
- **genes_only** (*bool*) – If True, then duplicate all of the genes and none of the terms. Default: False
- **levels** –

`Ontology.`**delete**(*to_delete=None*, *to_keep=None*, *preserve_transitivity=True*, *inplace=False*)

Delete genes and/or terms from the ontology.

Parameters

- **to_delete** (*array-like (optional)*) – Names of genes and/or terms to delete. Either to_delete or to_keep must be specified.
- **to_keep** (*array-like (optional)*) – Names of genes and/or terms to keep; all other genes/terms are delete. Only used if to_delete is not specified.
- **preserve_transitivity** (*bool*) – If True, then maintain transitive relations when deleting terms. For example, if the hierarchical structure consists of

  geneA –> term1 term1 –> term2 term2 –> term3 term2 –> term4

  then deleting term2 will result in the structure:

  geneA –> term1 term1 –> term3 term3 –> term4

  If False, then deleting term2 will result in a disconnected structure:

  geneA –> term1
- **inplace** (*bool*) – If True, then modify the ontology. If False, then create and modify a copy.

Returns

Return type ddot.Ontology.Ontology

`Ontology.`**`focus`**(*branches=None*, *genes=None*, *collapse=False*, *root=True*, *verbose=True*)

`Ontology.`**`propagate`**(*direction='forward'*, *gene_term=True*, *term_term=False*, *verbose=False*, *inplace=False*)

Propagate gene-term annotations through the ontology.

As an example, consider an ontology with one gene `g`, three terms `t1, t2, t3` and the following connections:

```
t1-->t2
t2-->t3
g-->t1
g-->t2
```

In "forward" propagation, a new relation `g-->t3` is added. In "reverse" propagation, the relation "g–>t2" is deleted because it is an indirect relation inferred from "g–>t1" and "t1–>t2".

Parameters

- **direction** (*str*) – The direction of propagation. Either 'forward' or 'reverse'
- **inplace** (*bool*) – If True, then modify the ontology. If False, then create and modify a copy.

Returns

Return type ddot.Ontology.Ontology

4.1.5 Inferring data-driven ontology

`Ontology.`**`flatten`**(*include_genes=True*, *include_terms=False*, *similarity='Resnik'*)

Flatten the hierarchy into a node-node similarity matrix by calculating a similarity between pair of genes in *genes_subset*. Currently, only the Resnik semantic similarity measure is implemented.

Parameters

- **include_genes** (*bool*) – If True, then calculate pairwise similarities between genes. If *include_terms* is also True, then also calculate similarities between genes and terms.

- **include_terms** (*bool*) – If True, then calculate pairwise similarities between terms. If *include_genes* is also True, then also calculate similarities between genes and terms.
- **similarity** (*str*) – Type of semantic similarity. (default: "Resnik")

  The Resnik similarity s(g1,g2) is defined as $-log_2(|T_{sca}|/|T_{root}|)$ where $|T|$ is the number of genes in *genes_subset* that are under term T. T_{sca} is the "smallest common ancestor", the common ancestral term with the smallest term size. T_{root} is the root term of the ontology.

  Resnik, P. (1999). Semantic similarity in a taxonomy: An information-based measured and its application to problems of ambiguity in natural language. J. Artif. Intell. Res. 11,95-130.

Returns A 2-tuple consisting of *sim*, a node-by-node NumPy array, and *nodes*, a NumPy array of the node names in *sim*.

Return type (sim, nodes)

classmethod Ontology.**run_clixo**(*graph*, *alpha=0.0*, *beta=None*, *newman_modularity=None*, *miyauchi_modularity=None*, *stop_score=None*, *min_dt=-10000000*, *timeout=100000000*, *square=False*, *square_names=None*, *output=None*, *output_log=None*, *clixo_cmd=None*, *clixo_version=None*, *verbose=False*, *debug=False*)

Runs the CLIXO algorithm and returns the result as an Ontology object.

Acts as a wrapper for the C++ packages for CLIXO v0.3 (*https://mhk7.github.io/clixo_0.3/*) and v1.0 (*https://github.com/fanzheng10/CliXO*).

Parameters

- **graph** (*pandas.DataFrame*) – Gene-gene similarities represented as a 3-column pandas.DataFrame (sparse) or square-shaped pandas.DataFrame (square format)
- **alpha** (*float*) – CLIXO alpha parameter
- **beta** (*float*) – CLIXO beta parameter
- **min_dt** (*float*) – Minimum similarity score
- **timeout** (*int*) – Maximum time (in seconds) allowed to run CLIXO
- **square** (*bool*) – If True, then <graph> is interpreted as a square-shaped DataFrame. Otherwise, it is interpreted as a 3-column DataFrame.
- **square_names** (*array-like (optional)*) – The names of the rows and columns in <graph>. Only used when square==True.
- **output** (*str*) – Filename to write the resulting Ontology as a table. Default: don't write to file
- **output_log** (*str*) – Filename to write log information from CLIXO. Default: don't write to file

Returns

Return type ddot.Ontology.Ontology

4.1.6 Aligning ontologies

Ontology.**align**(*hier*, *iterations=100*, *threads=None*, *update_self=False*, *update_ref=False*, *align_label=None*, *calculateFDRs=None*, *mutual_collapse=True*, *output=None*, *verbose=False*)

Identifies one-to-one matches between terms in this ontology with highly similar terms in another ontology.

This function wraps around the C++ code in the alignOntology package by Michael Kramer at https://github.com/mhk7/alignOntology

Reference:

Dutkowski, J., Kramer, M., Surma, M.A., Balakrishnan, R., Cherry, J.M., Krogan, N.J. and Ideker, T., 2013. "A gene ontology inferred from molecular networks." *Nature biotechnology*, 31(1).

Parameters

- **hier** (*ddot.Ontology.Ontology*) – The ontology to align against.
- **iterations** (*int*) – The number of null model randomizations to create FDR score.
- **threads** (*int*) – Number of CPU processes to run simultaneously. Used to parallelize the the null model randomizations. Default: The number of CPU cores returned by multiprocessing.cpu_count()
- **update_self** (*bool*) – If True, then import the node attributes from the reference hierarchy as attributes in this hierarchy
- **update_ref** (*bool*) – If True, then import the node attributes from the this hierarchy as attributes in the reference hierarchy
- **mutual_collapse** (*bool*) – If True, then remove genes that are unique to either ontology, and then remove redundant terms in both ontologies using Ontology.collapse_ontology().
- **calculate_FDRs** (*str*) – Filename of the 'calculateFDRs' scripts in the alignOntology C++package at https://github.com/mhk7/alignOntology. Default: use the 'calculateFDRs' script that comes built-in with ddot.
- **output** (*str*) – Filename to write the results of the alignment as a tab-delimited file. Default: don't write to a file

Returns Dataframe where index are names of terms in this ontology. There are three columns: 'Term' (name of the aligned term), 'Similarity' (the similarity score for the alignment), 'FDR' (the FDR of this alignment given the null models).

Return type pandas.DataFrame

4.2 Utility Functions

ddot.utils.**NdexGraph_to_nx**(*G*)

Converts a NetworkX into a NdexGraph object.

Parameters **G** (*ndex.networkn.NdexGraph*) –

Returns

Return type networkx.classes.DiGraph

ddot.utils.**bubble_layout_nx**(*G*, *xmin=-750*, *xmax=750*, *ymin=-750*, *ymax=750*, *verbose=False*)

Bubble-tree Layout using the Tulip library.

The input tree must be a graph. The layout is scaled so that it is fit exactly within a bounding box.

Grivet, S., Auber, D., Domenger, J. P., & Melancon, G. (2006). Bubble tree drawing algorithm. In Computer Vision and Graphics (pp. 633-641). Springer Netherlands.

Parameters

- **G** (*networkx.Graph*) – Tree

- **xmin** (*float, optional*) – Minimum x-coordinate of the bounding box
- **xmax** (*float, optional*) – Maximum x-coordinate of the bounding box
- **ymin** (*float, optional*) – Minimum y-coordinate of the bounding box
- **ymax** (*float, optional*) – Maximum y-coordinate of the bounding box

Returns Dictionary mapping nodes to 2D coordinates. pos[node_name] -> (x,y)

Return type dict

ddot.utils.**color_gradient**(*ratio*, *min_col='#FFFFFF'*, *max_col='#D65F5F'*, *output_hex=True*)
Calculate a proportional mix between two colors.

ddot.utils.**create_edgeMatrix**(*X*, *X_cols*, *X_rows*, *verbose=True*, *G=None*, *ndex2=True*)
Converts an NumPy array into a NdexGraph with a special CX aspect called "edge_matrix". The array is serialized using base64 encoding.

Parameters

- **X** (*np.ndarray*) –
- **X_cols** (*list*) – Column names
- **X_rows** (*list*) – Row names

Returns

Return type ndex.networkn.NdexGraph

ddot.utils.**expand_seed**(*seed*, *sim*, *sim_names*, *agg='mean'*, *min_sim=-inf*, *filter_perc=None*, *seed_perc=None*, *agg_perc=0.5*, *expand_size=None*, *include_seed=True*, *figure=False*, *verbose=False*)
Identify genes that are most similar to a seed set of genes.

A gene is included in the expanded set only if it meets all of the specified criteria. If include_seed is True, then genes that are seeds will be included regardless of the criteria. At the same time, the number of genes returned is still limited by expand_size. One way to get n novel genes returned is therefore to set expand_size = n + |seed| and include_seed = True, and then to remove the seed list from expand.

Parameters

- **seed** (*list*) –
- **sim** (*np.ndarray*) –
- **sim_names** (*list of str*) –
- **agg** (*str or function*) – Aggregation method. Possible values are mean, min, max, perc.
- **min_sim** (*float*) – Minimum similarity to the seed set.
- **filter_perc** (*float*) – Filter based on a percentile of similarities between all genes and the seed set.
- **seed_perc** (*float*) – Filter based on a percentile of similarities between seed set to itself.
- **agg_perc** (*float*) – The <agg_perc> percentile of similarities to the seed set. For example, if a gene has similarities of (0, 0.2, 0.4, 0.6, 0.8) to five seed genes, then the 10% similarity is 0.2
- **expand_size** (*int*) – Maximum limit on the number of returned genes.
- **include_seed** (*bool*) – Include the seed genes even if they didn't meet the criteria.

- **figure** (*bool*) – Generate a figure showing the average distances within the seed an d the average distances between seed and the background.

Returns

- *expand* – The list of expanded genes passing all filters.
- *expand_idx* – Indices of the ranking. I.e. *expand_idx[0]* is the index of the top gene, so you can get the name of the top gene with *sim_names[expand_idx[0]]* where *sim_names* is the input parameter.
- *sim_2_seed* – The returned array *sim_2_seed* is the calculated similarities of the genes to the seed set. So *sim_2_seed[0]* is the similarity of the gene
- *fig* – The generated figure. Can be saved like this: plt.savefig('foo.pdf')

ddot.utils.**gridify**(*parents*, *pos*, *G*)

Relayout leaf nodes into a grid.

Nodes must be connected and already laid out in "star"-like topologies. In each "star", a set of nodes are positioned to form the shape of a circle and connect to a common parent node that is positioned at the circle's center.

This function repositions the nodes in each start into a square grid that inscribes the circle.

Parameters

- **parents** (*list*) – For each parent, its children will be arranged in a grid.
- **pos** (*dict*) – Dictionary that maps names of nodes to their (x,y) coordinates
- **G** (*nx.Graph*) – Network

Returns Modifies <pos> inplace

Return type None

ddot.utils.**ig_edges_to_pandas**(*G*, *attr_list=None*)

Create pandas.DataFrame of edge attributes of a igraph Graph object.

Parameters

- **G** (*igraph.Graph*) –
- **attr_list** (*list, optional*) – Names of edge attributes. Default: all edge attributes

Returns DataFrame where index is a MultIndex with two levels (u,v) referring to edges and the columns refer to edge attributes.

Return type pandas.DataFrame

ddot.utils.**ig_nodes_to_pandas**(*G*, *attr_list=None*)

Create pandas.DataFrame of node attributes of a igraph.Graph object.

Parameters

- **G** (*igraph.Graph*) –
- **attr_list** (*list, optional*) – Names of node attributes. Default: all node attributes

Returns DataFrame where index is the names of nodes and the columns are node attributes.

Return type pandas.DataFrame

ddot.utils.**ig_unfold_tree_with_attr**(*g*, *sources*, *mode*)

Call igraph.Graph.unfold_tree while preserving vertex and edge attributes.

ddot.utils.**invert_dict**(*dic*, *sort=True*, *keymap={}*, *valmap={}*)
Inverts a dictionary of the form key1 : [val1, val2] key2 : [val1]

to a dictionary of the form

val1 : [key1, key2] val2 : [key2]

Parameters **dic** (*dict*) –

Returns

Return type dict

ddot.utils.**load_edgeMatrix**(*ndex_uuid*, *ndex_server*, *ndex_user*, *ndex_pass*, *ndex=None*, *json=None*, *verbose=True*)
Loads a NumPy array from a NdexGraph with a special CX aspect called "edge_matrix".

Parameters

- **ndex_uuid** (*str*) – NDEx UUID of ontology
- **ndex_server** (*str*) – URL of NDEx server
- **ndex_user** (*str*) – NDEx username
- **ndex_pass** (*str*) – NDEx password
- **json** (*module*) – JSON module with "loads" function. Default: the simplejson package (must be installed)

Returns

- **X** (*np.ndarray*)
- **X_cols** (*list*) – Column names
- **X_rows** (*list*) – Row names

ddot.utils.**make_index**(*it*)
Create a dictionary mapping elements of an iterable to the index position of that element

ddot.utils.**make_seed_ontology**(*sim*, *sim_names*, *expand_kwargs={}*, *build_kwargs={}*, *align_kwargs={}*, *ndex_kwargs={}*, *node_attr=None*, *verbose=False*, *ndex=True*)
Assembles and analyzes a data-driven ontology to study a process or disease

Parameters

- **sim** (*np.ndarray*) – gene-by-gene similarity array
- **sim_names** (*array-like*) – Names of genes as they appear in the rows and columns of <sim>
- **expand_kwargs** (*dict*) – Parameters for ddot.expand_seed() to identify an expanded set of genes
- **build_kwargs** (*dict*) – Parameters for Ontology.build_from_network(...) to build a data-driven ontology.
- **align_kwargs** (*dict*) – Parameters for Ontology.align() to align against a reference ontology.
- **ndex_kwargs** (*dict*) – Parameters for Ontology.to_ndex() to upload ontology to NDEx.
- **node_attr** (*pd.DataFrame*) – A DataFrame of node attributes to assign to the ontology.
- **ndex** (*bool*) – If True, then upload ontology to NDEx using parameters <ndex_kwargs>

ddot.utils.**melt_square**(*df, columns=['Gene1', 'Gene2'], similarity='similarity', empty_value=0, upper_triangle=True*)

Melts square dataframe into sparse representation.

Parameters

- **df** (*pandas.DataFrame*) – Square-shaped dataframe where df[i,j] is the value of edge (i,j)
- **columns** (*iterable*) – Column names for nodes in the output dataframe
- **similarity** (*string*) – Column for edge value in the output dataframe
- **empty_value** – Not yet supported
- **upper_triangle** (*bool*) – Only use the values in the upper-right triangle (including the diagonal) of the input square dataframe

Returns 3-column dataframe that provides a sparse representation of the edges. Two of the columns indicate the node name, and the third column indicates the edge value

Return type pandas.DataFrame

ddot.utils.**ndex_to_sim_matrix**(*ndex_url, ndex_server=None, ndex_user=None, ndex_pass=None, similarity=None, input_fmt='cx_matrix', output_fmt='matrix', subset=None, verbose=True*)

Read a similarity network from NDEx and return it as either a square np.array (compact representation) or a pandas.DataFrame of the non-zero similarity values (sparse representation)

Parameters

- **ndex_url** (*str*) – NDEx URL (or UUID) of ontology
- **ndex_server** (*str*) – URL of NDEx server
- **ndex_user** (*str*) – NDEx username
- **ndex_pass** (*str*) – NDEx password
- **similarity** (*str*) – Name of the edge attribute that represents the similarity/weight between two nodes. If None, then the name of the edge attribute in the output is named 'similarity' and all edges are assumed to have a similarity value of 1.
- **input_fmt** (*str*) –
- **output_fmt** (*str*) – If 'matrix', return a NumPy array. If 'sparse', return a pandas.DataFrame
- **subset** (*optional*) –

Returns

Return type np.ndarray or pandas.DataFrame

ddot.utils.**nx_edges_to_pandas**(*G, attr_list=None*)

Create pandas.DataFrame of edge attributes of a NetworkX graph.

Parameters

- **G** (*networkx.Graph*) –
- **attr_list** (*list, optional*) – Names of edge attributes. Default: all edge attributes

Returns DataFrame where index is a MultIndex with two levels (u,v) referring to edges and the columns refer to edge attributes. For multi(di)graphs, the MultiIndex have three levels of the form (u, v, key).

Return type pandas.DataFrame

ddot.utils.**nx_nodes_to_pandas**(*G*, *attr_list=None*)
Create pandas.DataFrame of node attributes of a NetworkX graph.

Parameters

- **G** (*networkx.Graph*) –
- **attr_list** (*list, optional*) – Names of node attributes. Default: all node attributes

Returns DataFrame where index is the names of nodes and the columns are node attributes.

Return type pandas.DataFrame

ddot.utils.**nx_to_NdexGraph**(*G_nx*, *discard_null=True*)
Converts a NetworkX into a NdexGraph object.

Parameters **G_nx** (*networkx.Graph*) –

Returns

Return type ndex.networkn.NdexGraph

ddot.utils.**parse_ndex_uuid**(*ndex_url*)
Extracts the NDEx UUID from a URL

Parameters **ndex_url** (*str*) – URL for a network stored on NDEx

Returns UUID of the network

Return type str

ddot.utils.**pivot_square**(*df*, *index*, *columns*, *values*, *fill_value=0*)
Convert a dataframe into a square compact representation.

Parameters **df** (*pandas.DataFrame*) – DataFrame in long-format where every row represents one gene pair

Returns **df** – DataFrame with gene-by-gene dimensions

Return type pandas.DataFrame

ddot.utils.**set_edge_attributes_from_pandas**(*G*, *edge_attr*)
Modify edge attributes according to a pandas.DataFrame.

Parameters

- **G** (*networkx.Graph*) –
- **edge_attr** (*pandas.DataFrame*) –

ddot.utils.**set_node_attributes_from_pandas**(*G*, *node_attr*)
Modify node attributes according to a pandas.DataFrame.

Parameters

- **G** (*networkx.Graph*) –
- **node_attr** (*pandas.DataFrame*) –

ddot.utils.**sim_matrix_to_NdexGraph**(*sim*, *names*, *similarity*, *output_fmt*, *node_attr=None*)
Convert similarity matrix into NdexGraph object

Parameters

- **sim** (*np.ndarray*) – Square-shaped NumPy array representing similarities
- **names** (*list*) – Genes names, in the same order as the rows and columns of sim

- **similarity** (*str*) – Edge attribute name for similarities in the resulting NdexGraph object
- **output_fmt** (*str*) – Either 'cx' (Standard CX format), or 'cx_matrix' (custom edgeMatrix aspect)
- **node_attr** (*pandas.DataFrame, optional*) – Node attributes, as a pandas.DataFrame, to be set in NdexGraph object

Returns

Return type ndex.networkn.NdexGraph

ddot.utils.**transform_pos**(*pos*, *xmin=-250*, *xmax=250*, *ymin=-250*, *ymax=250*)

Transforms coordinates to fit a bounding box.

Parameters

- **pos** (*dict*) – Dictionary mapping node names to (x,y) coordinates
- **xmin** (*float, optional*) – Minimum x-coordinate of the bounding box
- **xmax** (*float, optional*) – Maximum x-coordinate of the bounding box
- **ymin** (*float, optional*) – Minimum y-coordinate of the bounding box
- **ymax** (*float, optional*) – Maximum y-coordinate of the bounding box

Returns New dictionary with transformed coordinates

Return type dict

ddot.utils.**update_nx_with_alignment**(*G*, *alignment*, *term_descriptions=None*, *use_node_name=True*)

Add node attributes to a NetworkX graph.

Parameters

- **G** – NetworkX object
- **alignment** – pandas.DataFrame where the index is the name of terms, and where there are 3 columns: 'Term', 'Similarity', 'FDR'
- **use_node_name** (*bool*) –
- **term_descriptions** (*dict*) –

Returns

Return type None

Python Module Index

d

Index

U