graphtools Documentation

Release 1.0

Gautam Sisodia

September 26, 2014

CONTENTS

CHAPTER

ONE

GRAPHTOOLS PACKAGE

1.1 `graphtools` Package

This package includes `graphtools.gengraph.GenGraph`, a superclass implementing the descent algorithm, and subclasses of `GenGraph` which accomodate various graph data structures. Namely

- `graphtools.dbgraph.DBGraph` reads graph data stored in a sqlalchemy-supported database,
- `graphtools.sagegraph.SageGraph` wraps a `sage.graphs.digraph.DiGraph` object, and
- `graphtools.listgraph.ListGraph` reads graph data stored as a list of arrows.

Multiple types for the vertices are supported. The type *vertex* refers to the instance-specific type of the vertices.

1.2 `gengraph` Module

class `graphtools.gengraph.`**`GenGraph`**

A graph superclass implementing the descent algorithm. Subclasses should overwrite

- `get_num_arrows()`,
- `get_vert_list()`,
- `get_rank()`,
- `set_rank()` and
- `count_neighbors()`

The method `descent()` runs the algorithm, updating the ranks of the vertices.

`count_neighbors``(`*vert*, *out=True*, *cond=False*, *less=True*, *cutoff=0*`)`

Count the number of neighbors of a vertex.

Parameters

- **vert** (*vertex*) – the vertex to count the neighbors of
- **out** (*bool*) – if True, count out neighbors, else in
- **cond** (*bool*) – if True, count the neighbors satisfying a condition on rank
- **less** (*bool*) – if True, count the neighbors with rank less than or equal to *cutoff*, else more
- **cutoff** (*int*) – the cutoff rank for the conditional

Returns the number of (in or out) neighbors of *vert* (perhaps satisfying a condition on the rank)

Return type int

`descend`(*vert*, *debug=False*)
Run one iteration of the descend algorithm.

Parameters

- **vert** (*vertex*) – the vertex whose rank may change
- **debug** (*bool*) – if True, output debug code

`descent`(*num=1*, *debug=False*)
Run `descend()` a number of times on random vertices.

Parameters

- **num** (*int*) – the number of times to run `descend()`
- **debug** (*bool*) – if True, output debug code

`get_hierarchy_list`()
Return the list of hierarchy scores.

Returns the list of hierarchy scores

Return type list

`get_num_arrows`()
Return the number of arrows.

Returns the number of arrows in the graph

Return type int

`get_rank`(*vert*)
Return the rank of vertex vert.

Parameters **vert** (*vertex*) – the vertex to get the rank of

Returns the rank of *vert*

Return type int

`get_vert_list`()
Return a list of the vertices of the graph.

Returns a list of vertices of the graph

Return type list

`set_rank`(*vert*, *newrank*)
Set the rank of vertex *vert* to int *newrank*.

Parameters

- **vert** (*vertex*) – the vertex to set the rank of
- **newrank** (*int*) – the new rank of *vert*

1.3 `dbgraph` Module

class `graphtools.dbgraph.DBGraph`(*users*, *arrows*, *conn*, *group=None*)
Bases: `graphtools.gengraph.GenGraph`

A subclass of GenGraph for graphs stored in databases.

The vertices are stored in a table called **users** with columns

- *user_id* (any type) and
- *rank* (int)

(the name comes from the original motivation which was Twitter user subgraphs). The table may also have a column *group* (any type) specifying the particular graph that the user belongs to if the database contains multiple graphs. If the table has no *group* column, *user_id* should be a unique identifier; if there is a *group* column, *user_id* and *group* together should be unique.

The arrows are stored in a table called **arrows** with columns

- *follow_id* and
- *lead_id*

both refering to **users**.*user_id*. If **users** has a *group* column then **arrows** should have a corresponding *group* column.

Parameters

- **users** (*sqlalchemy.schema.Table*) – the table of vertices, described above
- **arrows** (*sqlalchemy.schema.Table*) – the table of arrows, described above
- **conn** (*sqlalchemy.engine.base.Connection*) – a connection to the database
- **group** (*any*) – an optional identifier, described above

The initializer sets all entries in **user**.*rank* to 0.

`count_neighbors`(*vert*, *out=True*, *cond=False*, *less=True*, *cutoff=0*)
See `graphtools.gengraph.GenGraph.count_neighbors()`.

`reset_ranks`()
Set all ranks to zero.

1.4 `listgraph` Module

class `graphtools.listgraph.`**`ListGraph`**(*arrows_list*)
Bases: `graphtools.gengraph.GenGraph`

A subclass of GenGraph for graphs given as a list of arrows.

Parameters **arrows_list** (*list*) – a list of the arrows in the graph; each arrow is an ordered list of 2 vertices (any type) where the first vertex is the tail of the arrow and the second is the head.

Graphs with isolated vertices (vertices with no neighbors) are not supported. Isolated vertices don't affect the hierarchy of the graph.

`neighbors_in`(*vert*)
Return the list of in neighbors of vertex *vert*.

Parameters **vert** (*vertex*) – the vertex to count the neighbors of

Returns the number of in neighbors of *vert*

Return type int

`neighbors_out`(*vert*)

Return the list of out neighbors of vertex *vert*.

Parameters **vert** (*vertex*) – the vertex to count the neighbors of

Returns the number of out neighbors of *vert*

Return type int

1.5 `sagegraph` Module

class `graphtools.sagegraph.`**`SageGraph`**(*dg*)

Bases: `graphtools.gengraph.GenGraph`

A subclass of GenGraph which wraps a Sage DiGraph object.

`rankdict` **= None**

The rank dictionary; keys are the vertices and values are the ranks.

PYTHON MODULE INDEX

INDEX