NetworkX - PostGIS Python interface

API Documentation

August 1, 2012

Contents

1 Module nx_pg

nx_pg.py - Read/write support for PostGIS tables in NetworkX

Introduction

NetworkX is a python library for graph analysis. Using edge and node attribution it can be used for spatial network analysis (with geography stored as node/edge attributes). This module supports the use of NetworkX for the development of network analysis of spatial networks stored in a PostGIS spatial database by acting as an interface to node and edge tables which contain graph nodes and edges.

Notes

Node support

Note that nodes are defined by the edges at network creation and the reading of node tables independently is not currently supported (because without defining a primary key/foreign key relationship this can easily break the network). This issue is solved in nx_pgnet which should be used for proper storage of networks in PostGIS tables. To read/write PostGIS networks (as defined by a network schema use) the nx_pgnet module.

Output tables

For each network written two tables are created: edges and nodes. This representation is similar to that of the nx_shp module (for reading/writing network shapefiles).

Coordinate system support

nx_pg has no support for defining a coordinate system of the output tables. Geometry is written without an SRS value, when viewing using a GIS you must specify the correct coordinate system for the network, nx_pgnet has coordinate systems support for network tables.

Graph/Network terms

Note that the terms 'graph' and 'network' are used interchangeably within the software and documentation. To some extent a 'graph' refers to a topological object (often in memory) with none or limited attribution, whilst a 'network' refers to a graph object with attribution of edges and nodes and with geography defined, although this is not always the case.

Module structure

The module has two key functions:

- read_pg: Function to create NetworkX graph instance from PostGIS table(s) representing edge and node objects.
- write_pg: Function to create PostGIS tables (edge and node) from NetworkX graph instance.
- Other functions support the read and write operations.

Database connections

Connections to PostGIS are created using the OGR simple feature library and are passed to the read() and write() classes. See *http://www.gdal.org/ogr*

Connections are mutually exclusive between read_pg and write_pg, although you can of course read and write to the same database. You must pass a valid connection to the read or write classes for the module to work.

To create a connection using the OGR python bindings to a database on localhost:

```
>>> import osgeo.ogr as ogr
>>> conn = ogr.Open("PG: host='127.0.0.1' dbname='database' user='postgres'
>>>                 password='password'")
```

Examples

The following are examples of high level read and write network operations. For more detailed information see method documentation below.

Reading a network from PostGIS table of LINESTRINGS representing edges:

```
>>> import nx_pg
>>> import osgeo.ogr as ogr

>>> # Create a connection
>>> conn = ogr.Open("PG: host='127.0.0.1' dbname='database' user='postgres'
>>>                 password='password'")

>>> # Read a network
>>> # Note 'my_network' is the name of the network stored in the 'Graphs' table

>>> network = nx_pg.read_pg(conn, 'edges_tablename')
```

Adding node attributes

Nodes are created automatically at the start/end of a line, or where there is a break in a line. A user can add node attributes from a table of point geometries which represent these locations. To add attributes to nodes use the nodes_tablename option in the read function:

```
>>> network = nx_pg.read_pg(conn, 'edge_tablename', 'node_tablename')
```

This will add attributes from the node table to nodes in the network where a network node geometry is equal to a point geometry in the specified node table.

Note that this will not add all points in the nodes table if not all points match the geometry of created nodes.

Also note that if two points share the same geometry as a node, only one of the point attributes will be added (whichever occurs later in the data).

Writing a NetworkX graph instance to a PostGIS schema:

Write the network to the same database but under a different name. Note if 'overwrite=True' then an existing network in the database of the same name will be overwritten.

```
>>> nx_pg.write_pg(conn, network, 'new_network, overwrite=False')
```

Dependencies

- Python 2.6 or later
- NetworkX 1.6 or later
- OGR 1.8.0 or later

Copyright

Tomas Holderness & Newcastle University

Developed by Tom Holderness at Newcastle University School of Civil Engineering and Geosciences, Geoinfomatics group:

Acknowledgement

Acknowledgement must be made to the nx_shp developers as much of the functionality of this module is the same.

License

This software is released under a BSD style license which must be compatible with the NetworkX license because of similarities with NetworkX source code.:

See LICENSE.TXT or type nx_pg.license() for full details.

Credits

Tomas Holderness, David Alderson, Alistair Ford, Stuart Barr and Craig Robson.

Contact

tom.holderness@ncl.ac.uk

www.staff.ncl.ac.uk/tom.holderness

Version: 0.9.1

Author: Tom Holderness

1.1 Functions

getfieldinfo(*lyr*, *feature*, *flds*)

Get information about fields from a table (as OGR feature).

read_pg(*conn*, *edgetable*, *nodetable*=`None`, *directed*=`False`)

Read a network from PostGIS table of line geometry.

Optionally takes a table of points and where point geometry is equal to that of nodes created, point attributes will be added to nodes.

Returns instance of networkx.Graph().

netgeometry(*key*, *data*)

Create OGR geometry from a NetworkX Graph using Wkb/Wkt attributes.

create_feature(*geometry*, *lyr*, *attributes*=`None`)

Wrapper for OGR CreateFeature function.

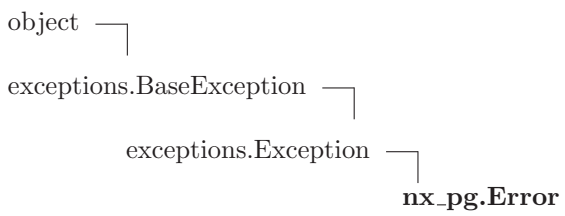
Creates a feature in the specified table with geometry and attributes.

write_pg(*conn*, *network*, *tablename_prefix*, *overwrite*=`False`)
Write NetworkX instance to PostGIS edge and node tables.

1.2 Variables

Name	Description
__created__	**Value:** `'Thu Jan 19 15:55:13 2012'`
__year__	**Value:** `'2011'`
__package__	**Value:** `None`

1.3 Class Error



Class to handle network IO errors.

1.3.1 Methods

__init__(*self*, *value*)
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
Overrides: object.__init__ extit(inherited documentation)

__str__(*self*)
str(x)
Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

1.3.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

2 Module nx_pgnet

nx_pgnet - Read/write support for PostGIS network schema in NetworkX.

Introduction

nx_pgnet is module for reading and writing NetworkX graphs to and from PostGIS tables as specified by the Newcastle University PostGIS network schema.

Note that the terms 'graph' and 'network' are used interchangeably within the software and documentation. To some extent a 'graph' refers to a topological object (often in memory) with none or limited attribution, whilst a 'network' refers to a graph object with attribution of edges and nodes and with geography defined, although this is not always the case.

nx_pgnet operations

read: PostGIS (network schema) –> NetworkX

write: PostGIS (network schema) <– NetworkX

Description

NetworkX is a python library for graph analysis. Using edge and node attribution it can be used for spatial network analysis (with geography stored as node/edge attributes). This module supports the use of NetworkX for the development of network analysis of spatial networks stored in a PostGIS spatial database by acting as an interface to a predefined table structure (the schema) in a PostGIS database from NetworkX Graph classes.

PostGIS Schema

This module assumes that the required PostGIS network schema is available within the target/source PostGIS database. The schema allows for a collection of tables to represent a spatial network, storing network geography and attributes. The definition of the schema and the associated scripts for network creation are outside the scope of this documentation however the schema can be briefly described as the following tables:

- Graphs: Holds a reference to all networks in the database, referencing Edge, Edge_Geometry and Node tables.
- Edges: Holds a representation of a network edge by storing source and destination nodes and edge attributes. Contains foreign keys to graph and edge geometry
- Edge_Geometry: Holds geometry (PostGIS binary LINESTRING/MULTILINESTRING representation). Edge geometry is stored separately to edges for storage/retrieval performance where more than one edge share the same geometry.
- Interdependency: Holds interdependencies between networks.
- Interdependency_Edges: Holds interdependency geometry.

Module structure

The module is split into three key classes:

- read: Contains methods to read data from PostGIS network schema to a NetworkX graph.
- write: Contains methods to write a NetworkX graph to PostGIS network schema tables.
- nisql: Contains methods which act as a wrapper to the special PostGIS network schema functions.
- errors: Class containing error catching, reporting and logging methods.

Detailed documentation for each class can be found below contained in class document strings. The highest level functions for reading and writing data are:

Read:

```
>>> nx_pgnet.read().pgnet()
>>> # Reads a network from PostGIS network schema into a NetworkX graph instance.
```

Write:

```
>>> nx_pgnet.write().pgnet()
>>> # Writes a NetworkX graph instance to PostGIS network schema tables.
```

Database connections

Connections to PostGIS are created using the OGR simple feature library and are passed to the read() and write() classes. See http://www.gdal.org/ogr

Connections are mutually exclusive between read() and write() and are contained within each class (i.e. all methods within those classes inherit the : connection), although you can of course read and write to the same database. You must pass a valid connection to the read or write classes for the module to work.

To create a connection using the OGR python bindings to a database on localhost:

```
>>> import osgeo.ogr as ogr
>>> conn = ogr.Open("PG: host='127.0.0.1' dbname='database' user='postgres'
>>>                 password='password'")
```

Examples

The following are examples of read and write network operations. For more detailed information see method documentation below.

Reading a network from PostGIS schema to a NetworkX graph instance:

```
>>> import nx_pgnet
>>> import osgeo.ogr as ogr

>>> # Create a connection
>>> conn = ogr.Open("PG: host='127.0.0.1' dbname='database' user='postgres'
```

```
>>>                   password='password'")

>>> # Read a network
>>> # Note 'my_network' is the name of the network stored in the 'Graphs' table
>>> network = nx_pgnet.read(conn).pgnet('my_network')
```

Writing a NetworkX graph instance to a PostGIS schema:

Write the network to the same database but under a different name. 'EPSG' is the EPSE code for the output network geometry. Note if 'overwrite=True' then an existing network in the database of the same name will be overwritten.

```
>>> epsg = 27700
>>> nx_pgnet.write(conn).pgnet(network, 'new_network', epsg, overwrite=False)
```

Dependencies

Python 2.6 or later NetworkX 1.6 or later OGR 1.8.0 or later

Copyright (C)

Tomas Holderness & Newcastle University

Developed by Tom Holderness at Newcastle University School of Civil Engineering and Geosciences, geoinfomatics group:

David Alderson, Alistair Ford, Stuart Barr, Craig Robson.

License

This software is released under a BSD style license. See LICENSE.TXT or type nx_pgnet.license() for full details.

Credits

Tomas Holderness, David Alderson, Alistair Ford, Stuart Barr and Craig Robson.

Contact

tom.holderness@ncl.ac.uk www.staff.ncl.ac.uk/tom.holderness

Development Notes

Where possible the PEP8/PEP257 style guide has been implemented.

To do:

1. Check attribution of nodes from schema and non-schema sources (blank old id fields are being copied over).
2. Error / warnings module.
3. Investigate bug: "Warning 1: Geometry to be inserted is of type Line String, whereas

the layer geometry type is Multi Line String. Insertion is likely to fail!"

4. Multi and directed graph support.
5. 3D geometry support.

Version: 0.9.2

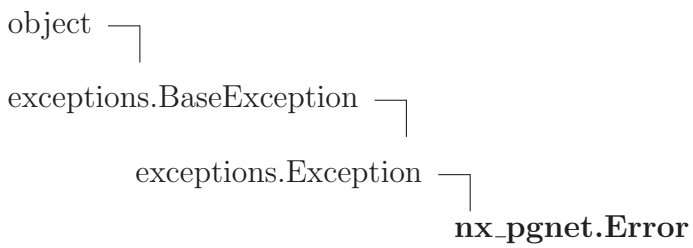
Author: Tomas Holderness (tom.holderness@ncl.ac.uk) David Alderson (david.alderson@ncl.ac.uk) Alistair Ford (a.c.ford@ncl.ac.uk) Stuart Barr (stuart.barr@ncl.ac.uk)

License: BSD style. See LICENSE.TXT

2.1 Variables

Name	Description
__created__	**Value:** `'January 2012'`
__package__	**Value:** `None`

2.2 Class Error



Class to handle network IO errors.

2.2.1 Methods

__init__(*self*, *value*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Overrides: object.__init__ extit(inherited documentation)

__str__(*self*)

str(x)

Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

2.2.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

2.3 Class nisql

Contains wrappers for PostGIS network schema functions.

Where possible avoid using this class directly. Uses the read and write classes instead.

2.3.1 Methods

__init__(*self*, *db_conn*)

Setup connection to be inherited by methods.

sql_function_check(*self*, *function_name*)

Checks Postgres database for existence of specified function, if not found raises error.

create_network_tables(*self*, *prefix*, *epsg*, *directed*, *multigraph*)

Wrapper for ni_create_network_tables function.

Creates empty network schema PostGIS tables. Requires graph 'prefix 'name and srid to create empty network schema PostGIS tables.

Returns True if succesful.

create_node_view(*self*, *prefix*)

Wrapper for ni_create_node_view function.

Creates a view containing node attributes and geometry values including int primary key suitable for QGIS.

Requires network name ('prefix').

Returns view name if succesful.

create_edge_view(*self*, *prefix*)

Wrapper for ni_create_edge_view function.

Creates a view containing edge attributes and edge geometry values. Requires network name ('prefix').

Returns view name if succesful.

add_graph_record(*self*, *prefix*, *directed*=`False`, *multipath*=`False`)

Wrapper for ni_add_graph_record function.

Creates a record in the Graphs table based on graph attributes.

Returns new graph id of succesful.

node_geometry_equaility_check(*self*, *prefix*, *wkt*, *srs*)

Wrapper for ni_node_geometry_equality_check function.

Checks if geometry already eixsts in nodes table.

If not, returns None

edge_geometry_equaility_check(*self*, *prefix*, *wkt*, *srs*)

Wrapper for ni_edge_geometry_equality_check function.

Checks if geometry already eixsts in nodes table.

If not, return None

delete_network(*self*, *prefix*)

Wrapper for ni_delete_network function.

Deletes a network entry from the Graphs table and drops associated tables.

2.4 Class read

Class to read and build networks from PostGIS schema network tables.

2.4.1 Methods

__init__(*self*, *db_conn*)

Setup connection to be inherited by methods.

getfieldinfo(*self*, *lyr*, *feature*, *flds*)

Get information about fields from a table (as OGR feature).

pgnet_edges(*self*, *graph*)

Reads edges from edge and edge_geometry tables and add to graph.

pgnet_nodes(*self*, *graph*)

Reads nodes from node table and add to graph.

graph_table(*self*, *prefix*)

Reads the attributes of a graph from the graph table.

Returns attributes as a dict of variables.

pgnet(*self*, *prefix*)

Read a network from PostGIS network schema tables.

Returns instance of networkx.Graph().

2.5 Class write

Class to write NetworkX instance to PostGIS network schema tables.

2.5.1 Methods

__init__(*self*, *db_conn*)

Setup connection to be inherited by methods.

getlayer(*self*, *tablename*)

Get a PostGIS table by name and return as OGR layer.

Else, return None.

netgeometry(*self*, *key*, *data*)

Create OGR geometry from a NetworkX Graph using Wkb/Wkt attributes.

create_feature(*self*, *lyr*, *attributes*=`None`, *geometry*=`None`)

Wrapper for OGR CreateFeature function.

Creates a feature in the specified table with geometry and attributes.

create_attribute_map(*self*, *lyr*, *g_obj*, *fields*)

Build a dict of attribute field names, data and OGR data types.

Accepts graph object (either node or edge), fields and returns attribute dictionary.

update_graph_table(*self*, *graph*)

Update the Graph table and return newly assigned Graph ID.

pgnet_edge(*self*, *edge_attributes*, *edge_geom*)

Write an edge to Edge and Edge_Geometry tables.

pgnet_node(*self*, *node_attributes*, *node_geom*)

Write a node to a Node table.

Return the newly assigned NodeID.

pgnet(*self*, *network*, *tablename_prefix*, *srs*, *overwrite*=`False`, *directed*=`False`, *multigraph*=`False`)

Write NetworkX instance to PostGIS network schema tables.

Updates Graph table with new network.

Note that schema constrains must be applied in database. There are no checks for database errors here.

Index