**Description:**

An extension to the ITRC interdependency network database model, including database schema, wrappers and functions, to handle explicitly network attributes and their functions for the field of network modelling and simulation.

**Developed by:** Craig Robson

Newcastle University

December 2014

# **Introduction**

The contents of this document sets out the form and structure of an extension to the ITRC database network schema which explicitly handles network attributes and the functions related to these. A schema diagram below shows the local structure for a single network. The schema can handle multiple networks. See the nx\_pgnet library for more details on the global structure of the database.

Section 2 introduces how the extension can be utilised to analyse networks. Section 3 provides some information on the key python functions which have been developed which enable this extension to work seamlessly with the ITRC database schema model. Section 4 introduces the key PostgreSQL functions which have been developed for the handling and manipulation database-side of the network tables.



# **Using the extended database schema and functions**

Key functions which facilitate the of the database schema for the storage, management and analysis of networks.

**Loading a network from the database**

This creates a NetworkX instance of the network from a database. The attributes variable allows the attributes to be added to the network, from a selection of up to five for the nodes and five for the edges) when returned to be specified. Function for each are also retrieved. If attributes set to ‘None’, a network is returned without any of the attributes are functions attached to nodes or edges.

G = nx\_pgnet\_atts.read(conn, network\_name).read\_from\_db(attributes)

attributes = [{'flow':False, 'capacity':True, 'storage':False, 'resistance':False, 'latency':False},

{'flow':False, 'capacity':True, 'length':False, 'resistance':False, 'stacking':False}]

**Adding a network to the database**

From a network instance a network can be written to the database schema.

If the network contains the attributes and functions for them, these can be added to the database schema through specifying in the attribute dict those which are present in the network. The contains\_atts variable and contains\_functions variable must also be set as True. The function will then add the functions to the function table for the network and then add the correct function id for each node and edge.

If the attributes are present in the network instance, but the functions, these can be added in a similar way, though the functioned for the nodes and edges will be left blank.

The overwrite variable allows the function to overwrite a network with the same name in the database with that selected as the input for this function.

G=nx\_pgnet\_atts.write(conn,network\_name).write\_to\_db(G,attributes,contains\_atts,contains\_functions,overwrite)

**Add functions to function table**

Where the functions for attributes are not present in the network, these can be added separately to the functions table. A list of lists is used so multiple functions can be added in a single process, where the format for each should be ‘type,function text, function id’.

result = nx\_pgnet\_atts.write(conn,name).add\_functions(functions)

**Update a function**

A function can be updated directly in the database. This updates the functions table, and thus any network instances will have to be re-created from the database or updated directly as a network instance.

result = nx\_pgnet\_atts.write(conn,name).update\_function(functioned,new\_function, function\_type)

**Find existing functions**

All functions in the network functions table are returned, along with their id’s and types.

result = nx\_pgnet\_atts.read(conn,name).return\_network\_functions()

# **Python functions developed**

**Write class**

|  |  |
| --- | --- |
| write\_to\_db() |  |
| Writes a network to the database, storing the functions and attributes in separate tables for nodes and edges if identified by the user. | |

|  |  |
| --- | --- |
| add\_functions() |  |
| Adds a function to the function table with the specified unique function id, the text for the function (written in a pythonic format) and the type of function it is. | |

|  |  |
| --- | --- |
| update\_functions () |  |
| Updates the function text (and the type is required) for a specified function in the function table. | |

**Read class**

|  |  |
| --- | --- |
| pull\_from\_db() |  |
| Loads a network from the database with the attributes and functions if both requested and present in the database, creating a networkx network instance. | |

|  |  |
| --- | --- |
| return\_network\_functions() |  |
| Returns a list of the functions in the function table. | |

Table\_sql class

# **PostgreSQL functions developed**

A number of PostgreSQL functions have been developed which enable the creation, manipulation and management of the network tables required to full fill the requirements of the developed model. The key functions which may be of interest to a user are specified below. For all others, please see the list of functions in the database itself.

|  |  |
| --- | --- |
| np\_add\_edge\_attribute() |  |
| Updates the attribute value (attribute and value user specified) and the functionID (providing that specified exists in the function table). | |

|  |  |
| --- | --- |
| np\_add\_function() |  |
| Adds a new function to the function table with the user specifying the functionID. | |

|  |  |
| --- | --- |
| np\_add\_functionid\_to\_edge\_attribute\_table() |  |
| Updates the functionID, provided by the user, if it exists in the function table, for the specified edge record in the specified attribute table. | |

|  |  |
| --- | --- |
| np\_add\_functionid\_to\_node\_attribute\_table() |  |
| Updates the functionID, provided by the user, if it exists in the function table, for the specified node record in the specified attribute table. | |

|  |  |
| --- | --- |
| np\_add\_node\_attribute() |  |
| Updates the attribute value (attribute and value user specified) and the functionID (providing that specified exists in the function table). | |

|  |  |
| --- | --- |
| np\_check\_attribute\_table\_exists() |  |
| Given an attribute, checks if it exists that a table exists for the specified network. | |

|  |  |
| --- | --- |
| np\_check\_edge\_columns() |  |
| Given an attribute checks if this exists in the main node table. If so, appends ‘\_1’ on to the name. | |

|  |  |
| --- | --- |
| np\_check\_node\_columns() |  |
| Given an attribute checks if this exists in the main node table. If so, appends ‘\_1’ on to the name. | |

|  |  |
| --- | --- |
| np\_create\_edge\_attribute\_table() |  |
| Build an attribute table for the edges in a network, provided with a name by the user. | |

|  |  |
| --- | --- |
| np\_create\_edge\_view() |  |
| Generates the edge view from which the network is built from. Adds attribute columns to the edges and their geometries as requested by the user. | |

|  |  |
| --- | --- |
| np\_create\_function\_table() |  |
| Creates the function table when the network is initially built using the network prefix. | |

|  |  |
| --- | --- |
| np\_create\_node\_attribute\_table() |  |
| Build an attribute table for the nodes in a network, provided with a name by the user. | |

|  |  |
| --- | --- |
| np\_create\_node\_view() |  |
| Generates the node view from which the network is built from. Adds attribute columns to the nodes, including their geometry, as requested by the user. | |

|  |  |
| --- | --- |
| np\_delete\_all\_tables() |  |
| Deletes all tables related to the network from the prefix provided including those for the network itself and the views. | |

|  |  |
| --- | --- |
| np\_update\_edge\_attribute() |  |
| Updates the attribute of the specified edge record for the supplied attribute. Checks the table exists for attempting to run the update commend. | |

|  |  |
| --- | --- |
| np\_update\_node\_attribute() |  |
| Updates the attribute of the specified node record for the supplied attribute. Checks the table exists for attempting to run the update commend. | |