**Resilience code documentation**

*For the analysis of networks for their robustness to failures as standalone networks and as systems where one network is dependent on another.*

Craig Robson

Newcastle University

September 2014

Contents

[1 Summary 2](#_Toc386004617)

[1.1 Utilisation once imported 2](#_Toc386004618)

[1.2 Dependencies 2](#_Toc386004619)

[2 Examples 3](#_Toc386004620)

[2.1 Quick analysis of a network 3](#_Toc386004621)

[2.2 Changing the simulation failure model 4](#_Toc386004622)

[2.3 Adding more metrics to the results 5](#_Toc386004623)

[4 Variables 7](#_Toc386004624)

[4.1 Analysis 7](#_Toc386004625)

[4.2 Metrics 8](#_Toc386004626)

[5 Module functions 9](#_Toc386004627)

[6 Subsidiary Modules 10](#_Toc386004628)

[6.1 Classes – Error Handling 10](#_Toc386004629)

[6.2 Failure methods functions 10](#_Toc386004630)

[6.3 Network handling functions 11](#_Toc386004631)

[6.4 Tools functions 12](#_Toc386004632)

[6.5 Outputs function 12](#_Toc386004633)

[7 Key function diagrams 14](#_Toc386004634)

[7.1 main function 14](#_Toc386004635)

[7.2 step function 15](#_Toc386004636)

[7.3 analysis\_B function 16](#_Toc386004637)

# Summary

The code can be used standalone, or imported and as a module. This has been built with python 2.7 and utilises networkx, a python package the analysis and manipulation of complex networks. The module allows the behaviour of a network to be analysed against perturbations, performed through the removal of nodes from the network, for which there are multiple options available within this module. The module allows the analysis of a single network, or a pair of networks where one is dependent on the other, or where both are dependent on the each other. The behaviour of the network(s) can be mapped through the use of a multitude of metrics which can be selected by the user (though a default set will always be used on top of these as well).

## Utilisation once imported

Once imported a failure simulation can be run through calling the main function. This will run the analysis from start to finish, saving the results to a text file upon completion. This gives limited options for the analysis compared to some of the more advanced options. An example of this is given in section 2.1. Further examples which use the main function but give a greater level of flexibility in the analysis which can be performed are given in sections 2.2 and 2.3.

Further customisation to the analysis process can be generated through the use of the other functions which have been made available, including the step function. When called, this performs the removal of nodes/edges for a single epoch (time/step), thus allowing for a greater level of customisation between epochs. A diagram of the process which this module handles can be found in section 7.2.

## Dependencies

This module is reliant on a number of other modules, all of which are standard with most python installations, with the exception of networkx, which can be installed from here <http://networkx.github.io/download.html>. Extra options/connectivity is possible also through the use of the nx\_pgnet module (which also is dependent on a number of other modules/applications, see the documentation for further details).

Required:

* networkx
* sys
* random
* time
* datetime

Optional:

* nx\_pgnet

# Examples

## Quick analysis of a network

The screen shot below shows a simple script which generates a network using networkx then using the default settings runs a failure simulation over the created network. The results are then output to a text file. (The default settings run a sequential random failure model on a single network with one simulation. It also uses only the basic set of metrics for recording the behaviour of the network.)

|  |
| --- |
| import networkx as nx  import sys  #specify location of module and import  sys.path.append(‘location of module’)  import interdependency\_analysis\_v\_5\_3\_2 as ia  #create a network using networkx  GA = nx.gnm\_random\_graph(34,154)  #create a second network, leave blank if only analysing one network  GB = None  #specify the location and name of output file  result\_file = ‘file location/file name.txt’  #specify location and name for logfile – None if not wanted  logfilepath = None  #get default set of parameters  parameters = ia.default\_parameters(result\_file)  #run analysis  completed = ia.main(GA, GB, parameters, logfilepath)  print ‘Did the simulation complete successfully (True/False):’, completed |

## Changing the simulation failure model

To change the type of failure model which is run over the network, a number of variables can be changed.

|  |
| --- |
| import networkx as nx  import sys  #specify location of module and import  sys.path.append(‘location of module’)  import interdependency\_analysis\_v\_5\_3\_2 as ia  #create a network using networkx  GA = nx.gnm\_random\_graph(34,154)  #create a second network, leave blank if only analysing one network  GB = None  #specify the location and name of output file  result\_file = ‘file location/file name.txt’  #specify location and name for logfile – None if not wanted  logfilepath = None  #set analysis methods  #standalone, dependency , interdependency  analysis\_method = True, False, False  #single, sequential, cascading  failure\_process = False, True, False  #random, degree, betweenness  selection\_method = False, True, False  #get default set of parameters  parameters = ia.default\_parameters(result\_file, analysis\_method, failure\_process, selection\_method)  #run analysis  completed = ia.main(GA, GB, parameters, logfilepath)  print ‘Did the simulation complete successfully (True/False):’, completed |

The introduction of the three set of options in the above (analysis method, failure process and selection, method), allows for any of the available failure models to be run. To change the type of analysis, simply set the appropriate item to True, making sure all others are set to False. For example, the above setting will run, for a single network (STAND ALONE = True (analysis\_method), a sequential failure model (SEQUENTIAL = True (failure\_process)), where the node to remove at each time step is selected using the degree based method (DEGREE = True (selection\_method)).

## Adding more metrics to the results

Again, this is dependent upon the setting of appropriate variables, as can be seen in the example below. **The code below replaces a single line in the above examples, so no call is made to the default\_parameters function**, but instead they are all defined within the script. The metrics are split into two categories; basic and option. The basic set are those which will always be computed as they are either needed for the analysis to work or are deemed fundamental to the interpretation of the results. The option set, which has 14 optional metrics, are those which add detail to the results, and thus can be set as either True or False (True meaning it will be computed at every time step). These need to be stated otherwise an error will be returned. The metrics can be set for each network when working with two, allowing for a flexible analysis approach.

|  |
| --- |
| #set parameters manually  #variables on how to handle subgraphs and isolated nodes  REMOVE\_SUBGRAPHS = False  REMOVE\_ISOLATES = False  NO\_ISOLATES = False  #set as a list of edges for dependency analysis  a\_to\_b\_edges = None  #basic metrics are:  basic\_A = True, True, True, True, True  basic\_B = None  #option metrics are:  size\_of\_components\_A = True  giant\_component\_size\_A = True  av\_nodes\_in\_components\_A = True  Isolated\_nodes\_A = True  Isolated\_n\_count\_A = True  Isolated\_n\_count\_removed\_A = True  subnodes\_A = True  subnodes\_count\_A = True  path\_length\_A = True  av\_path\_length\_components\_A = True  giant\_component\_av\_path\_length\_A = True  av\_path\_length\_geo\_A = True  average\_degree\_A = True  inter\_removed\_count\_A = False  option\_A = size\_of\_components\_A, giant\_component\_size\_A, av\_nodes\_in\_components\_A, isolated\_nodes\_A, isolated\_n\_count\_A, isolated\_n\_count\_removed\_A, subnodes\_A, subnodes\_count\_A, path\_length\_A, av\_path\_length\_components\_A, giant\_component\_av\_path\_length\_A, av\_path\_length\_geo\_A, average\_degree\_A, inter\_removed\_count\_A  option\_B = None  metrics = basic\_A, basic\_B, option\_A, option\_B  parameters = [metrics, failure\_1[0], failure\_1[1], failure\_1[2], failure\_2[0],  failure\_2[1], failure\_2[2], failure\_3[0], failure\_3[1], failure\_3[2],  REMOVE\_SUBGRAPHS, REMOVE\_ISOLATES, NO\_ISOLATES,  result\_file, a\_to\_b\_edges] |

# Functionality

## Failure methods

Nodes can be removed in a three different methods; single, sequential or cascading.

* Single

At each iteration a single node is selected to be removed, with it replaced for the next iteration, resetting the network. Allows an examination to find the most critical node in the network.

* Sequential

At each iteration a single node is selected to be removed, leading to a repetitive process until the network becomes fully disconnected.

* Cascading

At the first iteration a node is removed, then at the following all nodes linked to the original which failed are removed, and then so on until the network becomes disconnected.

Nodes which fail can be selected in three different methods; random, degree and betweenness.

* Random

A node is selected at random from those in the network.

* Degree

The node with the greatest degree is selected. If there is a number of nodes with the joint highest, one of these is picked at random. The degree of nodes is recalculated at each time step.

* Betweenness

The node with the greatest betweenness centrality is selected. If there is a number of nodes with the joint highest, one of these is picked at random. The betweenness centrality of nodes is recalculated at each time step.

Network complexity

* Stand alone

Analysis of a single network.

* Dependency (B on A)

Analysis of a system where nodes in the second network depend on nodes in in the first.

* Interdependency

Analysis of a system where nodes in the second network depend on nodes in the first, and nodes in the first depend on nodes in the second.

## Failure options

### Sources

Allows a network, or set of networks, to be analysed where all nodes must be connected to a single source node, otherwise they are deemed inactive (failed). This is done through the use of the source\_nodes\_A and source\_nodes\_B parameters, which store a list of the node ID’s which are the source nodes. (Has not been tested with the cascading failure model).

# Variables

## Analysis

* Failure dictionary (failure *(dict)*)

|  |  |  |
| --- | --- | --- |
| **Key** | **Description** |  |
| stand\_alone (*boolean*) | Analysis of a single network | Failure type  *(Only one should be set as True, all others should be set as False).* |
| dependency (*boolean*) | Analysis of a pair of networks where B is dependent upon A |
| interdependency (*boolean*) | Analysis of a pair of networks where A is dependent upon B and B is dependent upon A. |
| single (*boolean*) | Each node is removed to see how this affects the networks functionality. | Failure method  *(Only one should be set as True, all others should be set as False).* |
| sequential (*boolean*) | Each node is removed one after the next until no nodes are left. |
| cascading (*boolean*) | All nodes connected to the node(s) removed on the pervious time step are removed. This is started by the selection of a single node. |
| random (*boolean*) | Selects the node to remove at random. Valid for all three failure methods. | Node selection method  *(Only one should be set as True, all others should beset as False)* |
| degree (*boolean*) | Picks the node with the highest degree to remove from the network. Valid for SEQUENTIAL and CASCADING failures. |
| betweenness (*boolean*) | Select the node with the highest betweenness centrality value to be removed. Valid for SEQUENTIAL and CASCADING. |

* Handling variables (handling\_variables *(dict)*)

|  |  |
| --- | --- |
| **Key** | **Description** |
| remove\_subgraphs (*boolean*) | Remove all the nodes and edges associated with all subgraphs, leaving only the giant component. |
| remove\_isolates (*boolean*) | Remove all isolates node in the same time step as they appear. |
| no\_isolates (*boolean*) | Allow isolated node to be removed through being selected, e.g. through the random removal strategy. |

* Graph parameters (graphparameters (*list*))

|  |  |
| --- | --- |
| **Parameters** | **Description** |
| to\_b\_nodes (*list*) | Stores the nodes in network B which are dependent on nodes in network A. Set as *None* if not needed. |
| from\_a\_nodes (*list*) | Stores the nodes in network A which nodes in network B are dependent on. Set as *None* if not needed. |
| source\_nodes\_A (*list*) | Lists the nodes which are chosen as the source nodes for network A. Set as *None* if not using source nodes. |
| source\_nodes\_B (*list*) | Lists the nodes which are chosen as the source nodes for network B. Set as *None* if not using source nodes. |
| Junctions\_A (*list*) | Stores ID’s of nodes which are junctions in network A. Set as *None* if not using junctions. |
| Junctions\_B (*list*) | Stores ID’s of nodes which are junctions in network B. Set as *None* if not using junctions. |

## Metrics

The metrics listed as being part of the basic metric list are compulsory when analysing a network and thus the status of these cannot be changed. For those listed under the optional title, these can be used as and when necessary to record the behaviour of the network(s) as they are perturbed. To use any of the optional metrics, they need to be set as True initially, with those not being used set as False. An example of the use of the optional metrics is given in section 2.3.

* Basic metrics (basicA or basic\_metrics\_A *(dict)*)

|  |  |
| --- | --- |
| **Key** | **Description** |
| nodes\_removed (*list of lists*) | Lists the nodes removed. |
| no\_of\_nodes\_removed (*list*) | Running total of the nodes removed. |
| no\_of\_nodes (*list*) | Running total of the node left in the network. |
| no\_of\_edges (*list*) | Running total of the nodes left in the network. |
| no\_of\_components (*list*) | The number of connected components in the network at each time step. |
| no\_of\_isolated\_nodes (*list*) | The number of isolated nodes at each step. |
| isolated\_nodes\_removed (*list of lists*) | Lists of isolated nodes removed. |
| nodes\_selected\_to\_fail (*list*) | Nodes selected to fail e.g., node with the greatest degree. |

* Optional metrics (optionsA or option\_metrics\_A *(dict)*)

|  |  |
| --- | --- |
| **Key** | **Description** |
| size\_of\_components (*list*) | The size in terms of the number of nodes of all the connected subgraphs in the network at each time step. |
| giant\_component\_size (*list*) | The size of the largest connected component at each time step. |
| avg\_size\_of\_components (*list*) | The average number of nodes across all the components at each time step. |
| isolated\_nodes (*list*) | All isolated nodes at each time step. |
| no\_of\_isolated\_nodes\_removed (*list*) | A running total of the number of nodes removed due to being isolated. Updated at every time step. |
| subnodes (*list*) | All those nodes which are not part of the giant component at every time step. |
| no\_of\_subnodes (*list*) | A count of the nodes which are not part of the giant component. |
| avg\_path\_length (*list*) | The average shortest path length of the network. |
| avg\_path\_length\_of\_components (*list of lists*) | The average shortest path length for each connected component. |
| avg\_path\_length\_of\_giant\_component (*list*) | The average shortest path length of the giant component. |
| avg\_geo\_path\_length (*list*) | The geographic average path length of the whole network. |
| avg\_geo\_path\_length\_of\_components (*list of lists*) | The geographic average shortest path length for each network component. |
| avg\_geo\_path\_length\_of\_giant\_component (*list*) | The geographic average shortest path length or the giant component of the network. |
| avg\_degree (*list*) | The average degree of the nodes in the network. |
| density (l*ist*) |  |
| maximum\_betweenness\_centrality (*list*) | The maximum value of betweenness centrality in the network for all nodes. |
| avg\_betweenness\_centrality (*list*) | The average value across all nodes for the betweenness centrality. |
| assortativity\_coefficient (*list*) | The assortativity coefficient for the network. |
| clustering\_coefficient (*list*) | The clustering coefficient for the network, averaged across all nodes. |
| transitivity (*list*) | The transitivity value for the network as an average across all nodes in the network. |
| square\_clustering (*list*) | The square clustering coefficient or the network. |
| avg\_neighbor\_degree (*list*) |  |
| avg\_degree\_connectivity (*list*) |  |
| avg\_degree\_centrality (*list*) |  |
| avg\_closeness\_centrality (*list*) |  |
| diameter (*list*) |  |

* Dependency metrics (dependency (*dict*))

|  |  |
| --- | --- |
| **Key** | **Description** |
| nodes\_removed\_from\_A | Records a list of the nodes removed from network A when a dependency edge from network B fails. |
| no\_of\_nodes\_removed\_from\_A | Counts the number of nodes removed when a dependency edge fails from network B to A. |
| nodes\_removed\_from\_B | Records a list of the nodes removed from network B when a dependency edge from network A fails. |
| no\_of\_nodes\_removed\_from\_B | Counts the number of nodes removed when a dependency edge fails from network A to B. |

* Cascading metrics (cascading (*dict*))

|  |  |
| --- | --- |
| **Key** | **Description** |
| TBC | TBC |
|  |  |
|  |  |

# Module functions

These are the functions which can be directly accessed in the resilience module by users.

|  |
| --- |
| main(*GA, GB, parameters, logfilepath, viewfailure = False*)  Returns: *complete (boolean)*  Controls the running of the analysis. Returns a single boolean variable on if the analysis was completed. |
|  |
| step(*graphparameters, parameters,metrics, iterate, logfilepath*)  Returns: *graphparameters, parameters, metrics,iterate*  Performs a single step of the analysis of the network(s), including the selection and removal of the nose(s) and associated edge(s) to the computation of the metrics. Returns two variables; the first containing the parameters and metrics, the second a boolean vale indicating the need to perform another iteration or not. |
|  |
| analysis\_B(*parameters, iterate, Gtemp, i, to\_a\_nodes, from\_b\_nodes, node\_list, basic\_metrics, option\_metrics, to\_b\_nodes, from\_a\_nodes, net)*  Returns: i*terate, Gtemp, i, to\_b\_nodes, from\_a\_nodes, a\_to\_b\_edges, node\_list, basic\_metrics, option\_metrics*  Controls the removal of subnodes and isolated nodes (if either are set as True as parameters), and computes the required metric values. Returns all those received. |
|  |
| metrics\_initial(*GnetA, GnetB, metrics, failure, handling\_variables, store\_n\_e\_atts, length, a\_to\_b\_edges*)  Returns: *networks*, metrics, graphparameters  Calculates the initial values for the metrics which are not set as False. |
|  |
| default\_parameters(*result\_file, analysis\_method = None, failure\_process s= None, selection\_method = None*)  Returns:  Creates a default set of parameters required for the analysis to run. Allows for the user to provide some variables which describe the analysis wanting to be run |
|  |
| Import\_modules(resil\_mod\_loc)  Returns:  Imports the resilience modules using the file path provided in the config.txt file for the resilience module. |

# Subsidiary Modules

These are the modules which the functions which the main resilience module relies upon to perform different aspects of the analysis, and as such are key to the module.

## Classes – Error Handling

Classes for handling errors which occur during any of the resilience modules.

|  |
| --- |
| GeneralError(*Exception*)  Non-descript error handler. |
|  |
| GraphError(*Exception*)  Error relating to a network not being suitable to compute the reselected function on. |
|  |
| OutputError(*Exception*)  Error in the processes related to the outputting the results. |
|  |
| CalculationError(*Exception*)  When a calculation cannot be performed for some reason. |
|  |
| SearchError(*Exception*)  Handles errors associated to not being able to find the request node or edge in the network. |
|  |
| WriteError(*Exception*)  Could not open or read or write to the text file. |

## Failure methods functions

Contains the functions which select the appropriate nodes for the failure model.

|  |
| --- |
| sequential\_degree(network*, INTERDEPENDENCY*)  Returns: *network, node\_removed*  Calculates the degree of all nodes in *G*, and then removes the node with the greatest value and its edges. Returns *G* and the node removed. |
|  |
| sequential\_betweenness(*network, INTERDEPENDENCY*)  Returns: *network, node\_removed*  Calculates the betweenness centrality of all nodes, removes the one with the highest value and it’s the edges. Returns *G* and the node removed. |
|  |
| sequential\_random(*network, NO\_ISOLATES, INTERDEPENDENCY)*  Returns: *network, node\_removed*  Picks a node at random from the network, removes it and its edges. Returns *G* and the node removed. |
|  |
| sequential\_from\_list(*network,INTERDEPENDENCY,fail\_list,i*)  Returns: *network, node\_removed*  Removes a defined node based on a list of nodes in the order of removal. |
|  |
| cascading\_failure(network*, dlist, dead,k,subnodes\_A, isolated\_nodes\_A, removed\_nodes, INTERDEPENDENCY*)  Returns: network*, dlist, removed\_nodes, node*  Facilitates the running of the cascading failure model. Removes the nodes of those which were connected to the node(s) removed on the previous time step (*dlist*). |
|  |
| single\_random(network*, node\_list, INTERDEPENDENCY*)  Returns: *network, node\_removed*  Selects a node at random from *node\_list* and removes it from *G* along with its edges. Removes it from node\_list as well, which contains the nodes which have still not been removed. Returns an edited network and the node removed. |
|  |
| geo\_failure(network*, shp\_file*)  Returns: TBC  Identify the nodes and edges which fall within the extents of the shape file. |
|  |
| point\_in\_poly(*coord,poly*)  Returns: *inside* |

## Network handling functions

Functions with reference to the editing of the networks, including the removal of nodes and edges, subgraphs as well as other methods.

|  |
| --- |
| remove\_isolates(*Gtemp,node\_list,isolated\_nodes,isolated\_n\_count\_removed,node\_count\_removed,to\_b\_nodes,from\_a\_nodes*)  Returns: *G, node\_list, basic,option, isolatednodes, to\_b\_nodes, from\_a\_nodes, a\_to\_b\_edges*  Handles the removal of isolated nodes, adjusting the associated lists including the *node\_list*. |
|  |
| clean\_node\_lists(*subn,node\_list, to\_b\_nodes, from\_a\_nodes)*  Returns: *node\_list, to\_b\_nodes, from\_a\_nodes*  Clean the suite of node lists. Removes from the lists those nodes removed from the network due to being part of subgraphs. Returns the corrected lists. |
|  |
| handle\_sub\_graphs(*nodelists, edgelists*)  Returns: *G, subnodes, numofsubnodes, nodes, edges*  Removes subgraphs form the network if required. The network is built from the node and edge lists before doing so. Returns the network, a list and count of the nodes removed and the new node and edge lists. |
|  |
| check\_dependency\_edges(*networks,node,basic\_metrics\_A,basic\_metrics\_B,option\_metrics\_A,option\_metrics\_B,to\_b\_nodes,from\_a\_nodes,temp*)  Returns: *args(networks, nodes\_removed\_from\_b, basicA, basicB, optionA, optionB, to\_b\_nodes, from\_a\_nodes, a\_to\_b\_edges)*  Checks the dependency edges to see if any of them have been broken by the removal of nodes from the network. |
|  |
| check\_node\_removed(*node, subnodes, isolated\_nodes*)  Returns: *node\_removed*  Checks if a node has been removed from the network before trying to remove it. Returns boolean variable. |
|  |
| whole\_graph\_av\_path\_length(*Gtemp,length=''*)  Returns: *average* |

## Tools functions

This module contains functions which are commonly used and perform tasks.

|  |
| --- |
| write\_to\_log\_file(*logfileapth, text)*  Returns: NA  Writes to a file given a file path and a string of text. Also adds the time to every entry. If the filepath is set to *None(default),* nothing will be written. |
|  |
| max\_val\_random*(list)*  Returns: *max\_value, node*  Returns the maximum value in the list and the position in the list (or the node if applicable). If the maximum value is shared between more than one node (list entry), one of those is picked at random. |
|  |
| replace\_all(*text, dic*)  Returns: *text*  Given a string and a dict of characters, will where found, replace them. Returns a string. |
|  |
| failure\_type(*SINGLE, SEQUENTIAL, CASCADING, RANDOM, DEGREE, BETWEENNESS*)  Returns: *failuretype\_string*  Creates string which summarises the analysis being undertaken. |
|  |
| get\_nodes\_edges\_csv(*location*)  Returns: *node\_list, edge\_list*  Reads a csv file where the edges are listed on the top line and the nodes in the second line. Creates a list of nodes and edges which can be used to build a network from. |
|  |
| add\_node\_field(G,field\_name,data=None)  Returns: *network*  Add a field to the nodes in the network. |
|  |
| add\_edge\_field(G,field\_name,data=None)  Returns: *network*  Add a field to the edges in the network. |
|  |
| analyse\_existing\_networks(NETWORK\_NAME, conn, db, parameters, noioa, use\_db, use\_csv, logfilepath, nx\_location)  Returns: NA |

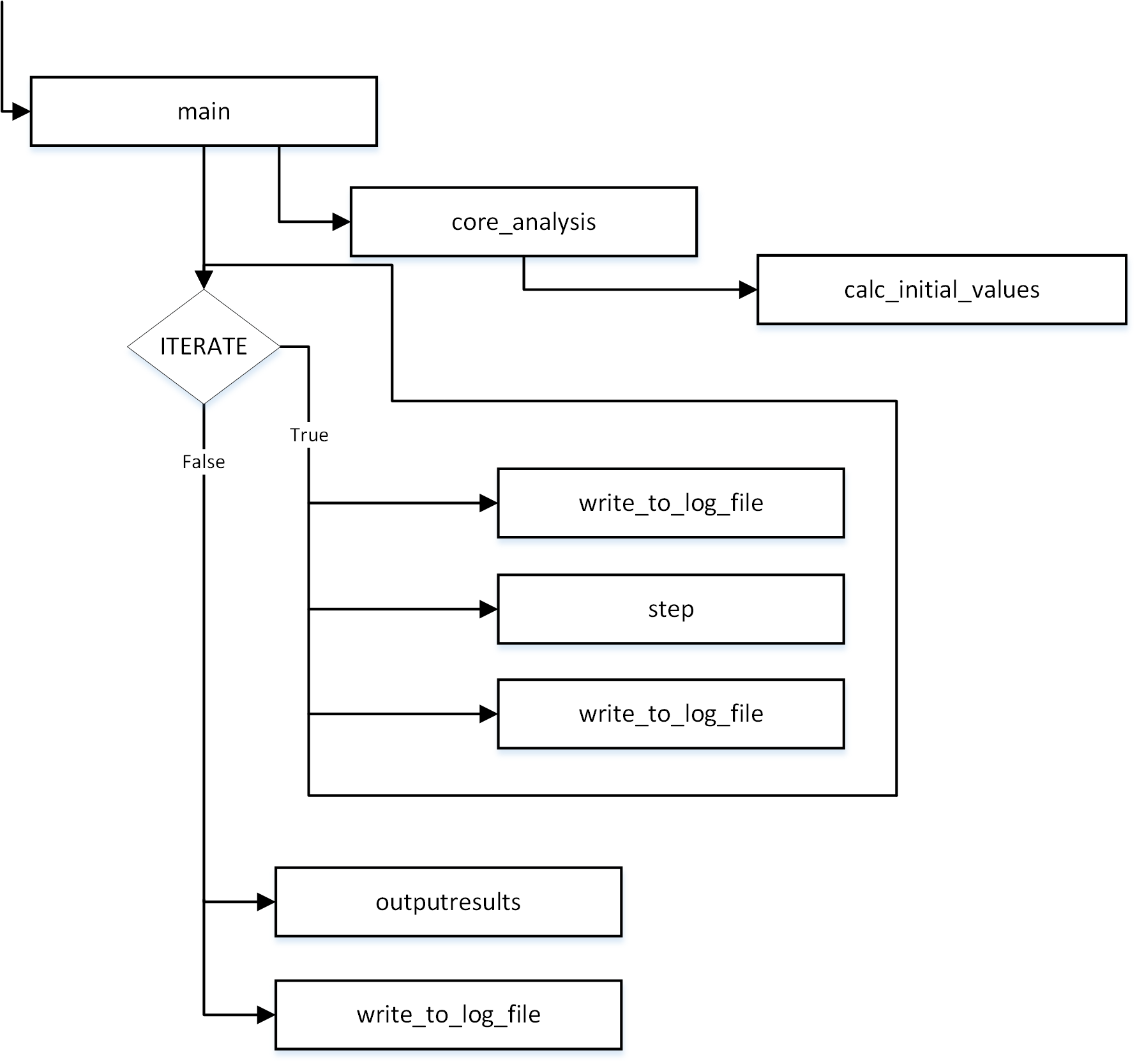
## Outputs function

Functions related to outputting (saving) the results to a text file.

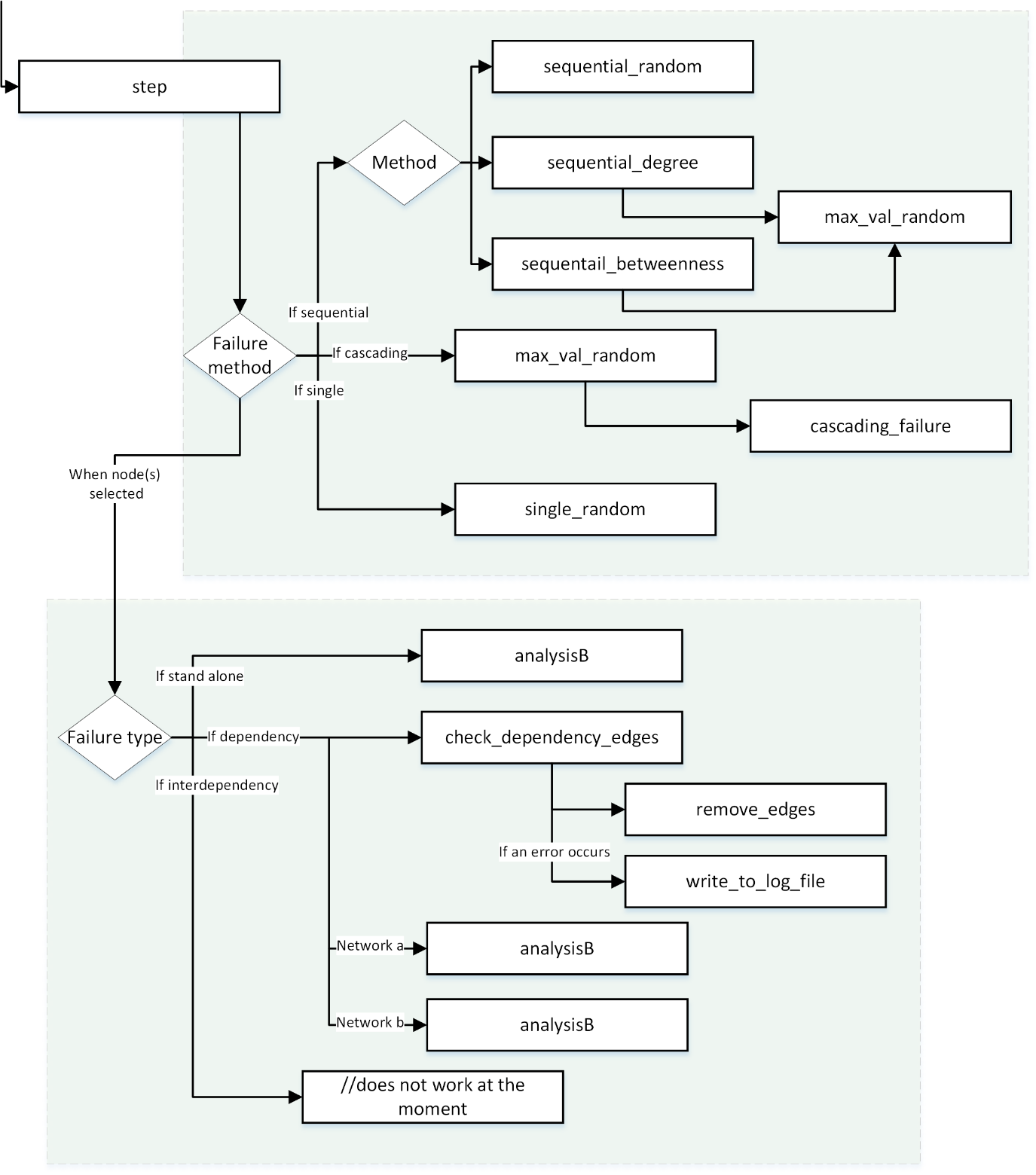
|  |
| --- |
| outputresults(*graphparameters, parameters,logfilepath=None,multiiterations=None*)  Returns: *metric\_values, error*  Control function for the output of the results. Returns the values of those metrics written to the text file. |
|  |
| average\_txtresults(*graphparameters, parameters,error*)  Returns: *basicA, basicB, optionA, optionB, error*  Reads in the text file with the results from the simulations and collates all the results together to produce an average for each metric at each time step. Returns the averages for each metric used. |
|  |
| results(*basic\_metrics\_A, basic\_metrics\_B*)  Returns: NA  Prints the results into the console window. |
|  |
| write\_text\_file(*outputfile,CASCADING,basic\_metrics,option\_metrics*)  Returns: NA  Write the metric values to the results text file. |
|  |
| txtout(*outputfile,graphparameters, parameters*)  Returns: NA  Write the parameters for the simulation(s) to the results text file. |
|  |
| write\_to\_db(*networks, a\_to\_b\_edges, failure, db\_parameters, i*)  Returns: NA  Save networks to database. |
|  |
| write\_results\_table(*metrics, i, failure, db\_parameters, k*)  Returns: NA  Write metrics to results table. |
|  |
| create\_db\_res\_table(*conn, table\_name, option, dependency, cascading, net*)  Returns: NA  Create table in database to store results from a network during failure analysis. |
|  |
| rename\_db\_table(*conn, table\_name\_old, table\_name\_new*)  Returns: NA  Rename a table in the database. |

# Key function diagrams

## main function



## step function



## analysis\_B function

