BooleanNetworksOfGRN

Location: `https://github.com/ClaudiaWinklmayr/BooleanNetworksOfGRN`

Dependencies:

- Graph generation: `networkx` (`https://networkx.github.io/`)
- Information theory: `dit` (`http://docs.dit.io`)
- Unique information: `computeUI` (`https://github.com/infodeco/computeUI`)

The Boolean Networks Class:

The `BooleanNetwork` class can be used to generate random Boolean networks or read in predefined network files. The main functionality of this class is the ability to manipulate single nodes (e.g fix their values, change update functions), perform updates on the entire network, find limit cycles and fixed point and display the network structure. This is the base class for all further analysis. A couple of important functions are listed below, for further information refer to the code documentation in `https://github.com/ClaudiaWinklmayr/BooleanNetworksOfGRN`. The `BooleanNetwork` class builds upon the `node` class, which is a collection of properties specifying a node in a Boolean network. These properties are listed in the box below:

The `node` class:	
`ID`	node label (string or int)
`Value`	current value (float)
`InputIDs`	the node's parents (list of node IDs)
`UpdateType`	flag for type of update function ('dict', 'function' or 'list)
`UpdateFunc`	The node's state as a function of it's parents dictionary - { input : output} lambda function - `lambda` input: output

The Boolean Network class provides several methods to create Boolean networks:

- read in a predefined Boolean network form a `.json` file
- build a sparse random network following the distributions presented in [2]
- build a random network of `N` nodes each of which has `k` randomly chosen inputs, where k is either the same for every node or drawn from a normal distribution $\mathcal{N}(k, \sigma_k)$
- create a Boolean network from a predefined `networkx` graph

If the Boolean functions are nor specified directly, e.g from a parameter or a network file, they will typically be generated at random. One can specify the p-Bias i.e. the probability that a Boolean functions outputs a 1 (this will affect the network's stability) and explicitly exclude functions that do not depend on all their inputs and specify specific update functions to root nodes.

Main properties

- `nodeDict:` dictionary where the keys are the node labels and the values are node objects from the `node` class
- `CurrentValues:` dictionary where the keys are the node labels and the values are the nodes values
- `Graph`: a networkx graph that represents the connections between the nodes and can be used for drawing

Main methods:

- `update_all(FixedNodes={})`: updates all nodes synchronously from the current value using the `node.update` function of the node class. The `FixedNodes` dictionary can be used to set some of the nodes to a defined value, they will be ignored in the update procedure
- `calc_all_updates(rounds=1,FixedNodes={},cheap=False)`: uses the `update_all` function to synchronously update all nodes from all possible input states. Returns a list of all input and output states. If rounds is set to a value greater than 1, several update steps can be performed at once.
- `update_until_cycle(FixedNodes={},verbose=True)`: starting form the networks current state the network is updated until it reaches a fied point or limit cycle
- `draw(RemoveNodes=[],positions={})`: draws the Boolean network. If not specified otherwise, nodes are placed on a circle with radius 1.

Boolean network functions
The python module `boolean_network_functions` contains numerous methods to calculate different measures of Boolean networks. In the following we list the most important functions, for further information and examples see the ExampleNotebook.ipynb at `https://github.com/ClaudiaWinklmayr/BooleanNetworks`.

- `scan_state_space(NetFixedNodes={},StartNodes={},display=True)`: finds all fixed points and limit cycles in the state space
- `NetworkCanalyzing(Net)`: finds out how many of the networks update functions are canalyzing.
- `Sensitivity`: calculates the Sensitivity of a Boolean function
- `find_basins_of_attraction(Net)`: uses the `scan_phase_space()` function to calculate the attractors of the network. For each attractor state the incoming states are recursively traced backwards to identify each attractors complete basin of attraction. This function can be used to calculate the *Basin entropy* of a network.
- `find_clusters(Net)`: Identifies clusters in a Boolean network using the approach described in the previous chapter.
- `bnf.calc_mutual_information_all_pairs(Net)`: calculates mutual information between a nodes X at time t and a node Y at time $t+1$ for all pairs of nodes independent of whether they are connected in the network graph
- `info_flow_in_tuples_and_triplets(Net,2)`: Calculates the information flow within pairs and triplets of nodes using mutual information and information decomposition for triplets
- `info_decomposition_clusters(Net,inputs1,inputs2,outputs)`: calculates information decomposition between groups of nodes by first generating the joint distribution of the groups.

The model files: The folder `model_files` contains predefined Boolean models in `.json` format. Apart from simple but frequently used types of networks such as Fan-Ins or Feed-Forward-Loops, these modesl files also include implementations of Boolean models from the literature. The references for these models are: `lacNet` & `lacNetTemporal` [5], `lacNetTemporal` ([3]), `Drosophila` ([4]), `YeastCellCycle` ([1]).

Bibliography

[1] Maria I Davidich and Stefan Bornholdt. Boolean network model predicts cell cycle sequence of fission yeast. *PloS one*, 3(2), 2008.

[2] Barbara Drossel. Random boolean networks. *Reviews of nonlinear dynamics and complexity*, 1:69–110, 2008.

[3] Franziska Hinkelmann and Reinhard Laubenbacher. Boolean models of bistable biological systems. *arXiv preprint arXiv:0912.2089*, 2009.

[4] Assieh Saadatpour and Réka Albert. Boolean modeling of biological regulatory networks: a methodology tutorial. *Methods*, 62(1):3–12, 2013.

[5] Alan Veliz-Cuba and Brandilyn Stigler. Boolean models can explain bistability in the lac operon. *Journal of computational biology*, 18(6):783–794, 2011.