DynamiX Documentation

Release 0.1

Pascal Held

July 01, 2014

Contents

Contents:

CHAPTER 1

Introduction

1.1 Recources

- Git - Repository: http://bitbucket.org/paheld/dynamix
- Documentation: http://dynamix.readthedocs.org

1.2 Requirements

- SciPy
- NetworkX
- Matplotlib
- twitter

CHAPTER 2

Event Management

This modules provides methods for event handling.

Every event is an dictionary in the following form:

```
event = {
    "time": [a float value],
    "sender": [a single sender (int or str) or a list of senders],
    "receiver": [a single receiver (int or str) or a list of receivers] (optional),
}
```

2.1 Tools

This file provides tools for event handling.

class `dynamix.events.tools.`**`MakeID`**

Converter object for sender/receiver labels.

Methods

`make_id`(*generator*)

Replaces all sender and receiver values by integer keys.

Parameters **generator** : iterable or single event

Events

`dynamix.events.tools.`**`event_to_string`**(*event*)

Converts the Event-Dictionary into a string representation.

Parameters **event** : dict

Event in dict representation

Returns **line** : str

String representation of the event

Notes

Example:

```
>>> event_to_string({"time":1,"sender":[1,2,3],"receiver":[4,5,6]})
'1;1,2,3;4,5,6'

>>> event_to_string({"time":1,"sender":[1,2,3]})
'1;1,2,3'

>>> event_to_string({"time":1,"sender":1,"receiver":2})
'1;1;2'
```

`dynamix.events.tools.`**`jitter_equal`**(*generator*, *interval=(-1, 1)*)

Adds noise to the event time.

This functions adds an equal distributed noise in the given interval to the time of the events.

Parameters **generator** : iterable or single event

Events

interval : tuple or list

2 element tuple with min and max jitter

Notes

Events could be out of order after adding jitter.

`dynamix.events.tools.`**`load`**(*filename*)

Loads events from file

Parameters **filename** : str

Filename which is used to load the data.

`dynamix.events.tools.`**`make_lists`**(*event*)

Converts all sender and receiver literals into lists

`dynamix.events.tools.`**`make_literals`**(*event*)

Converts all sender and receiver lists into literals if they contain only one element

`dynamix.events.tools.`**`random`**() → x in the interval [0, 1).

`dynamix.events.tools.`**`save`**(*generator*, *filename*)

Saves events to file

Parameters **generator** : iterable or single event

Events

filename : str

Filename which is used to save the file.

`dynamix.events.tools.`**`simplify`**(*generator*)

Simplifies the compact representation with multiple sender and receivers.

The functions uses an iterator of events and creates for every sender-receiver combination a new event.

Parameters **generator** : iterable or single event

Events

`dynamix.events.tools.`**`sort`**(*generator*, *window_size=5.0*)

Sorts events in an event stream.

Elements of an generator will be sorted in the given time window size.

Parameters **generator** : iterable or single event

Events

window_size : int or float

time window size

Notes

Events would be delayed until next event with timedelta > window_size arrives.

`dynamix.events.tools.`**`string_to_event`**`(`*string*`)`

Converts the string representation of an event into the dictionary representation.

Parameters **string** : str

Event in string representation

Returns **event** : dict

Dictionary representation of the event

Notes

Example:

```
>>> event = string_to_event('1;1,2,3;4,5,6')
>>> event ==  {'time': 1, 'sender': [1, 2, 3], 'receiver': [4, 5, 6]}
True

>>> event = string_to_event('1;1,2,3')
>>> event == {'time': 1, 'sender': [1, 2, 3]}
True

>>> event = string_to_event('1;1;2')
>>> event == {'time': 1, 'sender': 1, 'receiver': 2}
True
```

`dynamix.events.tools.`**`throttle`**`(`*generator*, *factor=1.0*`)`

Throttles event processing

The events will be repressed until the passing time from the first event to the current one is at least factor times the event time between the first event and the current one.

Parameters **generator** : iterable or single event

Events

factor : float

factor of time delay, default 1.0

Notes

The time will be interpreted as seconds.

2.2 Twitter - Stream

Barabasi Graph Operations

3.1 Barabasi Tools

`dynamix.generators.barabasi.tools.`**`add_nodes_to_barabasi_graph`**(*g*, *nodes_to_add*, *m*)

Adds nodes to existing graph of the Barabási-Albert using the preferential attachment model.

Graph g is grown by attaching new nodes each with m edges that are preferentially attached to existing nodes with high degree.

Parameters **g** : Graph

Graph where nodes will be added

nodes_to_add : List of nodelabels

nodes which will be added to graph g

m : int

number of edges per new node

Returns **g** : Graph

Notes

Nodes will be added in order of the list "nodes_to_add".

`dynamix.generators.barabasi.tools.`**`barabasi_graph`**(*nodes_prioritylist*, *m*)

Return random graph using Barabási-Albert preferential attachment model.

A graph of n nodes is grown by attaching new nodes each with m edges that are preferentially attached to existing nodes with high degree. Nodes are added in the same order as in nodes_prioritylist.

Parameters **nodes_prioritylist** : List, string/int

Number of nodes

m : int

Number of edges to attach from a new node to existing nodes

Returns **g** : Graph

Notes

The initialization is a full connected graph with with m nodes.

References

[R1]

`dynamix.generators.barabasi.tools.`**`check_power_law`**(*g*)

Estimates as power law fit and returns the exponent and the rmse

Uses the degree distribution to estimate the exponent a from P(k) ~ k ^ a

Parameters **graph** : Graph

Graph to analyse

Returns exponent: double

Exponent of function fit.

RMSE: double

Root mean squared error

`dynamix.generators.barabasi.tools.`**`compare_merge`**(*graph*, *subgraph_1*, *subgraph_2*)

Calls calculation of rank correlation coefficients and edge similarity measure.

Compares attributes of graph g1, g2 and their merged graph g.

Parameters **g1** : Subgraph 1 of g

nodes which will be added to graph g

g2 : Subgraph 2 of g

number of edges per new node

merged_graph : Graph

Graph where nodes will be added

Notes

Requirement: merged_graph.nodes() = g1.nodes + g2.nodes()

`dynamix.generators.barabasi.tools.`**`degree_rank_correlation`**(*graph*, *subgraph_1*, *subgraph_2*)

Calculates rank correlation coefficients.

Calculates Spearmans r (with tie correction) and Kendalls tau for the node degree distribution of g1, g2 and merged_graph

Parameters **g** : Graph

Initial graph

g1 : Graph

Subgraph 1 of g

g2 : Graph

Subgraph 2 of g

Notes

Requirement: g1.nodes + g2.nodes() = gn.nodes()

`dynamix.generators.barabasi.tools.`**`edge_similarity`**`(`*graph*, *subgraph_1*, *subgraph_2*`)`

Calculates Edge simlarity measures for graph and its subgraphs

Compares the number of edges represented in a graph and its subgraphs. Outputs the full similarity matrix for both subgraphs.

Parameters **graph** : Graph

Initial graph

subgraph1 : Graph

Subgraph 1 of g

subgraph2 : Graph

Subgraph 2 of g

Returns **similarity_matrix** : float

Notes

	G	
G'	a	b
	c	d

val	Desciption	index
a	in G and G'	[0,0]
b	only in G'	[0,1]
c	only in G	[1,0]
d	not present in G and G'	[1,1]

`dynamix.generators.barabasi.tools.`**`estimate_m`**`(`*g*, *g2=None*, *model_correction=True*`)`

Estimates parameter m for graphs following the Barabási-Albert model.

Estimates the parameter m by counting edges and vertices of graph g. Additionally estimates a shared m if two graphs are used as input parameters.

Parameters **g** : Graph

Graph for estimating m

g2 : Graph, optional

Second graph which will be included in the estimation for a shared m (default = None)

model_correction : boolean, optional

Was a model_correction used for the creation of graph g and g2. (default = True) True = Barabási-Albert model starting with a complete graph of m nodes False = Barabási-Albert model starting with an empty graph of m nodes

Returns **m** : int

Notes

Estimate will be influenced by nodes not following the Barabási-Albert model e.g. outliers connected to every node in the graph.

`dynamix.generators.barabasi.tools.`**`get_degree_distribution`**(*number_of_edges*, *number_of_nodes=None*, *exponent=-2.9*, *min_degree=1*, *max_degree=None*)

Estimates the degree distribution

Calculate estimated numbers ob nodes with the given node degree with P(k) ~ k ** exponent

Parameters **number_of_edges** : Integer

The number of estimated edges

number_of_nodes : Integer, optional

The number of nodes in the graph. If not given number_of_nodes = number_of_edges + 1. It is only used to get the maximum node degree

exponent : float, optional

The exponent used for the estimation. If not given -2.9 is used, as given in Barabasis Paper

Returns Dict

Estimated node degree distribution

`dynamix.generators.barabasi.tools.`**`get_repeated_node_list`**(*g*)

Returns a list which contains every node repeated by the number of his degree.

Parameters **g** : Graph

Graph from which the repeated_node_list should be created from

Returns **_** : list

`dynamix.generators.barabasi.tools.`**`random_subset`**(*seq*, *m*)

Return m unique elements from seq.

Parameters **seq** : list

Nodelabels

m : int

number of nodes to be returned

Returns **targets** : list

Notes

This differs from random.sample which can return repeated elements if seq holds repeated elements.

Elements of the returned list are in order! Do not use if random order ist desired.

`dynamix.generators.barabasi.tools.`**`repair_graph`**(*g*, *m*)

Try to repair a given graph so it is connected and follows the barabasi model.

Three step process 1) connect unconnected components 2) add edges for nodes with a degree less than m 3) add additional edges till g.edges = (g.node-m)*m + 0.5*m*(m-1)

Parameters **g** : Graph

Graph which will be splitted

m : int

Minimal number of edges per node

Notes

The given graph will be fixed inplace. No return needed.

`dynamix.generators.barabasi.tools.`**`sort_nodes_by_degree`**`(`*g1*, *g2=None*`)`

Returns a nodelist sorted by node degree in decreasing order

Parameters **g1** : Graph

Graph 1

g2 : Graph, optional

Graph 2

Returns **nodes** : list

nodelist sorted by node degree in decreasing order

3.2 Barabasi Merge Operations

`dynamix.generators.barabasi.merge.`**`minimal_merge`**`(`*g1*, *g2*`)`

Connects both graphs with minimal amount of edges

Adds a minimal amount of edges to connect g1 and g2. Therefore choose one node per graph by preferential attachment strategy and connect both.

Parameters **g1** : Graph

Graph 1

g2 : Graph

Graph 2

Returns **g** : Graph

merged Graph

`dynamix.generators.barabasi.merge.`**`node_degree_merge`**`(`*g1*, *g2*`)`

Creates merged graph following the Barabási-Albert model by adding nodes in order of their node-degree.

Merged graph g is grown by attaching the nodes of g1 and g2 in order of their node degree each with m edges that are preferentially attached to existing nodes with high degree.

Parameters **g1** : Graph

Graph where nodes will be added

g2 : Graph

nodes which will be added to graph g

Returns **g** : Graph

merged Graph

`dynamix.generators.barabasi.merge.`**`preserving_nodes_merge`**`(`*g1*, *g2*, *add_in_order=True*`)`

Creates merged graph following the Barabási-Albert model by adding nodes of g2 to graph 1.

Merged graph g is grown by attaching the nodes of g2 to the graph g1. The estimated m of g1 will be used for the preferential attachment process. This will result in a minimal change of g1.

Nodes of g2 are either added in order of their node degree (highest first) or in random order.

Parameters **g1** : Graph

Graph will be used as basis for merged graph

g2 : Graph

Nodes of Graph 2 will be added to g1

add_in_order: boolean, optional

Should nodes be sorted befor adding them to graph g1? (default = True) True: nodes of g2 will be added in order of their node degree False: nodes of g2 will be added in random order

Returns **g** : Graph

merged Graph

`dynamix.generators.barabasi.merge.`**`random_merge`**`(`*g1*, *g2*`)`

Creates merged graph following the Barabási-Albert model by adding nodes in random order.

Merged graph g is grown by attaching the nodes of g1 and g2 in random order each with m edges that are preferentially attached to existing nodes with high degree.

Parameters **g1** : Graph

Graph where nodes will be added

g2 : Graph

nodes which will be added to graph g

Returns **g** : Graph

merged Graph

3.3 Barabasi Divide Operations

`dynamix.generators.barabasi.divide.`**`maximum_cut_split`**`(`*g*, *n1*`)`

not documented yet

Parameters **g** : Graph

Graph which will be splitted

n1 : int

Size of Subgraph 1

Returns **g1** : Graph

Subgraph 1 of g

g2 : Graph

Subgraph 2 of g

Notes

n2 = len(g.nodes()) - n1

`dynamix.generators.barabasi.divide.`**`node_degree_divide_a`**`(`*g*, *n1*`)`

Splits the set of nodes by their node degree order and creates two new subgraphs following the Barabási-Albert model.

Strategy A: Chooses n1 highest ranked nodes (by node degree) of graph g to create a new graph using the Barabási-Albert model Nodes will be added in order of their rank. All other nodes will be used for the creation of the second subgraph following the same procedure.

Parameters **g** : Graph

Graph which will be splitted

n1 : int

Number of nodes used for subgraph 1

Returns **g1** : Graph

Subgraph 1 of g with n1 nodes

g2 : Graph

Subgraph 2 of g with len(g.nodes())-n1 nodes

Notes

n2 = len(g.nodes()) - n1

`dynamix.generators.barabasi.divide.`**`node_degree_divide_b`**`(`*g*, *n1*`)`

Splits the set of nodes by their node degree order and creates two new subgraphs following the Barabási-Albert model.

Strategy B: Sorts nodes in order of their node degree. Chooses nodes of n1 and n2 in alternating order to create both subgraphs using the Barabási-Albert model. Nodes will be added in order of their rank.

Parameters **g** : Graph

Graph which will be splitted

n1 : int

Number of nodes used for subgraph 1

Returns **g1** : Graph

Subgraph 1 of g with n1 nodes

g2 : Graph

Subgraph 2 of g with len(g.nodes())-n1 nodes

Notes

n2 = len(g.nodes()) - n1

`dynamix.generators.barabasi.divide.`**`random_divide`**(*g*, *n1*)

Splits the set of nodes randomly and creates two new subgraphs following the Barabási-Albert model.

Randomly chooses n1 nodes of graph g to create a new graph using the Barabási-Albert model. All other nodes will be used for the creation of the second subgraph following the same procedure.

Parameters **g** : Graph

Graph which will be splitted

n1 : int

Number of nodes used for subgraph 1

Returns **g1** : Graph

Subgraph 1 of g with n1 nodes

g2 : Graph

Subgraph 2 of g with len(g.nodes())-n1 nodes

Notes

n2 = len(g.nodes()) - n1

`dynamix.generators.barabasi.divide.`**`random_subgraph_divide`**(*g*, *n1*)

Splits the set of nodes randomly, preserves edges of the two groups and creates two new subgraphs following the Barabási-Albert model.

Randomly chooses n1 nodes of graph g to create a new graph using the Barabási-Albert model through adding edges to the induced subgraph. All other nodes and their intra-group-edges will be used for the creation of the second subgraph following the same procedure. The repair-operator is used for completing the graphs.

Parameters **g** : Graph

Graph which will be splitted

n1 : int

Number of nodes used for subgraph 1

Returns **g1** : Graph

Subgraph 1 of g with n1 nodes

g2 : Graph

Subgraph 2 of g with len(g.nodes())-n1 nodes

Notes

n2 = len(g.nodes()) - n1

`dynamix.generators.barabasi.divide.`**`subgraph_expansion_divide`**(*g*, *n1*)

Creates a new graph by choosing the node with the lowest node degree and iteratively expanding the subgraph using the neighborhood of the current subgraph.

Takes the node with the lowest node degree as starting point for the subgraph creation. Iteratively extends the subgraph by adding the neighbors of the last added node to the subgraph. Repeat this process till subgraph 1 consists of n1 nodes. All nodes not added to subgraph 1 will be used to create subgraph 2

Both graphs need to be repaired afterwards.

Parameters **g** : Graph

Graph which will be splitted

n1 : int

Number of nodes used for subgraph 1

Returns **g1** : Graph

Subgraph 1 of g with n1 nodes

g2 : Graph

Subgraph 2 of g with len(g.nodes())-n1 nodes

Notes

n2 = len(g.nodes()) - n1

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*

Bibliography

[R1] A. L. Barabási and R. Albert “Emergence of scaling in random networks”, Science 286, pp 509-512, 1999.