LineageTree

Leo Guignard

Mar 10, 2020

CONTENTS:

CHAPTER

ONE

MODULES

class LineageTree.**lineageTree**(*file_format=None*, *tb=None*, *te=None*, *z_mult=1.0*, *file_type=''*, *delim=', '*, *eigen=False*)

__init__(*file_format=None*, *tb=None*, *te=None*, *z_mult=1.0*, *file_type=''*, *delim=', '*, *eigen=False*)

Main library to build tree graph representation of lineage tree data It can read TGMM, ASTEC, SVF, MaMuT and TrackMate outputs.

Parameters

- **file_format** (*strr*) – either - path format to TGMM xmls - path to the MaMuT xml - path to the binary file
- **tb** (*int*) – first time point (necessary for TGMM xmls only)
- **te** (*int*) – last time point (necessary for TGMM xmls only)
- **z_mult** (*float*) – z aspect ratio if necessary (usually only for TGMM xmls)
- **file_type** (*str*) – type of input file. Accepts: 'TGMM, 'ASTEC', MaMuT', 'TrackMate', 'csv', 'celegans', 'binary' default is 'binary'

add_node(*t=None*, *succ=None*, *pos=None*, *id=None*, *reverse=False*)

Adds a node to the lineageTree and update it accordingly.

Parameters

- **t** (*int*) – int, time to which to add the node
- **succ** (*int*) – id of the node the new node is a successor to
- **pos** (*[float,]*) – list of three floats representing the 3D spatial position of the node
- **id** (*int*) – id value of the new node, to be used carefully, if None is provided the new id is automatically computed.
- **reverse** (*bool*) – True if in this lineageTree the predecessors are the successors and reciprocally. This is there for bacward compatibility, should be left at False.

Returns id of the new node.

Return type int

compute_k_nearest_neighbours(*k=10*)

Computes the k-nearest neighbors Writes the output in the attribute *kn_graph* and returns it.

Parameters **k** (*float*) – number of nearest neighours

Returns

dictionary that maps a cell id to its *k* nearest neighbors

Return type {int, set([int, ..])}

compute_spatial_density(*t_b=None*, *t_e=None*, *th=50*)
Computes the spatial density of cells between *t_b* and *t_e*. The spatial density is computed as follow: #cell/(4/3*pi*th^3) The results is stored in self.spatial_density is returned.

Parameters

- **t_b** (*int*) – starting time to look at, default first time point
- **t_e** (*int*) – ending time to look at, default last time point
- **th** (*float*) – size of the neighbourhood

Returns dictionary that maps a cell id to its spatial density

Return type {int, float}

compute_spatial_edges(*th=50*)
Computes the neighbors at a distance *th* Writes the output in the attribute *th_edge* and returns it.

Parameters **th** (*float*) – distance to consider neighbors

Returns

dictionary that maps a cell id to its neighbors at a distance *th*

Return type {int, set([int, ..])}

fuse_nodes(*c1*, *c2*)
Fuses together two nodes that belong to the same time point and update the lineageTree accordingly.

Parameters

- **c1** (*int*) – id of the first node to fuse
- **c2** (*int*) – id of the second node to fuse

get_cycle(*x*, *depth=None*, *depth_pred=None*, *depth_succ=None*)
Computes the predecessors and successors of the node *x* up to *depth_pred* predecessors plus *depth_succ* successors. If the value *depth* is provided and not None, *depth_pred* and *depth_succ* are overwrited by *depth*. The ordered list of ids is returned.

Parameters

- **x** (*int*) – id of the node to compute
- **depth** (*int*) – maximum number of predecessors and successor to return
- **depth_pred** (*int*) – maximum number of predecessors to return
- **depth_succ** (*int*) – maximum number of successors to return

Returns list of ids

Return type [int,]

get_gabriel_graph(*t*)
Build the Gabriel graph of the given graph for time point *t* The Garbiel graph is then stored in self.Gabriel_graph and returned *WARNING: the graph is not recomputed if already computed. even if nodes were added*.

Parameters **t** (*int*) – time

Returns

a dictionary that maps a node to the set of its neighbors

Return type {int, set([int,])}

get_idx3d(*t*)
Get a 3d kdtree for the dataset at time *t* . The kdtree is stored in *self.kdtrees[t]*

Parameters **t** (*int*) – time

Returns

the built kdtree and the correspondancy list, If the query in the kdtree gives you the value i, then it corresponds to the id in the tree to_check_self[i]

Return type (kdtree, [int,])

get_next_id()
Computes the next authorized id.

Returns next authorized id

Return type int

get_predecessors(*x*, *depth=None*)
Computes the predecessors of the node *x* up to *depth* predecessors. The ordered list of ids is returned.

Parameters

- **x** (*int*) – id of the node to compute
- **depth** (*int*) – maximum number of predecessors to return

Returns list of ids, the last id is *x*

Return type [int,]

get_sub_tree(*x*, *preorder=False*)
Computes the list of cells from the subtree spawned by *x* The default output order is breadth first traversal. Unless preorder is *True* in that case the order is Depth first traversal preordered.

Parameters

- **x** (*int*) – id of root node
- **preorder** (*bool*) – if True the output is preorder DFT

Returns the ordered list of node ids

Return type ([int, ..])

get_successors(*x*, *depth=None*)
Computes the successors of the node *x* up to *depth* successors. The ordered list of ids is returned.

Parameters

- **x** (*int*) – id of the node to compute
- **depth** (*int*) – maximum number of predecessors to return

Returns list of ids, the first id is *x*

Return type [int,]

read_from_ASTEC(*file_path*, *eigen=False*)
Read an *xml* or *pkl* file produced by the ASTEC algorithm.

Parameters

- **file_path** (*str*) – path to an output generated by ASTEC
- **eigen** (*bool*) – whether or not to read the eigen values, default False

read_from_binary (*fname*, *reverse_time=False*)
Reads a binary lineageTree file name. Format description: see self.to_binary

Parameters

- **fname** – string, path to the binary file
- **reverse_time** – bool, not used

read_from_csv (*file_path*, *z_mult*, *link=1*, *delim=', '*)

read_from_mamut_xml (*path*)
Read a lineage tree from a MaMuT xml.

Parameters **path** (*str*) – path to the MaMut xml

read_from_txt_for_celegans (*file*)

read_tgmm_xml (*file_format*, *tb*, *te*, *z_mult=1.0*)
Reads a lineage tree from TGMM xml output.

Parameters

- **file_format** (*str*) – path to the xmls location. it should be written as follow: path/to/xml/standard_name_t{t:06d}.xml where (as an example) {t:06d} means a series of 6 digits representing the time and if the time values is smaller that 6 digits, the missing digits are filed with 0s
- **tb** (*int*) – first time point to read
- **te** (*int*) – last time point to read
- **z_mult** (*float*) – aspect ratio

remove_node (*c*)
Removes a node and update the lineageTree accordingly

Parameters **c** (*int*) – id of the node to remove

to_binary (*fname*, *starting_points=None*)
Writes the lineage tree (a forest) as a binary structure (assuming it is a binary tree, it would not work for *n* ary tree with $2 < n$). The binary file is composed of 3 sequences of numbers and a header specifying the size of each of these sequences. The first sequence, *number_sequence*, represents the lineage tree as a DFT preporder transversal list. -1 signifying a leaf and -2 a branching The second sequence, *time_sequence*, represent the starting time of each tree. The third sequence, *pos_sequence*, reprensent the 3D coordinates of the objects. The header specify the size of each of these sequences. Each size is stored as a long long The *number_sequence* is stored as a list of long long (0 -> 2^(8*8)-1) The *time_sequence* is stored as a list of unsigned short (0 -> 2^(8*2)-1) The *pos_sequence* is stored as a list of double.

Parameters

- **fname** (*str*) – name of the binary file
- **starting_points** (*[int,]*) – list of the roots to be written. If None, all roots are written, default value, None

to_tlp (*fname*, *t_min=-1*, *t_max=inf*, *nodes_to_use=None*, *temporal=True*, *spatial=None*, *write_layout=True*, *node_properties=None*, *Names=False*)
Write a lineage tree into an understable tulip file.

Parameters

- **fname** (*str*) – path to the tulip file to create
- **t_min** (*int*) – minimum time to consider, default -1

- **t_max** (*int*) – maximum time to consider, default np.inf
- **nodes_to_use** (*[int,]*) – list of nodes to show in the graph, default *None*, then self.nodes is used (taking into account *t_min* and *t_max*)
- **temporal** (*bool*) – True if the temporal links should be printed, default True
- **spatial** (*str*) – Build spatial edges from a spatial neighbourhood graph. The graph has to be computed before running this function 'ball': neighbours at a given distance, 'kn': k-nearest neighbours, 'GG': gabriel graph, None: no spatial edges are writen. Default None
- **write_layout** (*bool*) – True, write the spatial position as layout, False, do not write spatial positionm default True
- **node_properties** ({*p_name*, [{id, p_value}, default]}) – a dictionary of properties to write To a key representing the name of the property is paired a dictionary that maps a cell id to a property and a default value for this property
- **Names** (*bool*) – Only works with ASTEC outputs, True to sort the cells by their names

write_to_am(*path_format*, *t_b=None*, *t_e=None*, *length=5*, *manual_labels=None*, *default_label=5*, *new_pos=None*)

Writes a lineageTree into an Amira readable data (.am format).

Parameters

- **path_format** (*str*) – path to the output. It should contain 1 %03d where the time step will be entered
- **t_b** (*int*) – first time point to write (if None, min(LT.to_take_time) is taken)
- **t_e** (*int*) – last time point to write (if None, max(LT.to_take_time) is taken) note, if there is no 'to_take_time' attribute, self.time_nodes is considered instead (historical)
- **length** (*int*) – length of the track to print (how many time before).
- **({id** (*new_pos*) – label, }): dictionary that maps cell ids to
- **default_label** (*int*) – default value for the manual label
- **({id** – [x, y, z]}): dictionary that maps a 3D position to a cell ID. if new_pos == None (default) then self.pos is considered.

write_to_svg(*file_name*, *roots=None*, *draw_nodes=True*, *draw_edges=True*, *order_key=None*, *vert_space_factor=0.5*, *horizontal_space=1*, *node_size=None*, *stroke_width=None*, *factor=1.0*, *node_color=None*, *stroke_color=None*, *positions=None*)

Writes the lineage tree to an SVG file. Node and edges coloring and size can be provided.

Parameters

- **file_name** – str, filesystem filename valid for *open()*
- **roots** – [int, ...], list of node ids to be drawn. If *None* all the nodes will be drawn. Default *None*
- **draw_nodes** – bool, wether to print the nodes or not, default *True*
- **draw_edges** – bool, wether to print the edges or not, default *True*
- **order_key** – function that would work for the attribute *key*= for the *sort*/*sorted* function
- **vert_space_factor** – float, the vertical position of a node is its time. *vert_space_factor* is a multiplier to space more or less nodes in time
- **horizontal_space** – float, space between two consecutive nodes

- **node_size** – func, a function that maps a node id to a *float* value that will determine the radius of the node. The default function return the constant value *vertical_space_factor/2.1*
- **stroke_width** – func, a function that maps a node id to a *float* value that will determine the width of the daughter edge. The default function return the constant value *vertical_space_factor/2.1*
- **factor** – float, scaling factor for nodes positions, default 1
- **node_color** – func, a function that maps a node id to a triplet between 0 and 255. The triplet will determine the color of the node.
- **stroke_color** – func, a function that maps a node id to a triplet between 0 and 255. The triplet will determine the color of the stroke of the inward edge.
- **positions** – {int: [float, float], ...}, dictionary that maps a node id to a 2D position. Default *None*. If provided it will be used to position the nodes.

INDEX

Symbols

A

C

F

G

L

R

T

W