LineageTree

Leo Guignard

Mar 10, 2020

CONTENTS:

CHAPTER

ONE

MODULES

class LineageTree.lineageTree.**lineageTree**(*file_format=None*, *tb=None*, *te=None*, *z_mult=1.0*, *mask=None*, *MaMuT=False*, *celegans=False*, *ASTEC=False*, *csv=False*, *link=True*, *delim=', '*, *eigen=False*)

lineageTree is a class container for lineage tree structures The main attributes are the following: self.nodes: [int,], list of node/cell ids self.edges: [(int, int),], a list of couple of cell/objects ids that represents the edges self.time_nodes: {int, [int,]}, a dictionary that maps time points to

a list of cell ids that belong to that time point

self.time_edges: {int, [(int, int),]}, a dictionary that maps time points to a list of edges couples that belong to that time point

self.successor: {int, [int,]}, a dictionary that maps a cell id to the list of its successors in time

self.predecessor: {int, [int,]}, a dictionary that maps a cell id to the list of its predecessors in time

self.time: {int: int, }, a dictionary that maps a cell to the time it belongs to.

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

Adds a node to the lineageTree and update it accordingly. :param t: int, time to which to add the node :param succ: id of the node the new node is a successor to :param pos: [float,], list of three floats representing the 3D spatial position of the node :param id: id value of the new node, to be used carefully,

if None is provided the new id is automatically computed.

Parameters **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 int, id of the new node.

Return type C_next

compute_spatial_density(*t_b=0*, *t_e=200*, *n_size=10*)

Computes the average distance between the *n_size* closest object for a set of time points The results is stored in self.spatial_density :param t_b: int, starting time to look at :param t_e: int, ending time to look at :param n_size: int, number of neighbors to look at

compute_spatial_edges(*th=50*)

Innefitiently computes the connection between cells at a given distance Writes the output in self.spatial_edges :param th: float, distance to consider neighbors

fuse_nodes(*c1*, *c2*)

Fuses together two nodes that belong to the same time point and update the lineageTree accordingly. :param c1: int, id of the first node to fuse :param 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 :param x: int, id of the node to compute :param depth: int, maximum number of predecessors and successor to return :param depth_pred: int, maximum number of predecessors to return :param depth_succ: int, maximum number of successors to return

Returns [int,], list of ids

Return type cycle

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. *WARNING: the graph is not recomputed if already computed. even if nodes were added*. :param t: int, time

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

Returns

kdtree, the built kdtree to_check_self: 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 idx3d

get_next_id()
Computes the next authorized id. :returns: int, next authorized id

get_predecessors(*x*, *depth=None*)
Computes the predecessors of the node *x* up to *depth* predecessors. The ordered list of ids is returned :param x: int, id of the node to compute :param depth: int, maximum number of predecessors to return

Returns [int,], list of ids, the last id is *x*

Return type cycle

get_successors(*x*, *depth=None*)
Computes the successors of the node *x* up to *depth* successors. The ordered list of ids is returned :param x: int, id of the node to compute :param depth: int, maximum number of predecessors to return

Returns [int,], list of ids, the first id is *x*

Return type cycle

print_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. :param file_name: str, filesystem filename valid for *open()* :param roots: [int, ...], list of node ids to be drawn. If *None* all the nodes will be drawn. Default *None* :param draw_nodes: bool, wether to print the nodes or not, default *True* :param draw_edges: bool, wether to print the edges or not, default *True* :param order_key: function that would work for the attribute *key=* for the *sort*/*sorted* function :param 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

Parameters

- **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.

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_mamut_xml (*path*)

Read a lineage tree from a MaMuT xml. :param path: string, path to the MaMut xml

read_tgmm_xml (*file_format*, *tb*, *te*, *z_mult=1.0*, *mask=None*)

Reads a lineage tree from TGMM xml output. :param file_format: string, path to the xmls location.

it should be written as follow: path/to/xml/standard_name_t%06d.xml where (as an example) %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

Parameters

- **tb** – int, first time point to read
- **te** – int, last time point to read
- **z_mult** – float, aspect ratio
- **mask** – SpatialImage, binary image that specify the region to read

remove_node (*c*)

Removes a node and update the lineageTree accordingly :param 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 :param fname: string, name of the binary file :param starting_points: [int,], list of the roots to be written.

If None, all roots are written Default: None

to_tlp(*fname*, *t_min=-1*, *t_max=inf*, *nodes_to_use=None*, *temporal=True*, *spatial=False*, *VF=False*, *write_layout=True*, *node_properties=None*, *Names=False*)

Write a lineage tree into an understable tulip file :param fname: string, path to the tulip file to create :param t_min: int, minimum time to consider, default -1 :param t_max: int, maximum time to consider, default np.inf :param 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*)

Parameters

- **temporal** – boolean, True if the temporal links should be printed, default True
- **spatial** – boolean, True if the special links should be printed, default True
- **VF** – boolean, useless
- **write_layout** – boolean, True, write the spatial position as layout, False, do not write spatial positionm default True
- **= {'p_name'** (*node_properties*) – [{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

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). :param path_format: string, path to the output. It should contain 1 %03d where the time step will be entered :param t_b: int, first time point to write (if None, min(LT.to_take_time) is taken) :param 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)

Parameters

- **length** – int, length of the track to print (how many time before).
- **manual_labels** – {id: label, }, dictionary that maps cell ids to
- **default_label** – int, default value for the manual label
- **new_pos** – {id: [x, y, z]}, dictionary that maps a 3D position to a cell ID. if new_pos == None (default) then self.pos is considered.

PYTHON MODULE INDEX

INDEX