Virtual Microbes Evolutionary Simulator Documentation

Release 0.1.5

Thomas Cuypers, Bram van Dijk

Jan 01, 2019

Contents

Contents:

CHAPTER 1

Overview

The VirtualMicrobes software package is a simulator of Virtual Microbe evolution. Virtual Microbes live in spatial environment where they compete for resources. In order to grow and divide they express their metabolic and regulatory genes to generate biomass. Mutations in their genome change the rates of metabolic reactions and gene expression levels enabling them to become better adapted over time.

1.1 Motivation

This software is being developed as part of the research in the EvoEvo project. The goal of the EvoEvo project is to understand how evolution can modify its own course. Mutations in the genome of an organism change the phenotype, thereby allowing adaptation to the environmental circumstances. Yet, how genotypic changes are translated to phenotypic changes, is itself an evolved property of species. Changing this genotype to phenotype mapping may make organisms more or less robust to mutations with respect to their functioning. Moreover, some structures of this mapping may make it more likely for random mutations to have large phenotypic effects, increasing the chance that a novel adaptive phenotype will be discovered. The goal of the VirtualMicrobes project is to study the *in silico* evolution of genome structure and of the genotype to phenotype mapping.

1.2 Features

- population of individuals
- spatial environment
- parameterized metabolic reaction space
- gene regulatory interaction network
- cell growth dynamics
- spatially extended genome
- genome and gene scale mutations

- lineage tracing
-

CHAPTER 2

Installing

Before installing VirtualMicrobes, you need to have setuptools installed.

2.1 Quick install

Get VirtualMicrobes from the Python Package Index at http://pypi.python.org/pypi/VirtualMicrobes

or install it with

```
pip install VirtualMicrobes
```

and an attempt will be made to find and install an appropriate version that matches your operating system and Python version.

The project can be cloned from bitbucket with:

```
git clone https://thocu@bitbucket.org/thocu/virtualmicrobes.git
```

2.2 Requirements

Installing the project via `pip`, the required packages will be checked against your installation and installed if necessary.

2.2.1 Python

To use VirtualMicrobes you need Python 2.7

2.2.2 ete3

This is a package for drawing and manipulating phylogenetic trees. It is used to keep track of phylogenetic relationships between `Virtual Microbes`.

- Download: http://etetoolkit.org/download/

ete3 has its own dependency on pyqt.

2.2.3 Matplotlib

Used for generating various plots during the simulation.

- Download: http://matplotlib.sourceforge.net/

2.2.4 Gnu Scientific Library

2.2.5 Other

- networkx
- attrdict
- blessings
- networkx
- pandas
- psutil
- errand_boy
- orderedset
- pyparsing
- setproctitle
- sortedcontainers
- mpl

2.3 Optional packages

2.3.1 Cython and CythonGSL

The package includes a few C Extension modules, originally written in the Cython language. If during the install both Cython and the CythonGSL package are detected, the extensions will be build from the original `.pyx` sources. Otherwise, the included pre-generated `.c` files will be used to build the extension.

- Download: http://http://cython.org/#download
- Download: https://github.com/twiecki/CythonGSL (follow the `build` and `install` instructions in the README)

CHAPTER 3

Reference

3.1 VirtualMicrobes package

3.1.1 Subpackages

VirtualMicrobes.Tree package

Submodules

VirtualMicrobes.Tree.PhyloTree module

class VirtualMicrobes.Tree.PhyloTree.**PhyloNode**(*val*, *time*)

Bases: object

Version

Author

add_root(*root_node*)

children

connect_phylo_offspring(*offspring*)

dist_to_parent(*parent*)

excise()

remove references to this node from other tree nodes and reconnect remaining nodes.

has_root()

id = 0

inherit_root(*node*)

is_leaf

is this a leaf of the tree

iter_prepostorder(*is_leaf_fn=None*)
Iterate over all nodes in a tree yielding every node in both pre and post order. Each iteration returns a postorder flag (True if node is being visited in postorder) and a node instance.

max_node_depth = 0

newick_id
id tag used in new hampshire (newick) tree format

nh_format_nr(*_format='newick'*, *formatter=<function nh_branch>*)
Format a newick string non-recursively, including internal nodes. (ete3, ape and iTol compatible!) NOTE: iTol doesn't like trees without leafs, so if you have a nh tree like (((A)B)C)D it will start complaining Add a comma to the deepest leaf to fix this, and delete it via iTol if you really want it. (((A,)B)C)D :) :param with_leafs: :param _format: :param formatter:

nhx_features
additional node features used in the 'extended' newick format (nhx)

parents

push_onto_internal_child(*child*)
makes this node an internal parent node of child :param child:

push_up_root()
Iteratively push up the root to the internal_parent_node.

remove_root_stem()

class VirtualMicrobes.Tree.PhyloTree.**PhyloTree**(*supertree=False*)
Bases: object

Primary use is a phylogenetic tree, representing reproduction/speciation events at particular points in time. Because generations are overlapping, a parent may have offspring at various time points. Therefore Nodes are not strictly parents, but rather, parent-reproduction-time tuples.

class SuperNode
Bases: object

add_leaf(*leaf*)

add_node(*phylo_unit*, *time*)

add_phylo_history(*phylo_unit*)

annotate_leafs(*ete_tree_struct*, *leaf*)
Annotate leaf labels

annotate_phylo_units_feature(*ete_tree_struct*, *phylo_units*, *feature_name*)

check_ete_mapping(*leafs=False*)

class_version = '1.0'

clear()

coalescent()
Find coalescent node and depth in the tree.

Returns (class

Return type *Tree.PhyloTree.PhyloNode*, time) : LCA, depth

connect_internal_node(*int_node*)
Connect internal node 'int_node' with nodes above and below it on its own branch (representing births and/or death in this phylogenetic unit's life history branch)

Parameters **int_node** – newly made internal node that should get connected up

and down in the tree.

connect_leaf(*leaf_node*)

connect_phylo_parent_child(*phylo_parent_node*, *phylo_child_node*)

create_leaf(*phylo_unit*)

create_root_stem(*phylo_root*)

create_stems(*phylo_unit*)

delete_empty_phylo_stems()

delete_node(*tree_node*)

delete_phylo_hist(*phylo_unit*)
Remove the branch representing the phylogenetic unit and disconnect it from all sub-branches (children) in the tree.

Parameters **phylo_unit** –

distances(*ete_tree*, *nodes*)
Pairwise distances between all nodes.

Parameters

- **ete_tree** (ete3.TreeNode) – ete tree root node
- **nodes** (*sequence of (phylo_node, ete_node) pairs*) –

Returns

- tree_dist is phylogenetic distance.
- top_dist is topological distance

Return type sequence of (phylo_node1, phylo_node2, tree_dist, top_dist)

ete_annotate_tree(*ete_tree_struct*, *features=[]*, *func_features={}*, *ete_root=None*)
Annotate the phylogenetic tree with cell data for tree plotting.

Assumes that the ete_tree has been constructed/updated. Creates a dictionary of feature dictionaries, keyed by the cells in the tree. Attaches the feature dictionaries to nodes in the ete_tree (annotate_ete_tree). Transforms some data to cummulative data along the branches of the tree. Optionally, prunes internal tree nodes (this will greatly simplify the tree drawing algorithm). Finally, transforms some data to rate of change data, using branch length for rate calculation.

Parameters **prune_internal** – if True, nodes with 1 offspring only will be

removed/collapsed and their branch length added to the preceding node on the branch.

ete_calc_lca_depth(*ete_tree*)
Calculate the distance from the last common ancestor to the farthest leaf in an ete tree.

Parameters **ete_tree** (*TreeNode*) – ete tree root node

Returns **int**

Return type depth of LCA

ete_convert_feature_to_cummulative(*ete_tree_struct*, *feature*, *ete_root*)
Convert annotated feature values to cummulative values.

By starting at the root, values further down the tree can be computed by adding values calculated for the parent node. It is useful when for example the aim is to prune away internal nodes and rates of change of a feature need to be calculated on the pruned tree.

ete_convert_feature_to_rate(*ete_root*, *feature*, *replace_tup=('count', 'rate')*)
Convert an annotated feature on the tree nodes to a rate.

The function assumes cummulative values of the feature. The rate is calculated by dividing the difference of the feature value between a parent and child by the corresponding branch length.

ete_cummulative_features(*ete_tree_struct*, *features=[]*, *func_features={}*, *ete_root=None*)

ete_get_lca(*ete_tree*, *nodes=None*)
Return last common ancestor (LCA) for a set of nodes.

(default: all leafs of the current tree)

Parameters

- **ete_tree** (*TreeNode*) – ete tree root node
- **nodes** (list of ete3.TreeNode) – nodes in tree for which to find LCA

Returns **TreeNode**

Return type the last common ancestor

ete_get_phylo2ete_dict(*ete_tree_struct*, *nodes=None*)
Return mapping from ete PhyloUnits to lists of ete TreeNode objects.

The mapping is constructed for a set of TreeNodes *nodes*. The default action is to map all the TreeNodes in the current ete_tree.

Parameters **nodes** (*sequence*) – sequence of TreeNodes

Returns

Return type dictionary from :class:‘VirtualMicrobes.virtual_cell.PhyloUnit.PhyloUnit‘s to list of :class:‘ete3.TreeNode‘s

ete_init_mappings(*ete_tree*)
Construct helper dictionaries for converting TreeNode names to TreeNodes and TreeNode names to PhyloUnits.

ete_n_most_distant_leafs(*ete_root*, *n*)
Find n leafs that are most diverged from eachother.

First, the oldest n subtrees are determined. Then, for all subtrees the farthest lying leaf is returned.

Parameters

- **n** – number of leafs
- **root** – starting point for search (default is the ete_tree root)

ete_n_most_distant_phylo_units(*ete_tree_struct*, *n=2*, *root=None*)
Convenience function to return phylo units after finding ‘ete_n_most_distant_leafs’ (see below) :param root: phylo unit used as root

ete_n_oldest_subtrees(*ete_root*, *n*)
Find n oldest subtrees under root.

Iteratively expands the oldest subtree root into its children until the desired number of subtrees is in the list. Return as a list of tuples of (subtree, distance-from-root).

Parameters

- **n** (*int*) – number of subtrees
- **root** (*TreeNode*) – start point for subtree search

Returns **subtrees**

Return type a list of (TreeNode, distance)

ete_named_node_dict
Return the ete_named_node_dict belonging to the first ete tree.

See also:

func *ete_tree*

ete_node_name_to_phylo_node
Return the ete_node_name_to_phylo_node belonging to the first ete tree.

See also:

func *ete_tree*

ete_node_to_phylo_unit(*ete_tree_struct*, *ete_node*)
Maps TreeNode to PhyloUnit.

Parameters **ete_node** (*TreeNode*) – node to map

Returns

Return type *PhyloUnit*

ete_nodes_to_phylo_units(*ete_tree_struct*, *nodes=None*)
Obtain a list of all the phylo units that are represented in the ete tree.

The ete_tree is traversed and ete node names are mapped first to PhyloTree nodes. The PhyloNodes have a reference to the PhyloUnit in their 'val' attribute.

ete_phylo_to_ete_birth_nodes(*ete_tree_struct*, *phylo_unit*)
Return iterator over birth nodes in the ete tree for a phylo unit.

Parameters **phylo_unit** (*VirtualMicrobes.virtual_cell.PhyloUnit.PhyloUnit*) – phylounit to map

Returns

Return type iterator of `ete3.TreeNode`

ete_phylo_to_ete_death_node(*ete_tree_struct*, *phylo_unit*)
Return TreeNode representing the death of a phylo_unit.

ete_phylo_to_ete_stem_nodes(*ete_tree_struct*, *phylo_unit*)
Return iterator over stem nodes in the ete tree for a phylo unit.

The stem nodes represent replication and death of the phylounit.

Parameters **phylo_unit** (*VirtualMicrobes.virtual_cell.PhyloUnit.PhyloUnit*) – phylounit to map

Returns

Return type iterator of `ete3.TreeNode`

ete_prune_external(*ete_tree_struct*, *prune_depth*)
Prune nodes of the external part of the tree beyond a certain depth.

The *prune_depth* is a time point in simulation time beyond which all subsequent nodes should be pruned and removed from the tree.

Parameters

- **ete_struct** (*VirtualMicrobes.my_tools.utility.ETEtreeStruct*) – struct holding data for the ete tree
- **prune_depth** (*int*) – simulation time point beyond which nodes should be pruned

Returns

Return type set of pruned internal `ete3.TreeNode` s

ete_prune_internal(*ete_tree_struct*)
Excise all internal nodes that have a single child, while preserving the length of the branch.

Parameters **ete_struct** (*VirtualMicrobes.my_tools.utility.ETEtreeStruct*) – struct holding data for the ete tree

Returns

Return type set of pruned internal `ete3.TreeNode` s

ete_prune_leafs(*ete_tree_struct*)
Prune the external nodes of an 'ete tree'.

Parameters **ete_tree_struct** (*VirtualMicrobes.my_tools.utility.ETEtreeStruct*) – struct holding data for the ete tree

Returns

Return type set of pruned internal `ete3.TreeNode` s

ete_rate_features(*ete_tree_struct*, *features=[]*, *func_features={}*, *ete_root=None*)

ete_tree
Return the first 'ete tree' in the ete trees dict if it exists.

This is an adapter function during the transition to working with the multiple ete tree dictionary.

find_lca()
Find the last common ancestor in the tree.

Start at the single root and go up until a branching point in the tree is found.

Returns **PhyloNode**

Return type LCA if it exists else None

nh_formats(*roots=None*, *supertree=None*, *_format='newick'*)

nodes_iter()

phylo_to_ete_nodes(*ete_tree_struct*, *phylo_unit*, *with_death=True*)
Return TreeNode objects in the ete_tree representing the birth and death nodes for this phylo_unit (e.g. cell or gene).

Parameters

- **ete_tree_struct** (*VirtualMicrobes.my_tools.utility.ETEtreeStruct*) – ete tree struct in which to find the phylo_unit

- **phylo_unit** (*VirtualMicrobes.virtual_cell.PhyloUnit.PhyloUnit*) – phylounit to map

Returns

Return type list of ete3.TreeNode

recalc_max_leaf_depth(*leafs=None*)

reconnect_internal_nodes(*internal_parent*, *internal_child*)

update(*new_phylo_units=[]*, *removed_phylo_units=[]*, *new_roots=[]*)

upgrade()

Upgrading from older pickled version of class to latest version. Version information is saved as class variable and should be updated when class invariants (e.g. fields) are added. Adapted from recipe at http://code.activestate.com/recipes/521901-upgradable-pickles/

VirtualMicrobes.Tree.PhyloTree.**newick**(*_id*, *time*, *branch_length*)

VirtualMicrobes.Tree.PhyloTree.**nh_branch**(*_id*, *branch_length*, *features=None*)

VirtualMicrobes.Tree.PhyloTree.**nhx**(*_id*, *time*, *branch_length*)

Module contents

VirtualMicrobes.cython_gsl_interface package

Submodules

VirtualMicrobes.cython_gsl_interface.integrate module

Document Manually ...

VirtualMicrobes.cython_gsl_interface.odes module

Document Manually ...

VirtualMicrobes.data_tools package

Submodules

VirtualMicrobes.data_tools.store module

class VirtualMicrobes.data_tools.store.**DataCollection**(*save_dir=''*, *name='dummy'*, *filename=None*)

Bases: object

change_save_location(*new_save_dir=None*, *copy_orig=True*, *current_save_path=None*)

get_data_point(*index*)

Get a data point at index.

Exception catching to handle case where due to code update a loaded form of a DataStore does not yet hold a particular DataCollection. In that case a dummy dict producing empty numpy arrays is returned.

Parameters **index** – key to a dict shaped data point

init_file(*name=None*, *labels=[]*, *suffix='.csv'*)

prune_data_file_to_time(*min_tp*, *max_tp*)
prune a data file by dropping lines starting from *max_tp*.

This function assumes that the first line contains column names and subsequent lines are time point data, starting with an comma separated (integer) time point.

Parameters

- **max_tp** – time point before which lines are dropped
- **max_tp** – time point after which lines are dropped

update_data_points(*index*, *dat*)

write_data()

class VirtualMicrobes.data_tools.store.**DataStore**(*base_save_dir*, *name*, *utility_path*, *n_most_frequent_metabolic_types*, *n_most_frequent_genomic_counts*, *species_markers*, *reactions_dict*, *small_mols*, *clean=True*, *create=True*)

Bases: object

Storage object for simulation data

Keep a store of simulation data that can be written to disk and retrieved for online plotting. Typically, the store will only hold the most recent data points before appending the data to relevant on disk storage files.

add_ancestry_data_point(*comp_dat*, *ref_lods*, *time_point*, *leaf_samples=100*)
Compare lines of descent in a reference tree to a population snapshot at a previous time point.

The *comp_dat* is a po

add_best_data_point(*best*, *attribute_mapper*, *time_point*, *affix=''*)

add_cell_tc(*cell*, *path*, *attribute_mapper*, *time_point*, *affix=''*)

add_collection(*dc*)

add_count_stats_dp(*dc_name*, *dat*, *time_point*)

add_dp(*dc_name*, *dat*, *time_point*)

add_eco_data_point(*system*, *time_point*)

add_expression_data_point(*system*, *time_point*)

add_frequency_stats_dp(*dc_name*, *dat*, *column_names*, *time_point*)

add_gain_loss_dp(*dc_name*, *cur_dat*, *prev_dat*, *time_point*)

add_list_dp(*dc_name*, *dat*, *time_point*)

add_lod_binding_conservation(*lod*, *stride*, *time_interval*, *lod_range*)

add_lod_data(*lod*, *pop*, *env*, *stride*, *time_interval*, *lod_range*)

add_lod_network_data(*lod*, *stride*, *time_interval*, *lod_range*, *save_dir=None*)

add_metabolic_stats_dp(*dc_name*, *dat*, *time_point*, *cutoff=0.05*)

add_pop_data_point(*system*, *time_point*)

add_raw_values_dp(*dc_name*, *dat*, *time_point*)

add_simple_stats_dp(*dc_name*, *dat*, *time_point*)

best_stats_dir = 'best_dat'

change_save_location(*base_save_dir=None*, *name=None*, *clean=False*, *copy_orig=True*, *create=True*, *current_save_path=None*)

class_version = '1.5'

copy_utility_files()

crossfeed_stats = ['crossfeeding', 'strict crossfeeding', 'exploitive crossfeeding']

eco_diversity_stats_dict = {'consumer type': <function <lambda> at 0x7fd6b5a09668>, '

eco_stats_dir = 'ecology_dat'

eco_type_stats = ['metabolic_type_vector', 'genotype_vector']

fit_stats_names = ['toxicity', 'toxicity_change_rate', 'raw_production', 'raw_producti

frequency_stats(*dat*, *column_names*)

functional_stats_names = ['conversions_type', 'genotype', 'reaction_genotype', 'metabo

gain_loss(*cur_dat*, *prev_dat*)

gain_loss_columns = ['gain', 'loss']

genome_dist_stats = ['copy_numbers', 'copy_numbers_tfs', 'copy_numbers_enzymes', 'copy

genome_simple_stats_names = ['tf_promoter_strengths', 'enz_promoter_strengths', 'pump_

genome_simple_val_stats_names = ['genome_size', 'chromosome_count', 'tf_count', 'enzym

grid_stats = ['neighbor crossfeeding', 'strict neighbor crossfeeding', 'exploitive nei

grn_edits = {'': <function <lambda> at 0x7fd6b5a09aa0>, '_pruned_1._cT_iT': <function

init_ancestry_compare_stores(*pop_hist*)

init_dict_stats_store(*save_dir*, *stats_name*, *column_names*, *index_name='time_point'*, ***kwargs*)

init_eco_data_stores(*save_dir=None*)

init_expression_data_stores(*save_dir=None*)

init_gain_loss_store(*save_dir*, *stats_name*, *index_name='time_point'*, *filename=None*)

init_list_stats_store(*save_dir*, *stats_name*)

init_lod_stores(*lod*, *met_classes*, *conversions*, *transports*, *first_anc*, *last_anc*)

init_phylo_hist_stores(*phylo_hist*)

init_pop_data_stores(*save_dir=None*)

init_save_dirs(*clean=False*, *create=True*)

create the paths to store various data types

Parameters

- **clean** – (bool) remove existing files in path
- **create** – create the path within the file system

init_simple_stats_plus_store(*save_dir*, *stats_name*, *index_name='time_point'*)

init_simple_stats_store(*save_dir*, *stats_name*, *index_name='time_point'*, *filename=None*)

init_simple_value_store(*save_dir*, *stats_name*, *index_name='time_point'*, *filename=None*)

init_stats_column_names(*n_most_freq_met_types*, *n_most_freq_genomic_counts*, *species_markers*, *reactions_dict*, *small_mols*)
Initialize column names for various data collections :param n_most_freq_met_types: :param n_most_genomic_counts:

lod_stats_dir = 'lod_dat'

meta_stats_names = ['providing_count', 'strict_providing_count', 'exploiting_count', '

metabolic_categories = ['producer', 'consumer', 'import', 'export']

mut_stats_names = ['point_mut_count', 'chromosomal_mut_count', 'stretch_mut_count', 'cl

network_stats_funcs = {'all_node_connectivities': <function <lambda> at 0x7fd6b5a098c

phy_dir = 'phylogeny_dat'

pop_genomic_stats_dict = {'chromosome counts': <function <lambda> at 0x7fd6b5a09848>,

pop_simple_stats_dict = {'cell sizes': <function <lambda> at 0x7fd6b5a08d70>, 'death

pop_stats_dir = 'population_dat'

prune_data_files_to_time(*min_tp*, *max_tp*)

save_anc_cells(*pop*, *time*)

save_dir

save_lod_cells(*lod*, *stride*, *time_interval*, *lod_range*, *runtime*)

save_phylo_tree(*pop*, *time*)

simple_stats(*dat*)

simple_stats_columns = ['avrg', 'min', 'max', 'median', 'std']

simple_value(*dat*)

simple_value_column = ['value']

snapshot_stats_names = ['historic_production_max']

trophic_type_columns = ['fac-mixotroph', 'autotroph', 'heterotroph', 'obl-mixotroph']

type_differences_stats(*types*, *column_names*)

type_totals_stats(*types*, *column_name*)

upgrade(*odict*)
Upgrading from older pickled version of class to latest version. Version information is saved as class variable and should be updated when class invariants (e.g. fields) are added. (see also __setstate__)

Adapted from recipe at http://code.activestate.com/recipes/521901-upgradable-pickles/

write_data()

write_genome_json(*save_dir*, *name*, *genome*, *attribute_mapper*, *labels*, *suffix='.json'*)

class VirtualMicrobes.data_tools.store.**DictDataCollection**(*column_names*, *index_name*, *to_string=None*, ***kwargs*)
Bases: *VirtualMicrobes.data_tools.store.DataCollection*

Entry point for storing and retrieving structured data

update_data_points(*index*, *data_dict*)

write_column_names(*columns=None*, *index_name=None*, *to_string=None*)

write_data(*sep=', '*)

class VirtualMicrobes.data_tools.store.**ListDataCollection**(***kwargs*)
Bases: *VirtualMicrobes.data_tools.store.DataCollection*

update_data_points(*index*, *data_vector*)

write_data(*sep=', '*)

VirtualMicrobes.data_tools.store.**create_gene_type_time_course_dfs**(*cell*)

VirtualMicrobes.data_tools.store.**create_tc_df**(*tc_dict*)

VirtualMicrobes.data_tools.store.**eco_type_vector_to_dict**(*vect_dict*)

VirtualMicrobes.data_tools.store.**tf_conservation_to_dict**(*tf_conservation_dict*)

VirtualMicrobes.data_tools.store.**tf_name**(*tf*)

Module contents

VirtualMicrobes.environment package

Submodules

VirtualMicrobes.environment.Environment module

Created on Mar 9, 2014

@author: thocu

class VirtualMicrobes.environment.Environment.**Environment**(*params*, *max_time_course_length=0*)
Bases: object

add_new_cells_to_grid(*cell_gp_dict*)

cells_grid_diffusion(*diff_rate=None*)
Cell diffusion on the grid.

diff_rate [float] Diffusion rate of cells to neighboring grid points.

class_version = '1.1'

clear_dead_cells_from_grid()

clear_mol_time_courses()

energy_mols

energy_precursors()
Return set of metabolites that are direct precursors for the energy class molecules.

Construct this set by iterating over energy classes and map them to reactions that produce the energy classes.

Returns

Return type class:OrderedSet

expression_grid_data_dict()
Return dictionary of spatial concentration data per metabolite (within cells)

external_molecules

find_reaction(*match_string*)
Returns reaction that matches stringrepr.

fluctuate(*time*, *p_fluct=None*, *influx_rows=None*, *influx_cols=None*, *mols=None*, *influx_dict=None*)
Influx fluctuation on the grid.

Sets new influx rates for influxed molecules depending on fluctuation frequency. Influx rates may vary between different sub-environments. Each sub-environment may define its own set of molecules that can be fluxed in.

Parameters

- **time** (*int*) – simulation time
- **p_fluct** (*float*) – fluctuation frequency per influxed molecule
- **influx_rows** (*iterable*) – a sequence of row indices on which molecules are exclusively influxed
- **influx_cols** (*iterable*) – a sequence of columnt indices on which molecules are exclusively influxed
- **mols** (list of :class:'VirtualMicrobes.event.Molecule.Molecule's) – molecules for which new influx rates should be set
- **influx_dict** (*dict*) – mapping from *VirtualMicrobes.event.Molecule.Molecule* to influx rate (float)

func_on_grid(*gp_func*, *rows=None*, *cols=None*)
Apply a function to a set of grid points.

The function *gp_func* should take a *VirtualMicrobes.environment.Environment.Locality* as argument. A subset of grid points are selected by using rows and cols lists. If both rows and cols are given, select gps as a mesh of intersecting rows and cols.

Parameters

- **gp_func** (*function*) – function on *VirtualMicrobes.environment.Environment.Locality*
- **rows** (*iterable*) – sequence of indices for grid row selection
- **cols** (*iterable*) – sequence of indices for grid column selection

grid_data_from_func(*func*)

grid_toggle_update(*updated=False*)

influx_change_gamma(*influx*, *variance_fact*, *upper=None*, *lower=None*)

influx_change_range(*param_space*)

init_anabolic_reactions(*nr_anabolic=None*, *nr_reactants=None*, *max_products=None*, *max_path_conversion=None*, *max_free_energy=None*)
Initialize the set of anabolic reactions.

Anabolic reactions combine small molecules into larger molecules. If the total free energy of product(s) is larger than that of the substrates, energy molecules will also be consumed in the reaction until reaction is (as) balanced (as possible).

Parameters

- **nr_catabolic** (*int*) – number of reactions to generate

- **max_products** (*int*) – maximum number of species produced in the reaction
- **min_energy** (*int*) – minimum amount of energy produced as energy molecules
- **max_path_conversion** (*int*) – maximum nr of reactions producing any molecule species
- **max_free_energy** (*int*) – maximum free energy loss from reaction

Returns

Return type list of *VirtualMicrobes.event.Reaction.Convert* reactions

init_catabolic_reactions(*nr_catabolic=None*, *max_products=None*, *min_energy=None*, *max_path_conversion=None*)

Initialize the set of catabolic reactions.

Catabolic reactions break down large molecules into smaller molecules. If the total free energy of products is lower than that of the substrate, energy molecules will also be produced in the reaction until reaction is (as) balanced (as possible).

Parameters

- **nr_catabolic** (*int*) – number of reactions to generate
- **max_products** (*int*) – maximum number of species produced in the reaction, excluding energy
- **min_energy** (*int*) – minimum amount of energy produced as energy molecules
- **max_path_conversion** (*int*) – maximum nr of reactions producing any molecule species

Returns

Return type list of *VirtualMicrobes.event.Reaction.Convert* reactions

init_class_conversions(*fraction_reactions*)

init_degradation()

init_degradation_dict(*degr_const=None*, *ene_degr_const=None*, *bb_degr_const=None*, *degradation_variance_shape=None*)

Initialize a global dictionary for degradation rates of molecules in the external environment.

init_diffusion()

init_external_mol_vals_on_grid(*init_conc=None*)

Initialize concentrations, degradation and influx rates of molecules on the grid.

init_global_influx_dict(*influx=None*)

Initialize a global dictionary for influx rates of molecules in the external environment.

init_grid(*verbose=False*)

Initialize the spatial grid. Set wrapping and barriers on particular neighborhoods.

init_influx()

init_influxed_mols(*fraction_influx=None*)

Select molecules that will be fluxed in in the external environment.

init_localities(*number_localities*, *max_time_course_length=0*, *params_dict=None*)

init_membrane_diffusion_dict(*diff_const=None*, *ene_diff_const=None*, *energy_proportional=None*, *diff_scaling_func=<function <lambda>>*)

init_microfluid_cycle()
Sets up a list of dictionaries to iterate cycles of fixed external concentrations

init_mol_class_sizes()

init_mol_classes()

init_mol_toxicities(*toxicity_avrg=None*, *toxicity_variance_shape=None*, *toxic_building_block=None*)

init_mols_to_reactions()
Create dictionaries of reactions, keyed on the molecules procuced and consumed in the reactions.

init_reaction_universe()

init_reactions()

init_sub_envs(*row_divs=None*, *col_divs=None*, *partial_influx=None*, *influx_combinations=None*)
Initialize sub-environments by dividing up the grid along rows and columns.

Within each subenvironment, influx can change independently. When partial_influx is chosen between 0 and 1, influxed molecules will only appear in a fraction of the subenvironments on the grid.

Parameters

- **row_divs** (*int*) – nr of divisions on y-axis
- **col_divs** (*int*) – nr of divisions on x-axis
- **partial_influx** (*float*) – fraction of molecules that will be influxed in each sub-environment

init_transports()

map_variables()

metabolite_grid_data_dict()
Return dictionary of spatial concentration data per metabolite

metabolite_internal_grid_data_dict()
Return dictionary of spatial concentration data per metabolite (within cells)

microfluid_chemostat(*run_time=None*)

molecule_classes

mols_per_class_dict

perfect_mix(*verbose=False*)
Perfect mix first shuffles all gps, then evens out all metabolites

pick_mol_class(*weighted_classes*, *rand_gen=None*)
Randomly select a molecule class with probability proportional to a weight.

Parameters

- **weighted_classes** (list of *VirtualMicrobes.event.Molecule.MoleculeClass*, weight (float) tuples) – molecule classes with their respective weights
- **rand_gen** (*RNG*) – RNG

Returns the selected class with its energy level and list index

Return type *VirtualMicrobes.event.Molecule.MoleculeClass*, energy, pos index

populate_localities(*population*)

population_grid_data(*data_func*)

population_grid_data_dict(*marker_names*, *marker_select_func*)
Return a dictionary of spatial distribution of values per marker

Parameters

- **marker_names** – markers to probe on the grid
- **marker_select_func** – how to select a marker value

when there are multiple individuals with different values per grid

population_grid_neighborhood_data(*neighborhood_pop_func*, *neighborhood='competition'*)

print_state()

print_values()

rand_anabolic_reaction_scheme(*resource_classes*, *energy_classes*, *product_classes*, *nr_reactants*, *max_products*, *max_free_energy*, *max_ene_energy=1*, *rand=None*)
Construct a random anabolic reaction scheme.

Construction is done by randomly selecting a molecule classes as the substrates of the reaction and than select the molecule classes that will be the product of the reaction. The total energy of reactants (+ energy) >= products.

Parameters

- **resource_classes** (list of VirtualMicrobes.event.Molecules.MoleculeClass) – all resource molecule classes
- **energy_classes** (list of VirtualMicrobes.event.Molecules.MoleculeClass) – energy molecule classes
- **max_products** (*int*) – maximum number of products in reaction scheme
- **max_ene_energy** (*int*) – maximum energy level that can be provided by energy metabolites as reaction substrate

Returns

Return type reaction scheme tuple (list of reactants, list of products, stoichiometries

rand_catabolic_reaction_scheme(*substrate_classes*, *product_classes*, *energy_classes*, *max_products*, *min_energy*, *rand_gen=None*)
Construct a random catabolic reaction scheme.

Typically a catabolic reaction breaks down a large, energy rich molecule into smaller molecules. Energy molecules may be produced in this reaction.

Construction is done by randomly selecting a molecule class as the substrate of the reaction and than select the molecule classes that will be the product of the reaction. The total energy of reactants = products (+ energy).

Parameters

- **substrate_classes** (list of VirtualMicrobes.event.Molecules.MoleculeClass) – all resource molecule classes
- **product_classes** (list of VirtualMicrobes.event.Molecules.MoleculeClass) – molecule classes that can be products of the reaction

- **energy_classes** (list of VirtualMicrobes.event.Molecules.MoleculeClass) – energy molecule classes
- **max_products** (*int*) – maximum number of products in reaction scheme
- **min_energy** (*int*) – minimum yield in energy metabolites

Returns

Return type reaction scheme tuple (list of reactants, list of products, stoichiometries

reactions_dict

repopulate_localities(*population*, *cells*, *mixed=False*)

reset_grid_concentrations(*conc=None*)

reset_grid_influx()
Reinitialize the global infux dict and the per sub environment influx rates.

reset_locality_updates()

resize_time_courses(*new_max_time_points*)

resource_combis_within_energy(*resource_classes*, *nr*, *total_energy*)
Return combinations of resource classes that have exactly a total sum in energy values.

Parameters

- **resource_classes** (set of :class:'VirtualMicrobes.event.Molecules.MoleculeClass's) – the set to make combinations from
- **nr** (*int*) – number of MCs to combine
- **total_energy** (*int*) – the sum of energy values that MCs should have

Returns

Return type combinations of MCs

select_building_blocks(*nr_bb*, *mol_classes*, *substrate_weight_function=<function <lambda>>*, *weight_scaling=2*, *rand_gen=None*)

set_mol_concentrations_from_time_point()

set_tot_volume()

start_concentration_dict(*init_conc=None*, *init_conc_dict=None*, *no_influx_conc=1e-20*)
Make dictionary of start concentrations for external molecules.

If the init_conc is not given, start with a concentration that is the equilibrium value based on influx and degradation rates of the metabolite.

Parameters **init_conc** –

sub_envs_influx_combinations(*richest_first=True*)
Assign a subset of all influxed molecules per individual sub-environment.

Parameters **fract** (*float*) – fraction of subenvironments where a molecule is fluxed in

sub_envs_partial_influx(*fract*)
Assign a subset of all influxed molecules per individual sub-environment.

Parameters **fract** (*float*) – fraction of subenvironments where a molecule is fluxed in

subgrids_gen(*row_divs=1*, *col_divs=1*)
Generate partitions of the grid.

Partitions are constructed from the product of row-chunks and column- chunks, such that the grid is divided (approximately) equally according to the number of row- and column divisions.

Parameters

- **row_divs** (*int*) – number of divisions in the row dimension
- **col_divs** (*int*) – number of divisions in the column dimension

Yields *iterable of '(row_nrs, col_nrs)' partitions of the grid*

update_cells_on_grid(*cell_gp_dict*)

update_localities()

update_reaction_universe(*path*)
Updates reaction universe by adding newly added molecules and reactions to the environment. Does NOT support the removal of metabolites / reactions, as this will often be very artificial for cells that are still using them.

update_volume(*volume=None*)

upgrade()
Upgrading from older pickled version of class to latest version. Version information is saved as class variable and should be updated when class invariants (e.g. fields) are added.

class VirtualMicrobes.environment.Environment.**Locality**(*params*, *internal_molecules*, *influx_reactions*, *degradation_reactions*, *env_rand*, *max_time_course_length=0*)

Bases: object

add_cell(*cell*)

add_small_molecule(*mol*, *concentration=0.0*)

class_version = '1.0'

clear_locality()
Clear the cells in this locality.

clear_mol_time_courses()

get_cells()

get_expression_level(*rea*, *exporting=False*)

get_gene_prod_conc(*gene*)

get_internal_mol_conc(*mol*)

get_mol_concentration_dict()
Get molecule concentration dictionary

get_mol_influx_dict()
Get influx dictionary from locality dictionary

get_mol_per_class_concentration_dict()

get_mol_per_class_influx_dict()

get_mol_time_course_dict(*max_tp=None*)
Get time course of molecule

get_small_mol_conc(*mol*)

init_external_mols(*conc=None*)

init_mol_time_course(*mol_struct*, *length=None*)

init_mol_views()
Sets alias to molecules in locality (QoL implementation)

init_time_courses(*length=None*)
initialize an array to hold time course data of molecules

Parameters **new_max_time_points** – max number of time points

init_variables_map()

map_cell_gene_products(*cell*)
The mapping is a dictionary of dictionaries; in this way, it is possible to recognize the same protein in the two different cells as two separate variables. (see also map_external_molecules)

Parameters **start_index** –

map_cell_internal_mols(*cell*)

map_external_molecules()
Mapping of the small molecules to indexes for use in the C integrator. The map is an initially empty dictionary. The first level of indexing is the container , while the second level is the molecules (type) . The container can be a cell or it can be the environment. (see also map_variables)

Parameters **start_index** –

map_variables()
Setup of the mapping of variables in the system to indexes used by the C encoded integrator part of the simulation:

- small molecules (metabolites, toxins, resources)
- gene products

In practice we start with an empty dictionary and let 'system' map molecules and gene products sepselfarately, creating the indexes on the go and reporting back on the last selfused index.

remove_cell(*cell*)

resize_time_courses(*new_max_time_points*)

set_gene_prod_conc(*gene*, *conc*)

set_mol_concentrations_from_time_point()

set_small_mol_conc(*mol*, *val*)

spill_dead_cell_contents(*cell*, *prod_as_bb=None*, *factor=None*)

tp_index

update_small_mol_concentrations(*concentration_dict*)

update_small_mol_degradation_rates(*degradation_dict*)

update_small_mol_influxes(*influx_dict*, *no_influx=1e-20*)

upgrade()
Upgrading from older pickled version of class to latest version. Version information is saved as class variable and should be updated when class invariants (e.g. fields) are added.

VirtualMicrobes.environment.Grid module

Created on Nov 3, 2014

@author: thocu

class VirtualMicrobes.environment.Grid.**Grid**(*rows*, *cols*, *neighborhood_dict=None*, *nei_wrapped_ew=None*, *nei_wrapped_ns=None*)

Bases: object

Grid will by default be wrapped, because the Neighborhood will index

as_numpy_array

columns_iter(*cols*, *order='tb_lr'*)

Iterate over gps in the selected columns :param cols: selected columns :param order: the order in which gps should be iterated

content_iter

disconnect_direction(*gp*, *area*, *neighborhood*)

For all neighboring grid points lieing in a particular *direction* in a named *neighborhood* of a grid point *gp*, remove the *gp* coordinate from their neighborhoods.

e.g. if area is 'south' , the southern neighbors of gp will remove gp's relative coordinate from their neighborhood

Parameters

- **gp** (*GridPoint*) – Grid point that will get unwrapped.
- **direction** (*{'north', 'south', 'east', 'west'}, optional*) – The direction relative to the *gp*.
- **neighborhood** (*str*) – named neighborhood for which the unwrapping is done

dummy()

fill_grid(*vals*, *order='lr_tb'*)

get_gp(*x*, *y*)

get_nei_gp(*gp*, *direction_vect*)

gp_iter

gp_iter_in_order(*order='lr_tb'*)

grid_barriers(*rand_gen*, *p_row=0.0*, *max_fraction_width=None*, *p_col=0.0*, *max_fraction_height=None*, *neighborhoods=None*)

Set up barriers in the grid

Parameters

- **p_row** –
- **max_fraction_width** –
- **p_col** –
- **max_fraction_height** –
- **neighborhood** –
- **rand_gen** –

init_grid(*rows*, *cols*, *neighborhood_dict*, *wrap_ew=None*, *wrap_ns=None*)

make_barrier(*start_coord*, *length*, *neighborhood_name*, *direction='lr'*)
Create a barrier on the grid where interaction between adjacent grid points is blocked.

Create a barrier in the given neighborhood_name by unwrapping gps on opposite sides of a line that is *length* long, starts at *start_gp* and extends along *direction*

mesh_iter(*rows*, *cols*, *order='lr_tb'*)
iterate over gps in a mesh, defined by intersections of rows and cols

normalize_coord(*coord*)
Transform coordinate *coord* to an absolute grid coordinate with non- wrapped index.

Applies modulo `cols` and `rows` on the coordinates x and y value, respectively, to obtain a normalized coordinate.

Parameters **coord** (*VirtualMicrobes.my_tools.utility.Coord*) – a coordinate on the grid that may be in wrapped index representation

Returns

Return type *VirtualMicrobes.my_tools.utility.Coord*

perfect_mix(*rand_gen*)

rows_iter(*rows*, *order='lr_tb'*)
Iterate over gps in the selected rows

set_gp(*x*, *y*, *gp*)

swap_content(*gp1*, *gp2*)

swap_gps(*gp1*, *gp2*)

swap_pos(*pos1*, *pos2*)

toggle_gps_updated(*updated=False*)

un_neighbor(*gp*, *nei_rel_coord*, *neighborhood*)

unwrap_ew(*neighborhood_name*)
Unwraps a grid neighborhood at its eastern and western borders.

Iterate over all grid points and detect when a grid point has neighbors that wrap over the east or west borders. Then remove the coordinate from the neighborhood.

Parameters **neighborhood_name** (*str*) – The name of the neighborhood to unwrap

Notes

For a wrapped neighbor it is true that the difference between the focal gp coordinate and this neighbors (normalized) grid-coordinate is not equal to the its relative-coordinate (to focal gp).

See also:

unwrap_ns(), *normalize_coord()*

unwrap_ns(*neighborhood_name*)
Unwraps a grid neighborhood at its northern and southern borders.

Iterate over all grid points and detect when a grid point has neighbors that wrap over the north or south borders. Then remove the coordinate from the neighborhood.

Parameters **neighborhood_name** (*str*) – The name of the neighborhood to unwrap

Notes

For a wrapped neighbor it is true that the difference between the focal gp coordinate and this neighbors (normalized) grid-coordinate is not equal to the its relative-coordinate (to focal gp).

See also:

unwrap_ew(), *normalize_coord()*

update_gp(*x*, *y*, *val*)

exception VirtualMicrobes.environment.Grid.**GridError**(*value*)
Bases: exceptions.Exception

class VirtualMicrobes.environment.Grid.**GridPoint**
Bases: object

content

coord

get_neighbor_at(*rel_coord*, *neighborhood_name*)
Get neighboring grid point at a relative coordinate.

Parameters

- **rel_coord** (*VirtualMicrobes.my_tools.utility.Coord*) – coordinate relative to self
- **neighborhood_name** (*str*) – Name of the neighborhood.

Returns The neighboring grid point.

Return type *GridPoint*

nei_grid_coords(*neighborhood_name*, *area=None*)

nei_rel_coord_to_gp(*neighborhood_name*, *area=None*)
List of (relative-coordinate,grid-point) tuples of the named neighborhood.

Parameters

- **neighborhood_name** (*str*) – Name of the neighborhood
- **area** (*{'north', 'south', 'east', 'west'}, optional*) – Select only neighbors lying in a specific area.

Returns

Return type list of (*VirtualMicrobes.my_tools.utility.Coord*, *GridPoint*) tuples

nei_rel_to_grid_coords(*neighborhood_name*, *area=None*)
List of (relative-coordinate, grid-coordinate) tuples of the named neighborhood.

Parameters

- **neighborhood_name** (*str*) – Name of the neighborhood
- **area** (*{'north', 'south', 'east', 'west'}, optional*) – Select only neighbors lying in a specific area

Returns

Return type list of (*VirtualMicrobes.my_tools.utility.Coord*, *VirtualMicrobes.my_tools.utility.Coord*) tuples

neighbor_gps(*neighborhood_name*, *area=None*)
Return list of grid points in a named neighborhood.

Parameters

- **neighborhood_name** (*str*) – Name of the neighborhood.
- **area** (*{'north', 'south', 'east', 'west'}, optional*) – Select only neighbors lying in a specific area.
- """ _

neighbors(*neighborhood_name*, *area=None*)
Return list of grid point content values of neighbors in named neighborhood.

Parameters

- **neighborhood_name** (*str*) – Name of the neighborhood.
- **area** (*{'north', 'south', 'east', 'west'}, optional*) – Select only neighbors lying in a specific area.

pos

random_neighbor(*neighborhood_name*, *rand_gen*, *area=None*)
Get random neighbor from a named neighborhood.

Parameters

- **neighborhood_name** (*str*) – Name of the neighborhood.
- **rand_gen** (*random generator*) – Random generator for neighbor drawing.
- **area** (*{'north', 'south', 'east', 'west'}, optional*) – Select only neighbors lying in a specific area.

remove_neighbor_at(*rel_coord*, *neighborhood_name*)
Remove a coordinate from a named neighborhood.

After removing the relative coordinate *rel_coord*, the corresponding grid point is no longer considered a neighbor of *self* when applying neighborhood functions.

Parameters

- **rel_coord** (*VirtualMicrobes.my_tools.utility.Coord*) – Coordinate relative to self.
- **neighborhood_name** (*str*) – Name of the neighborhood.

updated

class VirtualMicrobes.environment.Grid.**Neighborhood**(*neighbors*)
Bases: object

Neighborhood as grid indices (x,y coord tuples) relative to a focal gridpoint.

construct_moore_n(*perim=1*, *exclude_self=True*)

construct_named_neighborhood(*name*)

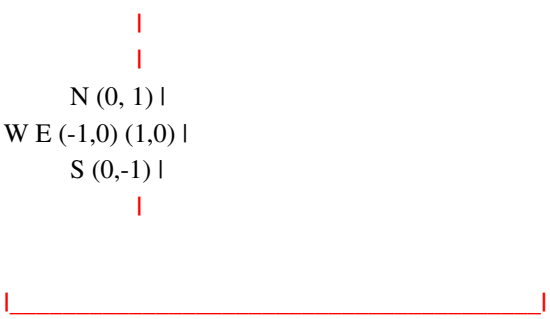
construct_neighbors(*neighbors*)

construct_neumann_n(*manhatten=1*, *exclude_self=True*)

get_rel_coords(*area=None*)
Return a list of neighborhood coordinates that lie in an area of the compass relative to (0,0)

Parameters **area** (*{'north', 'south', 'east', 'west'}*) – The named area (or 'hemisphere') in grid space relative to the origin (0,0).

Notes



Returns The coordinates that lie within the named area.

Return type list of :class:'VirtualMicrobes.my_tools.utility.Coord's

has_coord(*rel_coord*)
Check that relative coordinate is part of this neighborhood.

Parameters **rel_coord** (*VirtualMicrobes.my_tools.utility.Coord*) –

remove_coord(*remove_coord*)
Remove a relative coordinate from the neighborhood.

Parameters **remove_coord** (*VirtualMicrobes.my_tools.utility.Coord*) –

remove_direction(*direction*)
Remove coordinates lying to the given direction on the grid, where the main compass points are translated into coordinates as follows:

The main compass points are translated into coordinates as follows: N (0, 1)

W E (-1,0) (1,0) S (0,-1)

e.g. when direction is 'north', all relative coordinates that lie northerly of the (0,0) will be removed.

Parameters **direction** (*{'north', 'south', 'east', 'west'}, optional*) – Selects those coordinates in the neighborhood that lie in a particular direction (default is None, which implies returninng all coordinates.

exception VirtualMicrobes.environment.Grid.**PositionOutsideGridError**(*x*, *y*)
Bases: *VirtualMicrobes.environment.Grid.GridError*

VirtualMicrobes.environment.Grid.**mirror_rel_coord**(*coord*)

Module contents

VirtualMicrobes.event package

Submodules

VirtualMicrobes.event.Molecule module

class VirtualMicrobes.event.Molecule.**Molecule**(*name*, *toxic_level=None*, *is_internal=True*, *pair=True*, *is_building_block=False*, *is_gene_product=False*, *mol_class=None*, *is_energy=False*, *environment=None*, ***kwargs*)

Bases: object

Molecule species.

An internally and external variant of each molecule is defined. Molecules can act as metabolites that can be converted in (enzymatic) reactions. Can diffuse over membranes and may be transported into or out of the cell.

class_version = '1.0'

energy_level

environment

index = 0

is_building_block

is_energy

is_gene_product

is_influxed

is_internal

mol_class

pair_up()

Create a paired molecule for self on the other side of the Cell membrane.

When updating a property of self, the property of the paired molecule is automatically updated (if appropriate; e.g. toxic_level or is_energy)

set_building_block(*val=True*)

short_repr()

toxic_level

classmethod unique_index(*increase=None*)

upgrade()

Upgrading from older pickled version of class to latest version. Version information is saved as class variable and should be updated when class invariants (e.g. fields) are added.

class VirtualMicrobes.event.Molecule.**MoleculeClass**(*name*, *molecule_species=None*, *energy_level=1*, *is_energy=False*, *has_building_block=False*)

Bases: object

Defines a class of related molecule species.

add_molecule(*molecule*)
Add a molecule to this molecule class.

Parameters **molecule** (*VirtualMicrobes.event.Molecule*) – a molecule

short_repr()
A short string representation of the molecule class

class VirtualMicrobes.event.Molecule.**MoleculeIndexer**
Bases: object

VirtualMicrobes.event.Reaction module

exception VirtualMicrobes.event.Reaction.**BadStoichiometryException**
Bases: exceptions.Exception

class VirtualMicrobes.event.Reaction.**ClassConvert**(*substrate*, *energy*, *product*)
Bases: *VirtualMicrobes.event.Reaction.Convert*

Convert molecules within the same molecule class.

Parameters

- **substrate** (*VirtualMicrobes.event.Molecule.Molecule*) – molecule to convert
- **energy** (*VirtualMicrobes.event.Molecule.MoleculeClass*) – energy molecule class
- **product** (*VirtualMicrobes.event.Molecule.Molecule*) – product molecule

init_sub_reaction_dicts()
Write out dictionaries for the sub_reactions generated by sub_reactions()

class VirtualMicrobes.event.Reaction.**Convert**(*reactants*, *products*, *stoichiometry*)
Bases: *VirtualMicrobes.event.Reaction.Reaction*

Conversion reaction type.

In a conversion reaction a set of reactants react and a set of products are produced. Both *reactants* and *products* are defined as *VirtualMicrobes.event.Molecule.MoleculeClass*. Therefore, a single *Convert* reaction represent a set of sub-reactions of sets of reactant- *VirtualMicrobes.event.Molecule.Molecule*, sets of product- *VirtualMicrobes.event.Molecule.Molecule* combinations.

A pairing rule governs the exact set of sub-reactions that can take place.

Parameters

- **reactants** (*VirtualMicrobes.event.Molecule.MoleculeClass*) – reactants in reaction
- **products** (*VirtualMicrobes.event.Molecule.MoleculeClass*) – products of reaction
- **stoichiometry** (*list of ints*) – stoichiometric constants of reaction

init_sub_reaction_dicts()
Write out dictionaries for the sub_reactions generated by sub_reactions()

prod_species

reac_species

sub_reactions()
Representation of sub-reactions.

Returns a list of all potential reaction schemes in the following form: ([(reactant, stoichiometry), ..], [(product, stoichiometry) ..]). The general scheme for reactions from and to Molecule Classes maps molecules within a MolClass on lhs to molecules in another class on the rhs as follows:

Reaction scheme as MoleculeClass scheme: A + B -> C , where A :{a0, a1, a2}, B:{b0, b1}, C:{c0, c1, c2*} will be translated into:

a0 + b0 -> c0 a1 + b0 -> c1 a2 + b0 -> c2*

a0 + b1 -> c0 a1 + b1 -> c1 a2 + b1 -> c2*

- If certain molecule species do not exist (e.g. the c2 in the previous example does not

exist, the reaction is omitted from possible sub-reactions, and will therefor not take place. Note that products on the rhs will always be converted in to the species corresponding to the index of the substrate on the lhs. If there is more product than substrates, e.g. A -> C + D where D:{d0, d1}, then there will be subreactions for every possible species of D:

a0 -> c0 + d0 a0 -> c0 + d1

a1 -> c1 + d0 a1 -> c1 + d1

a2 -> c2 + d0 a2 -> c2 + d1

Example 2: F + G -> H + I , where F :{f0, f1, f2}, G:{g0}, H:{h0, h1} , I:{i0, i2} becomes:

f0 + g0 -> h0 + i0 f1 + g0 -> h1 + i0

.

class VirtualMicrobes.event.Reaction.**Degradation**(*substrate*, ***kwargs*)
Bases: *VirtualMicrobes.event.Reaction.Reaction*

Degradation type reaction.

Represents degradation of a molecule. The reaction has no products. Thus, when molecules degrade, mass is not strictly conserved.

Parameters **substrate** (*VirtualMicrobes.event.Molecule.Molecule*) – degrading molecule

reaction_scheme()
Reaction scheme dictionary.

sub_reactions()
Representation of sub-reactions.

In the case of Degradation reactions, only a single sub-reaction exists.

class VirtualMicrobes.event.Reaction.**Diffusion**(*substrate*, ***kwargs*)
Bases: *VirtualMicrobes.event.Reaction.Reaction*

Diffusion type reaction.

A reaction representing diffusion over a barrier (here the cell membrane). No products are defined for this reaction. Methods implementing this reaction type will convert the external/internal variant of a *VirtualMicrobes.event.Molecule.Molecule* into the linked internal/external molecule type.

Parameters **substrate** (*VirtualMicrobes.event.Molecule.Molecule*) – diffusing molecule

reaction_scheme()
Reaction scheme dictionary.

sub_reactions()
Representation of sub-reactions.

In the case of Diffusion reactions, only a single sub-reaction exists.

class VirtualMicrobes.event.Reaction.**Influx**(*substrate*, ***kwargs*)
Bases: *VirtualMicrobes.event.Reaction.Reaction*

Influx type reaction.

A reaction representing influx of molecules into the environment.

Parameters **substrate** (*VirtualMicrobes.event.Molecule.Molecule*) – molecule that fluxes in

reaction_scheme()
Reaction scheme dictionary.

class VirtualMicrobes.event.Reaction.**Reaction**(*type_*, *reactants*, *products*, *stoichiometry*)
Bases: object

Base class for reactions.

Reactions are rules to convert Molecules to other Molecules, or Transport Molecules from one volume to another volume. A reaction specifies its reactants and products as MoleculeClasses or Molecules. Because a MoleculeClass can hold multiple Molecules, one Reaction is in fact a collection of potential molecular reactions. The actualized reactions depend on the properties (binding affinities for specific Molecules) of the enzymes and the availability of Molecules.

Parameters

- **type** (*str*) – an identifier for the reaction type
- **reactants** (list of *VirtualMicrobes.event.Molecule.MoleculeClass* or) – *VirtualMicrobes.event.Molecule.Molecule* reactants in the reaction
- **products** (list of *VirtualMicrobes.event.Molecule.MoleculeClass* or) – *VirtualMicrobes.event.Molecule.Molecule* products of the reaction
- **stoichiometry** (*list of int*) – stoichiometric constants determine ratios of participants in the reaction

short_repr()
Shorter version of __str__

class VirtualMicrobes.event.Reaction.**Transport**(*substrate_class*, *energy_source_class*, *sub_stoi*, *cost*, ***kwargs*)
Bases: *VirtualMicrobes.event.Reaction.Reaction*

A transport type reaction.

Substrates are transported over a membrane, typically requiring the consumption of an energy source. The substrate and energy source are defined as *VirtualMicrobes.event.Molecule.MoleculeClass*. Therefore, a single *Transport* reaction represent a set of sub-reactions of substrate- *VirtualMicrobes.event.Molecule.Molecule*, energy- *VirtualMicrobes.event.Molecule.Molecule* combinations.

Parameters

- **substrate_class** (*VirtualMicrobes.event.Molecule.MoleculeClass*) – the substrate being transported
- **energy_source_class** (*VirtualMicrobes.event.Molecule.MoleculeClass*) – the energy source

- **sub_stoi** (*int*) – stoichiometric constant for substrate
- **cost** (*int*) – stoichiometry of energy cost

init_sub_reaction_dicts()
Write out dictionaries for the sub_reactions generated by sub_reactions()

sub_reactions()
Representation of sub-reactions.

Sub-reactions of Transporters link a specific external substrate to its internal counterpart. Different energy sources yield combinatorial expansion.

VirtualMicrobes.event.Reaction.**consumes**(*reactions*)
Set of consumed metabolic species.

Parameters **reactions** (iterable of *Convert* or dict) – set of reactions

Returns

Return type set of :class:'VirtualMicrobes.event.Molecule.Molecule's consumed in the reaction set.

VirtualMicrobes.event.Reaction.**find_metabolic_closure**(*input_set*, *conversions*)
Find the autocatalytic closure of metabolites given a set of inputs and reactions.

Iteratively overlap the produced + influxed and consumed metabolites of the set of conversion reactions, yielding a set of 'potentially autocatalytic metabolites'. Iterate over the set of 'potentially autocatalytic reactions' and require that all substrates of the reaction are in the set of 'potentially autocatalytic' metabolites. If not, remove the reaction from the 'potentially autocatalytic' reaction set.

Parameters

- **input_set** (iterable of *VirtualMicrobes.event.Molecule.Molecule*) – initial set to start expansion of metabolic set
- **conversions** (iterable of :class:'Conversion's) – enzymatic reactions that convert metabolites into other metabolites

Returns **tuple** – molecules and reactions in the core autocatalytic cycle.

Return type (set of :class:'VirtualMicrobes.event.Molecule.Molecule's , set of :class:'Conversion's)

VirtualMicrobes.event.Reaction.**find_product_set**(*input_mols*, *conversions*)
Find metabolites that can be produced from a set of input metabolites, given a set of reactions.

Parameters

- **input_mols** (iterable of *VirtualMicrobes.event.Molecule.Molecule*) – initial set to start expansion of metabolic set
- **conversions** (iterable of :class:'Conversion's) – enzymatic reactions that convert metabolites into other metabolites

Returns

Return type set of produced :class:'VirtualMicrobes.event.Molecule.Molecule's

VirtualMicrobes.event.Reaction.**produces**(*reactions*)
Set of produced metabolic species.

Parameters **reactions** (iterable of *Convert* or dict) – set of reactions

Returns

Return type set of:class:'VirtualMicrobes.event.Molecule.Molecule's produced in the reaction set.

Module contents

Created on Nov 12, 2013

author thocu

VirtualMicrobes.mutate package

Subpackages

VirtualMicrobes.mutate.test package

Submodules

VirtualMicrobes.mutate.test.TestMutate module

Module contents

Submodules

VirtualMicrobes.mutate.Mutation module

class VirtualMicrobes.mutate.Mutation.**ChromosomalMutation**(*chromosomes*, *genome*)
Bases: *VirtualMicrobes.mutate.Mutation.Mutation*

class VirtualMicrobes.mutate.Mutation.**ChromosomeDeletion**(*chromosome*, *genome*)
Bases: *VirtualMicrobes.mutate.Mutation.ChromosomalMutation*

mutate(*time*)
Apply mutation.

Parameters **time** (*int*) – simulation time

reapply()
Reapply mutation after rewinding

rewind()
Go back to the ancestral state.

class VirtualMicrobes.mutate.Mutation.**ChromosomeDuplication**(*chromosome*, *genome*)
Bases: *VirtualMicrobes.mutate.Mutation.ChromosomalMutation*

mutate(*time*)
Apply mutation.

Parameters **time** (*int*) – simulation time

reapply()
Reapply mutation after rewinding

rewind()
Go back to the ancestral state.

class VirtualMicrobes.mutate.Mutation.**Fission**(*chromosome*, *genome*, *pos*)
Bases: *VirtualMicrobes.mutate.Mutation.ChromosomalMutation*

mutate(*time*)
Apply mutation.

Parameters **time** (*int*) – simulation time

pos

reapply()
Reapply mutation after rewinding

rewind()
Go back to the ancestral state.

class VirtualMicrobes.mutate.Mutation.**Fusion**(*chrom1*, *chrom2*, *genome*, *end1*, *end2*)
Bases: *VirtualMicrobes.mutate.Mutation.ChromosomalMutation*

end1

end2

mutate(*time*)
Apply mutation.

Parameters **time** (*int*) – simulation time

reapply()
Reapply mutation after rewinding

rewind()
Go back to the ancestral state.

class VirtualMicrobes.mutate.Mutation.**Insertion**(*chromosome*, *genome*, *stretch*, *insert_pos*, *is_external*)
Bases: *VirtualMicrobes.mutate.Mutation.StretchMutation*

Insertion of a stretch of exogenous genomic material

insert_pos

is_external

mutate(*time*)
Apply mutation.

Parameters **time** (*int*) – simulation time

reapply()
Reapply mutation after rewinding

rewind()
Go back to the ancestral state.

class VirtualMicrobes.mutate.Mutation.**Inversion**(*chromosome*, *genome*, *start_pos*, *end_pos*)
Bases: *VirtualMicrobes.mutate.Mutation.StretchMutation*

mutate(*time*)
The invert is in place, hence pre- and post- mutation will appear the same

reapply()
Reapply mutation after rewinding

rewind()
Go back to the ancestral state.

class VirtualMicrobes.mutate.Mutation.**Mutation**(*target*, *genomic_unit*)
Bases: object

Base class for mutations.

applied
boolean – indicates that the mutation has been applied

genomic_target
target of the mutation

post_mutation
post mutation state of the *genomic_target*

genomic_unit
contains the *genomic_target*

time
int – simulation time when first mutated

applied

genomic_target

genomic_unit

mutate(*time*)
Apply mutation.

Parameters **time** (*int*) – simulation time

post_mutation

reapply()
Reapply mutation after rewinding

rewind()
Go back to the ancestral state.

time

uid = 0

exception VirtualMicrobes.mutate.Mutation.**MutationAlreadyAppliedError**(*value="Cannot 'reapply' if already applied"*)
Bases: *VirtualMicrobes.mutate.Mutation.MutationError*

exception VirtualMicrobes.mutate.Mutation.**MutationError**
Bases: exceptions.Exception

exception VirtualMicrobes.mutate.Mutation.**MutationNotAppliedError**(*value="Cannot 'rewind' if not already applied"*)
Bases: exceptions.Exception

class VirtualMicrobes.mutate.Mutation.**OperatorInsertion**(*gene*, *chromosome*, *new_val*, *pos*)
Bases: *VirtualMicrobes.mutate.Mutation.SingleGeneMutation*

mutate(*time*)
Apply mutation.

Parameters **time** (*int*) – simulation time

new_val

par

reapply()
Reapply mutation after rewinding

rewind()
Go back to the ancestral state.

class VirtualMicrobes.mutate.Mutation.**PointMutation**(*gene*, *chromosome*, *par*, *new_val*, *pos*)
Bases: *VirtualMicrobes.mutate.Mutation.SingleGeneMutation*

mutate(*time*)
Apply mutation.

Parameters **time** (*int*) – simulation time

new_val

par

reapply()
Reapply mutation after rewinding

rewind()
Go back to the ancestral state.

class VirtualMicrobes.mutate.Mutation.**SGDeletion**(*gene*, *chromosome*, *pos*)
Bases: *VirtualMicrobes.mutate.Mutation.SingleGeneMutation*

class VirtualMicrobes.mutate.Mutation.**SGDuplication**(*gene*, *chromosome*, *pos*)
Bases: *VirtualMicrobes.mutate.Mutation.SingleGeneMutation*

class VirtualMicrobes.mutate.Mutation.**SingleGeneMutation**(*gene*, *chromosome*, *pos*)
Bases: *VirtualMicrobes.mutate.Mutation.Mutation*

mutate(*time*)
Apply mutation.

Parameters **time** (*int*) – simulation time

pos

reapply()
Reapply mutation after rewinding

rewind()
Go back to the ancestral state.

class VirtualMicrobes.mutate.Mutation.**StretchDeletion**(*chromosome*, *genome*, *start_pos*, *end_pos*)
Bases: *VirtualMicrobes.mutate.Mutation.StretchMutation*

mutate(*time*)
Apply mutation.

Parameters **time** (*int*) – simulation time

reapply()
Reapply mutation after rewinding

rewind()
Go back to the ancestral state.

class VirtualMicrobes.mutate.Mutation.**StretchMutation**(*chromosome*, *genome*, *start_pos=None*, *end_pos=None*, *stretch=None*)
Bases: *VirtualMicrobes.mutate.Mutation.Mutation*

end_pos

positive_positions()

start_pos

stretch

class VirtualMicrobes.mutate.Mutation.**TandemDuplication**(*chromosome*, *genome*, *start_pos*, *end_pos*)
Bases: *VirtualMicrobes.mutate.Mutation.StretchMutation*

mutate(*time*)
Apply mutation.

Parameters **time** (*int*) – simulation time

reapply()
Reapply mutation after rewinding

rewind()
Go back to the ancestral state.

class VirtualMicrobes.mutate.Mutation.**Translocation**(*chromosome*, *genome*, *start_pos*, *end_pos*, *target_chrom*, *insert_pos*, *invert*)
Bases: *VirtualMicrobes.mutate.Mutation.StretchMutation*

insert_pos

invert

mutate(*time*)
Apply mutation.

Parameters **time** (*int*) – simulation time

positive_positions()

reapply()
Reapply mutation after rewinding

rewind()
Go back to the ancestral state.

Module contents

VirtualMicrobes.my_tools package

Submodules

VirtualMicrobes.my_tools.analysis_tools module

VirtualMicrobes.my_tools.analysis_tools.**set_differences**(*sets*)
Returns a measure of difference between sets in a Counter object

From the counter object, the n most frequent sets are extracted. Then, between each set and the remaining sets differences are computed and expressed as number of differing items for each pair compaired (all pairs formed by 'it.combinations'). This number is then scaled to the maximum possible difference, which would be the combined number of items in both members of a pair.

Parameters **sets** – (e.g. produced metabolites in a cell)

VirtualMicrobes.my_tools.monkey module

class VirtualMicrobes.my_tools.monkey.**Artist**
Bases: object

Abstract base class for someone who renders into a FigureCanvas.

add_callback(*func*)
Adds a callback function that will be called whenever one of the *Artist*'s properties changes.

Returns an *id* that is useful for removing the callback with *remove_callback()* later.

aname = u'Artist'

axes
The Axes instance the artist resides in, or *None*.

contains(*mouseevent*)
Test whether the artist contains the mouse event.

Returns the truth value and a dictionary of artist specific details of selection, such as which points are contained in the pick radius. See individual artists for details.

convert_xunits(*x*)
For artists in an axes, if the xaxis has units support, convert *x* using xaxis unit type

convert_yunits(*y*)
For artists in an axes, if the yaxis has units support, convert *y* using yaxis unit type

draw(*renderer*, **args*, ***kwargs*)
Derived classes drawing method

findobj(*match=None*, *include_self=True*)
Find artist objects.

Recursively find all Artist instances contained in self.

match can be

- None: return all objects contained in artist.
- function with signature boolean = match(artist) used to filter matches

- class instance: e.g., Line2D. Only return artists of class type.

If *include_self* is True (default), include self in the list to be checked for a match.

format_cursor_data(*data*)
Return *cursor data* string formatted.

get_agg_filter()
return filter function to be used for agg filter

get_alpha()
Return the alpha value used for blending - not supported on all backends

get_animated()
Return the artist's animated state

get_axes()
Return the `Axes` instance the artist resides in, or *None*.

This has been deprecated in mpl 1.5, please use the axes property. Will be removed in 1.7 or 2.0.

get_children()
Return a list of the child `Artist`s this :class:`Artist` contains.

get_clip_box()
Return artist clipbox

get_clip_on()
Return whether artist uses clipping

get_clip_path()
Return artist clip path

get_contains()
Return the _contains test used by the artist, or *None* for default.

get_cursor_data(*event*)
Get the cursor data for a given event.

get_figure()
Return the `Figure` instance the artist belongs to.

get_gid()
Returns the group id

get_label()
Get the label used for this artist in the legend.

get_path_effects()

get_picker()
Return the picker object used by this artist

get_rasterized()
return True if the artist is to be rasterized

get_sketch_params()
Returns the sketch parameters for the artist.

Returns

- **sketch_params** (tuple or *None*)
- *A 3-tuple with the following elements –*
  - *scale*: The amplitude of the wiggle perpendicular to the source line.

- *length*: The length of the wiggle along the line.
- *randomness*: The scale factor by which the length is shrunken or expanded.

• May return *None* if no sketch parameters were set.

get_snap()
Returns the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

get_transform()
Return the `Transform` instance used by this artist.

get_transformed_clip_path_and_affine()
Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.

get_url()
Returns the url

get_visible()
Return the artist's visiblity

get_window_extent(*renderer*)
Get the axes bounding box in display space. Subclasses should override for inclusion in the bounding box "tight" calculation. Default is to return an empty bounding box at 0, 0.

Be careful when using this function, the results will not update if the artist window extent of the artist changes. The extent can change due to any changes in the transform stack, such as changing the axes limits, the figure size, or the canvas used (as is done when saving a figure). This can lead to unexpected behavior where interactive figures will look fine on the screen, but will save incorrectly.

get_zorder()
Return the *Artist*'s zorder.

have_units()
Return *True* if units are set on the *x* or *y* axes

hitlist(*event*)
List the children of the artist which contain the mouse event *event*.

is_figure_set()
Returns True if the artist is assigned to a `Figure`.

is_transform_set()
Returns *True* if *Artist* has a transform explicitly set.

mouseover

pchanged()
Fire an event when property changed, calling all of the registered callbacks.

pick(*mouseevent*)
call signature:

```
pick(mouseevent)
```

each child artist will fire a pick event if *mouseevent* is over the artist and the artist has picker set

pickable()
Return *True* if *Artist* is pickable.

properties()
return a dictionary mapping property name -> value for all Artist props

remove()
Remove the artist from the figure if possible. The effect will not be visible until the figure is redrawn, e.g., with `matplotlib.axes.Axes.draw_idle()`. Call `matplotlib.axes.Axes.relim()` to update the axes limits if desired.

Note: `relim()` will not see collections even if the collection was added to axes with *autolim* = True.

Note: there is no support for removing the artist's legend entry.

remove_callback(*oid*)
Remove a callback based on its *id*.

See also:

add_callback() For adding callbacks

set(***kwargs*)
A property batch setter. Pass *kwargs* to set properties. Will handle property name collisions (e.g., if both 'color' and 'facecolor' are specified, the property with higher priority gets set last).

set_agg_filter(*filter_func*)
set agg_filter fuction.

set_alpha(*alpha*)
Set the alpha value used for blending - not supported on all backends.

ACCEPTS: float (0.0 transparent through 1.0 opaque)

set_animated(*b*)
Set the artist's animation state.

ACCEPTS: [True | False]

set_axes(*axes*)
Set the `Axes` instance in which the artist resides, if any.

This has been deprecated in mpl 1.5, please use the axes property. Will be removed in 1.7 or 2.0.

ACCEPTS: an `Axes` instance

set_clip_box(*clipbox*)
Set the artist's clip `Bbox`.

ACCEPTS: a `matplotlib.transforms.Bbox` instance

set_clip_on(*b*)
Set whether artist uses clipping.

When False artists will be visible out side of the axes which can lead to unexpected results.

ACCEPTS: [True | False]

set_clip_path(*path*, *transform=None*)
Set the artist's clip path, which may be:

- a `Patch` (or subclass) instance
- **a `Path` instance, in which case** an optional `Transform` instance may be provided, which will be applied to the path before using it for clipping.

- *None*, to remove the clipping path

For efficiency, if the path happens to be an axis-aligned rectangle, this method will set the clipping box to the corresponding rectangle and set the clipping path to *None*.

ACCEPTS: [(`Path`, `Transform`) | `Patch` | None]

set_contains(*picker*)
Replace the contains test used by this artist. The new picker should be a callable function which determines whether the artist is hit by the mouse event:

```
hit, props = picker(artist, mouseevent)
```

If the mouse event is over the artist, return *hit* = *True* and *props* is a dictionary of properties you want returned with the contains test.

ACCEPTS: a callable function

set_figure(*fig*)
Set the `Figure` instance the artist belongs to.

ACCEPTS: a `matplotlib.figure.Figure` instance

set_gid(*gid*)
Sets the (group) id for the artist

ACCEPTS: an id string

set_label(*s*)
Set the label to *s* for auto legend.

ACCEPTS: string or anything printable with '%s' conversion.

set_path_effects(*path_effects*)
set path_effects, which should be a list of instances of matplotlib.patheffect._Base class or its derivatives.

set_picker(*picker*)
Set the epsilon for picking used by this artist

picker can be one of the following:

- *None*: picking is disabled for this artist (default)
- A boolean: if *True* then picking will be enabled and the artist will fire a pick event if the mouse event is over the artist
- A float: if picker is a number it is interpreted as an epsilon tolerance in points and the artist will fire off an event if it's data is within epsilon of the mouse event. For some artists like lines and patch collections, the artist may provide additional data to the pick event that is generated, e.g., the indices of the data within epsilon of the pick event
- A function: if picker is callable, it is a user supplied function which determines whether the artist is hit by the mouse event:

  ```
  hit, props = picker(artist, mouseevent)
  ```

  to determine the hit test. if the mouse event is over the artist, return *hit=True* and props is a dictionary of properties you want added to the PickEvent attributes.

ACCEPTS: [None|float|boolean|callable]

set_rasterized(*rasterized*)
Force rasterized (bitmap) drawing in vector backend output.

Defaults to None, which implies the backend's default behavior

ACCEPTS: [True | False | None]

set_sketch_params(*scale=None*, *length=None*, *randomness=None*)
Sets the sketch parameters.

Parameters

- **scale** (*float, optional*) – The amplitude of the wiggle perpendicular to the source line, in pixels. If scale is *None*, or not provided, no sketch filter will be provided.
- **length** (*float, optional*) – The length of the wiggle along the line, in pixels (default 128.0)
- **randomness** (*float, optional*) – The scale factor by which the length is shrunken or expanded (default 16.0)

set_snap(*snap*)
Sets the snap setting which may be:

- True: snap vertices to the nearest pixel center
- False: leave vertices as-is
- None: (auto) If the path contains only rectilinear line segments, round to the nearest pixel center

Only supported by the Agg and MacOSX backends.

set_transform(*t*)
Set the `Transform` instance used by this artist.

ACCEPTS: `Transform` instance

set_url(*url*)
Sets the url for the artist

ACCEPTS: a url string

set_visible(*b*)
Set the artist's visiblity.

ACCEPTS: [True | False]

set_zorder(*level*)
Set the zorder for the artist. Artists with lower zorder values are drawn first.

ACCEPTS: any number

stale
If the artist is 'stale' and needs to be re-drawn for the output to match the internal state of the artist.

update(*props*)
Update the properties of this *Artist* from the dictionary *prop*.

update_from(*other*)
Copy properties from *other* to *self*.

zorder = 0

class VirtualMicrobes.my_tools.monkey.**Figure**(*figsize=None*, *dpi=None*, *facecolor=None*, *edgecolor=None*, *linewidth=0.0*, *frameon=None*, *subplotpars=None*, *tight_layout=None*)
Bases: *VirtualMicrobes.my_tools.monkey.Artist*

The Figure instance supports callbacks through a *callbacks* attribute which is a `matplotlib.cbook.CallbackRegistry` instance. The events you can connect to are 'dpi_changed', and the callback will be called with `func(fig)` where fig is the *Figure* instance.

patch The figure patch is drawn by a `matplotlib.patches.Rectangle` instance

suppressComposite For multiple figure images, the figure will make composite images depending on the renderer option_image_nocomposite function. If suppressComposite is True|False, this will override the renderer.

add_axes(**args*, ***kwargs*)

Add an axes at position *rect* [*left*, *bottom*, *width*, *height*] where all quantities are in fractions of figure width and height. kwargs are legal `Axes` kwargs plus *projection* which sets the projection type of the axes. (For backward compatibility, `polar=True` may also be provided, which is equivalent to `projection='polar'`). Valid values for *projection* are: [u'aitoff', u'hammer', u'lambert', u'mollweide', u'polar', u'rectilinear']. Some of these projections support additional kwargs, which may be provided to *add_axes()*. Typical usage:

```
rect = l,b,w,h
fig.add_axes(rect)
fig.add_axes(rect, frameon=False, axisbg='g')
fig.add_axes(rect, polar=True)
fig.add_axes(rect, projection='polar')
fig.add_axes(ax)
```

If the figure already has an axes with the same parameters, then it will simply make that axes current and return it. If you do not want this behavior, e.g., you want to force the creation of a new Axes, you must use a unique set of args and kwargs. The axes `label` attribute has been exposed for this purpose. e.g., if you want two axes that are otherwise identical to be added to the figure, make sure you give them unique labels:

```
fig.add_axes(rect, label='axes1')
fig.add_axes(rect, label='axes2')
```

In rare circumstances, add_axes may be called with a single argument, an Axes instance already created in the present figure but not in the figure's list of axes. For example, if an axes has been removed with *delaxes()*, it can be restored with:

```
fig.add_axes(ax)
```

In all cases, the `Axes` instance will be returned.

In addition to *projection*, the following kwargs are supported:

adjustable: ['box' | 'datalim' | 'box-forced'] agg_filter: unknown alpha: float (0.0 transparent through 1.0 opaque) anchor: unknown animated: [True | False] aspect: unknown autoscale_on: unknown autoscalex_on: unknown autoscaley_on: unknown axes: an `Axes` instance axes_locator: unknown axis_bgcolor: any matplotlib color - see `colors()` axisbelow: [*True* | *False*] clip_box: a `matplotlib.transforms.Bbox` instance clip_on: [True | False] clip_path: [(`Path`, `Transform`) | `Patch` | None] color_cycle: unknown contains: a callable function figure: unknown frame_on: [*True* | *False*] gid: an id string label: string or anything printable with '%s' conversion. navigate: [*True* | *False*] navigate_mode: unknown path_effects: unknown picker: [None|float|boolean|callable] position: unknown rasterization_zorder: unknown rasterized: [True | False | None] sketch_params: unknown snap: unknown title: unknown transform: `Transform` instance url: a url string visible: [True | False] xbound: unknown xlabel: unknown xlim: length 2 sequence of floats xmargin: unknown xscale: [u'linear' | u'log' | u'logit' | u'symlog'] xticklabels: sequence of strings xticks: sequence of floats ybound: unknown ylabel:

unknown ylim: length 2 sequence of floats ymargin: unknown yscale: [u'linear' | u'log' | u'logit' | u'symlog'] yticklabels: sequence of strings yticks: sequence of floats zorder: any number

add_axobserver(*func*)
whenever the axes state change, `func(self)` will be called

add_subplot(**args*, ***kwargs*)
Add a subplot. Examples:

```
fig.add_subplot(111)

# equivalent but more general
fig.add_subplot(1,1,1)

# add subplot with red background
fig.add_subplot(212, axisbg='r')

# add a polar subplot
fig.add_subplot(111, projection='polar')

# add Subplot instance sub
fig.add_subplot(sub)
```

kwargs are legal `Axes` kwargs plus *projection*, which chooses a projection type for the axes. (For backward compatibility, *polar=True* may also be provided, which is equivalent to *projection='polar'*). Valid values for *projection* are: [u'aitoff', u'hammer', u'lambert', u'mollweide', u'polar', u'rectilinear']. Some of these projections support additional *kwargs*, which may be provided to *add_axes()*.

The `Axes` instance will be returned.

If the figure already has a subplot with key (*args*, *kwargs*) then it will simply make that subplot current and return it.

See also:

`subplot()` for an explanation of the args.

The following kwargs are supported:

adjustable: ['box' | 'datalim' | 'box-forced'] agg_filter: unknown alpha: float (0.0 transparent through 1.0 opaque) anchor: unknown animated: [True | False] aspect: unknown autoscale_on: unknown autoscalex_on: unknown autoscaley_on: unknown axes: an `Axes` instance axes_locator: unknown axis_bgcolor: any matplotlib color - see `colors()` axisbelow: [*True* | *False*] clip_box: a `matplotlib.transforms.Bbox` instance clip_on: [True | False] clip_path: [(`Path`, `Transform`) | `Patch` | None] color_cycle: unknown contains: a callable function figure: unknown frame_on: [*True* | *False*] gid: an id string label: string or anything printable with '%s' conversion. navigate: [*True* | *False*] navigate_mode: unknown path_effects: unknown picker: [None|float|boolean|callable] position: unknown rasterization_zorder: unknown rasterized: [True | False | None] sketch_params: unknown snap: unknown title: unknown transform: `Transform` instance url: a url string visible: [True | False] xbound: unknown xlabel: unknown xlim: length 2 sequence of floats xmargin: unknown xscale: [u'linear' | u'log' | u'logit' | u'symlog'] xticklabels: sequence of strings xticks: sequence of floats ybound: unknown ylabel: unknown ylim: length 2 sequence of floats ymargin: unknown yscale: [u'linear' | u'log' | u'logit' | u'symlog'] yticklabels: sequence of strings yticks: sequence of floats zorder: any number

autofmt_xdate(*bottom=0.2*, *rotation=30*, *ha=u'right'*)
Date ticklabels often overlap, so it is useful to rotate them and right align them. Also, a common use case is a number of subplots with shared xaxes where the x-axis is date data. The ticklabels are often long, and it helps to rotate them on the bottom subplot and turn them off on other subplots, as well as turn off xlabels.

bottom The bottom of the subplots for *subplots_adjust()*

rotation The rotation of the xtick labels

ha The horizontal alignment of the xticklabels

axes
Read-only – list of axes in Figure

clear()
Clear the figure – synonym for *clf()*.

clf(*keep_observers=False*)
Clear the figure.

Set *keep_observers* to True if, for example, a gui widget is tracking the axes in the figure.

colorbar(*mappable*, *cax=None*, *ax=None*, *use_gridspec=True*, ***kw*)
Create a colorbar for a ScalarMappable instance, *mappable*.

Documentation for the pylab thin wrapper:

Add a colorbar to a plot.

Function signatures for the `pyplot` interface; all but the first are also method signatures for the `colorbar()` method:

```
colorbar(**kwargs)
colorbar(mappable, **kwargs)
colorbar(mappable, cax=cax, **kwargs)
colorbar(mappable, ax=ax, **kwargs)
```

Parameters ***mappable*** – the `Image`, `ContourSet`, etc. to which the colorbar applies; this argument is mandatory for the `colorbar()` method but optional for the `colorbar()` function, which sets the default to the current image.

Keyword Arguments

- ***cax*** – None | axes object into which the colorbar will be drawn
- ***ax*** – None | parent axes object(s) from which space for a new colorbar axes will be stolen. If a list of axes is given they will all be resized to make room for the colorbar axes.
- ***use_gridspec*** – False | If *cax* is None, a new *cax* is created as an instance of Axes. If *ax* is an instance of Subplot and *use_gridspec* is True, *cax* is created as an instance of Subplot using the grid_spec module.

Additional keyword arguments are of two kinds:

axes properties:

Property	Description
orientation	vertical or horizontal
fraction	0.15; fraction of original axes to use for colorbar
pad	0.05 if vertical, 0.15 if horizontal; fraction of original axes between colorbar and new image axes
shrink	1.0; fraction by which to shrink the colorbar
aspect	20; ratio of long to short dimensions
anchor	(0.0, 0.5) if vertical; (0.5, 1.0) if horizontal; the anchor point of the colorbar axes
panchor	(1.0, 0.5) if vertical; (0.5, 0.0) if horizontal; the anchor point of the colorbar parent axes. If False, the parent axes' anchor will be unchanged

colorbar properties:

Property	Description
extend	['neither' | 'both' | 'min' | 'max'] If not 'neither', make pointed end(s) for out-of- range values. These are set for a given colormap using the colormap set_under and set_over methods.
extendfrac	[*None* | 'auto' | length | lengths] If set to *None*, both the minimum and maximum triangular colorbar extensions with have a length of 5% of the interior colorbar length (this is the default setting). If set to 'auto', makes the triangular colorbar extensions the same lengths as the interior boxes (when *spacing* is set to 'uniform') or the same lengths as the respective adjacent interior boxes (when *spacing* is set to 'proportional'). If a scalar, indicates the length of both the minimum and maximum triangular colorbar extensions as a fraction of the interior colorbar length. A two-element sequence of fractions may also be given, indicating the lengths of the minimum and maximum colorbar extensions respectively as a fraction of the interior colorbar length.
extendrect	[*False* | *True*] If *False* the minimum and maximum colorbar extensions will be triangular (the default). If *True* the extensions will be rectangular.
spacing	['uniform' | 'proportional'] Uniform spacing gives each discrete color the same space; proportional makes the space proportional to the data interval.
ticks	[None | list of ticks | Locator object] If None, ticks are determined automatically from the input.
format	[None | format string | Formatter object] If None, the `ScalarFormatter` is used. If a format string is given, e.g., '%.3f', that is used. An alternative `Formatter` object may be given instead.
drawedges	[False | True] If true, draw lines at color boundaries.

The following will probably be useful only in the context of indexed colors (that is, when the mappable has norm=NoNorm()), or other unusual circumstances.

Property	Description
boundaries	None or a sequence
values	None or a sequence which must be of length 1 less than the sequence of *boundaries*. For each region delimited by adjacent entries in *boundaries*, the color mapped to the corresponding value in values will be used.

If *mappable* is a `ContourSet`, its *extend* kwarg is included automatically.

Note that the *shrink* kwarg provides a simple way to keep a vertical colorbar, for example, from being taller than the axes of the mappable to which the colorbar is attached; but it is a manual method requiring some trial and error. If the colorbar is too tall (or a horizontal colorbar is too wide) use a smaller value of *shrink*.

For more precise control, you can manually specify the positions of the axes objects in which the mappable and the colorbar are drawn. In this case, do not use any of the axes properties kwargs.

It is known that some vector graphics viewer (svg and pdf) renders white gaps between segments of the colorbar. This is due to bugs in the viewers not matplotlib. As a workaround the colorbar can be rendered with overlapping segments:

```
cbar = colorbar()
cbar.solids.set_edgecolor("face")
draw()
```

However this has negative consequences in other circumstances. Particularly with semi transparent images (alpha < 1) and colorbar extensions and is not enabled by default see (issue #1188).

Returns `Colorbar` instance; see also its base class, `ColorbarBase`. Call the `set_label()` method to label the colorbar.

contains(*mouseevent*)
Test whether the mouse event occurred on the figure.

Returns True,{}

delaxes(*a*)
remove a from the figure and update the current axes

dpi

draw(*renderer*, **args*, ***kwargs*)
Render the figure using `matplotlib.backend_bases.RendererBase` instance *renderer*.

draw_artist(*a*)
draw `matplotlib.artist.Artist` instance *a* only – this is available only after the figure is drawn

figimage(*X*, *xo=0*, *yo=0*, *alpha=None*, *norm=None*, *cmap=None*, *vmin=None*, *vmax=None*, *origin=None*, *resize=False*, ***kwargs*)
Adds a non-resampled image to the figure.

call signatures:

```
figimage(X, **kwargs)
```

adds a non-resampled array *X* to the figure.

```
figimage(X, xo, yo)
```

with pixel offsets *xo*, *yo*,

X must be a float array:

- If *X* is MxN, assume luminance (grayscale)
- If *X* is MxNx3, assume RGB
- If *X* is MxNx4, assume RGBA

Optional keyword arguments:

Keyword	Description
resize	a boolean, True or False. If "True", then re-size the Figure to match the given image size.
xo or yo	An integer, the *x* and *y* image offset in pixels
cmap	a `matplotlib.colors.Colormap` instance, e.g., cm.jet. If *None*, default to the rc `image.cmap` value
norm	a `matplotlib.colors.Normalize` instance. The default is normalization(). This scales luminance -> 0-1
vmin|vmax	used to scale a luminance image to 0-1. If either is *None*, the min and max of the luminance values will be used. Note if you pass a norm instance, the settings for *vmin* and *vmax* will be ignored.
alpha	the alpha blending value, default is *None*
origin	['upper' | 'lower'] Indicates where the [0,0] index of the array is in the upper left or lower left corner of the axes. Defaults to the rc image.origin value

figimage complements the axes image (`imshow()`) which will be resampled to fit the current axes. If you want a resampled image to fill the entire figure, you can define an `Axes` with size [0,1,0,1].

An `matplotlib.image.FigureImage` instance is returned.

Additional kwargs are Artist kwargs passed on to `FigureImage`

gca (***kwargs*)

Get the current axes, creating one if necessary

The following kwargs are supported for ensuring the returned axes adheres to the given projection etc., and for axes creation if the active axes does not exist:

adjustable: ['box' | 'datalim' | 'box-forced'] agg_filter: unknown alpha: float (0.0 transparent through 1.0 opaque) anchor: unknown animated: [True | False] aspect: unknown autoscale_on: unknown autoscalex_on: unknown autoscaley_on: unknown axes: an `Axes` instance axes_locator: unknown axis_bgcolor: any matplotlib color - see `colors()` axisbelow: [*True* | *False*] clip_box: a `matplotlib.transforms.Bbox` instance clip_on: [True | False] clip_path: [(`Path`, `Transform`) | `Patch` | None] color_cycle: unknown contains: a callable function figure: unknown frame_on: [*True* | *False*] gid: an id string label: string or anything printable with '%s' conversion. navigate: [*True* | *False*] navigate_mode: unknown path_effects: unknown picker: [None|float|boolean|callable] position: unknown rasterization_zorder: unknown rasterized: [True | False | None] sketch_params: unknown snap: unknown title: unknown transform: `Transform` instance url: a url string visible: [True | False] xbound: unknown xlabel: unknown xlim: length 2 sequence of floats xmargin: unknown xscale: [u'linear' | u'log' | u'logit' | u'symlog'] xticklabels: sequence of strings xticks: sequence of floats ybound: unknown ylabel:

unknown ylim: length 2 sequence of floats ymargin: unknown yscale: [u'linear' | u'log' | u'logit' | u'symlog'] yticklabels: sequence of strings yticks: sequence of floats zorder: any number

get_axes()

get_children()
get a list of artists contained in the figure

get_default_bbox_extra_artists()

get_dpi()
Return the dpi as a float

get_edgecolor()
Get the edge color of the Figure rectangle

get_facecolor()
Get the face color of the Figure rectangle

get_figheight()
Return the figheight as a float

get_figwidth()
Return the figwidth as a float

get_frameon()
get the boolean indicating frameon

get_size_inches()
Returns the current size of the figure in inches (1in == 2.54cm) as an numpy array.

Returns **size** – The size of the figure in inches

Return type ndarray

See also:

matplotlib.Figure.set_size_inches()

get_tight_layout()
Return the Boolean flag, True to use :meth'tight_layout' when drawing.

get_tightbbox(*renderer*)
Return a (tight) bounding box of the figure in inches.

It only accounts axes title, axis labels, and axis ticklabels. Needs improvement.

get_window_extent(**args*, ***kwargs*)
get the figure bounding box in display space; kwargs are void

ginput(*n=1*, *timeout=30*, *show_clicks=True*, *mouse_add=1*, *mouse_pop=3*, *mouse_stop=2*)
Call signature:

```
ginput(self, n=1, timeout=30, show_clicks=True,
       mouse_add=1, mouse_pop=3, mouse_stop=2)
```

Blocking call to interact with the figure.

This will wait for *n* clicks from the user and return a list of the coordinates of each click.

If *timeout* is zero or negative, does not timeout.

If *n* is zero or negative, accumulate clicks until a middle click (or potentially both mouse buttons at once) terminates the input.

Right clicking cancels last input.

The buttons used for the various actions (adding points, removing points, terminating the inputs) can be overriden via the arguments *mouse_add*, *mouse_pop* and *mouse_stop*, that give the associated mouse button: 1 for left, 2 for middle, 3 for right.

The keyboard can also be used to select points in case your mouse does not have one or more of the buttons. The delete and backspace keys act like right clicking (i.e., remove last point), the enter key terminates input and any other key (not already used by the window manager) selects a point.

hold(*b=None*)

Set the hold state. If hold is None (default), toggle the hold state. Else set the hold state to boolean value b.

e.g.:

```
hold()       # toggle hold
hold(True)   # hold is on
hold(False)  # hold is off
```

legend(*handles*, *labels*, **args*, ***kwargs*)

Place a legend in the figure. Labels are a sequence of strings, handles is a sequence of `Line2D` or `Patch` instances, and loc can be a string or an integer specifying the legend location

USAGE:

```
legend( (line1, line2, line3),
        ('label1', 'label2', 'label3'),
        'upper right')
```

The *loc* location codes are:

```
'best' : 0,          (currently not supported for figure legends)
'upper right'  : 1,
'upper left'   : 2,
'lower left'   : 3,
'lower right'  : 4,
'right'        : 5,
'center left'  : 6,
'center right' : 7,
'lower center' : 8,
'upper center' : 9,
'center'       : 10,
```

loc can also be an (x,y) tuple in figure coords, which specifies the lower left of the legend box. figure coords are (0,0) is the left, bottom of the figure and 1,1 is the right, top.

Keyword Arguments

- ***prop*** – [*None* | FontProperties | dict] A `matplotlib.font_manager.FontProperties` instance. If *prop* is a dictionary, a new instance will be created with *prop*. If *None*, use rc settings.
- ***numpoints*** – integer The number of points in the legend line, default is 4
- ***scatterpoints*** – integer The number of points in the legend line, default is 4
- ***scatteryoffsets*** – list of floats a list of yoffsets for scatter symbols in legend
- ***markerscale*** – [*None* | scalar] The relative size of legend markers vs. original. If *None*, use rc settings.
- ***markerfirst*** – [*True* | *False*] if *True*, legend marker is placed to the left of the legend label if *False*, legend marker is placed to the right of the legend label

- ***fancybox*** – [*None* | *False* | *True*] if *True*, draw a frame with a round fancybox. If *None*, use rc
- ***shadow*** – [*None* | *False* | *True*] If *True*, draw a shadow behind legend. If *None*, use rc settings.
- ***ncol*** – integer number of columns. default is 1
- ***mode*** – ["expand" | *None*] if mode is "expand", the legend will be horizontally expanded to fill the axes area (or *bbox_to_anchor*)
- ***title*** – string the legend title

Padding and spacing between various elements use following keywords parameters. The dimensions of these values are given as a fraction of the fontsize. Values from rcParams will be used if None.

Keyword	Description
borderpad	the fractional whitespace inside the legend border
labelspacing	the vertical space between the legend entries
handlelength	the length of the legend handles
handletextpad	the pad between the legend handle and text
borderaxespad	the pad between the axes and legend border
columnspacing	the spacing between columns

Note: Not all kinds of artist are supported by the legend. See LINK (FIXME) for details.

Example:

savefig (**args*, ***kwargs*)

Save the current figure.

Call signature:

```
savefig(fname, dpi=None, facecolor='w', edgecolor='w',
        orientation='portrait', papertype=None, format=None,
        transparent=False, bbox_inches=None, pad_inches=0.1,
        frameon=None)
```

The output formats available depend on the backend being used.

Parameters ***fname*** – A string containing a path to a filename, or a Python file-like object, or possibly some backend-dependent object such as `PdfPages`.

If *format* is *None* and *fname* is a string, the output format is deduced from the extension of the filename. If the filename has no extension, the value of the rc parameter `savefig.format` is used.

If *fname* is not a string, remember to specify *format* to ensure that the correct backend is used.

Keyword Arguments

- ***dpi*** – [*None* | `scalar > 0` | 'figure'] The resolution in dots per inch. If *None* it will default to the value `savefig.dpi` in the matplotlibrc file. If 'figure' it will set the dpi to be the value of the figure.
- **edgecolor** (*`*facecolor*,`*) – the colors of the figure rectangle

- ***orientation*** – ['landscape' | 'portrait'] not supported on all backends; currently only on postscript output
- ***papertype*** – One of 'letter', 'legal', 'executive', 'ledger', 'a0' through 'a10', 'b0' through 'b10'. Only supported for postscript output.
- ***format*** – One of the file extensions supported by the active backend. Most backends support png, pdf, ps, eps and svg.
- ***transparent*** – If *True*, the axes patches will all be transparent; the figure patch will also be transparent unless facecolor and/or edgecolor are specified via kwargs. This is useful, for example, for displaying a plot on top of a colored background on a web page. The transparency of these patches will be restored to their original values upon exit of this function.
- ***frameon*** – If *True*, the figure patch will be colored, if *False*, the figure background will be transparent. If not provided, the rcParam 'savefig.frameon' will be used.
- ***bbox_inches*** – Bbox in inches. Only the given portion of the figure is saved. If 'tight', try to figure out the tight bbox of the figure.
- ***pad_inches*** – Amount of padding around the figure when bbox_inches is 'tight'.
- ***bbox_extra_artists*** – A list of extra artists that will be considered when the tight bbox is calculated.

sca(*a*)
Set the current axes to be a and return a

set_canvas(*canvas*)
Set the canvas that contains the figure

ACCEPTS: a FigureCanvas instance

set_dpi(*val*)
Set the dots-per-inch of the figure

ACCEPTS: float

set_edgecolor(*color*)
Set the edge color of the Figure rectangle

ACCEPTS: any matplotlib color - see help(colors)

set_facecolor(*color*)
Set the face color of the Figure rectangle

ACCEPTS: any matplotlib color - see help(colors)

set_figheight(*val*, *forward=False*)
Set the height of the figure in inches

ACCEPTS: float

set_figwidth(*val*, *forward=False*)
Set the width of the figure in inches

ACCEPTS: float

set_frameon(*b*)
Set whether the figure frame (background) is displayed or invisible

ACCEPTS: boolean

set_size_inches(*w*, *h*, *forward=False*)
Set the figure size in inches (1in == 2.54cm)

Usage:

```
fig.set_size_inches(w,h)  # OR
fig.set_size_inches((w,h) )
```

optional kwarg *forward=True* will cause the canvas size to be automatically updated; e.g., you can resize the figure window from the shell

ACCEPTS: a w,h tuple with w,h in inches

See also:

`matplotlib.Figure.get_size_inches()`

set_tight_layout(*tight*)
Set whether *tight_layout()* is used upon drawing. If None, the rcParams['figure.autolayout'] value will be set.

When providing a dict containing the keys *pad*, *w_pad*, *h_pad* and *rect*, the default *tight_layout()* paddings will be overridden.

ACCEPTS: [True | False | dict | None]

show(*warn=True*)
If using a GUI backend with pyplot, display the figure window.

If the figure was not created using `figure()`, it will lack a `FigureManagerBase`, and will raise an AttributeError.

For non-GUI backends, this does nothing, in which case a warning will be issued if *warn* is True (default).

subplots_adjust(**args*, ***kwargs*)
Call signature:

```
subplots_adjust(left=None, bottom=None, right=None, top=None,
                    wspace=None, hspace=None)
```

Update the `SubplotParams` with *kwargs* (defaulting to rc when *None*) and update the subplot locations

suptitle(*t*, ***kwargs*)
Add a centered title to the figure.

kwargs are `matplotlib.text.Text` properties. Using figure coordinates, the defaults are:

x [0.5] The x location of the text in figure coords

y [0.98] The y location of the text in figure coords

horizontalalignment ['center'] The horizontal alignment of the text

verticalalignment ['top'] The vertical alignment of the text

A `matplotlib.text.Text` instance is returned.

Example:

```
fig.suptitle('this is the figure title', fontsize=12)
```

text(*x*, *y*, *s*, **args*, ***kwargs*)
Add text to figure.

Call signature:

```
text(x, y, s, fontdict=None, **kwargs)
```

Add text to figure at location *x*, *y* (relative 0-1 coords). See `text()` for the meaning of the other arguments.

kwargs control the `Text` properties:

agg_filter: unknown alpha: float (0.0 transparent through 1.0 opaque) animated: [True | False] axes: an `Axes` instance backgroundcolor: any matplotlib color bbox: FancyBboxPatch prop dict clip_box: a `matplotlib.transforms.Bbox` instance clip_on: [True | False] clip_path: [(`Path`, `Transform`) | `Patch` | None] color: any matplotlib color contains: a callable function family or fontfamily or fontname or name: [FONTNAME | 'serif' | 'sans-serif' | 'cursive' | 'fantasy' | 'monospace'] figure: a `matplotlib.figure.Figure` instance fontproperties or font_properties: a `matplotlib.font_manager.FontProperties` instance gid: an id string horizontalalignment or ha: ['center' | 'right' | 'left'] label: string or anything printable with '%s' conversion. linespacing: float (multiple of font size) multialignment: ['left' | 'right' | 'center'] path_effects: unknown picker: [None|float|boolean|callable] position: (x,y) rasterized: [True | False | None] rotation: [angle in degrees | 'vertical' | 'horizontal'] rotation_mode: unknown size or fontsize: [size in points | 'xx-small' | 'x-small' | 'small' | 'medium' | 'large' | 'x-large' | 'xx-large'] sketch_params: unknown snap: unknown stretch or fontstretch: [a numeric value in range 0-1000 | 'ultra-condensed' | 'extra-condensed' | 'condensed' | 'semi-condensed' | 'normal' | 'semi-expanded' | 'expanded' | 'extra-expanded' | 'ultra-expanded'] style or fontstyle: ['normal' | 'italic' | 'oblique'] text: string or anything printable with '%s' conversion. transform: `Transform` instance url: a url string usetex: unknown variant or fontvariant: ['normal' | 'small-caps'] verticalalignment or va or ma: ['center' | 'top' | 'bottom' | 'baseline'] visible: [True | False] weight or fontweight: [a numeric value in range 0-1000 | 'ultralight' | 'light' | 'normal' | 'regular' | 'book' | 'medium' | 'roman' | 'semibold' | 'demibold' | 'demi' | 'bold' | 'heavy' | 'extra bold' | 'black'] wrap: unknown x: float y: float zorder: any number

tight_layout(*renderer=None*, *pad=1.08*, *h_pad=None*, *w_pad=None*, *rect=None*)

Adjust subplot parameters to give specified padding.

Parameters

- ***pad*** – float padding between the figure edge and the edges of subplots, as a fraction of the font-size.
- **w_pad** (**h_pad*,*) – float padding (height/width) between edges of adjacent subplots. Defaults to *pad_inches*.
- ***rect*** – if rect is given, it is interpreted as a rectangle (left, bottom, right, top) in the normalized figure coordinate that the whole subplots area (including labels) will fit into. Default is (0, 0, 1, 1).

waitforbuttonpress(*timeout=-1*)

Call signature:

```
waitforbuttonpress(self, timeout=-1)
```

Blocking call to interact with the figure.

This will return True is a key was pressed, False if a mouse button was pressed and None if *timeout* was reached without either being pressed.

If *timeout* is negative, does not timeout.

`VirtualMicrobes.my_tools.monkey.`**MyArtist**

alias of *VirtualMicrobes.my_tools.monkey.Artist*

VirtualMicrobes.my_tools.monkey.**MyFigure**
alias of *VirtualMicrobes.my_tools.monkey.Figure*

VirtualMicrobes.my_tools.monkey.**MyTransformWrapper**
alias of *VirtualMicrobes.my_tools.monkey.TransformWrapper*

class VirtualMicrobes.my_tools.monkey.**TransformWrapper**(*child*)
Bases: matplotlib.transforms.Transform

A helper class that holds a single child transform and acts equivalently to it.

This is useful if a node of the transform tree must be replaced at run time with a transform of a different type. This class allows that replacement to correctly trigger invalidation.

Note that *TransformWrapper* instances must have the same input and output dimensions during their entire lifetime, so the child transform may only be replaced with another child transform of the same dimensions.

frozen()
Returns a frozen copy of this transform node. The frozen copy will not update when its children change. Useful for storing a previously known state of a transform where copy.deepcopy() might normally be used.

has_inverse

is_affine

is_separable

pass_through = True

set(*child*)
Replace the current child of this transform with another one.

The new child must have the same number of input and output dimensions as the current child.

VirtualMicrobes.my_tools.monkey.**monkeypatch_class**(*name*, *bases*, *namespace*)
https://mail.python.org/pipermail/python-dev/2008-January/076194.html

WHOOOHHHAAAAA this is a Monkey Patch. Eat that, unreliable package developers!

VirtualMicrobes.my_tools.utility module

class VirtualMicrobes.my_tools.utility.**CircularList**
Bases: list

A list that wraps around instead of throwing an index error.

Works like a regular list: >>> cl = CircularList([1,2,3]) >>> cl [1, 2, 3] >>> cl[0] 1 >>> cl[-1] 3 >>> cl[2] 3 Except wraps around: >>> cl[3] 1 >>> cl[-4] 3 Slices work >>> cl[0:2]

class VirtualMicrobes.my_tools.utility.**Consumer**(*task_queue*, *result_queue*, *task_timeout=120*)
Bases: multiprocessing.process.Process

Consumer Process that gets jobs from a Queue until receiving a 'poison pill' job that will terminate the process.

Jobs will timeout after a given time.

run()
Method to be run in sub-process; can be overridden in sub-class

class VirtualMicrobes.my_tools.utility.**Coord**(*x*, *y*)
Bases: tuple

x
Alias for field number 0

y
Alias for field number 1

class VirtualMicrobes.my_tools.utility.**ETEtreeStruct**(*tree*, *named_node_dict*, *node_name_to_phylo_node*)
Bases: tuple

named_node_dict
Alias for field number 1

node_name_to_phylo_node
Alias for field number 2

tree
Alias for field number 0

class VirtualMicrobes.my_tools.utility.**FIFOLarder**(*filename*, *flag='c'*, *protocol=None*)
Bases: shelve.Shelf

Maintain a hybrid cached/pickled dictionary that presents as a normal dictionary. Postpone pickling of dictionary values until (partial)sync calls. This requires both the pickled and the cached dict to be checked when lookup en set operations are applied. The cache is an OrderedDict so that it can be 'partially' synced to pickle in FIFO order. This structure is usefull when new data coming in is still handled often and may regularly change, while old data can be considered more stable and less likely to change, and thus amenable for more permanent storage.

close()

get(*key*, *default=None*)

has_key(*key*)

keys()

partial_sync(*part*, *select=<function <lambda>>*)
sync a part of the dict to the pickled representation.

@param part: can be an int or a float, specifying how much of cache will be synced in FIFO order.

reopen_db(*save_file=None*, *flag='c'*)

sync()

class VirtualMicrobes.my_tools.utility.**FormatMessageFile**(*name*, *mode='r'*, *replace_dict={'x08': '', 'r': 'n', '\x1b\[([0-9, A-Z]{1, 2}(;[0-9]{1, 2})?(;[0-9]{3})?)?[m|K]?': ''}*, ***kwargs*)
Bases: file

Class to remove console string formatting when writing to a file.

It is useful when redirecting stdout both to terminal and to a file. When writing to a file the special console formatting characters should be removed from the message line.

write(*str*) → None. Write string str to file.
Note that due to buffering, flush() or close() may be needed before the file on disk reflects the data written.

class VirtualMicrobes.my_tools.utility.**GeneProduct**(**iargs*, ***ikwargs*)
Bases: object

concentration

cum_time_course

degradation

diffusion

multiplicity

time_course

class VirtualMicrobes.my_tools.utility.**GeneTypeNumbers**(*tf*, *enz*, *pump*)
Bases: tuple

enz
Alias for field number 1

pump
Alias for field number 2

tf
Alias for field number 0

class VirtualMicrobes.my_tools.utility.**GridPos**(*row*, *col*)
Bases: tuple

col
Alias for field number 1

row
Alias for field number 0

class VirtualMicrobes.my_tools.utility.**GridSubDiv**(*row*, *col*)
Bases: tuple

col
Alias for field number 1

row
Alias for field number 0

class VirtualMicrobes.my_tools.utility.**LinkThroughSequence**(*data=[]*)
Bases: list

A list that applies an arbitrary element function before returning and storing

append(*val*)
L.append(object) – append object to end

insert(*ii*, *val*)
L.insert(index, object) – insert object before index

pop([*index*]) → item – remove and return item at index (default last).
Raises IndexError if list is empty or index is out of range.

remove(*val*)
L.remove(value) – remove first occurrence of value. Raises ValueError if the value is not present.

class VirtualMicrobes.my_tools.utility.**LinkThroughSet**(*data=[]*)
Bases: set

A dictionary that applies an arbitrary key-altering function before accessing the keys

add(*el*)
Add an element to a set.

This has no effect if the element is already present.

copy()
Return a shallow copy of a set.

difference(*new*)

difference_update(*new*)

intersection(*new*)

pop()
Remove and return an arbitrary set element. Raises KeyError if the set is empty.

symmetric_difference(*new*)

union(*new*)

update(*new*)

class VirtualMicrobes.my_tools.utility.**MutationParamSpace**(*base*, *lower*, *upper*, *min*, *max*, *uniform*, *randomize*)

Bases: tuple

base
Alias for field number 0

lower
Alias for field number 1

max
Alias for field number 4

min
Alias for field number 3

randomize
Alias for field number 6

uniform
Alias for field number 5

upper
Alias for field number 2

class VirtualMicrobes.my_tools.utility.**MutationRates**(*chrom_dup*, *chrom_del*, *chrom_fiss*, *chrom_fuse*, *point_mutation*, *tandem_dup*, *stretch_del*, *stretch_invert*, *stretch_translocate*, *stretch_exp_lambda*, *external_hgt*, *internal_hgt*, *regulatory_mutation*, *reg_stretch_exp_lambda*, *uptake_mutrate*)

Bases: tuple

chrom_del
Alias for field number 1

chrom_dup
Alias for field number 0

chrom_fiss
Alias for field number 2

chrom_fuse
Alias for field number 3

external_hgt
Alias for field number 10

internal_hgt
Alias for field number 11

point_mutation
Alias for field number 4

reg_stretch_exp_lambda
Alias for field number 13

regulatory_mutation
Alias for field number 12

stretch_del
Alias for field number 6

stretch_exp_lambda
Alias for field number 9

stretch_invert
Alias for field number 7

stretch_translocate
Alias for field number 8

tandem_dup
Alias for field number 5

uptake_mutrate
Alias for field number 14

class VirtualMicrobes.my_tools.utility.**OrderedDefaultdict**(**args*, ***kwargs*)
Bases: collections.OrderedDict

class VirtualMicrobes.my_tools.utility.**OrderedSet**(*iterable=None*)
Bases: _abcoll.MutableSet

add(*key*)
Add an element.

discard(*key*)
Remove an element. Do not raise an exception if absent.

pop(*last=True*)
Return the popped value. Raise KeyError if empty.

class VirtualMicrobes.my_tools.utility.**ParamSpace**(*base*, *lower*, *upper*)
Bases: tuple

base
Alias for field number 0

lower
Alias for field number 1

upper
Alias for field number 2

class VirtualMicrobes.my_tools.utility.**PartialLinkThroughDict**(*linker_dict*, **args*, ***kwargs*)

Bases: _abcoll.MutableMapping

Values objects can have a '_unique_key' attribute, in which case storage of the value is linked through to the linker_dict. Otherwise, the value goes into the local storage of the PartialLinkThroughDict instance.

close_larder()

partial_sync(*part*)

set_larder(*larder*)

class VirtualMicrobes.my_tools.utility.**PointMutationRatios**(*v_max*, *ene_ks*, *subs_ks*, *exporting*, *promoter*, *operator*, *eff_bound*, *eff_apo*, *ligand_ks*, *k_bind_op*, *ligand_class*, *bind*, *sense_external*)

Bases: tuple

bind
Alias for field number 11

eff_apo
Alias for field number 7

eff_bound
Alias for field number 6

ene_ks
Alias for field number 1

exporting
Alias for field number 3

k_bind_op
Alias for field number 9

ligand_class
Alias for field number 10

ligand_ks
Alias for field number 8

operator
Alias for field number 5

promoter
Alias for field number 4

sense_external
Alias for field number 12

subs_ks
Alias for field number 2

v_max
Alias for field number 0

class VirtualMicrobes.my_tools.utility.**PopulationWipe**(*interval*, *fraction*)
Bases: tuple

fraction
Alias for field number 1

interval
Alias for field number 0

class VirtualMicrobes.my_tools.utility.**ReactionScheme**(*reactants*, *products*)
Bases: tuple

products
Alias for field number 1

reactants
Alias for field number 0

class VirtualMicrobes.my_tools.utility.**RegulatorytMutationRatios**(*translocate*, *random_insert*)
Bases: tuple

random_insert
Alias for field number 1

translocate
Alias for field number 0

class VirtualMicrobes.my_tools.utility.**ReusableIndexDict**(*keys=[]*, *fixed_length=None*, *randomized=True*)
Bases: object

class_version = '0.0'

index_key(*key*)

init_rand_gen()

keys()

remove_key(*key*)

upgrade()
Upgrading from older pickled version of class to latest version. Version information is saved as class variable and should be updated when class invariants (e.g. fields) are added. Adapted from recipe at http://code.activestate.com/recipes/521901-upgradable-pickles/

class VirtualMicrobes.my_tools.utility.**SmallMol**(**iargs*, ***ikwargs*)
Bases: object

concentration

cum_time_course

degradation

diffusion

influx

time_course

VirtualMicrobes.my_tools.utility.**Struct**(**args*, ***kwargs*)

class VirtualMicrobes.my_tools.utility.**SubEnv**(*influx_dict*, *sub_grid*)
Bases: tuple

influx_dict
Alias for field number 0

sub_grid
Alias for field number 1

class VirtualMicrobes.my_tools.utility.**Task**(*obj*, *func_name*, *args=[]*, *kwargs={}*)
Bases: object

Task object class used to present jobs to Consumer processes. Tasks are assumed to be method functions associated with a class instance.

Task will except all errors generated by the method call so that the Consumer processing the Task will stay alive.

class VirtualMicrobes.my_tools.utility.**TracePrints**
Bases: object

flush()

write(*s*)

exception VirtualMicrobes.my_tools.utility.**ValueNotInRange**
Bases: exceptions.ValueError

VirtualMicrobes.my_tools.utility.**as_attrdict**(*val*)

VirtualMicrobes.my_tools.utility.**chunks**(*l*, *n*)
Yield successive n-sized chunks from l.

VirtualMicrobes.my_tools.utility.**detect_rel_path_change**(*old_save_dir*, *load_file*, *mount_path_depth=4*, *abs_root='/linuxhome/tmp/'*)
Detects the usage of a mounted path versus the abolute path on the server.

old_save_dir [path] Original simulation save path

load_file [file_path] Full path of the load file

mount_path_depth : path

Parameters **abs_root** –

VirtualMicrobes.my_tools.utility.**detect_sim_folder_move**(*stored_save_dir*, *load_path*)
Detect whether the load_path was placed in a different location from the original simulation path.

Parameters

- **stored_save_dir** (*path*) – original simulation directory
- **load_path** (*path*) – load file

VirtualMicrobes.my_tools.utility.**ensure_dir**(*d*, *message=None*, *remove_globs=[]*)

VirtualMicrobes.my_tools.utility.**errand_boy_server**(**args*, ***kwds*)

VirtualMicrobes.my_tools.utility.**flatten**(*nested_lists*)

VirtualMicrobes.my_tools.utility.**get_from_linker**(*key*)

VirtualMicrobes.my_tools.utility.**get_subpackages**(*module*)
Find all subpackages of a package/module

VirtualMicrobes.my_tools.utility.**get_unique_key**(*val*)

VirtualMicrobes.my_tools.utility.**grow_array**(*ar*, *factor=1.25*)

VirtualMicrobes.my_tools.utility.**json_dumper**(*obj*, **args*, ***kwargs*)

VirtualMicrobes.my_tools.utility.**kill_proc_tree**(*pid*, *including_parent=True*)

VirtualMicrobes.my_tools.utility.**linkthrough**(*f*)

VirtualMicrobes.my_tools.utility.**map_backward_func**(*key*)

VirtualMicrobes.my_tools.utility.**map_forward_func**(*val*)

VirtualMicrobes.my_tools.utility.**map_old_package_path**(*mod_name*, *kls_name*)
Mapping function for unpickler to map old to new package structure and returning the correct class.

This function works particularly for the new VirtualMicrobes package structure.

class VirtualMicrobes.my_tools.utility.**multifile**(*files*)
Bases: object

VirtualMicrobes.my_tools.utility.**namedtuple_with_defaults**(*typename*, *field_names*, *default_values=()*)

VirtualMicrobes.my_tools.utility.**open_fifolarder**(*filename*, *flag='c'*, *protocol=None*)

VirtualMicrobes.my_tools.utility.**opt_profile**(**args*, ***kwargs*)
Decorator for optional profiling.

If a global 'profile' flag has been set, profiling will be done by the profilestats decorator, if available. Else a dummy is applied and no profiling is done.

VirtualMicrobes.my_tools.utility.**pad_frequencies**(*a*, *pad_to*, *pad_with=(nan, nan)*)

VirtualMicrobes.my_tools.utility.**padded_most_frequent**(*a*, *freq_cutoff*)

class VirtualMicrobes.my_tools.utility.**pickles_adict**(**args*, ***kwargs*)
Bases: attrdict.AttrDict

VirtualMicrobes.my_tools.utility.**processify**(*func*)
Decorator to run a function as a process.

Be sure that every argument and the return value is *pickable*. The created process is joined, so the code does not run in parallel.

VirtualMicrobes.my_tools.utility.**queuedprocessor**(*as_subprocess=True*)
Decorator that can can cause a decorated method to be returned as a Task tuple, depending on the decorator argument.

VirtualMicrobes.my_tools.utility.**roulette_wheel_draw**(*events_rel_chances*, *rand_nr*, *non=0.0*)

VirtualMicrobes.my_tools.utility.**same_content**(*dir1*, *dir2*, *verbose=False*)
Compare two directory trees content. Return False if they differ, True if they are the same.

VirtualMicrobes.my_tools.utility.**sdi**(*data*)
Given a hash { 'species': count } , returns the SDI

```
>>> sdi({'a': 10, 'b': 20, 'c': 30,})
1.0114042647073518
```

VirtualMicrobes.my_tools.utility.**subprocessor**(*as_subprocess=True*)
Decorator that can can cause a decorated function or method to be returned as a subprocess or simply evaluated, depending on the decorator argument.

VirtualMicrobes.my_tools.utility.**time_course_array**(*length*)

class VirtualMicrobes.my_tools.utility.**timeout**(*seconds=1*, *error_message='Timeout'*)

handle_timeout(*signum*, *frame*)

VirtualMicrobes.my_tools.utility.**unique_count**(*a*)
Count values and return value-count pairs sorted on counts (highest first) :param a:

VirtualMicrobes.my_tools.utility.**within_range**(*val*, *rng*)

Module contents

VirtualMicrobes.plotting package

Submodules

VirtualMicrobes.plotting.Graphs module

class VirtualMicrobes.plotting.Graphs.**AttributeMap**(*mol_class_dict*, *reactions_dict*, *species_markers*)

Bases: object

activation_color(*effect*, *domain=(-1, 1)*)

color_mol(*mol*)

color_mol_class(*mc*)

color_protein(*g*)

color_reaction(*r*)

init_color_maps(*species_markers*)

init_mol_class_color_dict(*mol_class_dict*)

protein_type_line_style(*type_*)

class VirtualMicrobes.plotting.Graphs.**BindingNetwork**(*base_save_dir*, *name*, *attribute_dict=None*, *size=(35, 35)*, ***kwargs*)

Bases: *VirtualMicrobes.plotting.Graphs.Network*

draw_network(*self_marker=None*, *edge_effect='effect'*)

init_network(*cell*, *with_self_marker=True*)

layout_network_positions(*prog*)
Compute and store node positions using a layout algorithm.

Parameters **prog** (*str*) – layout algorithm

class VirtualMicrobes.plotting.Graphs.**Genome**(*base_save_dir*, *name*, *attribute_dict*, *size=(10, 10)*, *show=True*, ***kwargs*)

Bases: *VirtualMicrobes.plotting.Graphs.Grapher*

plot_genome_structure(*cell*, *labels*, *video=None*, *max_size=None*)
Render the genome structure as a circular (plasmid-like) representation. :param cell: which genome is plotted

exception VirtualMicrobes.plotting.Graphs.**GraphNotFoundError**(*value*)
Bases: *VirtualMicrobes.plotting.Graphs.GraphingError*

class VirtualMicrobes.plotting.Graphs.**Grapher**(*base_save_dir*, *name*, *show=True*, *attribute_dict=None*, *clean=True*, *create=True*)
Bases: object

backup_plots()

change_save_location(*base_dir=None*, *name=None*, *clean=False*, *create=True*)

class_version = '1.0'

init_attribute_mapper(*mol_class_dict*, *reactions_dict*, *species_markers*)

init_save_dir(*clean=False*, *create=True*)

save_dir

save_fig(*name=None*, *labels=[]*, *title=None*, *suffix='.svg'*, *copy_labels=None*, ***kwargs*)
Render and save a figure.

Parameters

- **name** (*str*) – base name of the figure
- **labels** (*iterable*) – additional labels in file name
- **title** (*str*) – printable title of figure
- **suffix** (*str*) – file extension suffix
- **copy_labels** (*iterable*) – additional labels for the filename of a copy of the figure

Returns

Return type list of filenames of the saved figure(s)

save_fig2(*ext*, *name=None*, *title=None*, ***kwargs*)
Render and save a figure.

Parameters

- **name** (*str*) – base name of the figure
- **title** (*str*) – printable title of figure
- **suffix** (*str*) – file extension suffix

Returns

Return type filename of the saved figure

save_video(*video=None*, *frame=None*)
Concat last plot to existing video (or, if first plot is made, make single framed video)

Parameters

- **video** (*str*) – alias to ffmpeg that is found during initialisation
- **frane** (*str*) – frame number
- **suffix** (*str*) – file extension suffix

Returns

Return type filename of the saved figure

update_figure(*show=None*)

upgrade(*odict*)
Upgrading from older pickled version of class to latest version. Version information is saved as class variable and should be updated when class invariants (e.g. fields) are added. (see also __setstate__)

Adapted from recipe at http://code.activestate.com/recipes/521901-upgradable-pickles/

exception VirtualMicrobes.plotting.Graphs.**GraphingError**(*value*)
Bases: exceptions.Exception

class VirtualMicrobes.plotting.Graphs.**Graphs**(*base_save_dir*, *name*, *utility_path*, *mol_class_dict*, *reactions_dict*, *population_markers*, *species_markers*, *show*, *clean=True*, *create=True*, ***kwargs*)
Bases: *VirtualMicrobes.plotting.Graphs.Grapher*

Produces static and online graphs of simulated data.

add_grid_graphs_data(*time_point*, *pop_grid_data_dict*, *small_mol_names*, *data_store*, *scaling_dict_updates*, *markers_range_dict*)

add_mol_evo_time_course_data(*time_point*, *ext_res_conc_dict*)

add_multigraph(*graph*)

add_pop_stats_data(*time*, *most_fecundant_death_rate*, *most_fecundant_production*, *data_store*, *high_freq_cutoff=10*)

add_prot_grid_graphs_data(*time_point*, *small_mol_names*, *reaction_dict*, *data_store*)

change_save_location(*base_save_dir=None*, *name=None*, *clean=False*, *create=True*)

init_binding_network(*show=None*, *clean=True*, ***kwargs*)

init_genome_structure(*show=None*, *clean=True*, ***kwargs*)

init_grid_graphs(*mol_class_dict*, *markers=[]*, *nr_cols_markers=3*, *show=None*, *mol_classes_per_row=4*, *clean=True*, ***kwargs*)

init_metabolic_network(*metabolites*, *conversions*, *imports*, *show=None*, *clean=True*, ***kwargs*)

init_phylo_tree_graph(*show=None*, *clean=True*, ***kwargs*)

init_pop_stats(*species_markers*, *reactions_dict*, *mol_class_dict*, *clean=True*, *show=None*, ***kwargs*)

init_prot_grid_graphs(*mol_class_dict*, *reactions_dict*, *nr_cols=4*, *show=None*, *mol_classes_per_row=4*, *reactions_per_row=4*, *clean=True*, ***kwargs*)

init_scaling_dict()

init_time_course_graph(*show=None*, *clean=True*, ***kwargs*)

line_colors(*name*, *nr*)

plot_binding_network(**args*, ***kwargs*)

Parameters

- **cell** – Who to plot
- **prog** – (???)
- **save** – Save the figure Y/N
- **write** – Save the network file (gml, dot, json, etc.)

Nodes: Pumping enzymes are BLUE squares (ip = importing pump, e-p = exporting pump) Generic enzymes are BLUE / TURQUOISE circles Specific enzymes are GREEN / YELLOW / RED circles TFs are BROWN diamonds Thick borders indicates self-binding

Node-labels: Labels: Metabolites with brackets are building blocks Metabolites with asterisks are energy carriers

Edges: Width shows the basal level of transcription for the TF Colours distinquish inhibiting (blue) vs activating (red) effects. Intermediates are white-ish.

!! Note: still needs the ReST references worked out !!

plot_genome_structure(***kwargs*)

plot_grid_concentrations(*dat*)

plot_grid_graphs(**args*, ***kwargs*)

plot_metabolic_network(**args*, ***kwargs*)

plot_mol_class_data(*ax*, *mol_tc_dict*, ***plot_params*)

plot_pop_stats(**args*, ***kwargs*)

plot_prot_data(*ax*, *prot_pert_type_tc_dict*, ***plot_params*)

plot_time_course(**args*, ***kwargs*)

class VirtualMicrobes.plotting.Graphs.**MetabolicNetwork**(*base_save_dir*, *name*, *mol_class_dict*, *conversions*, *imports*, *size=(30, 30)*, *attribute_dict=None*, ***kwargs*)

Bases: *VirtualMicrobes.plotting.Graphs.Network*

color_reactions(*conversions*, *imports*, *building_blocks=[]*, *self_marker=None*, *reac_color_func=None*, *mol_color_func=None*, *edge_color_func=None*)

draw_network(*reactions_dict=None*, *self_marker=None*, *building_blocks=None*)

init_network(*mol_class_dict*, *conversions*, *imports*)

layout_network_positions()

class VirtualMicrobes.plotting.Graphs.**MultiGraph**(*base_save_dir*, *name*, *rows*, *cols*, *row_heigth=2*, *col_width=4*, *attribute_dict=None*, *show=True*, ***kwargs*)

Bases: *VirtualMicrobes.plotting.Graphs.Grapher*

add_axis(*name*, *pos*, *rows=1*, *cols=1*, *min_max_y=None*, *nr_lines=0*, *auto_scale=True*, ***plot_params*)

add_lines(*ax_name*, *nr_lines*, ***line_customization*)

append_data(*axes_name*, *xdata*, *ydata*)

append_to_axes(*axes_name*, *xdata*, *ydata*, *line_in=[]*, *autoscale=True*)

append_to_lines(*axes_name*, *xdat*, *ydata_dict*, *autoscale=True*)

colors = array([[0. , 0. , 0.5 , 1.], [0. , 0. , 0.58912656, 1.], [0. , 0. , 0.67

cols

customize_lines(*lines*, *markers=None*, *line_styles=None*, *marker_sizes=None*, *marker_edge_widths=None*, *colors=None*)

fill_grid(*pos*, *rows*, *cols*)

grid_free(*pos*, *rows*, *cols*)

line_markers = (u'o', u'v', u'^', u'<', u'>', u'8', u's', u'p', u'*', u'h', u'H', u'D'

line_styles = ['-', ':', '--', '-.']

marker_edge_widths = [0.1]

marker_sizes = [2.0]

rows

tight_layout(***kwargs*)

within_grid(*pos*, *rows*, *cols*)

class VirtualMicrobes.plotting.Graphs.**MultiGridGraph**(*base_save_dir*, *name*, *rows*, *cols*, *row_heigth=4*, *col_width=4*, *attribute_dict=None*, *show=True*, ***kwargs*)

Bases: *VirtualMicrobes.plotting.Graphs.MultiGraph*

classdocs

add_axis(*name*, *pos*, *rows=1*, *cols=1*)

append_data(*axes_name*, *time_point*, *data*)

append_to_axes(*axes_name*, *time_point*, *data*)

plot_in_axes(*axes_name*, *cm_name*, *matrix_data=None*, *data_range=None*, *color_bar=True*)

set_data_range(*axes_name*, *_range*)

class VirtualMicrobes.plotting.Graphs.**Network**(*base_save_dir*, *name*, *attribute_dict*, *size=(10, 10)*, *show=True*, ***kwargs*)

Bases: *VirtualMicrobes.plotting.Graphs.Grapher*

add_self(*marker*)

clear_graph()

color_edge_direction(*i*)

e_attr_list(*attr*, *selectors=[]*)

edges_with_attr_list(*selectors=[]*)

gene_node_id(*gene*)

gene_node_label(*gene_node_id*, *depth=1*)

init_network()

metabolite_edge_width(*bb*, *cell_bb=False*)

n_attr_list(*attr*, *selectors=[]*)

nodes_with_attr_list(*selectors=[]*)

redraw_network(***kwargs*)

type_shape(*reac_type*)

write_to_file(*name=None*, *labels=[]*, *suffix='.gml'*, ***kwargs*)

class VirtualMicrobes.plotting.Graphs.**PhyloTreeGraph**(*base_save_dir*, *name*, *attribute_dict*, *show=True*, ***kwargs*)

Bases: *VirtualMicrobes.plotting.Graphs.Grapher*

Plot phylogenetic trees and represent the data in nodes with different layout styles.

compress_root_branch(*root_branch_dist=20*)

metabolic_type_layout(*node*)

metabolic_with_lod_layout(*node*)

mutation_rates_layout(*node*, *mut_type*)

node_layouts = ['metabolism', 'trophic_type', 'metabolic_with_lod']

rate_features = ['point_mut_rate', 'chromosomal_mut_rate', 'stretch_mut_rate', 'sequen

save_fig(*feature='metabolism'*, *name=None*, *labels=[]*, *suffix='.svg'*, *rescale=None*, *dpi=10*, ***kwargs*)

Render tree with layout function depending on the selected feature for tree representation.

Parameters

- **feature** – feature selects for a specific layout function, determining node style
- **name** – name for saving
- **labels** – labels to appear in save file name
- **suffix** – suffix of save file name

select_layout_fn(*data_type*)

trophic_type_layout(*node*, *trophic_type*)

update(*tree*)

Set new ete_tree.

update_figure(*show=None*, *feature='metabolism'*, ***kwargs*)

write_to_file(*name=None*, *labels=[]*, *suffix='.nw'*, ***kwargs*)

exception VirtualMicrobes.plotting.Graphs.**PositionOutsideGridError**(*value*)

Bases: *VirtualMicrobes.plotting.Graphs.GraphingError*

Module contents

VirtualMicrobes.post_analysis package

Submodules

VirtualMicrobes.post_analysis.lod module

class VirtualMicrobes.post_analysis.lod.**LOD**(*lod*, *name*, *stride*, *time_interval*, *lod_range*, *save_dir=None*)

Bases: `object`

classdocs

standardized_production(*test_params*)

strided_lod(*stride*, *time_interval*, *lod_range*)
Sample individuals within a range of the LOD at regular intervals.

Either use a stride or a time interval to sample individuals from the lod. If a time interval is provided, ancestors are sampled that have a time of birth that is approximately separated by *time_interval* in the evolutionary simulation.

Parameters

- **stride** (*int*) – stride in generations for sampling individuals along the *LOD*
- **time_interval** (*int*) – interval in simulation time for sampling individuals along the *LOD*
- **lod_range** (*(float, float)*) – bounds in fractions of the total range of the *LOD*

Returns

Return type list of ancestor *VirtualMicrobes.virtual_cell.Cell.Cell* s

t_interval_iter(*time_interval*)
Iterate ancestors that are approximately 'time_interval' timesteps apart in their time of birth.

class VirtualMicrobes.post_analysis.lod.**LOD_Analyser**(*args*)
Bases: object

Analyses the evolutionary history of a population by tracing ancestors in the line of descent.

Loads a simulation save from a file, keeping a reference in *ref_sim*. From this, initialise *ref_pop_hist* as a *PopulationHistory* object that analyses the phylogenetic tree of the population.

The *PopulationHistory* generates a *LOD* for 1 or more individuals in the saved population. For each *LOD*, evolutionary data and network and genome plots can be produced.

It is possible to load additional simulation snapshots that preceed the *ref_pop_hist* and compare individuals to their contemporaries present in the preceding populations. *compare_saves* contains a list of file names of populations-saves that should be compared.

anc_cells(*runtime=None*, *tcs=False*)
Dump all cells in the fossil record (e.g. to map onto the newick trees)

args = None
config and command line arguments used for initialisation

compare_saves = []
names of snapshot files to copmare to *ref_sim*

compare_to_pops()
Compare reference simulation to a set of previous population snapshots.

Compares each of the simulation snapshot saves in *compare_saves* to the *ref_pop_hist*. A *PopulationHistory* is constructed for each of the compare snapshots. Within the compare snapshot, individuals that correspond to the are part of (any of) the LOD`(s) of the :attr:`ref_pop_hist will be identified. Properties of these *ancestors* will then be compare with their statistical values for the whole population.

draw_ref_trees()
Draw a reference phylogenetic tree, with individual, selected *LOD*s marked

init_compare_saves(*compare_saves*)
Parse and check compare saves parameter.

Compare saves can be either a list of file names or a list of generation times (or None). In the latter case, the file names should be constructed using the time point and the file name of the reference simulation.

Checks are made to ensure files exist and also to ensure that no compares save points come after the reference simulation save point, as this would not make sense in the comparison functions.

init_ref_history(*ref_sim=None*, *nr_lods=None*, *prune_depth=0*, *pop_hist_dir='population_history'*)
Create a *PopulationHistory* from the *ref_sim VirtualMicrobes.simulation.Simulation.Simulation* object.

For the *PopulationHistory* object constructs its phylogenetic tree and prune back the tree to a maximum depth of (max_depth - prune_depth) counted from the root. Then create *LOD* objects representing the *line of descent* of the *nr_lods most diverged* branches in the tree.

Parameters

- **ref_sim** (*VirtualMicrobes.simulation.Simulation.Simulation* object) – simulation snapshot that is the basis for *LOD* analysis
- **nr_lods** (*int nr_lods*) – nr of separate (most distant) *LOD*s to initialize
- **prune_depth** (*int*) – prune back the phylogenetic tree with this many timesteps
- **pop_hist_dir** (*str*) – name of directory to store lod analysis output

lod_binding_conservation(*stride=None*, *time_interval=None*, *lod_range=None*)
Write time series for TF binding conservation for *LOD*s.

Parameters

- **stride** (*int*) – stride in generations for sampling individuals along the *LOD*
- **time_interval** (*int*) – interval in simulation time for sampling individuals along the *LOD*
- **lod_range** (*(float, float)*) – bounds in fractions of the total range of the *LOD*

lod_cells(*stride=None*, *time_interval=None*, *lod_range=None*, *runtime=None*)
Write time series of evolutionary changes along all *LOD*s.

Parameters

- **stride** (*int*) – stride in generations for sampling individuals along the *LOD*
- **time_interval** (*int*) – interval in simulation time for sampling individuals along the *LOD*
- **lod_range** (*(float, float)*) – bounds in fractions of the total range of the *LOD*

lod_graphs(*stride=None*, *time_interval=None*, *lod_range=None*, *formats=None*)
Draw network and genome graphs for *LOD*s

It is possible to set an interval and a range to sample individuals in the *LOD*.

Parameters

- **stride** (*int*) – stride in generations for sampling individuals along the *LOD*
- **time_interval** (*int*) – interval in simulation time for sampling individuals along the *LOD*
- **lod_range** (*(float, float)*) – bounds in fractions of the total range of the *LOD*

Note: Either use a stride or a time interval to sample individuals from the lod.

---

lod_network_stats(*stride=None*, *time_interval=None*, *lod_range=None*)
Write time series for evolutionary network property changes along all *LOD*s.

Parameters

- **stride** (*int*) – stride in generations for sampling individuals along the *LOD*
- **time_interval** (*int*) – interval in simulation time for sampling individuals along the *LOD*
- **lod_range** (*(float, float)*) – bounds in fractions of the total range of the *LOD*

lod_stats(*stride=None*, *time_interval=None*, *lod_range=None*)
Write time series of evolutionary changes along all *LOD*s.

Parameters

- **stride** (*int*) – stride in generations for sampling individuals along the *LOD*
- **time_interval** (*int*) – interval in simulation time for sampling individuals along the *LOD*
- **lod_range** (*(float, float)*) – bounds in fractions of the total range of the *LOD*

lod_time_course_plots(*stride=None*, *time_interval=None*, *lod_range=None*, *formats=None*)
Draw time course diagrams for individuals in the *LOD*s.

It is possible to set an interval and a range to sample individuals in the *LOD*.

Parameters

- **stride** (*int*) – stride in generations for sampling individuals along the *LOD*
- **time_interval** (*int*) – interval in simulation time for sampling individuals along the *LOD*
- **lod_range** (*(float, float)*) – bounds in fractions of the total range of the *LOD*

Note: Either use a stride or a time interval to sample individuals from the lod.

---

lod_time_courses(*lod_range=None*, *chunk_size=None*)
Write time series of molecule concentrations within the *LOD*

It is possible to set a range to sample individuals in the *LOD*.

Parameters

- **lod_range** (*(float, float)*) – bounds in fractions of the total range of the *LOD*
- **chunk_size** (*int*) – number of generations in LOD to concatenate per chunk

pop_cells(*runtime=None*, *tcs=False*)

ref_pop_hist = None
PopulationHistory for the reference simulation (*ref_sim*) snapshot

ref_sim = None
VirtualMicrobes.simulation.Simulation snapshot to analyse

write_newick_trees()
write newick trees for all phylogenies in attr:*ref_pop_hist*

class VirtualMicrobes.post_analysis.lod.**PopulationHistory**(*sim*, *params*, *save_dir=None*, *prune_depth=None*)

Bases: object

Performs and stores evolutionary history analysis of *VirtualMicrobes.simulation.Simulation.Simulation* snapshots.

Generates *LOD*s for 1 or more individuals in the population. Reconstruct the evolutionary events along the line of descent.

A reference *PopulationHistory* can also be compared to *population history* at earlier simulation time points. In this case the ancestors of individuals in the reference *population history* will be identified and compared to the rest of the population at that point in time. In this way, evolutionary biases on the line of descent can be brought to light.

anc_cells(*pop*, *time*)
Write cell files for all cells in the ancestry, which can be mapped on the newick tree :param pop: :type pop: current population that contains the current_ancestry list :param time: :type time: run_time

compare_to_pop(*compare_save*, *prune_depth=None*, *leafs_sample_size=None*)
Compare the reference *PopulationHistory* to an earlier population-save.

Parameters

- **compare_save** (*str*) – file location of population-save
- **prune_depth** (*int*) – prune back phylogeny of the :param:compare_save with this many timesteps
- **leafs_sample_size** (*int*) – maximum number of phylogenetic tree leafs to use for comparison

draw_ref_trees(*rescale=False*)
Output reference trees for phylogenetic trees with lods labeled.

Uses phylogenetic tree drawing methods to annotate the leaf nodes of lods. Reference trees give a visual overview of the position of the lods that are analysed in the tree.

dump_anc_cells(*time*)
Dump all ancestors (perfect fossil record) to files, and also save the newick tree. Should be all in there?

dump_lod_cells(*time*)
Dump all cells used in LOD analysis to files (i.o.w. a single lineages / subset of anc_cells)

dump_pop_cells(*time*)
Output current population cells as cellfiles

environment = None
Short cut to *VirtualMicrobes.environment.Environment* of *sim*.

identify_lod_ancestor(*ete_tree_struct*, *lod*)
Identify the individual in the population that is on the line of descent (lod) under consideration.

The nodes in the ete tree corresponding to the *lod* will be annotated with a tag.

Parameters

- **ete_tree_struct** (*VirtualMicrobes.my_tools.utility.ETEtreeStruct*) – container structure for phylogenetic tree representations
- **lod** (*LOD*) – line of descent

Returns

- (*VirtualMicrobes.virtual_cell.Cell.Cell*, ete3.TreeNode)
- *(oldest ancestor cell, its tree node representation)*

init_lods(*nr_lods*, *save_dir=None*, *stride=None*, *time_interval=None*, *lod_range=None*)
Initialize the line of descent (*LOD*) container objects.

Iterate over the phylogenetic trees of the *population* and for each tree select *nr_lods* leaf nodes that are at maximum phylogenetic distance.

For each of the selected leafs, construct a line of descent object (*LOD*).

Parameters

- **nr_lods** (*int*) – number of *LOD* objects per phylogenetic tree
- **save_dir** (*str*) –
- **stride** (*int*) – stride in generations for sampling individuals along the *LOD*
- **time_interval** (*int*) – interval in simulation time for sampling individuals along the *LOD*
- **lod_range** (*(float, float)*) – bounds in fractions of the total range of the *LOD*

init_phylo_tree(*prune_depth=None*)
Update the phylogenetic tree of the population.

Clears the change in the population of the final regular simulation step. Prunes back the tree to a maximum depth.

Parameters **prune_depth** (*int*) – number of generations to prune from the leafs of phylogenetic tree

lod_binding_conservation(*stride*, *time_interval*, *lod_range*)
Write time series for line of descent properties such as network connectivity, protein expression etc.

Either use a stride or a time interval to sample individuals from the lod.

Parameters

- **stride** (*int*) – stride in generations for sampling individuals along the *LOD*
- **time_interval** (*int*) – interval in simulation time for sampling individuals along the *LOD*
- **lod_range** (*(float, float)*) – bounds in fractions of the total range of the *LOD*

lod_cells(*stride*, *time_interval*, *lod_range*, *runtime*)
Write cell files for line of descent

The leaf of the tree is saved as CellLeaf<LOD_ID>, and all it's ancestors are saved as CellNode<BIRTHTIME>_<LOD_ID>.cell

Parameters

- **stride** (*int*) – stride in generations for sampling individuals along the *LOD*
- **time_interval** (*int*) – interval in simulation time for sampling individuals along the *LOD*
- **lod_range** (*(float, float)*) – bounds in fractions of the total range of the *LOD*

lod_network_stats(*stride*, *time_interval*, *lod_range*)
Write time series for line of descent properties such as network connectivity, protein expression etc.

Either use a stride or a time interval to sample individuals from the lod.

Parameters

- **stride** (*int*) – stride in generations for sampling individuals along the *LOD*

- **time_interval** (*int*) – interval in simulation time for sampling individuals along the *LOD*
- **lod_range** (*(float, float)*) – bounds in fractions of the total range of the *LOD*

lod_stats(*stride*, *time_interval*, *lod_range*)
Write time series for line of descent properties such as network connectivity, protein expression etc.

Either use a stride or a time interval to sample individuals from the lod.

Parameters

- **stride** (*int*) – stride in generations for sampling individuals along the *LOD*
- **time_interval** (*int*) – interval in simulation time for sampling individuals along the *LOD*
- **lod_range** (*(float, float)*) – bounds in fractions of the total range of the *LOD*

lods_time_course_data(*lod_range*, *chunk_size*)
Write time series data in the line of descent to files.

Concatenates time courses of individuals along a *LOD*. Concatenations are done in *chunks* of a chosen *chunk_size*. For each chunk **.csv** files are stored in a directory named part*n*, where *n* is the chunk number.

Parameters

- **ancestors** (list of *VirtualMicrobes.virtual_cell.Cell.Cell*s) –
- **base_save_dir** (*str*) –
- **viewer_path** (*str*) – path to utility files for html data viewer
- **chunk_size** (*int*) – length of chunks of concatenated data

lods_time_course_plots(*stride*, *time_interval*, *lod_range*, *formats*)
Output time course graphs for the line of descent.

Either use a stride or a time interval to sample individuals from the lod.

Parameters

- **stride** (*int*) – stride in generations for sampling individuals along the *LOD*
- **time_interval** (*int*) – interval in simulation time for sampling individuals along the *LOD*
- **lod_range** (*(float, float)*) – bounds in fractions of the total range of the *LOD*

params = None
The (updated) simulation parameters.

plot_lod_graphs(*stride*, *time_interval*, *lod_range*, *formats*)
Output metabolic, GRN and genome graphs for the line of descent.

Either use a stride or a time interval to sample individuals from the lod.

Parameters

- **stride** (*int*) – stride in generations for sampling individuals along the *LOD*
- **time_interval** (*int*) – interval in simulation time for sampling individuals along the *LOD*
- **lod_range** (*(float, float)*) – bounds in fractions of the total range of the *LOD*

population = None
Short cut to *VirtualMicrobes.virtual_cell.Population.Population* of *sim*.

prune_depth = 0
Number of generations from leaves to prune the phylogenetic tree of the pophist.

sim = None
The *VirtualMicrobes.simulation.Simulation.Simulation* snapshot for which this pophist was made.

time_point = None
Last simulation time of the *sim*.

tree_lods = []
List of lists of *LOD*s. One list for each independent phylogenetic tree within the population.

write_newick_trees()
Write newick representation of phylogenetic trees to files.

VirtualMicrobes.post_analysis.network_funcs module

VirtualMicrobes.post_analysis.network_funcs.**prune_GRN**(*grn*, *log_dif_effect=0.5*, *rescue_regulated=True*, *iterative=True*)

VirtualMicrobes.post_analysis.network_properties module

class VirtualMicrobes.post_analysis.network_properties.**PhyloGeneticAnalysis**
Bases: object

Analyze biological networks

VirtualMicrobes.post_analysis.network_properties.**calculate_overlap**(*tf_connections*, *connections_of_homologous_tfs*, *closest_bound_homologs_dict*)

Calculate the overlap in bound genes between tf homologs.

Parameters

- **tf_connections** (list of *VirtualMicrobes.virtual_cell.Gene.Gene*s) – Downstream connections of the reference TF
- **connections_of_homologous_tfs** (list of sets of *VirtualMicrobes.virtual_cell.Gene.Gene*s) – List of sets of downstream genes for each homolog of the reference TF
- **closest_bound_homologs_dict** (dict of sets of *VirtualMicrobes.virtual_cell.Gene.Gene*s) – Mapping of each original downstream gene of the reference TF to sets of homologs of these downstream genes.

Returns Tuple of fractions: [0]: Fraction of downstream genes who's homologs are bound by a homolog of the reference TF. [1]: Fraction of new connections (averaged over tf homologs) per original connection of the reference TF.

Return type float,float

VirtualMicrobes.post_analysis.network_properties.**find_homolog_distances**(*gene*, *genome*, *closest_homolog=False*)

Find homologs and their distance for a gene in a target genome.

Parameters

- **gene** (*VirtualMicrobes.virtual_cell.GenomicElement.GenomicElement*) – Reference gene for homology search.
- **genome** (*VirtualMicrobes.virtual_cell.Genome.Genome*) – Genome in which to search for homologs.
- **closest_homolog** (*bool*) – Flag to filter found homologs to those that have the shortest phylogenetic distance to the *gene*.

VirtualMicrobes.post_analysis.network_properties.**find_homologs**(*gene*, *genome*)

For a gene, find all its homologs in a given genome.

This is a naive approach that uses a combination of the gene's type and its *VirtualMicrobes.virtual_cell.Identifier.Identifier* attribute to detect common descent.

Parameters

- **gene** (*VirtualMicrobes.virtual_cell.GenomicElement.GenomicElement*) – Reference gene for homology search.
- **genome** (*VirtualMicrobes.virtual_cell.Genome.Genome*) – Genome in which to search for homologs.

Returns

Return type The set of homologs of *gene* in the *genome*.

VirtualMicrobes.post_analysis.network_properties.**tf_binding_overlap**(*cell1*, *cell2*, *closest_homolog=False*, *no_phylogeny=False*, *verbose=False*)

Measure the overlap in target genes for tf homologs in phylogenetically related individuals.

cell1 [*VirtualMicrobes.virtual_cell.Cell.Cell*] Reference individual for which to find homologs

cell2 [*VirtualMicrobes.virtual_cell.Cell.Cell*] Homologs of TFs and downstream targets will be detected in this individual.

closest_homolog [bool] Flag to filter found homologs to those that have the shortest phylogenetic distance to the *gene*.

verbose [bool] Print messages about homologs found.

Returns Mapping from *VirtualMicrobes.virtual_cell.Gene.TranscriptionFactor* to (maximum) binding overlap score.

Return type dict

Module contents

VirtualMicrobes.simulation package

Submodules

VirtualMicrobes.simulation.Simulation module

‘ Top level Simulation class that contains all objects and parameters of the simulation. Has a simulate method that executes the simulation loop, methods to save and reload, as well as initiate data storage and plotting.

class VirtualMicrobes.simulation.Simulation.**EvoSystem**(*params*)
Bases: object

Sets up the Environment and the Population.

reinit_system_params(*params*, ***param_updates*)

setup_environment()

class VirtualMicrobes.simulation.Simulation.**ODE_simulation**(*params*)
Bases: *VirtualMicrobes.simulation.Simulation.Simulation*

Set up a simulation. Initialize an EvoSystem and an Integrator. EvoSystem consists of a Population and Environment that are co-dependent. This is because a Cell in a Population can only choose its Reactions once the environment is set up and the reaction space is ready, while the reaction space can only be fully known, when all internal molecules have been defined in relation to Cells in the Population. To circumvent the problem, internal molecules will only exist as ‘ideal’ type molecules and then every Cell will contain a mapping from ‘ideal’ molecules to actual variables of the system.

init_spatial_integrator(*diffusion_steps=None*, *report_frequency=None*, *step_function=None*, *init_step_size=None*, *absolute_error=None*, *relative_error=None*, *global_time=0.0*)

init_time_course_lengths()

max_time_course_length()

simulation_burn_in(*burn_in_time=None*, *simulation_steps=1*)

simulation_step(*global_time=None*, *diffusion_steps=None*, *delta_t_between_diff=None*, *report_frequency=None*, *release_delay=None*, *update_relative_error=None*, *verbal=False*)
A simulation step will run the integration of cycle of a cell’s life:

- setup the variables_map needed
- simulate internal dynamics
- save variables states back to python objects (if no errors occured or > maxtries)

class VirtualMicrobes.simulation.Simulation.**Simulation**(*params*)
Bases: object

Base class for Simulation objects.

Stores the simulation parameters.

Initializes output and storage paths and file for stdout and stderr logging.

Initializes a data store class for storing data points and time series.

Initializes a graphs class for output graphing, that will be fed with data points from the data store.

Has method for the main simulation loop.

Stores simulation snapshots that can be reloaded for further simulation or post analysis.

add_graphs_data(*time_point*)

auto_restart_opts(*retry_list=None*, *time=None*)
Automatically restart the simulation after the population became extinct.

A simulation can be automatically restarted from a previous save point. The save points to retry from in the *retry_list* are updated such that the most recent point is retried first and removed from this list. This ensures that a subsequent restart will not be attempted from the same save point, but will instead skip back to a preceding save point. When restarting the simulation, new parameter values will be chosen for a selected set of parameters, by applying a scaling factor *v*.

Parameters **retry_list** (*list*) – list of population snapshots that can still be retried for restart

Returns

Return type (population save to retry, new parameter settings for retry)

backup_pop_stats()

class_version = '2.4'

close_phylo_shelf()

copy_config_files()

describe_environment()

init_data_store(*clean=True*, *create=True*)
Initialize a DataStore object for storage of simulation data.

Data are kept in storage for retrieval by plotting functions until a call is made to write the raw data to a file (write_data).

init_error_log_file(*save_dir=None*, *name='log'*, *suffix='.err'*, *labels=[]*)

init_ffmpeg()

init_graphs(*show=None*, *clean=True*, *create=True*)
Initialize a Graphs object that contains simulation graphs.

Parameters **show** – plot graphs on the X (blocking)

init_log_file(*save_dir=None*, *name='log'*, *suffix='.out'*, *labels=[]*)

init_log_files()

init_phylo_linker_dict()
Create a linker dict that maps phylogenetic units (PhyloUnit) to unique indices. This linker dict is used to mediate parent-child relations, while preventing that pickling recurses (heavily) on the phylogeny (no direct references to the objects)

init_save_dir(*clean=None, remove_globs=['*.txt', '*.sav', '*.pck', '*.log', '*.err']*)

init_unique_gene_key()
Initialize a generator for unique keys for use in the linker dict (see above).

init_unique_phylo_key()
Initialize a generator for unique keys for use in the linker dict (see above).

open_log_file(*name*)

open_phylo_shelf(*name*, *flag*)
Open a Shelf like database object that can store (part of) phylogenetic units (PhyloUnit) to disk.

plot_and_save_graphs(*time_point*)
Depending on the initialization parameter 'graphs_single_core' this will either run all plotting in functions in sequence (appending None to processes) or construct a list of job processes/ task tuples, to be run in parallel batch processes, separate from the main thread. These processes will be either put in a job queue or separately started by '_start_parallel_jobs'.

Parameters **time_point** – simulation time point

plot_best_genome_structure(*time_point*)

plot_grid_graphs(*time_point*)

plot_networks(*time_point*)

plot_pop_stats()

plot_time_course(*time_point*)

print_system_details()

prune_data_store_files()

reload_unique_gene_count()

reload_unique_phylo_count()

reopen_phylo_shelf(*save_file=None*)

save(*time=None*, *store=False*)
Save the simulation state as a snapshot.

Parameters **time** (*simulation time point*) –

Returns

Return type filename of simulation save

save_dir
Return the simulation save path.

save_phylo_shelf(*name=None*, *labels=[]*, *suffix='.pck'*)
Save a snapshot of the phylo_shelf, the global store of PhyloUnit objects.

Creating the snapshot enables reloading simulation saves with a correct state of the phylo_shelf.

set_phylo_shelf_file_name(*name=None*, *suffix='.pck'*, *labels=[]*)

simulate()
The main simulation loop.

Clear the population changes from previous iteration. Run the ode system. Apply HGT. Calculate death rates, determine deaths. Compete and reproduce. Mutate the offspring. Update the phylogeny. Store data and plot.

store_command_line_string(*fn='command_line_string.txt'*)

store_previous_save_dir()
Save the location for data storage presently used in the simulation.

Useful to keep a history of save locations when simulations are reloaded and run with different data storage locations.

update_data_location(*save_dir=None*, *graphs_name='plots'*, *data_name='data'*, *clean=False*, *copy_data=True*, *create=True*, *current_save_path=None*)
Moves existing data to a new location (e.g. when name of project has changed after propagating an existing population)

update_shelf_location(*current_save_path=None*)

update_sim_params(*params_dict*)
Update simulation parameters with values in an update dict.

Parameters **params_dict** (*dict*) – dict with updated parameter values

upgrade(*odict*)
Upgrading from older pickled version of class to latest version. Version information is saved as class variable and should be updated when class invariants (e.g. fields) are added. (see also __setstate__)

Adapted from recipe at http://code.activestate.com/recipes/521901-upgradable-pickles/

upgrade_graphs()

write_data()

write_params_to_file(*save_dir=None*, *name='params'*, *suffix='.txt'*, *labels=[]*)

VirtualMicrobes.simulation.Simulation.**create_simulation**(***param_updates*)

VirtualMicrobes.simulation.Simulation.**default_params**()

VirtualMicrobes.simulation.Simulation.**load_sim**(*file_name*, *verbose=False*, ***param_updates*)
Load a pickled representation of a saved simulation state.

Complementary method to :save_sim: to load and restore a simulation state. There is a possibility to update simulation parameters. The first stage of unpickling will reload the simulation parameters. This is necessary, because we need to set the 'as_subprocess' parameter for decorating Graph methods to be set before class initialization, by retrieving the 'graphs_single_core' parameter from the simulation 'temp_params' prior to reloading and recreating the pickled Graphs instance in the simulation object.

VirtualMicrobes.simulation.Simulation.**load_sim_params**(*file_name*)

VirtualMicrobes.simulation.Simulation.**load_simulation**(*file_name*, ***param_updates*)

VirtualMicrobes.simulation.Simulation.**mut_func_single_param**(*val*, *rand_g*, *mut_par_space*)

VirtualMicrobes.simulation.Simulation.**mut_func_single_param_step_uniform**(*val*, *rand_g*, *mut_par_space*)

VirtualMicrobes.simulation.Simulation.**mut_ks_dict_func**(*kss*, *rand_gen*, *mut_par_space*, *mutate_single_param*)

VirtualMicrobes.simulation.Simulation.**partial_mut_func_single_param**(*val*, *rand*, *mut_par_space*)

VirtualMicrobes.simulation.Simulation.**partial_mut_ks_dict_func**(*val*, *rand*, *mut_par_space*)

VirtualMicrobes.simulation.Simulation.**pump_exporting_mut_func**(*val*)

VirtualMicrobes.simulation.Simulation.**save_pop_cells**(*sim*, *name=None*, *save_dir=None*, *labels=[]*, *suffix='.sav'*)

VirtualMicrobes.simulation.Simulation.**save_sim**(**args*, ***kwargs*)

Make a pickled save state of the simulation.

It (nearly) completely creates a pickle representation of the simulation, from which the simulation can be reloaded and continued, with parameters and state fully restored. Because a global linker dict with database functionality is used to store phylogenetic elements such as Genes, Chromosomes and Cells, a snapshot of this database needs to be created simultaneously. The snapshot of the DB is remembered within the simulation, to allow it to be reloaded when the saved simulation state is reloaded.

VirtualMicrobes.simulation.Simulation.**save_single_cell**(*sim*, *cell*, *name=None*, *save_dir=None*, *labels=[]*, *suffix='.sav'*)

VirtualMicrobes.simulation.Simulation.**tf_ext_sense_mut_func**(*val*)

VirtualMicrobes.simulation.Simulation.**update_default_params**(*keep_unrecognized=False*, *verbose=False*, ***kwargs*)

Use simulate_params, pop_params and env_params as defaults and overwrite them with kwargs to construct a parameter dictionary. Warn for all kwargs that have not been consumed. (default_params() generates all default parameters)

Parameters

- **simulate_params** – dictionary with general simulation parameters
- **pop_params** – dictionary with population specific parameters
- **env_params** – dictionary with environment specific params

VirtualMicrobes.simulation.class_settings module

Created on Dec 16, 2014

@author: thocu

VirtualMicrobes.simulation.continue module

VirtualMicrobes.simulation.continue.**main**(*argv=None*)

VirtualMicrobes.simulation.continue.**parse_opts**(*opt_strings*)

VirtualMicrobes.simulation.continue.**to_argparse_opts**(*opts*)

VirtualMicrobes.simulation.start module

VirtualMicrobes.simulation.start.**main**(*argv=None*)

VirtualMicrobes.simulation.start.**parse_combining_opts**(*opt_strings*)

VirtualMicrobes.simulation.start.**parse_fixed_opts**(*opt_strings*)

VirtualMicrobes.simulation.start.**to_argparse_opts**(*opts*)

VirtualMicrobes.simulation.start_multi module

VirtualMicrobes.simulation.virtualmicrobes module

Command Line Interface to the Virtual Microbes Evolutionary Simulator package.

@license: MIT licence

@contact: thomas.cuypers@gmail.com @deffield updated: Updated

Module contents

VirtualMicrobes.virtual_cell package

Submodules

VirtualMicrobes.virtual_cell.Cell module

class VirtualMicrobes.virtual_cell.Cell.**Cell**(*params*, *environment*, *rand_gen=None*, *toxicity_function=None*, *toxicity_effect_function=None*, *time_birth=-1*, ***kwargs*)

Bases: *VirtualMicrobes.virtual_cell.PhyloUnit.PhyloUnit*

GRN(*genes=None*, *prot_color_func=<function <lambda>>*, *with_gene_refs=False*, *with_self_marker=False*)

Make Gene Regulatory Network representation of the regulated genome.

First, the lists of nodes and edges is generated for the given genes. Then, nodes are added to a Directed Graph (networkx.DiGraph) object with attributes that distinguish the gene type and other characteristics. Finally, edges are added with attributes that indicate e.g. binding strength.

Returns

Return type networkx.Digraph

add_gene_copy(*gene*, *concentration=0.0*, *diff_constant=None*, *degr_constant=None*)

Add gene product to the cell. If the 'gene' is already present as a key in the subdictionary, adding it again means we have to increase the multiplicity of the gene product, since there are multiple genes coding for this gene product. Otherwise, we add an entry for this gene product.

Parameters

- **gene** (*VirtualMicrobes.virtual_cell.Gene.Gene*) – The gene for which copy number is increased.
- **concentration** (*float*) – Initial concentration of gene product.
- **diff_constant** (*float*) – Diffusion constant of gene product over the cell membrane. (If None, no diffusion is modeled)
- **degr_constant** (*float*) – Degradation constant of gene product.

add_gene_product(*gene*, *concentration=None*)

Initialize gene products from the genes in the genome.

Gene products have a concentration and degradation rate. ODE system integration will use the gene products as input variables.

Parameters **concentration** (*float*) – initial concentration of gene products

add_small_molecule(*mol*, *env*, *concentration*, *diff_const*, *degr_const*)
Add a metabolite to the internal Cell molecules and set its state.

Parameters

- **mol** (*VirtualMicrobes.event.Molecule.Molecule*) – metabolite to be added to this Cell
- **env** (*VirtualMicrobes.environment.Environment.Environment*) – environment containing metabolites
- **conc** (*float*) – start concentration of this metabolite
- **diff_const** (*float*) – diffusion constant of this metabolite
- **degr_const** (*float*) – degradation constant of this metabolite

add_small_molecules(*env*, *conc*, *degr_const*, *ene_degr_const*, *bb_degr_const*)
Add the metabolites in the environment as internal variables to this individual.

Set a membraned diffusion rate and degradation rate for the molecules. Diffusion rates come from a dictionary of the environment. Degradation rates also come from a dictionary in the environment, but may be scaled with an 'internal' *degr_const* parameter to cause degradation rates to be different inside and outside of the cell.

Parameters

- **env** (*VirtualMicrobes.environment.Environment.Environment*) – environment containing metabolites
- **conc** (*float*) – start concentration of all metabolites
- **degr_const** (*float*) – degradation constant of metabolites
- **ene_degr_const** (*float*) – degradation constant of energy metabolites
- **bb_degr_const** (*float*) – degradation constant of building block metabolites

ancestral_mutations
Cumulative mutations in the line(s) of descent.

For each LOD of this individual concatenate all mutations in each mutation category that happened in the ancestors of this individual. Because multiple LODs might exist, return a list of dictionaries that hold all mutations that this lineage acumulated, per mutation category.

Returns

Return type list of mutation dictionaries of LODs

apply_hgt(*time*, *gp*, *environment*, *rand_gene_params=None*, *mutation_rates=None*, *global_mut_mod=None*, *rand_gen=None*, *rand_gen_np=None*, *verbose=False*)
Applies HGT to a cell, and updates the GRN.

If applied HGT is applied, the gp updated flag is set to True, to inform integrator update in next timestep.

Parameters

- **time** (*int*) – time point in the simulation
- **gp** (*VirtualMicrobes.environment.Grid.GridPoint*) – location of cell on the grid

- **environment** (*VirtualMicrobes.environment.Environment.Environment*) – simulation environment containing all possible reactions to draw a random gene for external HGT
- **rand_gene_params** (*dict*) – parameter space for properties of externally HGTed genes
- **mutation_rates** (*dict*) – contains rates for eHGT and iHGT
- **global_mut_mod** (*double*) – scaling factor for all mutation rates
- **rand_gen** (*RNG*) –
- **rand_gen_np** (*RNG numpy*) –

Returns

Return type The applied mutations are returned

asexual(*time*)
Asexual reproduction.

Parameters **time** (*int*) – simulation time point

Returns

Return type *VirtualMicrobes.virtual_cell.Cell.Cell*

avrg_promoter_strengths

building_blocks
Building Blocks in metabolism of this individual.

calculate_raw_death_rate(*base_rate*, *toxicity_scaling*, *toxic_effect_func=None*)
Raw death rate based on a base rate and the toxic effects of internal molecule concentrations.

Parameters

- **base_rate** (*float*) – base death rate
- **toxicity_scaling** (*float*) – scaling constant for toxic effect
- **toxic_effect_func** (*func*) – calculates surplus death rate due to toxicity

chromosomal_mut
List of chromosomal mutations of this individual.

chromosomal_mut_count
Number of chromosomal mutations of this individual.

chromosome_count

chromosome_del
List of chromosome deletions of this individual.

chromosome_del_count
Number of chromosomal deletions of this individual.

chromosome_dup
List of chromosome duplications of this individual.

chromosome_dup_count
Number of chromosomal duplications of this individual.

chromosome_fiss_count
Number of chromosomal fissions of this individual.

chromosome_fission
List of chromosome fissions of this individual.

chromosome_fuse_count
Number of chromosomal fusions of this individual.

chromosome_fusion
List of chromosome fusions of this individual.

chromosome_mutate_genome(*time*, *mutation_rates*, *global_mut_mod*, *rand_gen*, *rand_gen_np*, *verbose*)
Apply chromosomal mutations to the genome.

Each type of chromosome mutation is applied independently. When a mutation is selected, one or two random chromosomes are chosen for the mutation. The mutation method return a Mutation instance, that is appended to a list of chromosome mutations. This list is returned.

Parameters

- **time** (*int*) – simulation time
- **mutation_rates** (*attr_dict*) – dict with all mutation rates
- **global_mut_mod** (*float*) – modifier for Cell wide mutation rate
- **rand_gen** (*RNG*) –
- **rand_gen_np** (*RNG*) – numpy random number generator
- **verbose** (*bool*) – verbosity

Returns

Return type list of Mutation objects

class_version = '2.8'

clear_mol_time_courses()
Set time course arrays to None.

Notes

Reduces memory footprint of cell, for example before saving cell objects.

clone(*time*)
Make a clone of this cell.

Clones are identical, having no mutations, and maintain a reference to the parent.

Parameters **time** (*int*) – simulation time point

Returns

Return type *VirtualMicrobes.virtual_cell.Cell*

compute_product_toxicity(*toxicity_func=None*)
Set toxicity effect of product.

consumer_type
Return the set of all metabolic classes consumed in enzymatic reactions by this cell.

consumes
Set of consumed metabolic species.

consuming_count
Number of metabolic classes consumed by this individual.

conversions_type
The set of conversion reactions in the genome.

copy_numbers

copy_numbers_eff_pumps

copy_numbers_enzymes

copy_numbers_inf_pumps

copy_numbers_tfs

delete_chromosome(*chrom*, *time*)
Chromosome deletion.

Creates a mutation object that is then responsible for calling the appropriate genomic operations (deletion methods of chromosome and genome updation functions of genome. The Cell then updates its gene product counts accordingly.

Parameters

- **chrom** (*VirtualMicrobes.virtual_cell.Chromosome.Chromosome*) – chromosome to be deleted
- **time** (*int*) – simulation time

Returns

Return type VirtualMicrobes.mutation.Mutation.ChromosomeDeletion

delete_genes(*genes*, *time*)

delete_stretch(*chrom*, *start_pos*, *end_pos*, *time*, *verbose=False*)
Delete a stretch of genes in the genome.

The chromosome, start position and end position (exclusive) uniquely determine a sequence of genes that will be deleted. The copy number of gene products in the cell is reduced. A Mutation object will be returned.

Parameters

- **chrom** (*VirtualMicrobes.virtual_cell.Chromosome.Chromosome*) – target chromosome
- **start_pos** (*int*) – position index of the start of the stretch
- **end_pos** (*int*) – position index of the end of the stretch
- **time** (*int*) – simulation time
- **verbose** (*bool*) – verbosity

Returns

Return type VirtualMicrobes.mutation.Mutation.StretchDeletion

die(*time*, *verbose=False*, *clear_time_courses=False*, *wiped=False*)
Cell death.

Informs genomic units of cell death.

Parameters

- **time** (*float*) – simulation time point

- **verbose** (mol : *VirtualMicrobes.event.Molecule.Molecule*) – metabolitebool report on cell death
- **clear_time_courses** (*bool*) – set time courses to None
- **wiped** (*bool*) – set if cell death was due to wiping of (a fraction of) the population

divide_volume (*factor=2.0*)
Divide the volume of the cell.

Parameters **factor** (*float*) – division factor

duplicate_chromosome (*chrom*, *time*)
Chromosome duplication.

Creates a mutation object that is then responsible for calling the appropriate genomic operations (duplication methods in chromosome and genome updation functions of genome. The Cell then updates its gene product counts accordingly.

Parameters

- **chrom** (*VirtualMicrobes.virtual_cell.Chromosome.Chromosome*) – chromosome to be duplicated
- **time** (*int*) – simulation time

Returns

Return type VirtualMicrobes.mutation.Mutation.ChromosomeDuplication

eff_pump_count

enz_avrg_promoter_strengths

enz_promoter_strengths

enz_subs_differential_ks

enz_subs_ks

enz_sum_promoter_strengths

enz_vmaxs

enzyme_count

enzymes
Enzyme products in the Cell.

Notes

Can include gene products of genes that are no longer in the genome.

exploiting
Return the set of metabolic classes that are imported and consumed.

exploiting_count
Number of metabolic classes imported and consumed by this individual.

export_type
Return the set of all metabolic classes exported by this cell.

exporter_type
The set of export reactions in the genome.

exporting_count
Number of metabolic classes exported by this individual.

external_hgt
List of external Horizontal Gene Transfers of this individual.

external_hgt_count
Number of external Horizontal Gene Transfers of this individual.

fiss_chromosome(*chrom*, *pos*, *time*)
Chromosome fission.

Breaks apart a chromosome at a given position. The two resulting parts will become new chromosome replacing the original in the genome. Creates a mutation object that is then responsible for calling the appropriate genomic operations.

Parameters

- **chrom** (*VirtualMicrobes.virtual_cell.Chromosome.Chromosome*) – chromosome to be broken
- **pos** (*int*) – position in chromosome of the break
- **time** (*int*) – simulation time

Returns

Return type VirtualMicrobes.mutation.Mutation.Fission

fuse_chromosomes(*chrom1*, *chrom2*, *end1*, *end2*, *time*)
Fusion of two chromosomes.

Two chromosomes are fused end to end. The new product replaces the original two chromosomes in the genome.

chrom1 [*VirtualMicrobes.virtual_cell.Chromosome.Chromosome*] first chromosome to be fused

chrom2 [*VirtualMicrobes.virtual_cell.Chromosome.Chromosome*] second chromosome to be fused

end1 [bool] indicate end or start of the first chromosome will be fused

end2 [bool] indicate end or start of the second chromosome will be fused

Returns

Return type VirtualMicrobes.mutation.Mutation.Fusion

gene_substrate_differential_ks(*ks_param*, *genes*)
Calculate differences between K parameters per substrate for a list of genes.

Parameters

- **ks_param** (*str*) – type of parameter
- **genes** (*list*) – list of genes for which to calculate the diffs

Returns

Return type array of diffs per gene

gene_substrate_ks(*ks_param*, *genes*)

gene_type_counts

generate_enzymes(*nr_enzymes*, *env*, *rand_gen*, *metabolic_pathway_dict*, *prioritize_influxed=True*)
Generate the set of enzymes of the cell.

Uses heuristics for constructing 'viable' initial metabolic pathways for the production of building blocks and energy metabolites.

Parameters

- **nr_enzymes** (*int*) – number of enzymes to generate
- **env** (*VirtualMicrobes.environment.Environment.Environment*) – environment provides the set of metabolites and possible metabolic reactions
- **rand_gen** (*RNG*) –
- **metabolic_pathway_dict** (*dict*) – dictionary to keep track of metabolites consumed and produced by the cell.
- **prioritize_influxed** (*bool*) – first create enzymes that can utilize metabolites that are present (influxed) in environment

Returns

Return type list of :class:'VirtualMicrobes.virtual_cell.Gene.MetabolicGene's

generate_pumps(*nr_pumps*, *env*, *rand_gen*, *metabolic_pathway_dict*, *import_first=True*, *prioritize_influxed=True*)
Generate set of transporters of the cell.

Use heuristics for connecting to the metabolic pathways formed so far by the enzymes present in the cell.

Parameters

- **nr_pumps** (*int*) – number of transporters to generate
- **env** (*VirtualMicrobes.environment.Environment.Environment*) – environment provides the set of metabolites and possible metabolic reactions
- **rand_gen** (*RNG*) –
- **metabolic_pathway_dict** (*dict*) – dictionary to keep track of metabolites consumed and produced by the cell.
- **import_first** (*bool*) – first create import enzymes for all metabolites that can be consumed in metabolism
- **prioritize_influxed** (*bool*) – first create transporters that can import metabolites that are present (influxed) in environment

Returns

Return type list of :class:'VirtualMicrobes.virtual_cell.Gene.Transporter's

See also:

generate_enzymes

generate_tfs(*nr_tfs*, *env*, *rand_gen*, *metabolic_pathway_dict*, *bb_ene_only=False*)
Create TFs of the cell.

Prioritizes TFs that sense metabolites that are relevant for the cell, i.e. metabolites that are actively consumed or produced by the cell's enzymes (or production function).

Parameters

- **nr_tfs** (*int*) – number of tfs to generate

- **env** (*VirtualMicrobes.environment.Environment.Environment*) – environment provides the set of metabolites and possible metabolic reactions
- **rand_gen** (*RNG*) –
- **metabolic_pathway_dict** (*dict*) – dictionary to keep track of metabolites consumed and produced by the cell.
- **bb_ene_only** (*bool*) – create TFs with either a building block or an energy carrier as ligand exclusively

Returns

Return type list of :class:‘VirtualMicrobes.virtual_cell.Gene.TranscriptionFactor‘s

See also:

generate_enzymes()

genes_get_prop_vals(*genes*, *get_prop*)

Construct array of values of a gene property for all duplicates of a list of genes.

Parameters

- **genes** (list of `VirtualMicrobes.virtual_cell:Gene:Gene` objects) – The genes for which to list the proprty
- **get_prop** (func [`VirtualMicrobes.virtual_cell:Gene:Gene`] -> value) – A function that takes a gene as argument and returns a value

Returns

Return type numpy array

genome_size

genotype

Construct frozen set of the genotype classification of this cell.

The genotype represents the gene functionality that the cell is capable of. It is expressed as the total set of transport, enzymatic and transcription sensing capabilities of the cell.

Returns

Return type frozenset

genotype_vector(*env*)

Construct boolean vector of gene-type presence-absence.

Parameters **env** (*VirtualMicrobes.environment.Environment.Environment*) – environment relative to which the gene type presence absence is determined

Returns

- *mapping of gene-function to*
- *VirtualMicrobes.event.Reaction.Reaction*|
- :class:‘VirtualMicrobes.event.Molecule.MoleculeClass‘s presence/absence.

get_ancestor(*gen_back*, *verbose=False*)

get_ancestor_at_time(*time*)

get_cell_size_time_course()

Time course of cell size.

get_gene_concentration_dict()
Mapping of gene products to current concentration.

get_gene_degradation_dict()
Mapping of gene products to degradation rates.

get_gene_diffusion_dict()
Mapping of gene products to diffusion rates.

get_gene_multiplicities_dict()
Mapping of gene products to copy numbers in the genome.

get_gene_prod_conc(*gene*)
Get current gene product concentration.

Parameters **gene** (*VirtualMicrobes.virtual_cell.Gene.Gene*) – gene product

Returns

Return type concentration value (float)

get_gene_time_course_dict()
Fetches time courses for gene products (proteins)

Returns

Return type dictionary of all time courses

get_gene_type_time_course_dict()
Return a dictionary of concentration time course data for different gene types.

get_mol_concentration_dict()
Mapping of internal molecule species to current concentrations.

get_mol_degradation_dict()
Mapping of internal molecule species to degradation rates.

get_mol_diffusion_dict()
Mapping of internal molecule species to diffusion rates.

get_mol_time_course_dict()
Mapping of internal molecule species to concentration time course.

get_pos_prod_time_course()
Time course of positive component of production rate.

get_raw_production_time_course()
Time course of raw production value.

get_small_mol_conc(*mol*)
Get current metabolite concentration.

Parameters **mol** (*VirtualMicrobes.event.Molecule.Molecule*) – metabolite

Returns

Return type concentration value (float)

get_time_points()
Array of (data) time points stored.

get_total_expression_level(*reaction*, *exporting=False*)
Get total expression of a certain type of reaction

Parameters

- **reaction** (*VirtualMicrobes.event.Reaction*) –

  or VirtualMicrobes.event.MoleculeClass

  reaction or moleculeclass for TFs

- **exporting** – whether pump is exporting

Returns

Return type concentration valule (float)

get_total_reaction_type_time_course_dict()
Return a dictionary of summed time courses per reaction type.

Gets the time courses per gene type and then sums concentrations of gene products with the same reaction (enzymes/pumps) or ligand (tfs).

get_toxicity_time_course()
Time course of cell toxicity.

grow_time_course_arrays(*factor=1.5*)
Grow time course arrays if they cannot hold enough new time points.

Parameters **factor** (*float*) – increase capacity with this factor

hgt_external(*hgt_rate*, *global_mut_mod*, *environment*, *time*, *rand_gene_params*, *rand_gen*)
Insert a randomly generated (external) gene into the genome.

Gene is created with random parameters and inserted (as a stretch) into a randomly picked chromosome and position.

Parameters

- **hgt_rate** (*float*) – probability external hgt
- **global_mut_mod** (*float*) – mutation scaling parameter of all mutation rates
- **environment** (*VirtualMicrobes.environment.Environment.Environment*) – simulation environment containing all possible reactions to draw a random gene for external HGT.
- **time** (*int*) – simulation time
- **rand_gene_params** (*dict*) – parameter space for properties of externally HGTed genes
- **rand_gen** (*RNG*) –

Returns

Return type VirtualMicrobes.mutation.Mutation.Insertion

hgt_internal(*hgt_rate*, *global_mut_mod*, *gp*, *time*, *rand_gene_params*, *rand_gen*, *rand_gen_np*)
Insert a randomly chosen gene (stretch) from a neighboring individual into the genome.

Potential donor individuals come from the neighbourhood of a focal grid point. Once a donor individual is selected, a random gene is chosen to be transferred to this individual.

Parameters

- **hgt_rate** (*float*) – probability external hgt
- **global_mut_mod** (*float*) – mutation scaling parameter of all mutation rates
- **gp** (*VirtualMicrobes.environment.Grid.GridPoint*) – focal grid point to select donor individual

- **time** (*int*) – simulation time
- **rand_gene_params** (*dict*) – parameter space for properties of externally HGTed genes
- **rand_gen** (*RNG*) –

Returns

Return type VirtualMicrobes.mutation.Mutation.Insertion

hybridize(*second_parent*, *time*)

import_type
Return the set of all metabolic classes imported by this cell.

importer_type
The set of import reactions in the genome.

importing_count
Number of metabolic classes imported by this individual.

inf_pump_count

init_building_blocks_dict(*nr_blocks*, *rand_gen*, *stois*)
Initialize the dictionary of cellular building blocks and their stoichiometry.

Parameters

- **nr_blocks** (*int*) – number of different metabolic building blocks of the cell.
- **rand_gen** (*RNG*) –
- **stois** (*tuple of int*) – (lower, higher) range of stoichiometric constants from which to randomly draw stoichiometries.

init_cell_params()
Set cell parameters based on simulation parameter settings.

init_cell_time_courses(*length=None*)
Initialize arrays to hold time course data.

Parameters **length** (*int*) – initial length of array

init_energy_mols(*environment*)
Store the energy molecules of the cell.

Parameters **environment** (*VirtualMicrobes.environment.Environment.Environment*) – simulation environment that contains metabolic universe

Notes

energy_mols are used in odes.pyx

init_gene_products(*concentration=None*)
Initialize gene products from the genes in the genome.

Gene products have a concentration and degradation rate. ODE system integration will use the gene products as input variables.

Parameters **concentration** (*float*) – initial concentration of gene products

init_genome(*environment*, *chrom_compositions=None*, *min_bind_score=None*, *prioritize_influxed=None*, *rand_gene_params=None*, *circular_chromosomes=None*, *better_tf_params=None*, *rand_gen=None*, *randomize=True*)

Iniitialize the genome of the cell.

The genome is constructed according to a list of chromosome composition descriptions. These descriptions specify the number of each gene type a set of chromosomes. Sets of metabolic , transport and transcription factor genes are initialized using heuristic functions that guarantee basic viability of the cell and a basic logic in transported and sensed metabolites, by taking into account the set of indispensible metabolites (building blocks and energy carriers). Finally, kinetic and other parameters of all genes are randomized and the genes distributed over the chromosomes.

Parameters

- **environment** (*VirtualMicrobes.environment.Environment.Environment*) – simulation environment that determines metabolic universe
- **chrom_compositions** (list of *VirtualMicrobes.my_tools.utility.GeneTypeNumbers*) – per chromosome composition of different gene types
- **min_bind_score** (*float*) – parameter for minimal binding score for transcription regulation
- **prioritize_influxed** (*bool*) – first choose enzymatic reactions that use substrates that are influxed in *environment*
- **rand_gene_params** (*VirtualMicrobes.my_tools.utility.ParamSpace*) – range from which to draw randomized gene parameters
- **circular_chromosomes** (*bool*) – make the chromosomes circular
- **rand_gen** (*RNG*) –
- **randomize** (*bool*) – whether gene parameters should be randomized after creation

Returns

Return type *VirtualMicrobes.virtual_cell.Genome.Genome*

init_mol_time_course(*mol_struct*, *length=None*)

Initialize an array for time course data in a molecule structure SmallMol or GeneProduct.

Parameters

- **mol_struct** (*VirtualMicrobes.my_tools.utility.SmallMol*) – simple c-struct like object that holds data on small molecules or gene products
- **length** (*int*) – initial length of time course array

init_mol_views()

Initialize 'aliases' for the set of small molecules and that of gene products in the cell.

init_mutations_dict()

Initialize a dictionary for storing the mutations in the life time of this cell.

init_time_courses()

Initialize arrays that hold time course data for molecules and cell variables

insert_stretch(*chrom*, *insert_pos*, *stretch*, *time*, *is_external*, *verbose=False*)

Insert a stretch of exogenous genomic material. Also adds product to the dict of to proteins made by the cell

internal_hgt

List of internal Horizontal Gene Transfers of this individual.

internal_hgt_count
Number of internal Horizontal Gene Transfers of this individual.

invert_stretch(*chrom*, *start_pos*, *end_pos*, *time*, *verbose=False*)
Invert a stretch of genes in the genome in place.

The chromosome, start position and end position (exclusive) uniquely determine a sequence of genes that will be inverted. A Mutation object will be returned.

Parameters

- **chrom** (*VirtualMicrobes.virtual_cell.Chromosome.Chromosome*) – target chromosome
- **start_pos** (*int*) – position index of the start of the stretch
- **end_pos** (*int*) – position index of the end of the stretch
- **time** (*int*) – simulation time
- **verbose** (*bool*) – verbosity

Returns

Return type VirtualMicrobes.mutation.Mutation.Inversion

is_autotroph(*env*)
Determine if cell is autotrophic within an environment.

Autotrophy is defined as the ability to produce building blocks from precursors that are present 'natively' in the environment. This may be done in a multistep pathway in which the cell produces intermediates with its own metabolic enzymes.

Parameters **env** (*VirtualMicrobes.environment.Environment.Environment*) – the environment relative to which autotrophy is tested

See also:

is_heterotroph()

is_clone(*ref_cell*)
Is clone

Using string representation of genome to see if cells have the same genome

Parameters **ref_cell** (*VirtualMicrobes.virtual_cell.Cell.Cell*) –

Returns

Return type bool

is_heterotroph(*env*)
Determine if cell is heterotrophic within an environment.

Heterotrophy is defined as the ability to produce the building blocks from precursors that could only be present as (by)products from metabolism of other individuals in the environment, but not natively present (through influx).

Parameters **env** (*VirtualMicrobes.environment.Environment.Environment*) – the environment relative to which autotrophy is tested

See also:

is_autotroph()

mean_life_time_cell_size
Average cell size over the life time of the individual.

mean_life_time_pos_production
Average positive production value over the life time of the individual.

mean_life_time_production
Average production value over the life time of the individual.

mean_life_time_toxicity
Average toxicity value over the life time of the individual.

metabolic_type
Return a set of sets that uniquely defines the metabolic functions of this cell.

metabolic_type_vector(*env*)
Construct boolean vector of metabolic capacity of the cell.

Based on the complete set of environmental molecule classes, write out a cells metabolism in terms of produced, consumed, imported and exported molecule classes by the cells metabolism.

Parameters **env** (*VirtualMicrobes.environment.Environment.Environment*) – environment relative to which the metabolic capacity is determined

Returns

- *mapping of metabolic-function to*
- :class:'VirtualMicrobes.event.Molecule.MoleculeClass's presence/absence.

mutate(*time*, *environment*, *rand_gene_params=None*, *mutation_rates=None*, *mutation_param_space=None*, *global_mut_mod=None*, *excluded_genes=None*, *point_mutation_dict=None*, *point_mutation_ratios=None*, *regulatory_mutation_dict=None*, *regulatory_mutation_ratios=None*, *rand_gen=None*, *rand_gen_np=None*, *verbose=False*)
Apply mutations to the genome.

In turn, chromosome mutations, gene stretch mutations, point mutations and regulatory region mutations are applied. Mutations happen with probabilities supplied in a mutation_rates dict. The parameter spaces of the different types of mutations are supplied in the *mutation_param_space*

Parameters

- **time** (*int*) – simulation time
- **environment** (*VirtualMicrobes.environment.Environment.Environment*) – environment holds Metabolites that can be potential ligands
- **rand_gene_params** (*VirtualMicrobes.my_tools.utility.ParamSpace*) – parameter space to draw new random parameter values
- **mutation_rates** (*attr_dict*) – dict with all mutation rates
- **mutation_param_space** (*VirtualMicrobes.my_tools.utility.MutationParamSpace*) – parameter space and bounds for mutation effect
- **global_mut_mod** (*float*) – modifier for Cell wide mutation rate
- **point_mutation_dict** (*dict*) – maps gene types to type specific parameter-> modifier-function mapping
- **point_mutation_ratios** (*dict*) – mapping from parameter type to type specific relative mutation ratio
- **regulatory_mutation_dict** (*dict*) – maps gene types to type specific parameter-> modifier-function mapping
- **regulatory_mutation_ratios** (*dict*) – mapping from parameter type to type specific relative mutation ratio

- **rand_gen** (*RNG*) –
- **rand_gen_np** (*RNG*) – numpy random number generator
- **verbose** (*bool*) – verbosity

Returns

Return type list of mutations

mutate_uptake(*p_mutate*, *mutation_param_space*, *global_mut_mod*, *rand_gen*)

Apply mutations to rate of uptake of eDNA. Currently this uptake is a multiplier modulating the already existing natural occurence of transformation.

p_mutate [float] mutation probability

mutation_param_space [*VirtualMicrobes.my_tools.utility.MutationParamSpace*] parameter space and bounds for mutation effect

global_mut_mod [float] modifier for Cell wide mutation rate

rand_gen : RNG

Returns (void for now)

nodes_edges(*genes=None*)

Returns a list of nodes and edges of the individual's gene regulatory network.

Nodes are simply all the genes. Edges are TF binding site to operator interactions between genes in the nodes list.

Parameters **genes** (list of :class:'VirtualMicrobes.virtual_cell.Gene.Gene's) – genes for which to find nodes and edges. If None (default) all gene products in the cell that have copy nr > 0.

Returns

Return type nodes(list),edges(list of tuples)

point_mut

List of point mutations of this individual.

point_mut_count

Number of point mutations of this individual.

point_mutate_gene(*chrom*, *pos*, *mut_dict*, *point_mut_ratios*, *environment*, *mutation_param_space*, *rand_gene_params*, *time*, *rand_gen*)

Mutate a gene at a particular position of a chromosome.

Depending on the 'type' of the gene, a different set of parameters may be mutated. One parameter of the gene randomly selected to be mutated, according to a weighted roulette wheel draw, with the probabilities given by the *point_mut_ratios* associated with each parameter. The selected parameter for the gene will be modified using a particular *mut_modifier* function for this parameter type. A *VirtualMicrobes.mutate.Mutation.PointMutation* object is constructed that holds references to the original and mutated gene and the old and new parameter value, and allowing for reversal of the mutation operation. Finally, append the mutation in the Cell's list of single gene mutations.

Parameters

- **chrom** (*VirtualMicrobes.virtual_cell.Chromosome.Chromosome*) – chromosome that holds the gene to be mutated
- **pos** (*int*) – position index of the gene to mutate
- **mut_dict** (*dict*) – maps gene types to type specific parameter-> modifier-function mapping

- **point_mut_ratios** (*dict*) – mapping from parameter type to type specific relative mutation ratio
- **environment** (*VirtualMicrobes.environment.Environment.Environment*) – environment holds Metabolites that can be potential ligands
- **mutation_param_space** (*VirtualMicrobes.my_tools.utility.MutationParamSpace*) – parameter space and bounds for mutation effect
- **rand_gene_params** (*VirtualMicrobes.my_tools.utility.ParamSpace*) – parameter space to draw new random parameter values
- **time** (*int*) – simulation time point that mutation is applied
- **rand_gen** (*RNG*) –

Returns

Return type VirtualMicrobes.mutation.Mutation.PointMutation

point_mutate_gene_param(*chrom*, *pos*, *param*, *mut_modifier*, *environment*, *mutation_param_space*, *rand_gene_params*, *time*, *rand_gen*)

Mutate a particular parameter of a gene at a particular position.

A random new value is drawn for the parameter. Then the point mutation is initialized and applied. The Cell's gene products are updated to reflect the introduction of a new, mutated protein.

Parameters

- **chrom** (*VirtualMicrobes.virtual_cell.Chromosome.Chromosome*) – chromosome that holds the gene
- **pos** (*int*) – position index of the gene to mutate
- **param** (*str*) – name of parameter to mutate
- **mut_modifier** (*func*) – function that from parameter value to new parameter value
- **environment** (*VirtualMicrobes.environment.Environment.Environment*) – environment holds Metabolites that can be potential ligands
- **mutation_param_space** (*VirtualMicrobes.my_tools.utility.MutationParamSpace*) – parameter space and bounds for mutation effect
- **rand_gene_params** (*VirtualMicrobes.my_tools.utility.ParamSpace*) – parameter space to draw new random parameter values
- **time** (*int*) – simulation time point that mutation is applied
- **rand_gen** (*RNG*) –

Returns

Return type VirtualMicrobes.mutation.Mutation.PointMutation

point_mutate_genome(*p_mutate*, *mut_dict*, *global_mut_mod*, *point_mut_ratios*, *excluded_genes*, *environment*, *mutation_param_space*, *rand_gene_params*, *time*, *rand_gen*)

Apply point mutations to genes in the genome, according to point mutation rate.

p_mutate [float] point mutation probability

mut_dict [dict] maps gene types to type specific parameter-> modifier-function mapping

global_mut_mod [float] modifier for Cell wide mutation rate

point_mut_ratios [dict] mapping from parameter type to type specific relative mutation ratio

environment [*VirtualMicrobes.environment.Environment.Environment*] environment holds Metabolites that can be potential ligands

mutation_param_space [*VirtualMicrobes.my_tools.utility.MutationParamSpace*] parameter space and bounds for mutation effect

rand_gene_params [*VirtualMicrobes.my_tools.utility.ParamSpace*] parameter space to draw new random parameter values

time [int] simulation time point that mutation is applied

rand_gen : RNG

Returns

Return type list of mutations applied in this round

pos_production
Current bruto production value.

producer_type
Return the set of all metabolic classes produced in enzymatic reactions by this cell.

produces
Set of produced metabolic species.

producing_count
Number of metabolic classes produced by this individual.

promoter_strengths

providing
Return the set of metabolic classes that are produced AND exported.

See also:

producer_type(), *export_type()*

providing_count
Number of metabolic classes produced and exported by this individual.

prune_dead_phylo_branches()
Prune branches of phylogenetic trees of all phylogenetically tracked entities.

Recursively, prunes phylogenetic branches of this individual if it is not alive. Also prunes phylogenies of phylogenetic entities in the genome.

Returns

Return type pruned cells (set), pruned chromosomes (set), pruned genes (set)

pump_avrg_promoter_strengths

pump_count

pump_ene_differential_ks

pump_ene_ks

pump_promoter_strengths

pump_subs_differential_ks

pump_subs_ks

pump_sum_promoter_strengths

pump_vmaxs

pumps
Transporter products in the Cell.

Notes

Can include gene products of genes that are no longer in the genome.

qual_expr_diff(*ref_cell*)
Calculate a qualitative expression difference with a reference cell.

Parameters **ref_cell** (*VirtualMicrobes.virtual_cell.Cell.Cell*) – reference cell to compare gene expression.

Returns

Return type mean positive (relative) difference of all gene expression values.

raw_production
Current production value.

raw_production_change_rate
Current production change rate value.

reaction_genotype
Construct frozen set of the reaction genotype classification of this cell.

The genotype represents the enzyme functionality that the cell is capable of. It is expressed as the total set of transport, enzymatic capabilities of the cell, but excluding the tf sensing capabilities.

Returns

Return type frozenset

See also:

func *genotype*

reaction_set_dict
Dictionary of enzymatic reaction types to reactions.

Reactions of genes are mapped to their reaction types (Conversion, Transport).

reaction_set_dict2
Dictionary of enzymatic reaction types to reactions.

Reactions of genes are mapped to their reaction types (Conversion, Transport).

Notes

Different formatting from *reaction_set_dict*.

read_genome(*environment*, *gene_dict*, *genome_order*, *verbose=False*)
Reads and sets genome based on genes dictionary and a list of gene indexes representing the genome order.

Parameters

- **environment** (*VirtualMicrobes.environment.Environment.Environment*) – environment provides the reactions to match to the genes

- **gene_dict** (*dict*) – dictionary that contains all properties of the gene most are just values such as v_max, prom_str, type etc. ene_ks and subs_ks are themselves dicts for each mol
- **genome_order** (*list*) – a list of indexes that represent in what order the genes should be put into the genome

Returns

Return type class:'VirtualMicrobes.virtual_cell.Genome

reduce_gene_copies(*gene*)

Reduce the copy number for a gene.

Parameters **gene** (*VirtualMicrobes.virtual_cell.Gene.Gene*) – The gene for which copy number is reduced.

regulatory_region_mutate(*chrom*, *genome*, *pos*, *mut_dict*, *mut_ratios*, *stretch_exp_lambda*, *time*, *rand_gen*, *rand_gen_np*)

Mutate part of the sequence of a regulatory region of a gene.

The mutation may be either a copying and translocation of an existing sequence somewhere in the genome to new position, or the insertion of a random sequence.

chrom [*VirtualMicrobes.virtual_cell.Chromosome.Chromosome*] chromosome that holds the gene to be mutated

genome [*VirtualMicrobes.virtual_cell.Genome.Genome*] full genome to find donor sequence

pos [int] position on chromosome of gene to be mutated

mut_dict [dict] maps gene types to type specific parameter-> modifier-function mapping

mut_ratios [dict] mapping from parameter type to type specific relative mutation ratio

stretch_exp_lambda [float] geometric distribution parameter that determines expected stretch length

time [int] simulation time

rand_gen : RNG rand_gen_np : RNG

numpy random number generator

Returns

Return type VirtualMicrobes.mutation.Mutation.OperatorInsertion

regulatory_region_mutate_genome(*mutation_rates*, *mut_dict*, *global_mut_mod*, *reg_mut_ratios*, *time*, *rand_gen*, *rand_gen_np*)

Apply regulatory region mutations to genes in the genome.

mutation_rates [attr_dict] dict with all mutation rates

mut_dict [dict] maps gene types to type specific parameter-> modifier-function mapping

global_mut_mod [float] modifier for Cell wide mutation rate

reg_mut_ratios [dict] mapping from parameter type to type specific relative mutation ratio

time [int] simulation time point that mutation is applied

rand_gen : RNG rand_gen_np : RNG

numpy random number generator

Returns

Return type list of mutations applied in this round

remove_unproduced_gene_products (*conc_cutoff=None*)
Remove gene products when they are no longer produced and have a below threshold concentrations.

Parameters **conc_cutoff** (*float*) – threshold concentration below which gene product is removed

Returns True if any product was removed

Return type bool

reproduce (*spent_production*, *time*, *second_parent=None*)
Create a new child of this cell.

Copies all relevant properties, including the genome. Divides the original cell volume between the parent and child.

Parameters

- **spent_production** (*float*) – amount of the production variable spent for reproduction
- **time** (*int*) – simulation time point
- **second_parent** (*VirtualMicrobes.virtual_cell.Cell.Cell*) – an (optional) second parent that contributes to reproduction

Returns

Return type *VirtualMicrobes.virtual_cell.Cell.Cell*

reset_grn (*min_bind_score=None*)
Recalculate and reset all binding interactions in the genome.

Parameters **min_bind_score** (*float*) – minimum identity score to set a regulatory interaction.

resize_time_courses (*new_max_time_points*)
Set a new size for arrays that hold time course data.

Parameters **new_max_time_points** (*int*) – max number of time points

scale_mol_degr_rate (*mol*, *env*, *degr_const*, *ene_degr_const*, *bb_degr_const*)
Return scaled internal rate of metabolite degradation, relative to its external rate.

The base rate is the metabolite specific external degradation rate. If internal rates have been chosen (not None), the base rate will be scaled with the ratio of internal/external. Rates can be different for building block and energy metabolites.

Parameters

- **mol** (*VirtualMicrobes.event.Molecule.Molecule*) – metabolite
- **env** (*VirtualMicrobes.environment.Environment.Environment*) – environment containing metabolites
- **degr_const** (*float*) – degradation constant of metabolites that are not bbs or energy
- **ene_degr_const** (*float*) – degradation constant of energy metabolites
- **ene_degr_const** – degradation constant of building block metabolites

Returns

Return type scaled degradation rate (float)

sequence_mut
List of sequence mutations of this individual.

sequence_mut_count
Number of sequence mutations of this individual.

set_gene_prod_conc(*gene*, *conc*)
Set the new concentration of a gene product.

The last concentration time point will be overwritten.

Parameters

- **gene** (*VirtualMicrobes.virtual_cell.Gene.Gene*) – gene product
- **conc** (*float*) – new concentration

set_mol_concentrations_from_time_point()
Record cellular concentrations and values from time course arrays.

During the integration step, for each molecule or other variable time course data is stored in an array. The position that was last filled is the new concentration. The value stored under index pos will be copied to a dedicated concentration or 'value' member.

Parameters **pos** – position in array to

set_molconcs(*concdict*)
Set molecule concentrations based on dict.

Parameters **concdict** (*dict*) – dict with values for mol concentrations

set_properties(*celldict*)
Set cell properties based on dict.

Parameters **celldict** (*dict*) – dict with values for cell properties

set_small_mol_conc(*mol*, *conc*)
Set the new concentration of a metabolite.

The last concentration time point will be overwritten.

Parameters

- **mol** (*VirtualMicrobes.event.Molecule.Molecule*) – metabolite
- **conc** (*float*) – new concentration

set_small_mol_degr(*mol*, *reac*, *rate*)
Set metabolite degradation parameter.

Parameters

- **mol** (*VirtualMicrobes.event.Molecule.Molecule*) – metabolite
- **reac** (*VirtualMicrobes.event.Reaction.Degradation*) – degradation reaction
- **rate** (*float*) – degradation rate of this metabolite

set_small_mol_diff(*mol*, *reac*, *rate*)
Set metabolite diffusion parameter.

Parameters

- **mol** (*VirtualMicrobes.event.Molecule.Molecule*) – metabolite

- **reac** (*VirtualMicrobes.event.Reaction.Diffusion*) – diffusion reaction
- **rate** (*float*) – diffusion rate of this metabolite

set_state_from_file(*environment*, *filename*, *max_lin_marker*)
Set the state of the cell based on a configuration file.

Parameters

- **environment** (*VirtualMicrobes.environment.Environment.Environment*) – environment defines metabolic universe for the cell
- **filename** (*string*) – name of cell state file
- **lineagemarker** (*marker to distinguish lineage*) –

set_state_from_ref_cell_tp(*ref_cell*, *tp_index=0*, *verbose=False*)
Reset cell properties to the state of a reference cell at a chosen time point.

Parameters

- **ref_cell** (*VirtualMicrobes.virtual_cell.Cell.Cell*) – take values from this reference individual
- **tp_index** (*int*) – index of reference time point to take reset value
- **verbose** (*bool*) – verbosity flag

stretch_del
List of stretch deletions of this individual.

stretch_del_count
Number of stretch deletions of this individual.

stretch_invert
List of stretch inversions of this individual.

stretch_invert_count
Number of stretch inversions of this individual.

stretch_mut
List of stretch mutations of this individual.

stretch_mut_count
Number of stretch mutations of this individual.

stretch_mutate_genome(*time, mutation_rates, global_mut_mod, rand_gen, rand_gen_np, mut_types=['tandem_dup', 'stretch_del', 'stretch_invert', 'stretch_translocate'], verbose=False*)
Iterate over chromosomes and positions to select stretches of genes for mutational events.

For every chromosome, iterate over positions and select front and end positions of stretch mutations. The direction of iteration is randomly chosen (back-to-front or front-to-back). Multiple mutations per chromosome may occur. Every position may be independently selected as the front site of a stretch mutation. The length of the stretch is a geometrically distributed random variable, using a lambda parameter. The end position is the minimum of the remaining chromosome length and the randomly drawn stretch length. If any positions remain in the chromosome after the stretch mutation, these positions are then iterated over in a random direction (direction of iteration is reversed at random after the application of the stretch mutation). Reversing direction is significant, because it randomizes the locations of 'truncated' stretches when reaching the end of a chromosome.

Parameters

- **time** (*int*) – simulation time

- **mutation_rates** (*attr_dict*) – dict with all mutation rates
- **mut_dict** (*dict*) – maps gene types to type specific parameter-> modifier-function mapping
- **global_mut_mod** (*float*) – modifier for Cell wide mutation rate
- **rand_gen** (*RNG*) –
- **rand_gen_np** (*RNG*) – numpy random number generator
- **mut_types** (*list of str*) – mutation types to apply

strict_exploiting
Return exploited resources classes that are not produced by self.

See also:

exploiting()

strict_exploiting_count
Number of metabolic classes imported and consumed, but not produced by this individual.

strict_providing
Return provided resources classes that are not imported by self.

See also:

providing()

strict_providing_count
Number of metabolic classes produced, exported but not consumed by this individual.

sum_promoter_strengths

tandem_dup
List of stretch duplications of this individual.

tandem_dup_count
Number of stretch duplications of this individual.

tandem_duplicate_stretch(*chrom*, *start_pos*, *end_pos*, *time*, *verbose=False*)
Duplicate a stretch of genes in the genome in place.

The chromosome, start position and end position (exclusive) uniquely determine a sequence of genes that will be duplicated and inserted immediately behind the original stretch. A Mutation object will be returned.

Parameters

- **chrom** (*VirtualMicrobes.virtual_cell.Chromosome.Chromosome*) – target chromosome
- **start_pos** (*int*) – position index of the start of the stretch
- **end_pos** (*int*) – position index of the end of the stretch
- **time** (*int*) – simulation time
- **verbose** (*bool*) – verbosity

Returns

Return type VirtualMicrobes.mutation.Mutation.TandemDuplication

tf_avrg_promoter_strengths

tf_count

tf_differential_reg

tf_k_bind_ops

tf_ligand_differential_ks

tf_ligand_ks

tf_promoter_strengths

tf_sensed
The set of molecule classes that are sensed by TFs.

tf_sum_promoter_strengths

tfs
TF products in the Cell.

Notes

Can include gene products of genes that are no longer in the genome.

toxicity
Current toxicity value.

toxicity_change_rate
Current toxicity value change rate.

tp_index
Index of last time point stored.

translocate
List of stretch translocations of this individual.

translocate_count
Number of stretch translocations of this individual.

translocate_stretch(*chrom*, *start_pos*, *end_pos*, *target_chrom*, *insert_pos*, *invert*, *time*, *verbose=False*)
Translocate a stretch of genes in the genome in place.

The chromosome, start position and end position (exclusive) uniquely determine a sequence of genes that will be excised and inserted at a new position in target chromosome. A Mutation object will be returned.

Parameters

- **chrom** (*VirtualMicrobes.virtual_cell.Chromosome.Chromosome*) – chromosome with the original stretch
- **start_pos** (*int*) – position index of the start of the stretch
- **end_pos** (*int*) – position index of the end of the stretch
- **target_chrom** (*VirtualMicrobes.virtual_cell.Chromosome.Chromosome*) – target chromosome for insertion
- **insert_pos** (*int*) – position index where stretch will be inserted
- **time** (*int*) – simulation time
- **verbose** (*bool*) – verbosity

Returns

Return type VirtualMicrobes.mutation.Mutation.Translocation

trophic_type(*env*)
Trophic type classification of individual's metabolic capacities.

Based on the metabolic reactions present in the individual's genome an individual can be classified as being 'autotrophic' and/or 'heterotrophic'. Facultative mixotrophs are defined as having both autotrophic and heterotrophic metabolic capacities, while obligate mixotroph are neither self-sufficient autotrophs, nor heterotrophs.

Parameters **env** (*VirtualMicrobes.environment.Environment.Environment*) – the environment trophic type is determined in

See also:

is_autotroph(), *is_heterotroph()*

truncate_time_courses(*max_tp=None*)
Truncate the time course data.

If no maximum is supplied, truncates time courses to the parts of the arrays that have actually been filled with time points, discarding the empty last part of the array.

Intended to be called when a cell dies and no additional data points are expected to be stored.

Parameters **max_tp** (*int*) – maximum number of time points retained

uid = 0
A Cell in the Simulation.

Initializes the state of the cell (without genome); notably:

- internal molecule dictionaries
- arrays to hold time course data of internal molecule concentrations
- mutation dictionary, a record of mutations that occured during reproduction
- internal state variables for volume, toxicity etc

Parameters

- **params** (attrdict.AttrDict) – mapping object that holds global simulation parameters
- **environment** (*VirtualMicrobes.environment.Environment*) – environment that cell lives in; determines molecular and reaction universe
- **rand_gen** (*RNG*) –
- **fromfile** (*path to .cell-file to reproduce cell from*) –
- **toxicity_function** (*func*) – calculates toxicity based on actual and threshold level of a concentration
- **toxicity_effect_function** (*func*) – calculates added death rate from toxicity

update(*state*)

update_grn(*min_bind_score=None*)
Update the regulatory network, by finding (new) matches between TF binding sites and gene operators.

min_bind_score [float] minimum identity score to set a regulatory interaction.

update_mutated_gene_product(*old*, *new*)
Decrease the copy number of the old gene and increase/initialize the new gene.

Parameters

- **old** (*VirtualMicrobes.virtual_cell.Gene.Gene*) – pre mutation gene
- **new** (*VirtualMicrobes.virtual_cell.Gene.Gene*) – post mutation gene

update_small_molecules (*env*, *conc*)
Used when a new molecule enters the game

update_small_molecules_degr (*env*, *degr_const*, *ene_degr_const*, *bb_degr_const*)
Update membrane degradation parameter of metabolites.

env [*VirtualMicrobes.environment.Environment.Environment*] environment containing reactions and base degradation rates.

update_small_molecules_diff (*env*)
Update membrane diffusion parameter of metabolites.

env [*VirtualMicrobes.environment.Environment.Environment*] environment containing reactions and membrane diffusion rates.

upgrade ()
Upgrading from older pickled version of class to latest version. Version information is saved as class variable and should be updated when class invariants (e.g. fields) are added.

volume
Current cell volume.

VirtualMicrobes.virtual_cell.Cell.**make_inherited_hgt_dict**()

VirtualMicrobes.virtual_cell.Cell.**make_mutations_dict**()

VirtualMicrobes.virtual_cell.Chromosome module

class VirtualMicrobes.virtual_cell.Chromosome.**Chromosome**(*gene_list=None*, *positions=None*, *time_birth=0*, *circular=False*, ***kwargs*)
Bases: *VirtualMicrobes.virtual_cell.PhyloUnit.PhyloBase*

A chromosome contains genes in spatial order.

The chromosome class defines methods for chromosomal mutations * duplication * deltion * fission * fusion and mutations on the level of gene stretches * stretch deltion * stretch duplication * stretch inversion * stretch insertions

append_gene (*g*)
Append gene at end of chromosome.

Parameters **g** (class *virtual_cell.Gene.Gene*) – gene to add

delete_stretch (*start_pos*, *end_pos*)
Deletes a stretch of genes.

Parameters

- **start_pos** (*int*) – first position of stretch
- **end_pos** (*int*) – last position (exclusive) of stretch

Returns

Return type iterable of genes; the deleted stretch

duplicate(*time*)
Duplicate the chromosome.

Creates two identical copies of the chromosome. The original version is deleted.

Parameters **time** (*int*) – simulation time

Returns

Return type Returns tuple of two chromosomes

fiss(*pos*, *time*)
Chromosome fission.

A fission splits a chromosome, creating two new chromosomes.

Parameters

- **pos** (*int*) – position of fission
- **time** (*int*) – simulation time

Returns

Return type returns tuple of two chromosomes

classmethod fuse(*chrom1*, *chrom2*, *time*, *end1=True*, *end2=True*)
Fuse two chromosomes.

Parameters

- **chrom1** (*VirtualMicrobes.virtual_cell.Chromosome.Chromosome*) – first chromosome to fuse
- **chrom2** (*VirtualMicrobes.virtual_cell.Chromosome.Chromosome*) – second chromosome to fuse
- **time** (*int*) – simulation time
- **end1** (*bool*) – the end of chromosome1 will be fused
- **end2** (*bool*) – the end of chromosome2 will be fused

Returns

Return type returns the new fused chromosome

init_positions(*circular=False*)
Initialize chromosome positions.

Parameters **circular** (*boolean*) – if true define as circular list

insert_stretch(*stretch*, *pos*)
Insert a stretch of genes at position.

Parameters

- **stretch** (*iterable*) – stretch of genes to insert
- **pos** (*int*) – insert position

invert(*start_pos*, *end_pos*)
Invert a stretch of genes.

Parameters

- **start_pos** (*int*) – first position of stretch
- **end_pos** (*int*) – last position (exclusive) of stretch

Returns

Return type iterable of genes; the inverted stretch

positions

tandem_duplicate(*start_pos*, *end_pos*)
Duplicates a stretch of genes 'in place', and insert after end_pos.

Parameters

- **start_pos** (*int*) – first position of stretch
- **end_pos** (*int*) – last position (exclusive) of stretch

Returns

Return type iterable of genes; the duplicated stretch

toJSON(*index*, **args*, ***kwargs*)
Create JSON representation of chromosome.

Parameters **index** (*int*) – a position index of the chromosome

Returns

Return type json dict

translocate_stretch(*start_pos*, *end_pos*, *target_pos*, *target_chrom*)
Translocate a stretch of genes to a target chromosome and position.

Parameters

- **start_pos** (*int*) – first position of stretch
- **end_pos** (*int*) – last position (exclusive) of stretch
- **target_pos** (*int*) – position to insert stretch
- **target_chrom** (*VirtualMicrobes.virtual_cell.Chromosome.Chromosome*) – target chromosome for insertion

uid = 0

VirtualMicrobes.virtual_cell.Gene module

class VirtualMicrobes.virtual_cell.Gene.**Gene**(*type_*, *pr_str=1.0*, *operator_seq_len=10*, *operator=None*, *fixed_length=None*, *is_enzyme=False*, *promoter_phylo_type='base'*, *operator_phylo_type='base'*, ***kwargs*)
Bases: *VirtualMicrobes.virtual_cell.GenomicElement.GenomicElement*

Representation of gene in the genome.

Gene is the base class for various types of genes in the genome. A gene has * a set of type specific parameters * a promoter that determines basal gene expression level * an operator that allows interaction and expression modulation by

transcription factors

Parameters

- **type** (*string*) – type of gene

- **pr_str** (*float*) – promoter strength
- **operator_seq_len** (*int*) – sequence length of operator
- **operator** (*iterable*) – operator sequence as bitstring
- **fixed_length** (*bool*) – if false, operator length can change
- **is_enzyme** (*bool*) – if true, is an enzyme type

is_enzyme

mutated(*param*, *new_val*, *time*, *verbose=False*)
Mutates a parameter of the gene.

To maintain a full ancestry, the mutation should be applied to a (shallow) copy of the gene and this copy reinserted in the original ancestral position. The shallow copy will however have a new deepcopied version of the parameter dictionary so that the mutation will not affect the ancestral gene state.

Parameters

- **param** (*string*) – parameter to mutate
- **new_val** (*new parameter value*) –

Returns

Return type Returns the mutated copy of the gene.

operator

params

promoter

randomize(*rand_gene_params*, *rand_gen*, *better_tfs=False*, ***kwargs*)
Randomizes gene properties.

Parameters

- **rand_gene_params** (*dict*) – parameters of randomization function
- **rand_gen** (*RNG*) –

randomize_params(*rand_gene_params*, *rand_generator*)

toJSON(*attr_mapper*, *index*, *d=None*, **args*, ***kwargs*)
Create JSON representation of gene.

Parameters **attr_mapper** (*dict*) – mapping from properties to colours

Returns

Return type JSON dict

class VirtualMicrobes.virtual_cell.Gene.**MetabolicGene**(*reaction*, *substrates_ks=10.0*, *v_max=1.0*, *forward=True*, ***kwargs*)

Bases: *VirtualMicrobes.virtual_cell.Gene.Gene*

Version

Author

ode_params()
Returns a list of dictionaries of parameters necessary and sufficient to parameterize an ODE for all the sub-reactions associated with this Gene.

randomize_params(*rand_gene_params*, *rand_generator*)

reaction

simple_str()

toJSON(**args*, ***kwargs*)
Create JSON representation of gene.

Parameters **attr_mapper** (*dict*) – mapping from properties to colours

Returns

Return type JSON dict

class VirtualMicrobes.virtual_cell.Gene.**Promoter**(*pr_str*, *time_birth=0*, ***kwargs*)
Bases: *VirtualMicrobes.virtual_cell.PhyloUnit.PhyloBase*

A promoter sequence object.

A promoter is exclusively associated with a genomic element. It encodes a basal promoter strength that is a factor in the gene expresion level.

mutate(*mut_modifier*, *rand_gen*)
Mutates the promoter.

Parameters

- **mut_modifier** (*func*) – a function that changes the promoter parameters.
- **rand_gen** (*RNG*) –

randomize(*rand_gene_params*, *rand_generator*)
Randomize the promoter parameters.

Parameters

- **rand_gene_params** (*dict*) – parameters for the randomization function
- **rand_generator** (*RNG*) –

strength

toJSON(*attr_mapper*, **args*, ***kwargs*)
Creates JSON representation of promoter.

Parameters **attr_mapper** (*dict*) – mapping from attributes to colors

Returns

Return type A JSON dictionary.

uid = 0

class VirtualMicrobes.virtual_cell.Gene.**TranscriptionFactor**(*ligand_mol_class*, *ligand_ks=10.0*, *ligand_cooperativity=1.0*, *binding_seq_len=10*, *binding_seq=None*, *eff_apo=1.0*, *eff_bound=1.0*, *k_bind_op=1.0*, *binding_cooperativity=2*, *sense_external=False*, ***kwargs*)

Bases: *VirtualMicrobes.virtual_cell.Gene.Gene*

Version

Author

binding_sequence

init_ligand(*ligand_class*, *ligand_ks=10*)

Set ligand class and the kinetic (K) constants of binding affinity for individual molecule species.

Parameters

- **ligand_class** (*VirtualMicrobes.event.Molecule.MoleculeClass*) –
- **ligand_ks** (*float or list of floats*) – binding affinity (K) values for individual Molecule species

ligand_class

randomize(*rand_gene_params*, *rand_gen*, ***kwargs*)

Randomizes gene properties.

Parameters

- **rand_gene_params** (*dict*) – parameters of randomization function
- **rand_gen** (*RNG*) –

randomize_good_params(*rand_gene_params*, *rand_generator*)

randomize_params(*rand_gene_params*, *rand_generator*)

simple_str()

toJSON(**args*, ***kwargs*)

Create JSON representation of gene.

Parameters **attr_mapper** (*dict*) – mapping from properties to colours

Returns

Return type JSON dict

class VirtualMicrobes.virtual_cell.Gene.**Transporter**(*reaction*, *ene_ks=10.0*, *substrates_ks=10.0*, *v_max=1.0*, *exporting=False*, ***kwargs*)

Bases: *VirtualMicrobes.virtual_cell.Gene.Gene*

Transporter gene class.

Transporters can transport metabolites across the cell membrane. The class defines the kinetic parameters of the transport reaction. The *exporting* parameter determines the direction of transport.

Parameters

- **reaction** (VirtualMicrobes.virtual_cell.event.Reaction.Reaction) – the transport reaction of the gene
- **ene_ks** (*float or list of floats*) – energy molecule binding constant
- **subtrate_ks** (*float or list of floats*) – substrate binding constants
- **v_max** (*float*) – max reaction flux constant
- **exporting** (*bool*) – if true, set to exporting

ode_params()
Returns a list of dictionaries of parameters necessary and sufficient to parameterize an ODE for all the sub-reactions associated with this Gene.

randomize_params(*rand_gene_params*, *rand_generator*, *rand_direction=False*)
Randomizes gene properties.

Parameters

- **rand_gene_params** (*dict*) – parameters of randomization function
- **rand_generator** (*RNG*) –
- **rand_direction** (*bool*) – if true, randomize the direction of transport

reaction

simple_str()

toJSON(**args*, ***kwargs*)
Create JSON representation of gene.

Parameters **attr_mapper** (*dict*) – mapping from properties to colours

Returns

Return type JSON dict

VirtualMicrobes.virtual_cell.Gene.**convert_rates**(*enzyme*, *enzyme_conc*, *metabolite_conc_dict*)
Estimate of conversion rates per substrate

Parameters

- **enzyme** – enzyme gene
- **enzyme_conc** – internal enzyme concentrations
- **metabolite_conc_dict** – concentrations of metabolites

VirtualMicrobes.virtual_cell.Gene.**init_molecule_ks**(*mol_class*, *mol_ks*, *external=False*)
Initialize the ordered mapping from molecules to Ks.

Each molecule in a moleculeclass has an associated K value, that is the binding affinity.

Parameters

- **mol_class** (*VirtualMicrobes.event.Molecule.MoleculeClass*) –
- **mol_ks** (*float or list*) – the K values for individual molecules

Returns

Return type mapping from *VirtualMicrobes.event.Molecule.Molecule* to K values (float)

VirtualMicrobes.virtual_cell.Gene.**init_substrates_ks**(*reaction*, *kss*)
Initialize the ordered mapping from substrate molecules to Ks for a Conversion reaction.

Each molecule in a moleculeclass has an associated K value, that is the binding affinity.

Parameters

- **reaction** (VirtualMicrobes.event.Reaction.Conversion) –
- **mol_ks** (*float or list*) – the K values for individual molecules

Returns

Return type mapping from *VirtualMicrobes.event.Molecule.Molecule* to K values (float)

VirtualMicrobes.virtual_cell.Gene.**pump_rates**(*pump*, *pump_conc*, *metabolite_conc_dict*)
Estimate of pumping rates for each substrate

Parameters

- **pump** – pump gene
- **pump_conc** – internal pump concentration
- **metabolite_conc_dict** – concentrations of metabolites

VirtualMicrobes.virtual_cell.Gene.**random_gene**(*environment*, *rand_gene_params*, *rand_gen*, *params*, *keep_transport_direction=True*)

VirtualMicrobes.virtual_cell.Gene.**randomized_param**(*rand_gene_params*, *rand_generator*)

VirtualMicrobes.virtual_cell.Gene.**read_gene**(*environment*, *gene*, *gene_index*, *verbose=False*)

VirtualMicrobes.virtual_cell.Genome module

class VirtualMicrobes.virtual_cell.Genome.**Genome**(*chromosomes*, *min_bind_score*)
Bases: object

add_chromosome(*chrom*, *verbose=False*)
Add a chromosome to the list of chromosomes.

binding_sequences

binding_tfs_scores(*op*)
Return tfs that bind this operator and their scores.

Parameters **op** (*VirtualMicrobes.virtual_cell.Sequence.Operator*) – operator sequence

Returns

Return type list of *VirtualMicrobes.virtual_cell.Gene.TranscriptionFactor*, float tuples

bs_to_tfs_dict()
Create mapping from binding sequences to tfs.

For each binding sequence in the genome map to the set of tfs that contain this binding sequence

Returns

- mapping from *VirtualMicrobes.virtual_cell.Sequence.BindingSequence* to set
- **of** (class:‘VirtualMicrobes.virtual_cell.Gene.TranscriptionFactor)

class_version = '1.0'

copy_number_dist

copy_numbers

copy_numbers_eff_pumps

copy_numbers_enzymes

copy_numbers_inf_pumps

copy_numbers_tfs

del_chromosome(*chrom*, *remove_genes=True*, *verbose=False*)
Delete a chromosome.

Remove a chromosome from the list of chromosomes. If *remove_genes* is True the genome will be further updated to reflect deletion of genes. E.g. the sequence bindings should be updated when genes are removed from the genome. It may be useful to defer updating if it is already known that the genes will be readded immediately. This may be the case when a chromosome is split (fission) or fused and no genes will be actually lost from the genome.

Parameters

- **chrom** (*VirtualMicrobes.virtual_cell.Chromosome.Chromosome*) – chromosome to be removed
- **remove_genes** (*bool*) – if True update the genome
- **verbose** (*bool*) – be verbose

die(*time*)
Record death of phylogenetic units in the genome.

Typically called from the cell when it dies. All phylogenetic units in the genome are instructed to record their death. When phylogenetic units are no longer alive, they may be pruned from their respective phylogenetic trees if there are no more living descendants of the phylogenetic unit.

Parameters **time** (*float*) – simulation time

eff_pumps

enzymes

inf_pumps

init_chromosomes(*chromosomes*)
Initialize chromosomes.

Add preinitialized chromosomes to the genome.

Parameters **chromosomes** (iterable of *VirtualMicrobes.virtual_cell.Chromosome.Chromosome*) – chromosomes to add

init_regulatory_network(*min_bind_score*)
Initialize the binding state of the regulatory network.

Iterate over all :class:‘VirtualMicrobes.virtual_cell.Sequence.Operator‘s in the genome and match them against all :class:‘VirtualMicrobes.virtual_cell.Sequence.BindingSequence‘s.

Parameters **min_bind_score** (*float*) – minimum binding score for sequence matching

op_to_tfs_scores_dict()
Create mapping from operators to the tfs that bind them, with their scores.

For each operator in the genome map the set of tfs, together with their binding scores.

Returns

- mapping from *VirtualMicrobes.virtual_cell.Sequence.Operator* to set
- **of** (class:‘VirtualMicrobes.virtual_cell.Gene.TranscriptionFactor, binding-score (float) tuples.)

operators

prune_genomic_ancestries()
Prune the phylogenetic trees of phylogenetic units in the genome.

Returns

- tuple (set of *VirtualMicrobes.virtual_cell.Chromosome.Chromosome*,
- set of *VirtualMicrobes.virtual_cell.GenomicElement.GenomicElement*)

pumps

reset_regulatory_network(*min_bind_score*)
Reset the binding state of the regulatory network.

Iterate over all Sequences in the genome and clear all bindings. Then re-initialize the regulatory network.

Parameters **min_bind_score** (*float*) – minimum binding score for sequence matching

size

tf_connections_dict()
A dictionry of TFs to sets of downstream bound genes.

tfs

toJSON(**args*, ***kwargs*)

update(*state*)

update_genome_removed_gene(*gene*)
Remove a gene from the genome if no more copies exist in the genome.

Updates the genome

Parameters **gene** (*VirtualMicrobes.virtual_cell.GenomicElement.GenomicElement*) – gene to be removed

update_genome_removed_genes(*genes*)
Update the genome to reflect gene deletions.

After the deletion of (part of) a chromosome, the genome has to be updated to reflect the change. Because exact copies of deleted genes may still be present in another part of the genome a check has to be performed before definitive removal.

Parameters **genes** (iterable of *VirtualMicrobes.virtual_cell.GenomicElement.GenomicElement*) – genes that were targeted by a deletion operation.

update_regulatory_network(*min_bind_score*)
Update the binding state of the regulatory network.

Iterate over all Sequences in the genome and if their check_binding flag is set, match the sequence against all potential binders in the genome.

Parameters **min_bind_score** (*float*) – minimum binding score for sequence matching

upgrade()

VirtualMicrobes.virtual_cell.GenomicElement module

class VirtualMicrobes.virtual_cell.GenomicElement.**GenomicElement**(*time_birth=0*, ***kwargs*)

Bases: *VirtualMicrobes.virtual_cell.PhyloUnit.PhyloBase*

Version

Author

types = ['tf', 'pump', 'enz']

uid = 0

VirtualMicrobes.virtual_cell.Identifier module

class VirtualMicrobes.virtual_cell.Identifier.**Identifier**(*obj*, *versioned_id=1*, *increment_func=None*)

Bases: object

clear_offspring()

classmethod count_class_types(*obj*)

from_parent(*parent*, *flat=True*, *pos=-1*)
Set an Identifier from a parent id.

If id is incremented in a 'flat' way, the new id is the unique count of objects of the (parent) type that this id belongs to. Else, the id increment is done in a 'versioned' manner. If parent has 0 offspring ids so far, the parent id is simply copied and no increment is done. If parent already has > 0 offspring ids, then a version element is added that indicates this id as the "n'th" offspring of the parent id. E.g. if parent id is 2.3 and it has 2 offspring already: from_parent(parent, flat=False) -> 2.3.2

Parameters

- **parent** (object with an *Identifier* attribute) – The parent of this Identifier.
- **flat** (*bool*) – If True, set versioned id position as the total number of counted objects of a the type of *parent*. Else, add versioned id information.
- **pos** (*int (index)*) – index of the version bit to update

increment_func

increment_offspring()

is_copy(*identifier*)
Test if identifier and self are different copies of the same major_id.

identifier [*Identifier*] Identifier to compare to.

Returns

Return type bool

major_id
Return the first part (highest order) of identifier.

Returns

Return type int (or other id format)

minor_id
Return version part of the identifier.

Returns

Return type list of int (or other id format)

offspring_count

parse(*versioned_id*)

unique_unit_dict = {}

versioned_id

VirtualMicrobes.virtual_cell.Identifier.**increment**(*x*)

VirtualMicrobes.virtual_cell.PhyloUnit module

class VirtualMicrobes.virtual_cell.PhyloUnit.**AddInheritanceType**
Bases: type

A metaclass that can set a class instances base type to support phylogenetic linking

The base type of the instantiated class can be either a PhyloBase or a PhyloUnit, depending on its _phylo_type class attribute. This enables an at run-time decision (via a program options) to fix the ancestry structure the class supports. PhyloBase instances keep references to neither parents nor children and hence do not need use a linker dict or unique_key generator. PhyloUnit does support ancestry. Phylogenetic linking is delegated to a global linker dict.

class VirtualMicrobes.virtual_cell.PhyloUnit.**PhyloBase**(*time_birth*)
Bases: object

Base class for all classes that can behave as phylogenetic units of inheritance.

Phylogenetic Base units record their time of birth and death and have an identifier field that can indicate a relation to parents and offspring. PhyloBase object may come into existence when a mother cell gives rise to a daughter cell and all units of inheritance that it contains (i.e. when a genome get's copied), but also when a phylogenetic unit (such as a gene or chromosome) mutates and the ancestral version will be kept intact for analysis purposes.

alive

die(*time*)
Death of a phylo unit happens either when a cell dies and all its genetic material with it, or when a mutation gives rise to a new variant of the unit.

Parameters **time** – Time of death in simulation time units.

id

living_offspring()

mark(*marker*, *mark*)
Set a marker on the phylo unit.

Parameters

- **marker** – marker type
- **mark** – value

marker_dict

prune_dead_branch()
Return self to be removed from the global phylo linker dict if not alive.

This is the degenerate base version of pruning. See the version in Phylo Unit for the case when units keep track of parent-child relationships.

time_birth

time_death

class VirtualMicrobes.virtual_cell.PhyloUnit.**PhyloUnit**(*time_birth*)
Bases: *VirtualMicrobes.virtual_cell.PhyloUnit.PhyloBase*

Extended Base class for all classes that can be represented in phylogenies. These classes should support ancestor and child retrieval and setting time of birth and death.

child_of(*phylo_unit*)
Return whether this PhyloUnit is the child of another PhyloUnit.

phylo_unit : *PhyloUnit*

children

common_ancestors(*phylo_unit*)

die(**args*, ***kwargs*)
Death of a phylo unit happens either when a cell dies and all its genetic material with it, or when a mutation gives rise to a new variant of the unit.

Parameters **time** – Time of death in simulation time units.

has_living_offspring(*exclude_set=set([])*)
Returns True if any of the phylo units descendants are alive

init_phylo_dicts()

living_offspring()
Returns a list of all offspring of this phylo unit that are currently alive.

lod_down_single()
Proceed down a single branch on the line of descent until there is a branch point or terminal node.

lod_up_single()
Proceed up a single branch on the line of descent until there is a branch point or terminal node.

lods_down()
Composes all the lines of descent leading down (forward in time) from phylo unit in a non- recursive way (compare lods_up).

lods_up()
Composes all the lines of descent leading up (back in time) from phylo unit (compare lods_down).

parent_of(*phylo_unit*)
Return whether this PhyloUnit is the parent of another PhyloUnit.

phylo_unit : *PhyloUnit*

parents

prune_dead_branch(*exclude_offspring_check_set=set([])*)
Return a set of phylogenetically related units that represent a dead phylogenetic branch.

Recursively checks for parent nodes whether the nodes descendants are all dead. In that case, the node can be pruned and its parents may additionally be checked for being part of the extended dead branch. The exclude is used to prevent superfluous checks of living offspring when it is already known that the current phylo_unit has no living_offspring.

Parameters **exclude_offspring_check_set** (set of *VirtualMicrobes.virtual_cell.PhyloUnit.PhyloUnit*) –

set_ancestor(*ge*)

set_unique_key()
Generate a unique key that can be used for mapping in a global linker dict.

VirtualMicrobes.virtual_cell.Population module

class VirtualMicrobes.virtual_cell.Population.**Population**(*params*, *environment*)
Bases: object

add_cell(*cell*)
Add an individual to the population.

A cell that is added is stored in a dictionary that maps the individual to a dictionary of properties, e.g. the numer of offspring.

Increase the population size.

Parameters **cell** (*VirtualMicrobes.virtual_cell.Cell.Cell*) – the individual to add

average_death_rate(*cells=None*)

average_production(*cells=None*)

average_promoter_strengths(*cells=None*)
Return array of individual average genomic promoter strength.

best_producer()
Return the individual and the value of highest production in the population.

calculate_death_rates(*base_death_rate=None*, *max_die_off_fract=None*, *toxicity_scaling=None*, *cells=None*)
Calculate and store death rates of individuals.

Uses a *base_death_rate* and *toxicity_scaling* parameter to calculate the death rate of individuals.

Parameters

- **base_death_rate** (*float, optional*) – base death rate for all individuals
- **max_die_off_fract** (*float, optional*) – maximum fraction of individuals that can die (stochastically)
- **toxicity_scaling** (*float, optional*) – scaling parameter for toxicity death rate effect
- **cells** (sequence of *VirtualMicrobes.virtual_cell.Cell.Cell*) – individuals in calculation

Returns

Return type returns mapping of cells to death rates

calculate_reference_production(*pressure=None*, *historic_production_weight=None*)
Calculates a reference production value for competition.

Reference production is used to scale the reproductive potential of cells during competition to reproduce.

Parameters

- **pressure** (*str*) – type of selection pressure scaling
- **historic_production_weight** (*float, optional*) – weighting of historic production values

cell_death(*cell*, *time*, *wiped=False*)
Kill an individual.

Updates the population size. Sets the time of death of individual.

Parameters

- **cell** (*VirtualMicrobes.virtual_cell.Cell.Cell*) – individual that is killed
- **time** (*int*) – simulation time
- **wiped** (*bool*) – indicate if cell died by *wipe_pop*

cell_markers(*marker*, *cells=None*)

cell_sizes(*cells=None*)
Return array of individual cell volumes.

chromosomal_mut_counts(*cells=None*)
Return array of individual counts of life time chromosomal mutations.

chromosome_counts(*cells=None*)
Return array of individual chromosome counts.

class_version = '1.0'
A population of individual Cells that reproduce and die during a Simulation.

The class defines methods for life history events of the cell population:

- death of individuals
- competition and reproduction
- population level gene exchange (HGT)

Phylogenetic relationships between individuals in the population are tracked and a phylogenetic tree is maintained and pruned when individuals reproduce and die.

Parameters

- **params** (*dict*) – a dictionary of simulation parameters
- **environment** (*VirtualMicrobes.environment.Environment*) – environment that is home to the population; determines molecular and reaction universe

params
dict – reference of Simulation parameters

historic_production_max
float – historic maximum of production value in the population

production_val_history
list of floats – history of all population production maxima

pop_rand_gen
RNG – RNG for drawing population events

evo_rand_gen
RNG – RNG for drawing evolutionary events

markers_range_dict
dict – stores ranges for cell markers used in plotting

value_range_dict
dict – ranges of attributes on phylogenetic tree

current_pop_size
int – population size

died
sequence of *VirtualMicrobes.virtual_cell.Cell.Cell* s – individuals that have died in the current simulation step

cell_dict
dict – mapping from *VirtualMicrobes.virtual_cell.Cell.Cell* to attributes

cells
view on keys – the living *VirtualMicrobes.virtual_cell.Cell.Cell* s in the population

new_offspring
sequence of *VirtualMicrobes.virtual_cell.Cell.Cell* s – individuals born in current time step

pruned_cells
set of *VirtualMicrobes.virtual_cell.Cell.Cell* s – individuals that are dead and do not have living offspring

current_ancestors
sequence of *VirtualMicrobes.virtual_cell.Cell.Cell* s – living individuals and ancestors that contribute offspring to current population

roots
sequence of *VirtualMicrobes.virtual_cell.Cell.Cell* s – first common ancestors of current population

phylo_tree
VirtualMicrobes.Tree.PhyloTree.PhyloTree – represent phylogenetic tree of current population

pruned_cells
set of *VirtualMicrobes.virtual_cell.PhyloUnit.PhyloUnit* – cell phylo units to be pruned from global phylo dict

pruned_chromosomes
set of *VirtualMicrobes.virtual_cell.PhyloUnit.PhyloUnit* – chromosome phylo units to be pruned from global phylo dict

pruned_genes
set of *VirtualMicrobes.virtual_cell.PhyloUnit.PhyloUnit* – gene phylo units to be pruned from global phylo dict

clear_mol_time_courses()

clear_pop_changes()
Reset population change containers.

cloned_pop(*pop_size*, *environment*, *params_dict*, *time=0*)
Creates a single ancestor *VirtualMicrobes.virtual_cell.Cell.Cell* individual and initialises its genomes. Genome initialisation depends on the *environment*, because this determines the set of possible gene types and minimum viable metabolism that can be constructed.

The population is then generated by cloning the ancestor individual. Clones are identical and store a reference to the ancestor.

Parameters

- **pop_size** (*int*) – population size
- **environment** (*VirtualMicrobes.environment.Environment*) – environment that is home to the population; determines molecular and reaction universe
- **params_dict** (*dict, optional*) – a dictionary of simulation parameters
- **time** (*int, optional*) – time of birth of the clone

Returns **common ancestor** – the common ancestor of the cloned population

Return type *VirtualMicrobes.virtual_cell.Cell.Cell*

cloned_pop_from_files(*pop_size*, *environment*, *cell_files=None*, *time=0*)
Create population of individuals initialised from cell parameter files.

Creates *VirtualMicrobes.virtual_cell.Cell.Cell* individuals and reads their genome and state from cell parameter files.

Parameters

- **pop_size** (*int*) – population size
- **environment** (*VirtualMicrobes.environment.Environment*) – environment that is home to the population; determines molecular and reaction universe
- **time** (*int, optional*) – time of birth of the clone

Returns **common_ancestors** – the common ancestors of the new population

Return type list of *VirtualMicrobes.virtual_cell.Cell.Cell* objects

consumer_type_counts(*cells=None*)
Return counts of cell sets with equal consumption metabolomes.

death_rates(*cells=None*)
Return array of individual death rate parameters.

die_off(*time*, *cells=None*)
Kill individual cells.

Individuals die deterministically if their cell volume is too low, or stochastically, with a probability : *death_rate* .

Parameters

- **time** (*int*) – simulation time
- **max_die_off_frac** (*float, optional*) – maximum fraction of individuals that can die
- **min_cell_volume** (*float, optional*) – minimum cell volume for survival

- **stochastic_death** (*bool, optional*) – if true, cells die stochastically, according to a *death_rate*
- **cells** (sequence of *VirtualMicrobes.virtual_cell.Cell.Cell*) – individuals that can die

differential_regulation (*cells=None*)
Return array of individual average differential regulation value.

enz_average_promoter_strengths (*cells=None*)
Return array of individual average enzyme promoter strength.

enzyme_average_vmaxs (*cells=None*)
Return array of individual average enzyme vmax values.

enzyme_counts (*cells=None*)
Return array of individual enzyme gene counts.

enzyme_substrate_ks (*cells=None*)
Return array of individual average enzyme substrate binding strength values.

export_type_counts (*cells=None*)
Return counts of cell sets with equal export genotypes

exporter_counts (*cells=None*)
Return array of individual exporting pump gene counts.

genome_sizes (*cells=None*)
Return array of individual genome sizes.

genotype_counts (*cells=None*)
Return counts of cell sets with equal genotypes

get_cell_death_rate_dict (*cells=None*)

get_cell_pos_production_dict (*cells=None*)
Return dict of cell production values.

get_cell_production_dict (*cells=None*, *life_time_prod=None*)
Return a mapping of cells and their last or life time production value.

Parameters

- **cells** (*sequence of Cell objects*) – individuals to map, default is the current population
- **life_time_prod** (*bool*) – take life time mean production instead of current

get_cell_production_rate_dict (*cells=None*)
Return dict of cell production rates.

get_cell_reproduction_dict (*cells=None*)
Counts of the number of reproduction events for each cell (living and dead direct children)

get_cell_size_dict (*cells=None*)
Return dict of cell sizes.

get_cell_toxicity_rate_dict (*cells=None*)
Return dict of cell toxicity change rates.

grow_time_course_arrays ()

horizontal_transfer (*time*, *grid*, *environment*, *rand_gen=None*, *rand_gen_np=None*)
Applies HGT to all cells in the grid

Parameters

- **grid** (*needed for internal HGT and setting the update-flags*) –
- **environment** (*contains all possible reactions to draw a random gene for external HGT*) –
- **rand_gen** (*RNG*) –

Returns

Return type

-

import_type_counts (*cells=None*)
Return counts of cell sets with equal import genotypes

importer_counts (*cells=None*)
Return array of individual importing pump gene counts.

init_cells_view ()
Initialise the view on the keys of the cell dict.

init_current_ancestors (*cells=None*)
Initialise the current ancestors.

init_evo_rand_gens (*evo_rand_seed=None*)
Initialise random generator for evolutionary events

init_phylo_tree (*supertree=False*)
Initialise the phylogenetic tree of the population.

Parameters **supertree** (*bool, optional*) – if true create supertree of all independent phylogenetic lineages

init_pop (*environment*, *pop_size=None*, *params_dict=None*)
Initialise the population.

Population initialisation is determined by the set of simulation parameters available in the params_dict or the params stored during construction of the *VirtualMicrobes.virtual_cell.Population.Population*. Several methods for creating a population are available:

- a new population of unique individuals is created with randomised

cell parameters. * individuals are initialised from a set of cell configuration files, describing their parameters * a population of identical clones is generated from a single randomly initialised individual.

Initialise the phylogenetic tree.

Parameters

- **environment** (*VirtualMicrobes.environment.Environment*) – environment that is home to the population; determines molecular and reaction universe
- **pop_size** (*int, optional*) – size of population
- **params_dict** (*dict, optional*) – a dictionary of simulation parameters

Returns the newly created individuals

Return type iterator of *VirtualMicrobes.virtual_cell.Cell.Cell* objects

init_pop_rand_gen (*pop_rand_seed=None*)
Initialise random generator for population events

init_range_dicts()
Initialise mappings used in colour coding individuals in graphical output.

The *markers_range_dict* stores lower and upper bounds on discrete population statistics.

The *metabolic_type_marker_dict* maps metabolic types to markers that are used for colour coding.

The *value_range_dict* stores lower and upper limits on various celll properties that are represented on the phylogenetic tree.

init_roots(*roots=None*)
Initialise the phylogenetic roots.

If no *roots* are given, initialise roots with the *current_ancestors*.

Parameters **roots** (sequence of *VirtualMicrobes.virtual_cell.Cell.Cell*, optional) – roots ancestors of the population

iterages(*cells=None*)
Return array of individual line of descent ages.

kill_cells_lineage(*lineages*, *time*, *fract*, *cells=None*)

lineages(*cells=None*)
Return array of individual linages

make_anc_clones(*generations_ago*, *density*)

make_cell_clones(*environment*, *list_of_cell_files*, *density*)

mark_cells_lineage(*cells=None*)

mark_cells_metabolic_type(*cells=None*)

mark_for_death(*max_die_off_frac=None*, *min_cell_volume=None*, *stochastic_death=None*, *cells=None*)
Kill individual cells.

Individuals die deterministically if their cell volume is too low, or stochastically, with a probability : *death_rate* .

Parameters

- **time** (*int*) – simulation time
- **max_die_off_frac** (*float, optional*) – maximum fraction of individuals that can die
- **min_cell_volume** (*float, optional*) – minimum cell volume for survival
- **stochastic_death** (*bool, optional*) – if true, cells die stochastically, according to a *death_rate*
- **cells** (sequence of *VirtualMicrobes.virtual_cell.Cell.Cell*) – individuals that can die

marker_counts(*marker*, *cells=None*)

classmethod metabolic_complementarity(*cells*, *strict_providing=False*, *strict_exploiting=False*)
Determine for the list of cells what the overlap is in metabolites provided and exploited.

To provide a metabolite a cell should simultaneous produce and export the metabolite. To exploit, it should be imported and consumed in a reaction.

metabolic_complementarity_pop(*strict=False*)
Return the set of metabolites that are provided and exploited simultaneously by the population.

See also:

func() *metabolic_complementarity*

metabolic_type_color(*cell*)

metabolic_type_counts(*cells=None*)
Return frequencies of cell sets with identical metabolic types.

A metabolic type is defined on the bases of the full set of metabolic reactions that an individual can perform using its metabolic gene set. A frequency spectrum of these types is than produced in the form of a collections.Counter object. From this object we can ask things like: most_common(N) N elements etc.

metabolic_types(*cells=None*)
Return array of individual metabolic types

most_abundant_marker(*marker_name*, *cells=None*)

most_offspring()
Return individual and value of highest offspring count.

mutate_new_offspring(*time*, *environment*, *rand_gen=None*, *rand_gen_np=None*)

offspring_counts(*cells=None*)
Return array of individual offspring counts.

oldest_cell()
Return oldest living individual.

pan_reactome_dict(*cells=None*)
Return the pan reactome of the population.

The pan-reactome is the combined set of reactions present in the population.

Returns

- **pan_reactome_dict** (*mapping of reaction type to sets of*)
- *VirtualMicrobes.event.Reaction.Reaction* s

Notes

For historic reasons the set of all transport reactions, either importing or exporting are keyd under 'import'. This type as stored in the *type_* attribute of the *VirtualMicrobes.event.Reaction.Reaction* and should not be confused with the *type_* attribute of the :class:VirtualMicrobes.virtual_cell.Gene.Gene' object.

point_mut_counts(*cells=None*)
Return array of individual counts of life time point mutations.

pop_marker_counts(*marker_name*, *cells=None*)

pos_production(*cells=None*)
Return array of individual positive production rates.

print_state()
Print population state.

producer_type_counts (*cells=None*)
Return counts of cell sets with equal production metabolomes.

production_rates (*cells=None*)
Return array of individual netto production rates.

production_values (*cells=None*)
Return array of individual production values.

prune_metabolic_types (*cells=None*)

pump_average_promoter_strengths (*cells=None*)
Return array of individual average pump promoter strength.

pump_average_vmaxs (*cells=None*)
Return array of individual average pump vmax values.

pump_energy_ks (*cells=None*)
Return array of individual average pump energy binding strength values.

pump_substrate_ks (*cells=None*)
Return array of individual average pump substrate binding strength values.

reaction_counts (*cells=None*)
Return counts of all reactions found in the population.

reaction_counts_split (*cells=None*)
Return counts of all reactions found in the population per reaction type.

reaction_genotype_counts (*cells=None*)
Return counts of cell sets with equal reaction genotypes

regulator_score (*cells=None*)

reproduce_at_minimum_production (*time*, *competitors=None*, *max_reproduce=None*, *reproduction_cost=None*)

reproduce_cell (*cell*, *time*, *spent_production=0.0*, *report=False*)
Reproduction of individual cell.

Parameters

- **cell** (*VirtualMicrobes.virtual_cell.Cell.Cell*) – reproducing individual
- **time** (*int*) – simulation time
- **spent_production** (*float, optional*) – production spent on reproducing
- **report** (*bool, optional*) – reporting

Returns **offspring** – new individual

Return type *VirtualMicrobes.virtual_cell.Cell.Cell*

reproduce_neutral (*time*, *competitors*, *max_reproduce=None*)
Individuals compete and reproduce proportional to NOTHING :) Note that cells would shrink if you keep doing this! Therefore we choose to reset the volumes continuously.

Parameters

- **time** (*int*) – simulation time
- **competitors** (list of *VirtualMicrobes.virtual_cell.Cell.Cell*) – competing individuals
- **max_reproduce** (*int, optional*) – maximum allowed reproduction events

Returns **new_offspring** – new offspring produced in this function

Return type list of *VirtualMicrobes.virtual_cell.Cell.Cell* s

reproduce_on_grid(*grid*, *max_pop_per_gp*, *time*, *neighborhood='competition'*, *non=None*, *selection_pressure=None*)

Reproduction of the population on the spatial grid.

Parameters

- **grid** (VirtualMicrobes.Environment.Grid.Grid) – spatial grid environment
- **max_pop_per_gp** (*int*) – maximum number of individuals per grid point
- **time** (*int*) – simulation time
- **neighborhood** (*str, optional*) – key to select neighborhood shape
- **non** (*float, optional*) – chance of no competitor winning competition
- **selection_pressure** (*str*) – type of selection pressure

Returns **new_offspring_gp_dict** – *VirtualMicrobes.environment.Grid.GridPoint* mapping of new offspring to the spatial grid point that they are born in.

Return type *VirtualMicrobes.virtual_cell.Cell.Cell* ->

reproduce_production_proportional(*time*, *competitors*, *max_reproduce=None*, *production_spending_fract=None*, *non=0.0*)

Individuals compete and reproduce proportional to their production value.

Parameters

- **time** (*int*) – simulation time
- **competitors** (list of *VirtualMicrobes.virtual_cell.Cell.Cell*) – competing individuals
- **max_reproduce** (*int, optional*) – maximum allowed reproduction events
- **production_spending_fract** (*float, optional*) – fraction of cell production value spent on reproduction
- **non** (*float, optional*) – chance of no competitor winning competition

Returns **new_offspring** – new offspring produced in this function

Return type list of *VirtualMicrobes.virtual_cell.Cell.Cell* s

reproduce_size_proportional(*time*, *competitors*, *max_reproduce=None*, *non=0.0*)

Individuals compete and reproduce proportional to their cell size.

Parameters

- **time** (*int*) – simulation time
- **competitors** (list of *VirtualMicrobes.virtual_cell.Cell.Cell*) – competing individuals
- **max_reproduce** (*int, optional*) – maximum allowed reproduction events
- **non** (*float, optional*) – chance of no competitor winning competition

Returns **new_offspring** – new offspring produced in this function

Return type list of *VirtualMicrobes.virtual_cell.Cell.Cell* s

reset_divided()
Reset flag for recent cell division.

reset_production_toxicity_volume(*cells=None*)

resize_time_courses(*new_max_time_points*)
resize the arrays that can hold time course information of cellular concentrations etc.

Parameters **new_max_time_points** – new length of time course array

scale_death_rates(*max_die_off_fract*, *cells=None*)
Scale death rates to give a maximum rate of dieing individuals.

If individual death rate are too high, these are scaled to have a maximum fraction of deaths, on average, in the population.

Parameters

- **max_die_of_fract** (*float*) – maximum allowed fraction of deaths
- **cells** (sequence of *VirtualMicrobes.virtual_cell.Cell.Cell*) – individuals in calculation

Returns

Return type returns mapping of cells to death rates

select_reproducing_cell(*cells_competition_value*, *rand_nr*, *non=0.0*, *competition_scaling_fact=None*)
Select a competing individual for reproduction.

Parameters

- **cells_competition_value** (list of (class:*VirtualMicrobes.virtual_cell.Cell.Cell*, float)) – competing cells with competition values
- **rand_nr** (*float*) – randomly drawn value between 0 and 1.
- **non** (*float, optional*) – value between 0 and 1 that represents no individual is chosen
- **competition_scaling_fact** (*float, optional*) – factor that can skew competition to be more or less severe

Returns

Return type (chosen individual, competition_value, index in competition list)

store_pop_characters()

tf_average_promoter_strengths(*cells=None*)
Return array of individual average transcription factor promoter strength.

tf_counts(*cells=None*)
Return array of individual transcription factor gene counts.

tf_k_bind_operators(*cells=None*)
Return array of individual average transcription factor operator binding strength values.

tf_ligand_ks(*cells=None*)
Return array of individual average transcription factor ligand binding strength values.

toxicity_rates(*cells=None*)
Return array of individual toxicity change rates.

trophic_type_counts(*env*, *cells=None*)
Return counts of trophic types in the population.

unique_pop (*pop_size*, *environment*, *params_dict*)
Create population of unique, randomised individuals.

Creates new class:*VirtualMicrobes.virtual_cell.Cell.Cell* individuals and initialises their genomes. Genome initialisation depends on the *environment*, because this determines the set of possible gene types and minimum viable metabolism that can be constructed.

Parameters

- **pop_size** (*int*) – population size
- **environment** (*VirtualMicrobes.environment.Environment*) – environment that is home to the population; determines molecular and reaction universe
- **params_dict** (*dict, optional*) – a dictionary of simulation parameters

Returns the newly created individuals

Return type view keys of *VirtualMicrobes.virtual_cell.Cell.Cell* objects

update_cell_params (*cells=None*)
Update the the cell parameters.

Sets cell parameters from the Simulation *params* dict.

Parameters **cells** (sequence of *VirtualMicrobes.virtual_cell.Cell.Cell*, optional) – the cells to update

update_ete_tree ()
Update the ete tree representation of the phylo_tree.

update_lineage_markers (*cells=None*, *min_nr_marks=None*)

update_offspring_regulatory_network (*min_bind_score=None*)

update_phylogeny (*new_roots=None*, *verbose=True*, *add_living=None*)
Update the phylogeny of the current population.

current_ancestors contains all living individuals and those that have contributed to the current population.

Individuals are only added to the *phylo_tree* representation after they have died, unless the add_living option is used. Nodes will only remain in the *phylo_tree* as long as the branch they're on contains a living individual in the current population. If a lineage dies out, its corresponding branch (and its constituent tree nodes) is pruned from the phylogenetic tree.

- Add new offspring to the current ancestors.
- Prune extinct branches in the phylogeny and remove ancestors without living offspring
- **Update the phylogenetic tree structure:**
  - add new root nodes
  - add new intermediate nodes
  - prune dead branches and nodes

Parameters

- **new_roots** (sequence of *VirtualMicrobes.virtual_cell.Cell.Cell*, optional) – new roots in the *phylo_tree*
- **verbose** (*bool, optional*) – print tree changes

Returns **phylo_tree** – phylogenetic tree representation of current population

Return type *VirtualMicrobes.Tree.PhyloTree.PhyloTree*

update_prod_val_hist(*hist_prod_func=<function median>*, *historic_production_window=None*, *pop_size_scaling=None*)
Keep a sliding window view on historic production values.

Parameters

- **hist_prod_func** (*func, optional*) – calculates the population production value
- **historic_production_window** (*int, optional*) – length of the sliding window
- **pop_size_scaling** (*bool, optional*) – scale production value by population size

update_stored_variables()
Syncs all local variables of class Cell.py (small_molecules) with the time course data

upgrade()
Upgrading from older pickled version of class to latest version. Version information is saved as class variable and should be updated when class invariants (e.g. fields) are added.

uptake_rates(*cells=None*)
Return array of individual uptake multipliers.

wipe_pop(*fract*, *time*, *min_surv=None*, *cells=None*)
Kill a fraction of the individuals.

Sets a flag on individuals that were killed by *wipe_pop*.

Parameters

- **fract** (*float*) – fraction to kill
- **time** (*int*) – simulation time
- **min_surv** (*int, optional*) – minimum number of surviving individuals
- **cells** (sequence of *VirtualMicrobes.virtual_cell.Cell.Cell*) – individuals in calculation

Returns

Return type list of individuals that were killed

VirtualMicrobes.virtual_cell.Sequence module

class VirtualMicrobes.virtual_cell.Sequence.**BindingSequence**(*sequence=None, length=None, elements=['0', '1'], flip_dict={'0': '1', '1': '0'}, **kwargs*)
Bases: *VirtualMicrobes.virtual_cell.Sequence.Sequence*

Binding sequence of a Transcription Factor

bound_operators

clear_bound_operators()

inform_operators()

init_bound_operators()

match_operators(*operators*, *minimum_score*)

remove_bound_operator(*op*)

toJSON(**args*, ***kwargs*)

class VirtualMicrobes.virtual_cell.Sequence.**Operator**(*sequence=None, length=None, elements=['0', '1'], flip_dict={'0': '1', '1': '0'}, **kwargs*)

Bases: *VirtualMicrobes.virtual_cell.Sequence.Sequence*

Version

Author

bind_to_bs(*bs*, *minimum_score*, *report=False*)

binding_sequences

calc_score_for_bs(*binding_site*, *minimum_score=1.0*, *report=False*)

calculate_regulation()

clear_binding_sequences()

inform_bss()

init_binding_sequences()
A dictionary from binding sequences that bind to this operator to binding scores

remove_binding_sequence(*bs*)

toJSON(**args*, ***kwargs*)

update_binding_sequences(*all_binding_sequences*, *minimum_score*)
Find the Binding Sequences that match this Operator and update dictionaries accordingly. Matching depends on a threshold "minimum_score".

Parameters

- **binding_sequences** – set of BS
- **minimum_score** – threshold for calling a match between this

Operator and a BS

class VirtualMicrobes.virtual_cell.Sequence.**Sequence**(*sequence*, *elements*, *length*, *flip_dict*, *time_birth=0*, ***kwargs*)

Bases: *VirtualMicrobes.virtual_cell.PhyloUnit.PhyloBase*

Version

Author

best_match(*s2*, *at_least=0*, *penalty=0.5*, *report=False*)
Finds the best match possible between self and s2 anywhere in both sequences, but only if the match score reaches at least a minimum threshold. If the returned score is below this threshold it is not guaranteed to be the best possible match. No gaps are allowed.

Parameters

- **s2** – sequence to compare to
- **at_least** – the minimum score that should still be attainable by

the scoring algorithm for it to proceed computing the scoring matrix :param penalty: mismatch penalty

bit_flip(*bit*, *flip_dict=None*)

check_binding

elements

flip_dict

insert_mutate(*pos*, *sequence_stretch*, *constant_length=True*)
Insert a sequence stretch into the current sequence.

Sets the check_binding flag to indicate that the sequence should be checked for changed binding status.

Parameters

- **pos** (*int*) – position in current sequence to insert
- **sequence_stretch** (*str*) – stretch to be inserted
- **constant_length** (*bool*) – whether to truncate after insertion to maintain the original sequence length

Returns

Return type newly mutated sequences

match(*sequence*, *func*)

mutate(*rand_gen=None*, *change_magnitude=None*)
Mutates the sequence.

Number of bits to mutate is either 1 or an amount of bits determined by the probability per bit to be changed or a given number of bits, depending on the value of "change_magnitude". Sets the check_binding flag to indicate that the sequence should be checked for changed binding status.

Parameters

- **rand_gen** (*RNG*) –
- **change_magnitude** (*float or int*) – when < 1, it is a probability, otherwise it is assumed to be the number of bits that should be changed (rounded up to nearest integer).

Returns

Return type newly mutated sequences

mutate_bits(*nr=1*, *rand_gen=None*)
Mutates a given number of random bits in the sequence to new values, chosen from the set of elements (possible values) that a bit can take, randomly.

Parameters

- **nr** – number of bits to mutate
- **rand_gen** – random generator

random_sequence(*n*, *rand_gen*)
Create random sequence of length n from *elements*.

Parameters

- **n** (*int*) – length of sequence
- **rand_gen** (*RNG*) –

Returns

Return type sequence string

randomize(*rand_gen=None*)

sequence

classmethod substring_scoring_matrix(*s1*, *s2*, *at_least=0*, *penalty=0.5*)

Computes a scoring matrix for matching between 2 sequences. Starts with a matrix filled in with all -1 . Comparisons between the strings continue as long as it is still possible to obtain the 'at_least' score when comparing the remainder of the strings. When the score is too low and the remaining substring length that can still be matched too short, the algorithm will stop, leaving the rest of the scores uncomputed. In that case, the _max is not guaranteed to be the maximum attainable matching score.

example matrix when matching sequences 0000000000 and 0000010100 with the default mismatch penalty of 0.5 (penalty substracted from score attained up to that point).

0 0 0 0 0 0 0 0 0 0 <- sequence 1

0 0 0 0 0 0 0 0 0 0 0

0 0 1 1 1 1 1 1 1 1 1 0 0 1 2 2 2 2 2 2 2 2 2 0 0 1 2 3 3 3 3 3 3 3 3 0 0 1 2 3 4 4 4 4 4 4 4 0 0 1 2 3 4 5 5 5 5 5 5 1 0 0 0.5 1.5 2.5 3.5 4.5 4.5 4.5 4.5 4.5 0 0 1 1 1.5 2.5 3.5 4.5 5.5 5.5 5.5 5.5 1 0 0 0.5 0.5 1.0 2.0 3.0 4.0 5.0 5.0 5.0 0 0 0 1 1 1.5 1.5 2.0 3.0 4.0 5.0 6.0 6.0 0 0 1 2 2 2.5 2.5 3.0 4.0 5.0 6.0 7.0

sequence 2

sequence 1: 0000000000 sequence 2: 0000010100 match: 0.7

uid = 0

VirtualMicrobes.virtual_cell.Sequence.**pretty_scoring_matrix**(*seq1*, *seq2*, *scoring_mat*, *width=4*)

Module contents

3.1.2 Module contents

Created on Aug 26, 2016 Modified in November, 2018* @author: thocu @author2: brem*

3.2 Glossary

building block A metabolite that is used by microbes to produce biomass. Biomass production may depend on the simultaneous conversion of a set of building blocks.

LOD The line of descent is the genealogical succession of individuals that connects some ancestor with one of its descendants. In the model these are constructed by storing all parent-offspring relations.

metabolite Any molecule that are not enzymes and that can undergo a reaction, including influx and degradation reactions.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search
- *Glossary*

Python Module Index

V

Index

A

B

C

D

E

F

G

H

I

M

N

Q

R

S

T

U

V

W

X

Y

Z