pyvipr Documentation

Release 0.3.5

Oscar O Ortega

Nov 21, 2019

CONTENTS:

1 Installation **3**

2 PyViPR Tutorial and PySB interface **5**
- 2.1 Start Jupyter Notebook . . . 5
- 2.2 How to interact with the widget . . . 5

3 PySB interface **7**
- 3.1 Import pyvipr pysb_viz module and a PySB model . . . 7
- 3.2 Species view . . . 7
- 3.3 Communities view . . . 8
- 3.4 Bipartite graph with species and bidirectional reactions nodes . . . 8
- 3.5 Bipartite graph with species and rules nodes . . . 8
- 3.6 Bipartite graph with species and rules nodes from incorrect model . . . 8
- 3.7 Bipartite graph with species and rules nodes. Rules are grouped by the functions that were used to create them. . . . 9
- 3.8 Bipartite graph with species and rules nodes. Rules are grouped by the modules they come from . . . 9
- 3.9 Species graph grouped by the compartment in which they are located . . . 9
- 3.10 Using a BioNetGen file (.bngl) to visualize the model . . . 9
- 3.11 Visualizing a large rule-based model using the atom-rules graph . . . 10
- 3.12 Dynamic visualization of a model . . . 10

4 Tellurium interface **11**
- 4.1 Static network visualizations: . . . 11
- 4.2 Visualization of the species network clustered with different algorithms: . . . 11
- 4.3 Dynamic visualization: . . . 11
  - 4.3.1 Species view . . . 12
  - 4.3.2 Species reactions view . . . 12
  - 4.3.3 Communities view . . . 12
  - 4.3.4 Dynamic visualization of a Tellurium model . . . 12

5 Other graph formats interface **13**
- 5.1 Networkx graph . . . 13
- 5.2 GRAPHML format . . . 14
- 5.3 SIF format . . . 14
- 5.4 SBGN XML format . . . 14
- 5.5 GEXF format . . . 14
- 5.6 GML format . . . 14
- 5.7 YAML format . . . 14
- 5.8 CYTOSCAPE JSON format . . . 15

6 PyViPR Core Modules Reference **17**

6.1 PySB static model visualizations (pyvipr.pysb_viz.static_viz) 17
6.2 PySB Dynamic model visualizations (pyvipr.pysb_viz.dynamic_viz) 22
6.3 PySB visualization views (pyvipr.pysb_viz.views) 23
6.4 Tellurium static model visualizations (pyvipr.tellurium_viz.static_viz) 29
6.5 Tellurium Dynamic model visualizations (pyvipr.tellurium_viz.dynamic_viz) 30
6.6 Tellurium visualization views (pyvipr.tellurium_viz.views) 30
6.7 NetworkX static and dynamic visualizations (pyvipr.networkx_viz.network_viz) 32
6.8 NetworkX visualization views (pyvipr.networkx_viz.views) 32

7 Indices and tables **35**

Python Module Index **37**

Index **39**

pyvipr is an ipython widget for interactively visualizing systems biology models. It uses PySB for generating network data and simulating trajectories and Cytoscape.js to render them. It supports BioNetGen and SBML models through the pysb importer module.

CHAPTER

ONE

INSTALLATION

To install the pyvipr widget using pip:

```
pip install pyvipr
jupyter nbextension enable --py --sys-prefix pyvipr
```

CHAPTER

TWO

PYVIPR TUTORIAL AND PYSB INTERFACE

Pyvipr is an ipython widget for interactively visualizing systems biology models. It has an interface to both PySB and Tellurium for generating network data and simulating trajectories, and uses Cytoscape.js to render static and dynamic networks. It supports visualization of BioNetGen and SBML models through the PySB importer module.

2.1 Start Jupyter Notebook

To start the jupyter notebook just run the following command in the terminal

```
jupyter notebook
```

2.2 How to interact with the widget

All visualizations have a search button that can be used to find nodes in large networks. This search function displays information about the species label and the type of node (species, reaction, rule, ...). Also, there is a fit button to center the nodes into the display area, a layout dropdown to select a layout for the network, and a button to save the visualization into a png file. Additionally, there is a Group button that can be use to embbed selected nodes into a hyper node.

Dynamic visualizations have a play a pause and refresh button to control the visualization. In addition, there is a slider that can be grabbed and dragged to go to a specific time point of the simulation.

Gestures supported by cytoscape.js to interact with the widget:

- Grab and drag background to pan : touch & desktop
- Pinch to zoom : touch & desktop (with supported trackpad)
- Mouse wheel to zoom : desktop
- Two finger trackpad up or down to zoom : desktop
- Tap to select : touch & desktop
- Tap background to unselect : desktop
- Taphold background to unselect : desktop & touch
- Multiple selection via modifier key (shift, command, control, alt) + tap : desktop
- Box selection : touch (three finger swipe) & desktop (modifier key + mousedown then drag)
- Grab and drag nodes : touch & desktop

Additional gestures added by the widget * Click on a nodes to display connecting nodes: touch & desktop * Click on compound nodes to show containing nodes: touch & desktop

CHAPTER

THREE

PYSB INTERFACE

Function	Description
`sp_view(model)`	Shows network of interacting species
`sp_comp_view(model)`	Shows network of species in their respective compartments
`sp_comm_louvain_view(model)`	Shows network of species grouped in communities
`sp_rxns_bidirectional_view(model)`	Shows bipartite network with species and bidirectional rections nodes
`sp_rxns_view(model)`	Shows bipartite network with species and unidirectional rections nodes
`sp_rules_view(model)`	Shows bipartite network with species and rules nodes
`sp_rules_fxns_view(model)`	Shows bipartite network with species and rules nodes. Rules nodes are grouped in the functions they come from
`sp_rules_mod_view(model)`	Shows bipartite network with species and rules nodes. Rules nodes are grouped in the file modules they come from
`projected_species_from_bireactions_view(model)`	Shows network of species projected from the bipartite(species, reactions) graph
`projected_bireactions_view(model)`	Shows network of reactions projected from the bipartite(species, reactions) graph
`projected_rules_view(model)`	Shows network of rules projected from the bipartite(species, rules) graph
`projected_species_rules_view(model)`	Shows network of species projected from the bipartite(species, rules) graph
`highlight_nodes_view(model, species, reactions)`	Shows network of species and highlights the species and reactions passed as arguments
`atom_rules_view(model, visualize_args, ...)`	Uses the BioNetGen atom-rules to visualize large rule-base models. Please visit PyViPR documentation for parameter details.
`sp_dyn_view(SimulationResult)`	Shows a species network. Edges size and color are updated according to reaction rate values. Nodes filling are updated according to concentration
`sp_comp_dyn_view(SimulationResult)`	Same as sp_dyn_view but species nodes are grouped by the compartments on which they are located
`sp_comm_dyn_view(SimulationResult)`	Same as sp_dyn_view but species nodes are grouped by communities

3.1 Import pyvipr pysb_viz module and a PySB model

[1]:
```
from pyvipr.examples_models.lopez_embedded import model
import pyvipr.pysb_viz as viz
```

3.2 Species view

In this type of visualization nodes represent molecular species of the model, and the edges represent the reactions that occur among different species.

[2]:
```
viz.sp_view(model)
```

```
Viz(data=<Model 'pyvipr.examples_models.lopez_embedded' (monomers: 23, rules: 62,␣
↪parameters: 126, expressions...
```

3.3 Communities view

In this type of visualization nodes represent molecular species of the model, and the edges represent the reaction that occur among different species. Densely connected nodes are grouped into communities that are represented by compound nodes.

[3]:
```
viz.sp_comm_louvain_view(model, layout_name='klay', random_state=1)
```

```
Viz(data=<Model 'pyvipr.examples_models.lopez_embedded' (monomers: 23, rules: 62,␣
↪parameters: 126, expressions...
```

3.4 Bipartite graph with species and bidirectional reactions nodes

There are two different sets of nodes in this visualization. Molecular species nodes and reaction nodes that indicate how species react. Reaction nodes have incoming edges that connect it with reactant species and outgoing edges that connect it with the product of the reaction.

[4]:
```
viz.sp_rxns_bidirectional_view(model)
```

```
Viz(data=<Model 'pyvipr.examples_models.lopez_embedded' (monomers: 23, rules: 62,␣
↪parameters: 126, expressions...
```

3.5 Bipartite graph with species and rules nodes

There are two different sets of nodes in this visualization. Molecular species nodes and rules nodes that indicate how species react. Rules nodes have incoming edges that connect it with reactant species and outgoing edges that connect it with the product of the rule.

[5]:
```
viz.sp_rules_view(model, layout_name='cose-bilkent')
```

```
Viz(data=<Model 'pyvipr.examples_models.lopez_embedded' (monomers: 23, rules: 62,␣
↪parameters: 126, expressions...
```

3.6 Bipartite graph with species and rules nodes from incorrect model

[6]:
```
from pyvipr.examples_models.earm_incorrect import model as model_incorrect
viz.sp_rules_view(model_incorrect, layout_name='cose-bilkent')
```

```
Viz(data=<Model 'pyvipr.examples_models.earm_incorrect' (monomers: 23, rules: 62,␣
↪parameters: 127, expressions...
```

3.7 Bipartite graph with species and rules nodes. Rules are grouped by the functions that were used to create them.

There are two different sets of nodes in this visualization. Molecular species nodes and rules nodes that indicate how species react. Rules nodes have incoming edges that connect it with reactant species and outgoing edges that connect it with the product of the rule. Additionally, rules are grouped by the python functions that were used to create them.

[7]:
```
viz.sp_rules_fxns_view(model, layout_name='fcose')
```

```
Viz(data=<Model 'pyvipr.examples_models.lopez_embedded' (monomers: 23, rules: 62,␣
↪parameters: 126, expressions...
```

3.8 Bipartite graph with species and rules nodes. Rules are grouped by the modules they come from

There are two different sets of nodes in this visualization. Molecular species nodes and rules nodes that indicate how species react. Rules nodes have incoming edges that connect it with reactant species and outgoing edges that connect it with the product of the rule. Additionally, rules are grouped by the python files where the rules were defined.

[8]:
```
viz.sp_rules_mod_view(model, layout_name='fcose')
```

```
Viz(data=<Model 'pyvipr.examples_models.lopez_embedded' (monomers: 23, rules: 62,␣
↪parameters: 126, expressions...
```

3.9 Species graph grouped by the compartment in which they are located

In this type of visualization nodes represent molecular species of the model, and the edges represent the reaction that occur among different species. Additionally, nodes are grouped by the cellular compartment they belong to.

Note: In order to use this type of visualization your model must have compartments defined.

[9]:
```
from pyvipr.examples_models.organelle_transport import model as model_compartments
viz.sp_comp_view(model_compartments)
```

```
Viz(data=<Model 'pyvipr.examples_models.organelle_transport' (monomers: 8, rules: 6,␣
↪parameters: 19, expressio...
```

3.10 Using a BioNetGen file (.bngl) to visualize the model

This widget accepts models defined in the BioNetGen format. All the static visualizations are available for this format. It requires the extension of the file to be .bngl

[10]:
```
import os
import pyvipr.examples_models as models
models_path = os.path.dirname(models.__file__)
organelle_model_path = os.path.join(models_path, 'organelle_transport.bngl')
```

[11]:
```
viz.sp_view(organelle_model_path)
```

```
Viz(data='/Users/ortega/miniconda3/envs/pyvipr/lib/python3.6/site-packages/pyvipr/
↪examples_models/organelle_tr...
```

3.11 Visualizing a large rule-based model using the atom-rules graph

[12]:
```
opts_path = os.path.join(models_path, 'ensemble_1_bng/ensemble_1_opts.txt')
ensemble_model_path = os.path.join(models_path, 'ensemble_1_bng/ensemble_1.bngl')
visualize_compressed = {'type': 'regulatory',
                        'opts': opts_path,
                        'groups': 1, 'collapse': 1, 'doNotUseContextWhenGrouping': 1,
↪'ruleNames': 1,
                        'removeReactantContext': 1, 'suffix': 'compressed'}
viz.atom_rules_view(ensemble_model_path, visualize_compressed)
```

```
Viz(data={'data': {'name': 'ensemble_1', 'style': 'atom'}, 'elements': {'nodes': [{
↪'data': {'label': '_R1', 'b...
```

3.12 Dynamic visualization of a model

In this type of visualization nodes represent molecular species of the model, and the edges represent the reaction that occur among different species. The node pie charts are a representation of the concentration relative to the maximum concentration across all time points. The thickness of the edges is a representation of the order of magnitude of the reaction rates.

We first simulate the model with pysb and pass the SimulationResult to the widget

[13]:
```
import numpy as np
from pysb.simulator import ScipyOdeSimulator
from pyvipr.examples_models.mm_two_paths_model import model as model_dynamic

tspan = np.linspace(0, 20000, 100)
sim_compartments = ScipyOdeSimulator(model, tspan, compiler='python').run()
```

[14]:
```
viz.sp_comm_dyn_view(sim_compartments, random_state=1)
```

```
Viz(data=<pysb.simulator.base.SimulationResult object at 0x1246fee10>, layout_name=
↪'klay', process='consumptio...
```

[ ]:
```

```

CHAPTER

FOUR

TELLURIUM INTERFACE

4.1 Static network visualizations:

- sp_view(model)
- sp_rxns_view(model)

4.2 Visualization of the species network clustered with different algorithms:

- sp_comm_louvain_view(model)
- sp_comm_greedy_view(model)
- sp_comm_asyn_lpa_view(model)
- sp_comm_label_propagation_view(model)
- sp_comm_girvan_newman_view(model)
- sp_comm_asyn_fluidc_view(model)

4.3 Dynamic visualization:

- sp_dyn_view(simulation)

In the future, we plan to add more visualizations of Tellurium models

[1]:
```
import tellurium as te
import pyvipr.tellurium_viz as tviz

r = te.loada("""
    J1:S1 -> S2; k1*S1;
    J2:S2 -> S3; k2*S2;
    J3:S4 -> S3; k2*S4;

    k1= 0.1; k2 = 0.2;
    S1 = 10; S2 = 0; S3 = 0; S4 = 20;
""")
```

4.3.1 Species view

[2]:
```
tviz.sp_view(r)
```

```
Viz(data=<roadrunner.RoadRunner() { this = 0x7fee43193900 }>, layout_name='cose-
↪bilkent', type_of_viz='sp_view...
```

4.3.2 Species reactions view

[3]:
```
tviz.sp_rxns_view(r)
```

```
Viz(data=<roadrunner.RoadRunner() { this = 0x7fee43193900 }>, layout_name='cose-
↪bilkent', type_of_viz='sp_rxns...
```

4.3.3 Communities view

[4]:
```
tviz.sp_comm_louvain_view(r)
```

```
Viz(data=<roadrunner.RoadRunner() { this = 0x7fee43193900 }>, layout_name='klay',␣
↪type_of_viz='sp_comm_louvain...
```

4.3.4 Dynamic visualization of a Tellurium model

To obtain the dynamic visualization of a Tellurium model users have to pass an specific selection to the simulate function. This selection has to contain the `time` variable, and all the species and reactions defined in the model.

[5]:
```
# Obtaining species and reactions defined in a model
selections = ['time'] + r.getFloatingSpeciesIds() + r.getReactionIds()

r.simulate(0, 40, selections=selections)
tviz.sp_dyn_view(r)
```

```
Viz(data=<roadrunner.RoadRunner() { this = 0x7fee43193900 }>, layout_name='cose-
↪bilkent', process='consumption...
```

[ ]:
```

```

CHAPTER

FIVE

OTHER GRAPH FORMATS INTERFACE

PyViPR uses NetworkX functions and cytoscape.js extensions to enable the visualization of the following graph formats:

- nx.Graph, nx.DiGraph, nx.MultiDiGraph
- GRAPHML
- SIF
- SBGN XML
- GEXF
- GML
- YAML
- CYTOSCAPE JSON

5.1 Networkx graph

[1]:
```
import networkx as nx
import pyvipr.network_viz as nviz
from pyvipr.util_networkx import network_dynamic_data
```

[2]:
```
G = nx.Graph()
G.add_edge(1, 2)
e = (2, 3)
G.add_edge(*e)  # unpack edge tuple*

node_rel = {1:[50, 100, 0],
            2:[50, 100, 0],
            3:[50, 100, 0]}

edge_colors = {(1,2):['#2b913a', '#2b913a', '#2b913a'],
               (2, 3):['#2b913a', '#2b913a', '#2b913a'],
               (1, 3):['#2b913a', '#2b913a', '#2b913a']}
```

[3]:
```
nviz.nx_graph_dyn_view(G, tspan=[1,2,3], node_rel=node_rel,
                       edge_colors=edge_colors, layout_name='fcose')
```

```
Viz(data=<networkx.classes.graph.Graph object at 0x116af25f8>, layout_name='fcose',␣
↪type_of_viz='dynamic_netwo...
```

5.2 GRAPHML format

[4]:
```
nviz.graphml_view('graphs_formats/graphml_example2.graphml', layout_name='fcose')
```

```
Viz(data='graphs_formats/graphml_example2.graphml', layout_name='fcose', type_of_viz=
↪'graphml')
```

5.3 SIF format

[5]:
```
nviz.sif_view('graphs_formats/bid_network.sif', layout_name='fcose')
```

```
Viz(data='graphs_formats/bid_network.sif', layout_name='fcose', type_of_viz='sif')
```

5.4 SBGN XML format

[6]:
```
nviz.sbgn_xml_view('graphs_formats/activated_stat1alpha_induction_of_the_irf1_gene.xml
↪', layout_name='fcose')
```

```
Viz(data='graphs_formats/activated_stat1alpha_induction_of_the_irf1_gene.xml', layout_
↪name='fcose', type_of_vi...
```

5.5 GEXF format

[7]:
```
nviz.gexf_view('graphs_formats/gexf_network.gexf')
```

```
Viz(data=<networkx.classes.digraph.DiGraph object at 0x116af2be0>, layout_name='fcose
↪', type_of_viz='network_s...
```

5.6 GML format

[8]:
```
nviz.gml_view('graphs_formats/karate.gml', label='id')
```

```
Viz(data=<networkx.classes.graph.Graph object at 0x116af2fd0>, layout_name='fcose',␣
↪type_of_viz='network_stati...
```

5.7 YAML format

[9]:
```
nviz.yaml_view('graphs_formats/yaml_network.yaml')
```

```
Viz(data=<networkx.classes.graph.Graph object at 0x116b99198>, layout_name='fcose',␣
↪type_of_viz='network_stati...
```

5.8 CYTOSCAPE JSON format

[10]:
```
nviz.json_view('graphs_formats/earm.json', layout_name='fcose')
```
```
Viz(data='graphs_formats/earm.json', layout_name='fcose', type_of_viz='json')
```

[ ]:
```

```

CHAPTER

SIX

PYVIPR CORE MODULES REFERENCE

6.1 PySB static model visualizations (pyvipr.pysb_viz.static_viz)

class pyvipr.pysb_viz.static_viz.**PysbStaticViz**(*model*, *generate_eqs=True*)

Class to generate static visualizations of systems biology models

Parameters

- **model** (*pysb.Model*) – PySB Model to visualize.
- **generate_eqs** (*bool*) – If True, generate math expressions for reaction rates and species in a model

atom_rules_view(*visualize_args*, *rule_name=None*, *verbose=False*, *cleanup=True*)

Uses the BioNetGen atom-rules to visualize large rule-base models. For more information regarding atom-rules and its parameters please visit: Sekar et al (2017), Automated visualization of rule-based models https://doi.org/10.1371/journal.pcbi.1005857

The visualize_args parameter contains all the arguments that will be passed to the BioNetGen visualize function. It is a dictionary and supports the following key, value pairs.

- *type*
  - *conventional* => Conventional rule visualization
  - *compact* => Compact rule visualization (using graph operation nodes)
  - *regulatory* => Rule-derived regulatory graph
  - *opts* => Options template for regulatory graph
  - *contactmap* => Contact map
  - *reaction_network* => Reaction network
- *suffix*
- str => add suffix string to output filename
- *each*
  - 1 => Show all rules in separate GML files
  - 0 => Show all rules the same GML file.
- *opts*
  - file path => import options from file

- *background*
  - 1 => Enable background
  - 0 => Disable background
- *groups*
  - 1 => Enable groups
  - 0 => Disable groups
- *collapse*
  - 1 => Enable collapsing of groups
  - 0 => Disable collapsing of groups
- *ruleNames*
  - 1 => Enable display of rule names
  - 0 => Disable display of rule names
- *doNotUseContextWhenGrouping*
  - 1 => Use permissive edge signature
  - 0 => Use strict edge signature
- *doNotCollapseEdges*:
  - 1 => When collapsing nodes, retain duplicate edges
  - 0 => When collapsing nodes, remove duplicate edges

visualize_args: dict Contains all the arguments that will be passed to the BioNetGen visualize function. The following key, value pairs are available

rule_name [str] Name of the rule to visualize, when *each* is set to 1 in visualize_args.

cleanup [bool, optional] If True (default), delete the temporary files after the simulation is finished. If False, leave them in place. Useful for debugging.

verbose [bool or int, optional (default: False)] Sets the verbosity level of the logger. See the logging levels and constants from Python's logging module for interpretation of integer values. False is equal to the PySB default level (currently WARNING), True is equal to DEBUG.

cluster_rxns_by_rules_view()
Cluster reaction nodes into the rules that generated them

Returns A Dictionary object that can be converted into Cytoscape.js JSON. This dictionary contains all the information (nodes,edges, parent nodes, positions) to generate a cytoscapejs network.

Return type dict

compartments_data_graph()
Create a networkx DiGraph. Check for compartments in a model and add the compartments as compound nodes where the species are located

Returns Graph with model species and compartments

Return type nx.Digraph

Raises **ValueError** – Model has not compartments

static graph_merge_pair_edges(*graph*, *reactions=None*)
Merges pair of edges that are reversed

Parameters

- **graph** (*nx.DiGraph or nx.MultiDiGraph*) – The networkx directed graph whose pairs of edges ((u, v), (v, u)) are going to be merged
- **reactions** (*pysb.ComponentSet*) – Model reactions

Returns Graph that has the information for the visualization of the model

Return type nx.Digraph

highlight_nodes_view(*species=None*, *reactions=None*)
Highlights the species and/or reactions passed as arguments

Parameters

- **species** (*list-like*) – It can be a vector with the indices of the species to be highlighted, or a vector with the concrete pysb.ComplexPattern species to be highlighted
- **reactions** (*list-like*) – A vector of tuples of length 2, where the first entry is the edge source and the second entry is the edge target, entries can be species indices or complex patterns. Or it can be a vector of integers that represent the indices of the reactions to highlight.

Returns A Dictionary object that can be converted into Cytoscape.js JSON. This dictionary contains all the information (nodes,edges, positions) to generate a cytoscapejs network.

Return type dict

static merge_nodes(*G*, *nodes*, *new_node*, ***attr*)
Merges the selected *nodes* of the graph G into one *new_node*, meaning that all the edges that pointed to or from one of these *nodes* will point to or from the *new_node*. attr_dict and ***attr* are defined as in *G.add_node*.

merge_reactions2rules(*graph*)
Merges the model reactions into each of the rules from which the reactions come form.

Returns Dictionary whose keys are tuples of rule name and rule index and the values are the reactions that are generated by each rule

Return type dict

projected_graph(*graph*, *project_to*, *reactions=None*)
Project a bipartite graph into one of the sets of nodes

Parameters

- **graph** (*nx.DiGraph*) – a networkx bipartite graph
- **project_to** (*str*) – One of the following options *species_from_bireactions*, *species_from_rules*, *bireactions*, *rules*
- **reactions** (*pysb.ComponentSet*) – Model reactions

Returns Projected graph

Return type nx.DiGraph

projected_species_from_bireactions_view()
This is a projection from the species & bidirectioanl reactions bipartite graph

sp_comm_louvain_hierarchy_view(*random_state=None*)
Use the Louvain algorithm https://en.wikipedia.org/wiki/Louvain_Modularity for community detection to find groups of nodes that are densely connected. It generates the data of all the intermediate clusters obtained during the Louvain algorithm generate to create a network with compound nodes that hold the communities.

Parameters **random_state** (*int, optional*) – Random state seed use by the community detection algorithm, by default None

Returns A Dictionary object that can be converted into Cytoscape.js JSON. This dictionary contains all the information (nodes,edges, parent nodes, positions) to generate a cytoscapejs network.

Return type dict

sp_comm_louvain_view(*random_state=None*)
Use the Louvain algorithm https://en.wikipedia.org/wiki/Louvain_Modularity for community detection to find groups of nodes that are densely connected. It generates the data to create a network with compound nodes that hold the communities.

Parameters **random_state** (*int, optional*) – Random state seed use by the community detection algorithm, by default None

Returns A Dictionary object that can be converted into Cytoscape.js JSON. This dictionary contains all the information (nodes,edges, parent nodes, positions) to generate a cytoscapejs network.

Return type dict

sp_comp_view()
Generate a dictionary that contains the information about the species network. Species are grouped by the compartments they belong to

Returns A Dictionary object that can be converted into Cytoscape.js JSON. This dictionary contains all the information (nodes, parent_nodes, edges, positions) to generate a cytoscapejs network.

Return type dict

sp_rules_fxns_view()
Generates a dictionary with the info of a bipartite graph where one set of nodes is the model species and the other set is the model rules. Additionally, it adds information of the functions from which the rules come from.

Returns A Dictionary object that can be converted into Cytoscape.js JSON. This dictionary contains all the information (nodes,edges, parent nodes, positions) to generate a cytoscapejs network.

Return type dict

sp_rules_graph()
Creates a bipartite nx.DiGraph graph where one set of nodes is the model species and the other set is the model rules.

Returns Graph that has the information for the visualization of the model

Return type nx.Digraph

sp_rules_mod_view()
Generates a dictionary with the info of a bipartite graph where one set of nodes is the model species and the other set is the model rules. Additionally, it adds information of the modules from which the rules come from.

Returns A Dictionary object that can be converted into Cytoscape.js JSON. This dictionary contains all the information (nodes,edges, parent nodes, positions) to generate a cytoscapejs network.

Return type dict

sp_rules_view()
Generates a dictionary with the info of a bipartite graph where one set of nodes is the model species and the other set is the model rules

Returns A Dictionary object that can be converted into Cytoscape.js JSON. This dictionary contains all the information (nodes,edges, parent nodes, positions) to generate a cytoscapejs network.

Return type dict

sp_rxns_bidirectional_graph(*two_edges=False*)
Creates a bipartite nx.DiGraph graph where one set of nodes is the model species and the other set is the model bidirectional reactions.

Parameters **two_edges** (*bool*) – If true, it draws two edges (in opposite directions) for each reversible reaction

Returns Graph that has the information for the visualization of the model

Return type nx.Digraph

sp_rxns_bidirectional_view()
Generate a dictionary with the info of a bipartite graph where one set of nodes is the model species and the other set is the model bidirectional reactions

Returns A Dictionary object that can be converted into Cytoscape.js JSON. This dictionary contains all the information (nodes,edges, parent nodes, positions) to generate a cytoscapejs network.

Return type dict

sp_rxns_graph()
Creates a bipartite nx.DiGraph graph where one set of nodes is the model species and the other set is the model unidirectional reactions.

Returns Graph that has the information for the visualization of the model

Return type nx.Digraph

sp_rxns_view()
Generates a dictionary with the info of a bipartite graph where one set of nodes is the model species and the other set is the unidirectional reactions

Returns A Dictionary object that can be converted into Cytoscape.js JSON. This dictionary contains all the information (nodes,edges, parent nodes, positions) to generate a cytoscapejs network.

Return type dict

sp_view()
Generate a dictionary that contains the species network information

Examples

```
>>> from pysb.examples.earm_1_0 import model
>>> viz = PysbStaticViz(model)
>>> data = viz.sp_view()
```

Returns A Dictionary object that can be converted into Cytoscape.js JSON. This dictionary contains all the information (nodes,edges, positions) to generate a cytoscapejs network.

Return type dict

species_graph()
Creates a nx.DiGraph graph of the model species interactions

Returns Graph that has the information for the visualization of the model

Return type nx.Digraph

pyvipr.pysb_viz.static_viz.**parse_name**(*spec*)
Function that writes short names of the species to name the nodes. It counts how many times a monomer_pattern is present in the complex pattern an its states then it takes only the monomer name and its state to write a shorter name to name the nodes.

Parameters **spec** (*pysb.ComplexPattern*) – Name of species to parse

Returns

Return type Parsed name of species

6.2 PySB Dynamic model visualizations (pyvipr.pysb_viz.dynamic_viz)

class pyvipr.pysb_viz.dynamic_viz.**PysbDynamicViz**(*simulation*, *sim_idx=0*, *cmap='RdBu_r'*)
Class to visualize the dynamics of systems biology models defined in PySB format.

Parameters

- **simulation** (*pysb SimulationResult*) – A SimulationResult instance of the model that is going to be visualized.
- **sim_idx** (*Index of simulation to be visualized*) –
- **cmap** (*str or Colormap instance*) – The colormap used to map the reaction rate values to RGBA colors. For more information visit: https://matplotlib.org/3.1.0/tutorials/colors/colormaps.html

dynamic_sp_comm_view(*type_viz='consumption'*, *random_state=None*)
Same as dynamic_view() but the species nodes are grouped by the communities they belong to. Communities are obtained using the Louvain algorithm.

Parameters

- **type_viz** (*str*) – Type of visualization. It can be *consumption* to see how species are being consumed or *production* to see how the species are being produced.
- **random_state** (*int*) – Seed used by the random generator in community detection

Returns A Dictionary Object with all nodes and edges information that can be converted into Cytoscape.js JSON to be visualized

Return type dict

dynamic_sp_comp_view(*type_viz='consumption'*)
Same as `dynamic_view()` but the species nodes are grouped by the compartments they belong to

dynamic_sp_view(*type_viz='consumption'*)
Generates a dictionary with the model dynamics data that can be converted in the Cytoscape.js JSON format

Parameters **type_viz** (*str*) – Type of the dynamic visualization, it can be 'consumption' or 'production'

Examples

```
>>> from pysb.examples.earm_1_0 import model
>>> from pysb.simulator import ScipyOdeSimulator
>>> import pyvipr.pysb_viz.dynamic_viz as viz
>>> import numpy as np
>>> tspan = np.linspace(0, 20000)
>>> sim = ScipyOdeSimulator(model, tspan).run()
>>> dyn_viz = viz.PysbDynamicViz(sim)
>>> data = dyn_viz.dynamic_sp_view()
```

Returns A Dictionary Object with all nodes and edges information that can be converted into Cytoscape.js JSON to be visualized

Return type dict

edges_colors_sizes()
This function obtains values for the size and color of the edges in the network. The color is a representation of the percentage of flux going through an edge. The edge size is a representation of the relative value of the reaction normalized to the maximum value that the edge can attain during the whole simulation.

Returns Three dictionaries. The first one contains the information of the edge sizes at all time points. The second one contains the information of the edge colors at all time points. The third one contains the values of the reaction rates at all time points.

Return type tuple

matrix_bidirectional_rates(*rxns_idxs=None*)
Obtains the values of the reaction rates at all the time points of the simulation

Returns Array with the reaction rates values

Return type np.ndarray

node_data()
Obtains the species concentration values and the relative concentration compared with the maximum concentration across all time points

Returns Two dictionaries. The first one has the species concentration. The second one has the relative species concentrations

Return type tuple

6.3 PySB visualization views (pyvipr.pysb_viz.views)

pyvipr.pysb_viz.views.**sp_view**(*model*, *layout_name='cose-bilkent'*)
Render a visualization of the interactions between the species in a model.

Parameters

- **model** (*pysb.model or str*) – Model to visualize. It can be a pysb model, or the file path to an an SBML or BNGL model
- **layout_name** (*str*) – Layout to use

pyvipr.pysb_viz.views.**sp_comp_view**(*model*, *layout_name='cose-bilkent'*)

Render a visualization of the interactions between the species in a model. The species nodes are grouped by the compartments they belong to.

Parameters

- **model** (*pysb.model or str*) – Model to visualize. It can be a pysb model, or the file path to an an SBML or BNGL model
- **layout_name** (*str*) – Layout to use

pyvipr.pysb_viz.views.**sp_comm_louvain_view**(*model*, *layout_name='klay'*, *random_state=None*)

Render a visualization of the interactions between the species in a model. The species nodes are grouped by the communities detected by the Louvain algorithm: https://en.wikipedia.org/wiki/Louvain_Modularity.

Parameters

- **model** (*pysb.model or str*) – Model to visualize. It can be a pysb model, or the file path to an an SBML or BNGL model
- **layout_name** (*str*) – Layout to use
- **random_state** (*int*) – Random state seed use by the community detection algorithm

pyvipr.pysb_viz.views.**sp_comm_louvain_hierarchy_view**(*model*, *layout_name='klay'*, *random_state=None*)

Render a visualization of the interactions between the species in a model. The species nodes are grouped by the communities detected by the Louvain algorithm: https://en.wikipedia.org/wiki/Louvain_Modularity.

Parameters

- **model** (*pysb.model or str*) – Model to visualize. It can be a pysb model, or the file path to an an SBML or BNGL model
- **layout_name** (*str*) – Layout to use
- **random_state** (*int*) – Random state seed use by the community detection algorithm

pyvipr.pysb_viz.views.**sp_comm_greedy_view**(*model*, *layout_name='klay'*)

Render a visualization of the interactions between the species in a model. The species nodes are grouped by the communities detected by the Clauset-Newman-Moore greedy modularity maximization algorithm implemented in Networkx

Parameters

- **model** (*pysb.model or str*) – Model to visualize. It can be a pysb model, or the file path to an an SBML or BNGL model
- **layout_name** (*str*) – Layout to use

pyvipr.pysb_viz.views.**sp_comm_asyn_lpa_view**(*model*, *random_state=None*, *layout_name='klay'*)

Render a visualization of the interactions between the species in a model. The species nodes are grouped by the communities detected by the asynchronous label propagation algorithm implemented in Networkx.

Parameters

- **model** (*pysb.model or str*) – Model to visualize. It can be a pysb model, or the file path to an an SBML or BNGL model
- **layout_name** (*str*) – Layout to use
- **random_state** (*int*) – Random state seed use by the community detection algorithm

pyvipr.pysb_viz.views.**sp_comm_label_propagation_view**(*model*, *layout_name='klay'*)

Render a visualization of the interactions between the species in a model. The species nodes are grouped by the communities detected by the label propagation algorithm implemented in Networkx.

Parameters

- **model** (*pysb.model or str*) – Model to visualize. It can be a pysb model, or the file path to an an SBML or BNGL model
- **layout_name** (*str*) – Layout to use

pyvipr.pysb_viz.views.**sp_comm_girvan_newman_view**(*model*, *layout_name='klay'*)

Render a visualization of the interactions between the species in a model. The species nodes are grouped by the communities detected by the Girvan-Newman method implemented in Networkx.

Parameters

- **model** (*pysb.model or str*) – Model to visualize. It can be a pysb model, or the file path to an an SBML or BNGL model
- **layout_name** (*str*) – Layout to use

pyvipr.pysb_viz.views.**sp_comm_asyn_fluidc_view**(*model*, *k*, *max_iter=100*, *random_state=None*, *layout_name='fcose'*)

Render a visualization of the interactions between the species in a model. The species nodes are grouped by the communities detected by the asynchronous label propagation algorithm implemented in Networkx.

Parameters

- **model** (*pysb.model or str*) – Model to visualize. It can be a pysb model, or the file path to an an SBML or BNGL model
- **k** (*int*) – The number of communities to be found
- **max_iter** (*int*) – The number of maximum iterations allowed
- **random_state** (*int*) – Random state seed use by the community detection algorithm
- **layout_name** (*str*) – Layout to use

pyvipr.pysb_viz.views.**sp_rxns_bidirectional_view**(*model*, *layout_name='cose-bilkent'*)

Render a visualization of a bipartite graph where one set of nodes are the molecular species in the model and the other set are the bidirectional reactions.

Parameters

- **model** (*pysb.model or str*) – Model to visualize. It can be a pysb model, or the file path to an an SBML or BNGL model
- **layout_name** (*str*) – Layout to use

pyvipr.pysb_viz.views.**sp_rxns_view**(*model*, *layout_name='cose-bilkent'*)

Render a visualization of a bipartite graph where one set of nodes are the molecular species in the model and the other set are the unidirectional reactions.

Parameters

- **model** (*pysb.model or str*) – Model to visualize. It can be a pysb model, or the file path to an an SBML or BNGL model
- **layout_name** (*str*) – Layout to use

pyvipr.pysb_viz.views.**sp_rules_view**(*model*, *layout_name='cose-bilkent'*)

Render a visualization of a bipartite graph where one set of nodes are the molecular species in the model and the other set are the rules.

Parameters

- **model** (*pysb.model or str*) – Model to visualize. It can be a pysb model, or the file path to an an SBML or BNGL model
- **layout_name** (*str*) – Layout to use

pyvipr.pysb_viz.views.**sp_rules_fxns_view**(*model*, *layout_name='cose-bilkent'*)

Render a visualization of a bipartite graph where one set of nodes are the molecular species in the model and the other set are the rules. Additionally, rules are grouped by the macros that created them.

Parameters

- **model** (*pysb.model or str*) – Model to visualize. It can be a pysb model, or the file path to an an SBML or BNGL model
- **layout_name** (*str*) – Layout to use

pyvipr.pysb_viz.views.**sp_rules_mod_view**(*model*, *layout_name='cose-bilkent'*)

Render a visualization of a bipartite graph where one set of nodes are the molecular species in the model and the other set are the rules. Additionally, rules are grouped by the modules where they are defined.

Parameters

- **model** (*pysb.model or str*) – Model to visualize. It can be a pysb model, or the file path to an an SBML or BNGL model
- **layout_name** (*str*) – Layout to use

pyvipr.pysb_viz.views.**projected_species_from_bireactions_view**(*model*, *layout_name='cose-bilkent'*)

Render a visualization of the interactions between species in a model.

Parameters

- **model** (*pysb.model or str*) – Model to visualize. It can be a pysb model, or the file path to an an SBML or BNGL model
- **layout_name** (*str*) – Layout to use

pyvipr.pysb_viz.views.**projected_bireactions_view**(*model*, *layout_name='cose-bilkent'*)

Render a visualization of the interaction between the reaction in a model

Parameters

- **model** (*pysb.model or str*) – Model to visualize. It can be a pysb model, or the file path to an an SBML or BNGL model
- **layout_name** (*str*) – Layout to use

pyvipr.pysb_viz.views.**projected_rules_view**(*model*, *layout_name='cose-bilkent'*)

Render a visualization of the interactions between rules in a model.

Parameters

- **model** (*pysb.model or str*) – Model to visualize. It can be a pysb model, or the file path to an an SBML or BNGL model
- **layout_name** (*str*) – Layout to use

pyvipr.pysb_viz.views.**projected_species_from_rules_view**(*model*, *layout_name='cose-bilkent'*)

Render a visualization of a bipartite graph where one set of nodes are the molecular species in the model and the other set are the rules.

Parameters

- **model** (*pysb.model or str*) – Model to visualize. It can be a pysb model, or the file path to an an SBML or BNGL model
- **layout_name** (*str*) – Layout to use

pyvipr.pysb_viz.views.**cluster_rxns_by_rules_view**(*model*, *layout_name='fcose'*)

Render a visualization of the interactions between the reactions in a model. Reaction nodes are grouped by the rules that generated them.

Parameters

- **model** (*pysb.model or str*) – Model to visualize. It can be a pysb model, or the file path to an an SBML or BNGL model
- **layout_name** (*str*) – Layout to use

pyvipr.pysb_viz.views.**sp_dyn_view**(*simulation*, *sim_idx=0*, *process='consumption'*, *layout_name='cose-bilkent'*, *cmap='RdBu_r'*)

Render a dynamic visualization of the simulation

Parameters

- **simulation** (*pysb.SimulationResult*) – Simulation result to visualize
- **sim_idx** (*int*) – Index of simulation to be visualized
- **process** (*str*) – Type of the dynamic visualization, it can be 'consumption' or 'production'
- **layout_name** (*str*) – Layout to use
- **cmap** (*str or Colormap instance*) – The colormap used to map the reaction rate values to RGBA colors. For more information visit: https://matplotlib.org/3.1.0/tutorials/colors/colormaps.html

pyvipr.pysb_viz.views.**sp_comp_dyn_view**(*simulation*, *sim_idx=0*, *process='consumption'*, *layout_name='cose-bilkent'*, *cmap='RdBu_r'*)

Render a dynamic visualization of the simulation. The species nodes are grouped by the compartments they belong to.

Parameters

- **simulation** (*pysb.SimulationResult object*) – Simulation result to visualize dynamically
- **sim_idx** (*int*) – Index of simulation to be visualized
- **process** (*str*) – Type of the dynamic visualization, it can be 'consumption' or 'production'
- **layout_name** (*str*) – Layout to use

- **cmap** (*str or Colormap instance*) – The colormap used to map the reaction rate values to RGBA colors. For more information visit: https://matplotlib.org/3.1.0/tutorials/colors/colormaps.html

pyvipr.pysb_viz.views.**sp_comm_dyn_view**(*simulation*, *sim_idx=0*, *process='consumption'*, *layout_name='klay'*, *cmap='RdBu_r'*, *random_state=None*)

Render a dynamic visualization of the simulation. The species nodes are grouped by the communities detected by the Louvain algorithm

Parameters

- **simulation** (*pysb.SimulationResult object*) – Simulation result to visualize dynamically
- **sim_idx** (*int*) – Index of simulation to be visualized
- **process** (*str*) – Type of the dynamic visualization, it can be 'consumption' or 'production'
- **layout_name** (*str*) – Layout to use
- **cmap** (*str or Colormap instance*) – The colormap used to map the reaction rate values to RGBA colors. For more information visit: https://matplotlib.org/3.1.0/tutorials/colors/colormaps.html
- **random_state** (*int*) – Random state seed use by the community detection algorithm

pyvipr.pysb_viz.views.**sim_model_dyn_view**(*model*, *tspan*, *param_values=None*, *type_of_viz='dynamic_view'*, *process='consumption'*, *cmap='RdBu_r'*, *layout_name='cose-bilkent'*)

Render a dynamic visualization of the model using the tspan and param_values passed to the function

Parameters

- **model** (*pysb.model or str*) – Model to visualize. It can be a pysb model, or the file path to an an SBML or BNGL model
- **tspan** (*vector-like, optional*) – Time values over which to simulate. The first and last values define the time range.
- **param_values** (*vector-like or dict, optional*) – Values to use for every parameter in the model. Ordering is determined by the order of model.parameters. If passed as a dictionary, keys must be parameter names. If not specified, parameter values will be taken directly from model.parameters.
- **type_of_viz** (*str*) – Type of visualization. It can only be *sp_dyn_view*, *sp_comp_dyn_view* or *sp_comm_dyn_view*
- **process** (*str*) – Type of the dynamic visualization, it can be 'consumption' or 'production'
- **cmap** (*str or Colormap instance*) – The colormap used to map the reaction rate values to RGBA colors. For more information visit: https://matplotlib.org/3.1.0/tutorials/colors/colormaps.html
- **layout_name** (*str*) – Layout name to use

6.4 Tellurium static model visualizations (pyvipr.tellurium_viz.static_viz)

class pyvipr.tellurium_viz.static_viz.**TelluriumStaticViz**(*model*)
Class to generate static visualization of sbml models from tellurium

sp_comm_louvain_hierarchy_view(*random_state=None*)
Use the Louvain algorithm https://en.wikipedia.org/wiki/Louvain_Modularity for community detection to find groups of nodes that are densely connected. It generates the data of all the intermediate clusters obtained during the Louvain algorithm generate to create a network with compound nodes that hold the communities.

Parameters **random_state** (*int, optional*) – Random state seed use by the community detection algorithm, by default None

Returns A Dictionary object that can be converted into Cytoscape.js JSON. This dictionary contains all the information (nodes,edges, parent nodes, positions) to generate a cytoscapejs network.

Return type dict

sp_comm_louvain_view(*random_state=None*)
Use the Louvain algorithm https://en.wikipedia.org/wiki/Louvain_Modularity for community detection to find groups of nodes that are densely connected. It generates the data to create a network with compound nodes that hold the communities.

Parameters **random_state** (*int, optional*) – Random state seed use by the community detection algorithm, by default None

Returns A Dictionary object that can be converted into Cytoscape.js JSON. This dictionary contains all the information (nodes,edges, parent nodes, positions) to generate a cytoscapejs network.

Return type dict

sp_rxns_graph()
Creates a bipartite nx.DiGraph graph where one set of nodes is the model species and the other set is the model bidirectional reactions.

Returns Graph that has the information for the visualization of the model

Return type nx.Digraph

sp_rxns_view()
Generate a dictionary that contains the species and reactions network information

Returns A Dictionary object that can be converted into Cytoscape.js JSON. This dictionary contains all the information (nodes,edges, positions) to generate a cytoscapejs network.

Return type dict

sp_view()
Generate a dictionary that contains the species network information

Returns A Dictionary object that can be converted into Cytoscape.js JSON. This dictionary contains all the information (nodes,edges, positions) to generate a cytoscapejs network.

Return type dict

species_graph()
Creates a graph of the model species interactions :returns: Graph that has the information for the visualization of the model :rtype: nx.Digraph

6.5 Tellurium Dynamic model visualizations (pyvipr.tellurium_viz.dynamic_viz)

class pyvipr.tellurium_viz.dynamic_viz.**TelluriumDynamicViz**(*sim_model*, *cmap='RdBu_r'*)

class to visualize the dynamics of systems biology models defined in sbml or antimony format

Parameters

- **sim_model** (*tellurium roadrunner*) – A roadrunner instance after a simulation
- **cmap** (*str or Colormap instance*) – The colormap used to map the reaction rate values to RGBA colors. For more information visit: https://matplotlib.org/3.1.0/tutorials/colors/colormaps.html

dynamic_sp_view(*type_viz='consumption'*)

Generates a dictionary with the model dynamics data that can be converted in the Cytoscape.js JSON format

Parameters **type_viz** (*str*) – Type of the dynamic visualization, it can be 'consumption' or 'production'

Returns A Dictionary Object with all nodes and edges information that can be converted into Cytoscape.js JSON to be visualized

Return type dict

edges_colors_sizes()

This function obtains values for the size and color of the edges in the network. The color is a representation of the percentage of flux going through an edge. The edge size is a representation of the relative value of the reaction normalized to the maximum value that the edge can attain during the whole simulation.

Returns Three dictionaries. The first one contains the information of the edge sizes at all time points. The second one contains the information of the edge colors at all time points. The third one contains the values of the reaction rates at all time points.

Return type tuple

node_data()

Obtains the species concentration values and the relative concentration compared with the maximum concentration across all time points

Returns Two dictionaries. The first one has the species concentration. The second one has the relative species concentrations

Return type tuple

6.6 Tellurium visualization views (pyvipr.tellurium_viz.views)

pyvipr.tellurium_viz.views.**sp_view**(*model*, *layout_name='cose-bilkent'*)

Render a visualization of the interactions between the species in a model.

Parameters

- **model** (*tellurium model*) – Model to visualize. an SBML or BNGL model
- **layout_name** (*str*) – Layout to use

pyvipr.tellurium_viz.views.**sp_rxns_view**(*model*, *layout_name='cose-bilkent'*)

Render a visualization of the interactions between the species and reactions in a model.

Parameters

- **model** (*tellurium model*) – Model to visualize. an SBML or BNGL model
- **layout_name** (*str*) – Layout to use

pyvipr.tellurium_viz.views.**sp_comm_louvain_view**(*model*, *layout_name='klay'*, *random_state=None*)

Render a visualization of the interactions between the species in a model. The species nodes are grouped by the communities detected by the Louvain algorithm: https://en.wikipedia.org/wiki/Louvain_Modularity.

Parameters

- **model** (*tellurium model*) – Model to visualize.
- **layout_name** (*str*) – Layout to use
- **random_state** (*int*) – Random state seed use by the community detection algorithm

pyvipr.tellurium_viz.views.**sp_dyn_view**(*simulation*, *process='consumption'*, *layout_name='cose-bilkent'*, *cmap='RdBu_r'*)

Render a dynamic visualization of the simulation

Parameters

- **simulation** (*tellurium simulation*) – Simulation to visualize
- **process** (*str*) – Type of the dynamic visualization, it can be 'consumption' or 'production'
- **layout_name** (*str*) – Layout to use
- **cmap** (*str or Colormap instance*) – The colormap used to map the reaction rate values to RGBA colors. For more information visit: https://matplotlib.org/3.1.0/tutorials/colors/colormaps.html

pyvipr.tellurium_viz.views.**sp_comm_greedy_view**(*model*, *layout_name='klay'*)

Render a visualization of the interactions between the species in a model. The species nodes are grouped by the communities detected by the Clauset-Newman-Moore greedy modularity maximization algorithm implemented in Networkx

Parameters

- **model** (*pysb.model or str*) – Model to visualize. It can be a pysb model, or the file path to an an SBML or BNGL model
- **layout_name** (*str*) – Layout to use

pyvipr.tellurium_viz.views.**sp_comm_asyn_lpa_view**(*model*, *random_state=None*, *layout_name='klay'*)

Render a visualization of the interactions between the species in a model. The species nodes are grouped by the communities detected by the asynchronous label propagation algorithm implemented in Networkx.

Parameters

- **model** (*pysb.model or str*) – Model to visualize. It can be a pysb model, or the file path to an an SBML or BNGL model
- **layout_name** (*str*) – Layout to use
- **random_state** (*int*) – Random state seed use by the community detection algorithm

pyvipr.tellurium_viz.views.**sp_comm_label_propagation_view**(*model*, *layout_name='klay'*)

Render a visualization of the interactions between the species in a model. The species nodes are grouped by the communities detected by the label propagation algorithm implemented in Networkx.

Parameters

- **model** (*pysb.model or str*) – Model to visualize. It can be a pysb model, or the file path to an an SBML or BNGL model
- **layout_name** (*str*) – Layout to use

pyvipr.tellurium_viz.views.**sp_comm_girvan_newman_view**(*model*, *layout_name='klay'*)

Render a visualization of the interactions between the species in a model. The species nodes are grouped by the communities detected by the Girvan-Newman method implemented in Networkx.

Parameters

- **model** (*pysb.model or str*) – Model to visualize. It can be a pysb model, or the file path to an an SBML or BNGL model
- **layout_name** (*str*) – Layout to use

pyvipr.tellurium_viz.views.**sp_comm_asyn_fluidc_view**(*model*, *k*, *max_iter=100*, *seed=None*, *layout_name='fcose'*)

Render a visualization of the interactions between the species in a model. The species nodes are grouped by the communities detected by the asynchronous label propagation algorithm implemented in Networkx.

Parameters

- **model** (*pysb.model or str*) – Model to visualize. It can be a pysb model, or the file path to an an SBML or BNGL model
- **k** (*int*) – The number of communities to be found
- **max_iter** (*int*) – The number of maximum iterations allowed
- **random_state** (*int*) – Random state seed use by the community detection algorithm
- **layout_name** (*str*) – Layout to use

6.7 NetworkX static and dynamic visualizations (pyvipr.networkx_viz.network_viz)

6.8 NetworkX visualization views (pyvipr.networkx_viz.views)

pyvipr.network_viz.views.**nx_graph_view**(*graph*, *layout_name='cose'*)

Render a networkx Graph or DiGraph :param graph: Graph to render :type graph: nx.Graph or nx.DiGraph :param layout_name: Layout to use :type layout_name: str

pyvipr.network_viz.views.**nx_graph_dyn_view**(*graph*, *tspan*, *node_rel=None*, *node_tip=None*, *edge_colors=None*, *edge_sizes=None*, *edge_tips=None*, *layout_name='cose'*)

Render a dynamic visualization of a networkx graph

Parameters

- **graph** (*nx.DiGraph or nx.Graph*) – Graph to visualize
- **tspan** (*vector-like, optional*) – Time values over which to simulate. The first and last values define

- **node_rel** (*dict*) – A dictionary where the keys are the node ids and the values are lists that contain (0-100) values that are represented in a pie chart within the node
- **node_tip** (*dict*) – A dictionary where the keys are the node ids and the values are lists that contain any value that can be accessed as a tooltip in the rendered network
- **edge_colors** (*dict*) – A dictionary where the keys are the edge ids and the values are lists that contain any hexadecimal color value that are represented in the edge colors
- **edge_sizes** (*dict*) – A dictionary where the keys are the edge ids and the values are lists that contain any numerical value that are represented in the edge size
- **edge_tips** (*dict*) – A dictionary where the keys are the edge ids and the values are lists that contain any value that can be accessed as a tooltip in the rendered network
- **layout_name** (*str*) – Layout to use

pyvipr.network_viz.views.**graphml_view**(*file*, *layout_name='fcose'*)
Read graph stored in GRAPHML format using NetworkX and render a visualization of it

Parameters

- **file** (*str*) – Path to file in graphml format
- **layout_name** (*str*) – Name of layout to use

pyvipr.network_viz.views.**sif_view**(*file*, *layout_name='fcose'*)
Read graph stored in SIF format using NetworkX and render a visualization of it

Parameters

- **file** (*str*) – Path to file in sif format
- **layout_name** (*str*) – Name of layout to use

pyvipr.network_viz.views.**sbgn_xml_view**(*file*, *layout_name='fcose'*)
Read graph stored in SBGN XML format using NetworkX and render a visualization of it

Parameters

- **file** (*str*) – Path to file in SBGN XML format
- **layout_name** (*str*) – Name of layout to use

pyvipr.network_viz.views.**json_view**(*file*, *layout_name='fcose'*)
Read graph stored in cytoscape json format using NetworkX and render a visualization of it

Parameters

- **file** (*str*) – Path to file in cytoscape json format
- **layout_name** (*str*) – Name of layout to use

pyvipr.network_viz.views.**dynamic_json_view**(*file*, *layout_name='fcose'*)
Read graph stored in cytoscape json format using NetworkX and render a visualization of it. This function is for graphs saved from dynamic visualizations.

Parameters

- **file** (*str*) – Path to file in cytoscape json format
- **layout_name** (*str*) – Name of layout to use

pyvipr.network_viz.views.**gexf_view**(*file*, *node_type=None*, *relabel=False*, *version='1.2draft'*, *layout_name='fcose'*)
Read graph stored in GEXF format using NetworkX and render a visualization of it

Parameters

- **file** (*str*) – Path to file in gexf format
- **node_type** (*Python type (default: none)*) – Convert node ids to this type if not None
- **relabel** (*bool (default: False)*) – If True relabel the nodes to use the GEXF node "label attribute" instead of the node "id" attribute as the NetworkX node label
- **version** (*str (default: 1.2draft)*) – Version of GEFX File Format (see https://gephi.org/gexf/format/schema.html). Supported values: "1.1draft", "1.2dra
- **layout_name** (*str*) – Name of layout to use

pyvipr.network_viz.views.**gml_view**(*file*, *label='label'*, *destringizer=None*, *layout_name='fcose'*)

Read graph stored in GML format using NetworkX and render a visualization of it

Parameters

- **file** (*str*) – Path to file in cytoscape json format
- **label** (*str, optional*) – If not None, the pased nodes will be renamed according to node attributes indicated by label. Default value: 'label'
- **destringizer** (*callable, optional*) – A destringizer that recovers values stored as strings in GML. If it cannot convert a string to a value, a ValueError is raised. Default value: None
- **layout_name** (*str*) – Name of layout to use

pyvipr.network_viz.views.**yaml_view**(*file*, *layout_name='fcose'*)

Read graph stored in YAML format using NetworkX and render a visualization of it

Parameters

- **file** (*str*) – Path to file in YAML format
- **layout_name** (*str*) – Name of layout to use

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

pyvipr.network_viz.network_viz, 32
pyvipr.network_viz.views, 32
pyvipr.pysb_viz.dynamic_viz, 22
pyvipr.pysb_viz.static_viz, 17
pyvipr.pysb_viz.views, 23
pyvipr.tellurium_viz.dynamic_viz, 30
pyvipr.tellurium_viz.static_viz, 29
pyvipr.tellurium_viz.views, 30

INDEX

A

atom_rules_view() (*pyvipr.pysb_viz.static_viz.PysbStaticViz method*), 17

C

cluster_rxns_by_rules_view() (*in module pyvipr.pysb_viz.views*), 27
cluster_rxns_by_rules_view() (*pyvipr.pysb_viz.static_viz.PysbStaticViz method*), 18
compartments_data_graph() (*pyvipr.pysb_viz.static_viz.PysbStaticViz method*), 18

D

dynamic_json_view() (*in module pyvipr.network_viz.views*), 33
dynamic_sp_comm_view() (*pyvipr.pysb_viz.dynamic_viz.PysbDynamicViz method*), 22
dynamic_sp_comp_view() (*pyvipr.pysb_viz.dynamic_viz.PysbDynamicViz method*), 23
dynamic_sp_view() (*pyvipr.pysb_viz.dynamic_viz.PysbDynamicViz method*), 23
dynamic_sp_view() (*pyvipr.tellurium_viz.dynamic_viz.TelluriumDynamicViz method*), 30

E

edges_colors_sizes() (*pyvipr.pysb_viz.dynamic_viz.PysbDynamicViz method*), 23
edges_colors_sizes() (*pyvipr.tellurium_viz.dynamic_viz.TelluriumDynamicViz method*), 30

G

gexf_view() (*in module pyvipr.network_viz.views*), 33
gml_view() (*in module pyvipr.network_viz.views*), 34
graph_merge_pair_edges() (*pyvipr.pysb_viz.static_viz.PysbStaticViz static method*), 18
graphml_view() (*in module pyvipr.network_viz.views*), 33

H

highlight_nodes_view() (*pyvipr.pysb_viz.static_viz.PysbStaticViz method*), 19

J

json_view() (*in module pyvipr.network_viz.views*), 33

M

matrix_bidirectional_rates() (*pyvipr.pysb_viz.dynamic_viz.PysbDynamicViz method*), 23
merge_nodes() (*pyvipr.pysb_viz.static_viz.PysbStaticViz static method*), 19
merge_reactions2rules() (*pyvipr.pysb_viz.static_viz.PysbStaticViz method*), 19

N

node_data() (*pyvipr.pysb_viz.dynamic_viz.PysbDynamicViz method*), 23
node_data() (*pyvipr.tellurium_viz.dynamic_viz.TelluriumDynamicViz method*), 30
nx_graph_dyn_view() (*in module pyvipr.network_viz.views*), 32
nx_graph_view() (*in module pyvipr.network_viz.views*), 32

P

parse_name() (*in module pyvipr.pysb_viz.static_viz*), 22
projected_bireactions_view() (*in module pyvipr.pysb_viz.views*), 26

projected_graph() (pyvipr.pysb_viz.static_viz.PysbStaticViz method), 19
projected_rules_view() (in module pyvipr.pysb_viz.views), 26
projected_species_from_bireactions_view() (in module pyvipr.pysb_viz.views), 26
projected_species_from_bireactions_view() (pyvipr.pysb_viz.static_viz.PysbStaticViz method), 19
projected_species_from_rules_view() (in module pyvipr.pysb_viz.views), 27
PysbDynamicViz (class in pyvipr.pysb_viz.dynamic_viz), 22
PysbStaticViz (class in pyvipr.pysb_viz.static_viz), 17
pyvipr.network_viz.network_viz (module), 32
pyvipr.network_viz.views (module), 32
pyvipr.pysb_viz.dynamic_viz (module), 22
pyvipr.pysb_viz.static_viz (module), 17
pyvipr.pysb_viz.views (module), 23
pyvipr.tellurium_viz.dynamic_viz (module), 30
pyvipr.tellurium_viz.static_viz (module), 29
pyvipr.tellurium_viz.views (module), 30

S

sbgn_xml_view() (in module pyvipr.network_viz.views), 33
sif_view() (in module pyvipr.network_viz.views), 33
sim_model_dyn_view() (in module pyvipr.pysb_viz.views), 28
sp_comm_asyn_fluidc_view() (in module pyvipr.pysb_viz.views), 25
sp_comm_asyn_fluidc_view() (in module pyvipr.tellurium_viz.views), 32
sp_comm_asyn_lpa_view() (in module pyvipr.pysb_viz.views), 24
sp_comm_asyn_lpa_view() (in module pyvipr.tellurium_viz.views), 31
sp_comm_dyn_view() (in module pyvipr.pysb_viz.views), 28
sp_comm_girvan_newman_view() (in module pyvipr.pysb_viz.views), 25
sp_comm_girvan_newman_view() (in module pyvipr.tellurium_viz.views), 32
sp_comm_greedy_view() (in module pyvipr.pysb_viz.views), 24
sp_comm_greedy_view() (in module pyvipr.tellurium_viz.views), 31
sp_comm_label_propagation_view() (in module pyvipr.pysb_viz.views), 25
sp_comm_label_propagation_view() (in module pyvipr.tellurium_viz.views), 31
sp_comm_louvain_hierarchy_view() (in module pyvipr.pysb_viz.views), 24
sp_comm_louvain_hierarchy_view() (pyvipr.pysb_viz.static_viz.PysbStaticViz method), 19
sp_comm_louvain_hierarchy_view() (pyvipr.tellurium_viz.static_viz.TelluriumStaticViz method), 29
sp_comm_louvain_view() (in module pyvipr.pysb_viz.views), 24
sp_comm_louvain_view() (in module pyvipr.tellurium_viz.views), 31
sp_comm_louvain_view() (pyvipr.pysb_viz.static_viz.PysbStaticViz method), 20
sp_comm_louvain_view() (pyvipr.tellurium_viz.static_viz.TelluriumStaticViz method), 29
sp_comp_dyn_view() (in module pyvipr.pysb_viz.views), 27
sp_comp_view() (in module pyvipr.pysb_viz.views), 24
sp_comp_view() (pyvipr.pysb_viz.static_viz.PysbStaticViz method), 20
sp_dyn_view() (in module pyvipr.pysb_viz.views), 27
sp_dyn_view() (in module pyvipr.tellurium_viz.views), 31
sp_rules_fxns_view() (in module pyvipr.pysb_viz.views), 26
sp_rules_fxns_view() (pyvipr.pysb_viz.static_viz.PysbStaticViz method), 20
sp_rules_graph() (pyvipr.pysb_viz.static_viz.PysbStaticViz method), 20
sp_rules_mod_view() (in module pyvipr.pysb_viz.views), 26
sp_rules_mod_view() (pyvipr.pysb_viz.static_viz.PysbStaticViz method), 20
sp_rules_view() (in module pyvipr.pysb_viz.views), 26
sp_rules_view() (pyvipr.pysb_viz.static_viz.PysbStaticViz method), 21
sp_rxns_bidirectional_graph() (pyvipr.pysb_viz.static_viz.PysbStaticViz method), 21
sp_rxns_bidirectional_view() (in module pyvipr.pysb_viz.views), 25
sp_rxns_bidirectional_view() (pyvipr.pysb_viz.static_viz.PysbStaticViz method), 21
sp_rxns_graph() (pyvipr.pysb_viz.static_viz.PysbStaticViz

method), 21
sp_rxns_graph() (*pyvipr.tellurium_viz.static_viz.TelluriumStaticViz method*), 29
sp_rxns_view() (*in module pyvipr.pysb_viz.views*), 25
sp_rxns_view() (*in module pyvipr.tellurium_viz.views*), 30
sp_rxns_view() (*pyvipr.pysb_viz.static_viz.PysbStaticViz method*), 21
sp_rxns_view() (*pyvipr.tellurium_viz.static_viz.TelluriumStaticViz method*), 29
sp_view() (*in module pyvipr.pysb_viz.views*), 23
sp_view() (*in module pyvipr.tellurium_viz.views*), 30
sp_view() (*pyvipr.pysb_viz.static_viz.PysbStaticViz method*), 21
sp_view() (*pyvipr.tellurium_viz.static_viz.TelluriumStaticViz method*), 29
species_graph() (*pyvipr.pysb_viz.static_viz.PysbStaticViz method*), 22
species_graph() (*pyvipr.tellurium_viz.static_viz.TelluriumStaticViz method*), 29

T

TelluriumDynamicViz (*class in pyvipr.tellurium_viz.dynamic_viz*), 30
TelluriumStaticViz (*class in pyvipr.tellurium_viz.static_viz*), 29

Y

yaml_view() (*in module pyvipr.network_viz.views*), 34