pyvipr Documentation

Release 1.0.6

Oscar O Ortega

Apr 04, 2022

CONTENTS:

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_from_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

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search