GraSPy: Graph Statistics in Python

Jaewon Chung\textsuperscript{1,}\textsuperscript{†} \hfill J1C@JHU.EDU
Benjamin D. Pedigo\textsuperscript{1,}\textsuperscript{†} \hfill BPEDIGO@JHU.EDU
Eric W. Bridgeford\textsuperscript{2} \hfill EBRIDGE2@JHU.EDU
Bijan K. Varjavand\textsuperscript{1} \hfill BVARJAV1@JHU.EDU
Hayden S. Helm\textsuperscript{3} \hfill HH@JHU.EDU
Joshua T. Vogelstein\textsuperscript{1,3,4,⋆} \hfill JOVO@JHU.EDU

\textsuperscript{1}Department of Biomedical Engineering, Johns Hopkins University, Baltimore, MD 21218
\textsuperscript{2}Department of Biostatistics, Johns Hopkins University, Baltimore, MD 21218
\textsuperscript{3}Center for Imaging Science, Johns Hopkins University, Baltimore, MD 21218
\textsuperscript{4}Kavli Neuroscience Discovery Institute, Institute for Computational Medicine, Johns Hopkins University, Baltimore, MD 21218
\textsuperscript{†}Denotes equal contribution
\textsuperscript{⋆}Corresponding author

Editor: Alexandre Gramfort

Abstract

We introduce GraSPy, a Python library devoted to statistical inference, machine learning, and visualization of random graphs and graph populations. This package provides flexible and easy-to-use algorithms for analyzing and understanding graphs with a \texttt{scikit-learn} compliant API. GraSPy can be downloaded from Python Package Index (PyPi), and is released under the Apache 2.0 open-source license. The documentation and all releases are available at \url{https://neurodata.io/graspy}.

Keywords: Python, graph analysis, network analysis, statistical inference, machine learning

1. Introduction

Graphs, or networks, are a mathematical representation of data that consists of discrete objects (nodes or vertices) and relationships between these objects (edges). For example, a brain can be represented with nodes corresponding to each brain region and edges representing the presence of a structural connection between regions (Vogelstein et al., 2019). Since graphs necessarily deal with relationships between nodes, classical statistical assumptions about independence are violated. Thus, novel methodology is required for performing statistical inference on graphs and populations of graphs (Athreya et al., 2018). While the theory for inference on graphs is highly developed, to date, there has not existed a numerical package implementing these methods. GraSPy fills this gap by providing implementations of algorithms with strong statistical guarantees, such as graph and multi-graph embedding methods, two-graph hypothesis testing, and clustering of vertices of graphs. Many of the algorithms implemented in GraSPy are flexible and can operate on graphs that are weighted or unweighted, as well as directed or undirected. Table 1 provides a comparison of GraSPy
Figure 1: Illustration of modules and procedure for statistical inference on graphs, populations of graphs, or simulated data. A detailed description of each module is given in Section 2.

to other existing graph analysis packages (Hagberg et al., 2008; Peixoto. 2014. Leskovec and Sosič, 2016).

2. Library Overview

GraSPy includes functionality for fitting and sampling from random graph models, performing dimensionality reduction on graphs or populations of graphs (embedding), testing hypotheses on graphs, and plotting of graphs and embeddings. The following provides brief overview of different modules of GraSPy. An example workflow using these modules is shown in Figure 1. More detailed overview and code usage can be found in the tutorial section of GraSPy documentation at https://graspy.neurodata.io/tutorial All descriptions here correspond to GraSPy version 0.1.1.

Simulations Several random graph models are implemented in GraSPy, including the Erdős-Rényi (ER) model, stochastic block model (SBM), degree-corrected Erdős-Rényi (DCER) model, degree-corrected stochastic block model (DCSBM), and random dot product graph (RDPG) (Holland et al., 1983; Karrer and Newman, 2011; Young and Scheinerman, 2007). The simulations module allows the user to sample random graphs given the parameters of one of these models. Additionally, the user can specify a distribution on the weights of graph edges.

Utils GraSPy includes a variety of utility functions for graph and graph population importing and preprocessing. GraSPy can import graphs represented as NetworkX objects or NumPy arrays. Preprocessing examples include finding the largest connected component of a graph, finding the intersection or union of connected components across multiple graphs, or checking whether a graph is directed.

Embed Inferences on random graphs can leverage low-dimensional Euclidean representations of the vertices, known as latent positions. Adjacency spectral embedding (ASE)
Table 1: Qualitative comparison of Python graph analysis packages. GraSPy is largely complementary to existing graph analysis packages in Python. GraSPy does not implement many of the essential algorithms for operating on graphs (rather, it leverages NetworkX for these implementations). The focus of GraSPy is on statistical modeling of populations of networks, with features such as multiple graph embeddings, model fitting, and hypothesis testing. A ✓ is given for packages that implement the respective feature, a √ for packages that partially implement the respective feature, and a ✗ is given for packages that do not implement the respective feature. Note that while a ✓ shows that the feature exists in the corresponding package, it does not imply that the specific algorithms are the same for all packages.

<table>
<thead>
<tr>
<th>package</th>
<th>traversal</th>
<th>network stats.</th>
<th>communities</th>
<th>embed</th>
<th>mult. embed</th>
<th>model fit</th>
<th>simulations</th>
<th>hyp. test</th>
<th>plotting</th>
<th>pip install</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraSPy 0.1.1</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NetworkX 2.3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>graph-tool 2.29</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Snap.py 4.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>

Models GraSPy includes classes for fitting random graph models to an input graph (Figure 2). Currently, ER, SBM, DCER, DCSBM, and RDPG are supported for model estimation. After fitting a model to data, the model class can also output goodness-of-fit metrics (mean squared error, likelihood) and the number of estimated model parameters, allowing for model selection. The model classes can also be used to sample new simulated graphs based on the fit model.

Inference Given two graphs, a natural question to ask is whether these graphs are both random samples from the same generative distribution. GraSPy provides two types of test for this null hypothesis: a latent position test and a latent distribution test. Both tests are framed under the RDPG model, where the generative distribution for the graph can be modeled as a set of latent positions. The latent position test can only be performed on two graphs of the same size and with known correspondence between the vertices of the two graphs (Tang et al., 2017). The latent distribution test
Figure 2: Connectome model fitting using GraSPy. Heatmaps show the probability of potential edges for models of graphs fit to the Drosophila larva right mushroom body connectome (unweighted, directed) (Eichler et al., 2017). The node labels correspond to cell types: P) projection neurons, O) mushroom body output neurons, I) mushroom body input neurons. The graph models are: inhomogeneous Erdős-Rényi (IER) model in which all potential edges are specified, random dot product graph (RDPG), degree-corrected stochastic block model (DCSBM), degree-corrected Erdős-Rényi (DCER), stochastic block model (SBM), and Erdős-Rényi (ER). Blocks (cell types) are sorted by number of member vertices and nodes are sorted by degree within blocks. The code used to generate the figure is a tutorial section at https://neurodata.io/graspy.

can be performed on graphs without vertex alignment, or even with slightly different numbers of vertices (Tang et al., 2014).

Cluster GraSPy extends Gaussian mixture models (GMM) and k-means from scikit-learn to sweep over a specified range of parameters and choose the best clustering (Pedregosa et al., 2011). The number of clusters and covariance structure for each GMM is chosen by Bayesian information criterion (BIC), which is a penalized likelihood function

4
to evaluate the quality of estimators (Schwarz et al., 1978). Silhouette score is used to choose the number of clusters for k-means (Rousseeuw, 1987). Clustering is often useful for computing the the community structure of vertices after embedding.

**Plot GraSPy** extends **seaborn** to visualize graphs as adjacency matrices and embedded graphs as paired scatter plots (Waskom et al., 2018). Individual graphs can be visualized using heatmap function, and multiple graphs can be overlaid on top of each other using gridplot. The nodes in both graph visualizations can be sorted by various node metadata, such as node degree or assigned node labels. Pairplot can visualize high dimensional data, such as embeddings, as a pairwise scatter plot.

3. **Code example**

Given the connectomes of the *Drosophila* larva left and right mushroom bodies, one natural question to ask is: how similar are these graphs (Eichler et al., 2017)? We can frame this question as whether these graphs are generated from the same distribution of latent positions (Tang et al., 2014). We can use the latent distribution test to test this hypothesis:

```python
from graspy.datasets import load_drosophila_left, load_drosophila_right
from graspy.inference import LatentDistributionTest

# Load data
left_graph = load_drosophila_left()
right_graph = load_drosophila_right()

# Initialize hypothesis test object and compute p-value
ldt = LatentDistributionTest(n_components=3, n_bootstraps=500)
p_value = ldt.fit(left_graph, right_graph)
print("p-value: " + str(p_value))
p-value: 0.002
```

4. **Conclusion**

**GraSPy** is an open-source Python package to perform statistical analysis on graphs and graph populations. Its compliance with the **scikit-learn** API makes it an easy-to-use tool for anyone familiar with machine learning in Python (Buitinck et al., 2013). In addition, **GraSPy** is implemented with an extensible class structure, making it easy to modify and add new algorithms to the package. As **GraSPy** continues to grow and add functionality, we believe it will accelerate statistically principled discovery in any field of study concerned with graphs or populations of graphs.

**Acknowledgments**

This work is graciously supported by the DARPA, under agreement numbers FA8650-18-2-7834 and FA8750-17-2-0112. We thank all the contributors for assisting with writing **GraSPy**. We thank the NeuroData Design class, the NeuroData lab, and Carey E. Priebe at Johns Hopkins University for helpful feedback.
References


