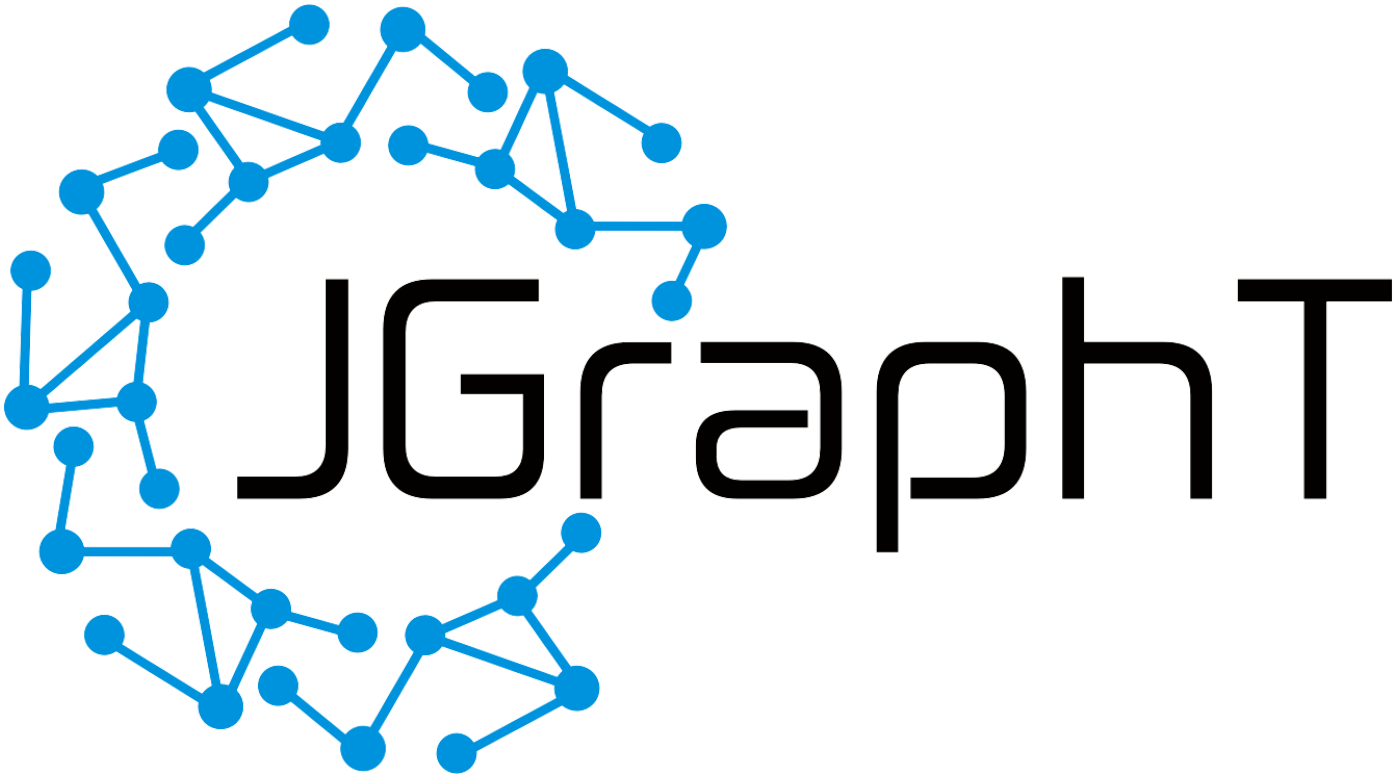
Dimitrios Michail Follow
Jun 29, 2020 · 4 min read · ✦ · Listen

Announcing the Python Bindings of JGraphT



The JGraphT is a stable and mature graph library targeting the JVM ecosystem. It contains very efficient and generic graph data-structures along with a large collection of state-of-the-art algorithms.

Today, we are delighted to announce the availability of the JGraphT library in the Python ecosystem. The library is named `jgrapht` and can be directly installed using

The best of both worlds

While the JGraphT library is written in Java, the Python Bindings for JGraphT is a pure/native Python package which is completely independent from the JVM. This means that the users are not required to install the JVM or anything related. How is this achieved? We use the GraalVM in order to build a native shared library of the JGraphT, which we call the backend library. Then, using SWIG and custom wrappers we create a Python graph library. The result is a very efficient and easy to use library.

First official version is `1.5.0.0` which is build using the `1.5.0` backend version.

Who is it for?

The library is designed to be usable by all kinds of different users. It tries to satisfy both performance and usability requirements. This is achieved by providing two main graph implementations.

- (integer) graph
- (any-hashable) graph

These two implementations are able to cover almost all use-cases of graphs or networks which can be found both in industry and academia. Are you a data scientist who is performing graph mining? Are you a software developer who wants to do some graph analytics? Are you using graphs to model dependencies using DAGs (directed acyclic graphs)? Are you modelling networks solving some flow problem? Are you trying to compute TSP tours?
These use-cases and much more are easily handled in jgrapht.

Installation

If your pip version is old, you might need to upgrade first using `pip install — upgrade pip`. Otherwise, the library can be installed by performing:




The integer graph

The (integer) graph is a graph implementation which uses integers for the vertices and edges of the graph. It is tailored towards performance. Let us create a small graph and compute pagerank:

```
import jgrapht
import jgrapht.algorithms.scoring as scoring

g = jgrapht.create_graph(directed=True)

for v in range(0, 11):
    g.add_vertex(v)

g.add_edges_from([(1, 2), (2, 1), (3, 0), (3, 1), (4, 1),
   (4, 3), (4, 5), (5, 1), (5, 4), (6, 1), (6, 4), (7, 1),
   (7, 4), (8, 1),  (8, 4), (9, 4), (10, 4)])

print(g)

pagerank = scoring.pagerank(g)

result = [pagerank[v] for v in g.vertices]
print(result)
```

The any-hashable graph

The any-hashable graph is a graph which can use any Python hashable as vertices and edges. It also supports to associate attributes/properties directly with vertex and edges, instead of maintaining external dictionaries. This graph is tailored towards usability at the expense of some performance overhead.

Let us see a small example by creating the Königsberg bridge problem and trying to find an Euler cycle.

```
import jgrapht
import jgrapht.algorithms.cycles as cycles

g = jgrapht.create_graph(directed=False, allowing_multiple_edges=True,
any_hashable=True)
```




```
g.add_edge('a', 'c', edge='b1')
g.add_edge('a', 'c', edge='b2')
g.add_edge('c', 'b', edge='b3')
g.add_edge('c', 'b', edge='b4')
g.add_edge('c', 'd', edge='b5')
g.add_edge('a', 'd', edge='b6')
g.add_edge('b', 'd', edge='b7')

g.vertex_attrs['a']['type'] = 'river bank'
g.vertex_attrs['b']['type'] = 'river bank'
g.vertex_attrs['c']['type'] = 'island'
g.vertex_attrs['d']['type'] = 'island'

print(g)

c = cycles.eulerian_cycle(g)
if c is None:
    print ('No Eulerian cycle exists')
```

Since it does not exist, the above code will output that ‘No Eulerian cycle exists’. We can instead solve the Chinese Postman Problem which allows a minimum amount of edge duplicates:

```
p = cycles.chinese_postman(g)
print (','.join([e for e in p]))
```

The output will be `b7,b7,b6,b1,b4,b3,b2,b1,b5`!

Algorithms

The library contains a large collection of algorithms including:

- Graph traversals (BFS, DFS, etc.)
- Graph metrics such as radius, diameter, etc.
- Graph generators using famous models
- Connectivity




- Cycles
- Coloring
- Clustering
- Centralitites, Pagerank, etc.
- Planarity Detection
- Flows and Cuts
- Drawing algorithms
- Isomorphism detection
- Tours (TSP, Metric TSP)
- Clique enumeration
- Vertex Cover

Importers/Exporters

The library contains importers/exporters for the following formats:

- GraphML
- Dot
- GEXF
- Json
- GML
- CSV
- Dimacs




Graphs with any-hashable as vertices/edges which also support attributes internally, automatically define these callback functions, to make this even easier.

Documentation

See https://python-jgrapht.readthedocs.io/ for the official docs. It contains a very detailed description of the API, as well as tutorials and examples.

More information

More information can be found at the paper:

> *Michail, D., Kinable, J., Naveh, B., & Sichi, J. V., JGraphT — A Java Library for Graph Data Structures and Algorithms. ACM Transactions on Mathematical Software, 46(2), May 2020.*

at https://doi.org/10.1145/3381449 together with a thorough description of the backend library including performance experiments.

Links

- https://python-jgrapht.readthedocs.io/
- https://pypi.org/project/jgrapht
- https://github.com/d-michail/python-jgrapht
- https://jgrapht.org/
- https://github.com/jgrapht/jgrapht