odgf-python – A Python Interface for the Open Graph Drawing Framework*

Simon D. Fink[0000−0002−2754−1195], Andreas Strobl

Faculty for Mathematics and Computer Science, University of Passau, Passau, Germany
{finksim,strobland}@fim.uni-passau.de

Abstract. The Open Graph Drawing Framework (OGDF) is a C++ library that contains a vast amount of algorithms and data structures for automatic graph drawing. While the library is powerful, it is also not easily accessible for new users and the nature of C++ makes implementing even simple algorithms cumbersome to non-experts. The `odgf-python` project remedies these problems by making the full functionality of the OGDF available from Python, providing a visual way to iteratively develop graph algorithms.

Keywords: OGDF · Graph Algorithms · Python · Jupyter Notebooks

The Open Graph Drawing Framework (OGDF) [1] is a C++ library that contains a vast amount of algorithms and data structures for automatic graph drawing. While the library is powerful, it is also not easily accessible for new users and the nature of C++ makes implementing even simple algorithms cumbersome to non-experts. New C++ projects require non-trivial set-up and distinct code-compile-execute iterations make it hard to incrementally develop a new algorithm. This is because they separate the code from its results and there is no easy way to debug and visually analyze the current state of the program.

Many of these issues have been resolved within the Python ecosystem [7]. The `odgf-python`[1] project remedies these problems also for the OGDF by making its full functionality available from Python. A similar approach has recently been taken to make the Computational Geometry Algorithms Library (CGAL) more accessible [2]. A Python interface greatly reduces the overhead and complications when using a library for the first time and unlocks a large ecosystem of other libraries and tools, for example the interactive computing Notebooks provided by Project Jupyter [3]. These Notebooks allow iteratively developing algorithms and visualize their results inline right next to their code [7]. They are widely used in different scientific contexts and constitute the de facto standard in data science [6].

A key component of the `odgf-python` library is its graph display widget for Jupyter Notebooks [8]. It can be used as-is to quickly explore and modify the

* Funded by the Deutsche Forschungsgemeinschaft (German Research Foundation, DFG) under grant RU-1903/3-1.

[1] `https://github.com/N-Coder/ogdf-python`

current graph in-memory. At the same time, it can be used as a building block for more complex user interfaces, where its interactivity can be fully customized to suit different applications. Combining this with the UI components of Jupyter's `ipywidgets` allows to easily build platform-independent user interfaces for interacting with graphs and algorithms. Example use cases include the iterative development of new graph algorithms, step-by-step debugging of implemented algorithms, visual editing of in-memory graphs and variables, interactive visualization of algorithms for teaching, and flexible user interfaces for domain-specific applications.

Internally, `ogdf-python` uses the `cppyy` library [4,5] to automatically generate python bindings for the C++ OGDF library. One of the main advantages of `cppyy` is that there is no need for explicitly declared bindings or interfaces. This makes `ogdf-python` future-proof by being automatically compatible with future versions of the OGDF without the need for adaptions or manual updates to declarations. The `cppyy` library also allows for arbitrary C++ code to be loaded and called from Python. This allows for variables, functions and even classes to be created and used in either language.

Features

- **No C++ skills needed**: The full OGDF API is available from Python.
- **Rapid prototyping**: Python needs less boilerplate and allows more idiomatic constructions, without the need to configure and compile anything.
- **Iterative execution**: Jupyter Notebooks allow individual lines of code to be adapted and re-run, retaining all previous variable values.
- **Inline results**: Graphs are visually displayed right next to the code that generates them.
- **Interactive graph exploration**: The inline display allows interactive zooming and panning to easily explore the graph.
- **Extensible building block**: The library can be easily combined with other projects from the Python ecosystem, for example with `ipywidgets` to build portable user interfaces for graphs.

Installation

The `ogdf-python` library comes with a prebuilt version of the OGDF called `ogdf-wheel` that works on Linux, macOS and in the Windows Subsystem for Linux. Both components can easily be installed using the Python package manager pip:

```
pip3 install ogdf-python ogdf-wheel notebook
jupyter notebook # start an interactive notebook
```

References

1. Chimani, M., Gutwenger, C., Jünger, M., Klau, G.W., Klein, K., Mutzel, P.: The Open Graph Drawing Framework (OGDF). In: Tamassia, R. (ed.) Handbook on Graph Drawing and Visualization, pp. 543–569. Chapman and Hall/CRC (2013)
2. Goren, N., Fogel, E., Halperin, D.: CGAL made more accessible. CoRR (2022). `https://doi.org/10.48550/arXiv.2202.13889`
3. Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., Willing, C., development team, J.: Jupyter notebooks – a publishing format for reproducible computational workflows. In: Loizides, F., Scmidt, B. (eds.) Proceedings of the 20th International Conference on Electronic Publishing. pp. 87–90. IOS Press (2016), `https://eprints.soton.ac.uk/403913/`
4. Kundu, B., Vassilev, V., Lavrijsen, W.: Efficient and accurate automatic python bindings with cppyy & cling. CoRR (2023). `https://doi.org/10.48550/arXiv.2304.02712`
5. Lavrijsen, W.T., Dutta, A.: High-performance python-c++ bindings with PyPy and cling. In: Proceedings of the 6th Workshop on Python for High-Performance and Scientific Computing (PyHPC 2016). IEEE (2016). `https://doi.org/10.1109/pyhpc.2016.008`
6. Perkel, J.M.: Why jupyter is data scientists' computational notebook of choice. Nature **563**(7729), 145–146 (oct 2018). `https://doi.org/10.1038/d41586-018-07196-1`
7. Perkel, J.M.: Ten computer codes that transformed science. Nature **589**(7842), 344–348 (jan 2021). `https://doi.org/10.1038/d41586-021-00075-2`
8. Strobl, A.: A generic widget library for rapid-prototyping of graph algorithms in jupyter notebooks (2020), `https://www.fim.uni-passau.de/fileadmin/dokumente/fakultaeten/fim/lehrstuhl/rutter/abschlussarbeiten/2022-Andreas_Strobl-BA.pdf`