The Journal of Open Source Software

retworkx: A High-Performance Graph Library for Python

Matthew Treinish[1], **Ivan Carvalho**[2], **Georgios Tsilimigkounakis**[3], and **Nahum Sá**[4]

1 IBM Quantum, IBM T.J. Watson Research Center, Yorktown Heights, USA
2 University of British Columbia, Kelowna, Canada
3 National Technical University of Athens, Athens, Greece
4 Centro Brasileiro de Pesquisas Físicas, Rio de Janeiro, Brazil

DOI: 10.21105/joss.03968

Software

- Review
- Repository
- Archive

Editor: Vincent Knight

Reviewers:

- @szhorvat
- @inakleinbottle

Submitted: 28 October 2021
Published: 01 December 2021

Network and graph analysis is a widely applicable field of research, and Python is a popular programming language. In *retworkx*, we provide a high-performance, flexible graph and network analysis library for Python. *retworkx* is inspired by *NetworkX* (Hagberg et al., 2008) but addresses many performance concerns of the latter. *retworkx* is particularly suited for performance-sensitive applications that use graph representations.

Statement of need

retworkx is a general-purpose graph theory library focused on performance. It wraps low-level Rust code (Matsakis & Klock, 2014) into a flexible Python API, providing fast implementations for popular graph algorithms.

retworkx originated from the performance demands of the Qiskit compiler (Treinish et al., 2021). At first, Qiskit used the *NetworkX* library (Hagberg et al., 2008) to construct directed acyclic graph (DAG) representations of quantum circuits which the compiler operates on to perform analysis and transformations (Childs et al., 2019). As the development of Qiskit progressed, the input size of the executed quantum circuits grew, and *NetworkX* started to become a bottleneck. Hence, *retworkx* development emerged to cover the graph usage in Qiskit. The library is now also used by other projects (Jha et al., 2021; Ullberg, 2021).

Related work

To address the performance issues in Qiskit, we explored several graph library alternatives. *igraph* (Csardi & Nepusz, 2006), *graph-tool* (Peixoto, 2014), and *SNAP* (Leskovec & Sosič, 2016) are Python libraries written in C or C++ that can replace *NetworkX*.

However, there was a strong desire to keep the flexibility that *NetworkX* provided for exploring and interacting with the graphs, which precluded custom application-specific graph data structures. The graph libraries mentioned above either had issues integrating with Qiskit or APIs that were too rigid, such that the migration of existing code was more complex than desired. Thus, the main contribution of *retworkx* is keeping the ease of use of *NetworkX* without sacrificing performance.

The Journal of Open Source Software

Graph data structures

retworkx provides two core data structures: `PyGraph` and `PyDiGraph`. They correspond to undirected and directed graphs, respectively. Graphs describe a set of nodes and the edges connecting pairs of those nodes. Internally, *retworkx* leverages the *petgraph* library (bluss et al., 2021) to store the graphs and the *PyO3* library (Hewitt et al., 2021) for the Python bindings.

Nodes and edges of the graph may also be associated with weights. Weights can contain arbitrary data, such as node labels or edge lengths. Any Python object can be a weight, which makes the library flexible because no assumptions are made about the weight types.

retworkx operates on weights with callbacks. Callbacks are functions that take weights and return statically typed data. They resemble the named attributes in *NetworkX*. Callbacks are beneficial because they bridge the arbitrary stored data with the static types *retworkx* expects.

A defining characteristic of *retworkx* graphs is that each node maps to a non-negative integer node index, and similarly, each edge maps to an edge index. Those indices uniquely determine nodes and edges in the graph. Moreover, the indices provide a clear separation between the underlying graph structure and the data associated with weights.

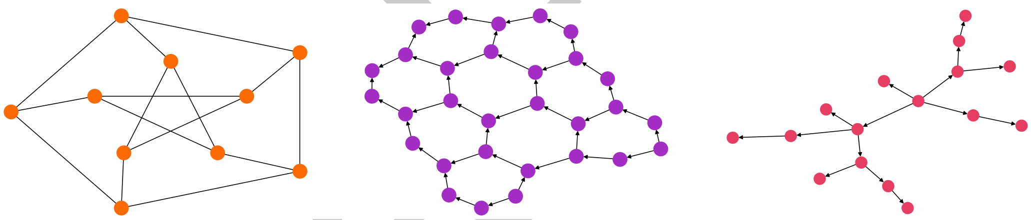

Figure 1: A Petersen graph, a hexagonal lattice graph, and a binomial tree graph visualized with the `retworkx.visualization` module.

Use Cases

retworkx is suitable for modeling graphs ranging from a few nodes scaling up to millions. The library is particularly suited for applications that have core routines executing graph algorithms, such as Qiskit. In those applications, the performance of *retworkx* considerably reduces computation time.

We demonstrate the library's performance and use cases comparing *retworkx* to other popular graph libraries[1] on a benchmark:

Library	*retworkx*	*NetworkX*	*python-igraph*	*graph-tool*
Version[2,3,4]	0.10.2	2.6.3	0.9.6	2.43

The benchmark is available on Github[5] for reproducibility. We present results conducted on the same machine running Python 3.9.7, with 128GB of DDR4 RAM @ 3200MHz and Intel(R)

[1] *SNAP* was not included in the benchmarks because its Python wrapper did not contain the required functions

[2] *retworkx*, *NetworkX*, and *igraph* were installed from PyPI (Python Package Index) with the `pip` command

[3] *graph-tool* does not publish packages on PyPI and had to be compiled and installed manually

[4] *retworkx* and *igraph* provide precompiled wheel binaries (Holth, 2012), hence no Rust or C compilers were required. *graph-tool* required a C++ compiler and all of the C++ library dependencies

[5] https://github.com/mtreinish/retworkx-comparison-benchmarks

Core i7-6900K CPU @ 3.20GHz with eight cores and 16 threads.

Graph Creation

The first use case is to represent real-world networks by creating graphs with their respective nodes and edges. We compare the time to create graphs representing the USA road network from the 9th DIMACS challenge dataset (Demetrescu et al., 2009). Each graph contains $|V| = 23,947,347$ nodes and $|E| = 58,333,344$ weighted edges.

The results in Figure 2 shows that *retworkx* is on average 3x faster than *NetworkX* on this benchmark. *retworkx* is also the fastest among all libraries, being at least 5x faster than *igraph* and *graph-tool*.

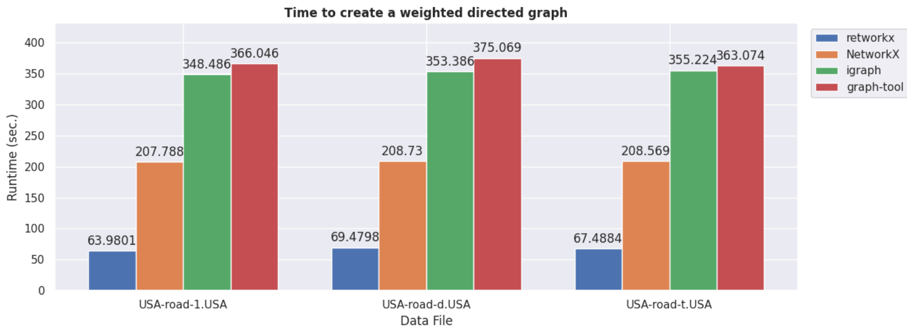

Figure 2: Time to create the USA road network graph with 23,947,347 nodes and 58,333,344 edges.

Shortest Path

The second use case is to calculate the distance among nodes in a graph using Dijkstra's algorithm (Dijkstra, 1959)[6]. We compare two scenarios. In the first scenario, we calculate the distance between the first and the last node in the USA road network. In the second scenario, we calculate the distance among all nodes in the City of Rome road network, with the dataset also coming from the 9th DIMACS challenge (Demetrescu et al., 2009). The City of Rome network has $|V| = 3,353$ nodes and $|E| = 8,870$ weighted edges.

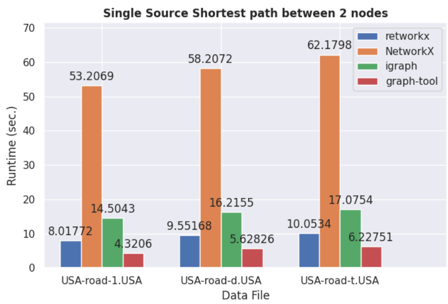

Figure 3: Time to find the shortest path between two nodes in the USA road network.

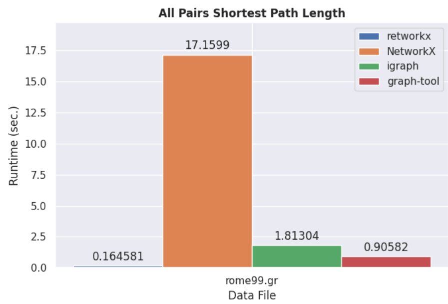

Figure 4: Time to find the shortest path among all nodes in the City of Rome road network.

retworkx is 6x faster than *NetworkX* on the single-source scenario, and 104x faster on the all-pairs scenario as shown in Figures 3 and 4. We highlight that *NetworkX* is the slowest

[6] *igraph* and *graph-tool* use Johnson's algorithm (Johnson, 1977) for all-pairs shortest paths, which contains Dijkstra's as a subroutine

library among all in the benchmark because it was designed with different goals in mind, such as readability and ease of distribution.

retworkx is the second-fastest in the single-source scenario after *graph-tool*, but we interpret the result as a trade-off. *graph-tool* is 1.6-1.8x faster than *retworkx* on the single-source shortest-path calculation but takes 5x longer to create the graph, as shown in the other benchmark. *retworkx* creates graphs faster and trades some of the shortest-path calculation performance to accept Python callbacks that work with arbitrary data types. In the all-pairs scenario, *retworkx* is the fastest with a 5.6x speedup compared to *graph-tool* which is the second-fastest.

Subgraph Isomorphism

The third use case is to detect a pattern graph within a larger graph using the VF2 or VF2++ algorithms (Cordella et al., 2004; Jüttner & Madarasi, 2018). We compare the time to answer if pairs of graphs from the ARG Database are subgraph-isomorphic (De Santo et al., 2003). The graphs are unlabeled, bounded-valence graphs ranging from 20 to 1000 nodes with valence $v \in \{3, 6, 9\}$. They are organized in pairs such that the subgraph size is either 20%, 40% or 60% of the full graph.

The results in Figure 5 show that *retworkx* consistently outperforms *NetworkX* by two orders of magnitude. For $n = 1000$ nodes, *retworkx* has averaged around the order of 10^{-1} seconds while *NetworkX* is closer to the order of 10^{1} seconds. Compared to other libraries, *retworkx* leads the benchmark together with *igraph*, albeit *retworkx* performs slightly better as the number of nodes grows larger.

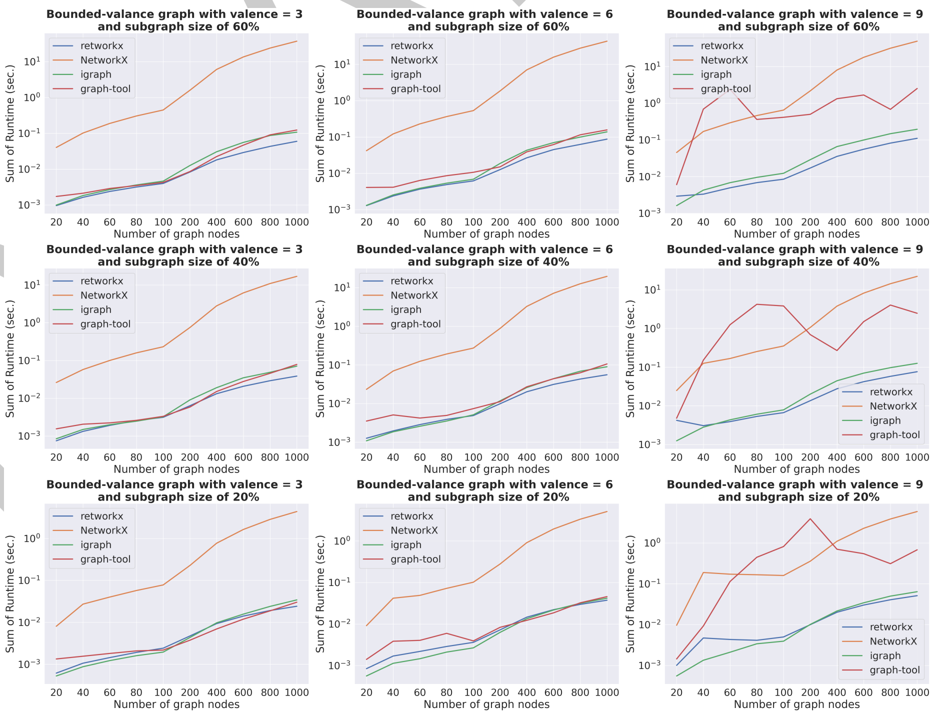

Figure 5: Average time to verify subgraph isomorphism versus the number of graph nodes, grouped by valence number and subgraph size.

Acknowledgements

We thank Kevin Krsulich for his help in getting *retworkx* ready for use by Qiskit; Lauren Capelluto and Toshinari Itoko for their continued support and help with code review; and all of the *retworkx* contributors who have helped the library improve over time.

References

bluss, Comets, J.-M., Borgna, A., Larralde, M., Mitchener, B., & Kochkov, A. (2021). Petgraph. In *GitHub repository*. GitHub. https://github.com/petgraph/petgraph

Childs, A. M., Schoute, E., & Unsal, C. M. (2019). Circuit Transformations for Quantum Architectures. In W. van Dam & L. Mancinska (Eds.), *14th conference on the theory of quantum computation, communication and cryptography (TQC 2019)* (Vol. 135, pp. 3:1–3:24). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. https://doi.org/10.4230/LIPIcs.TQC.2019.3

Cordella, L. P., Foggia, P., Sansone, C., & Vento, M. (2004). A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *26*(10), 1367–1372. https://doi.org/10.1109/TPAMI.2004.75

Csardi, G., & Nepusz, T. (2006). The igraph software package for complex network research. *InterJournal*, *Complex Systems*, 1695. https://igraph.org

De Santo, M., Foggia, P., Sansone, C., & Vento, M. (2003). A large database of graphs and its use for benchmarking graph isomorphism algorithms. *Pattern Recogn. Lett.*, *24*(8), 1067–1079. https://doi.org/10.1016/S0167-8655(02)00253-2

Demetrescu, C., Goldberg, A. V., & Johnson, D. (Eds.). (2009). *The shortest path problem: Ninth DIMACS implementation challenge*. American Mathematical Society. https://doi.org/10.1090/dimacs/074

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, *1*, 269–271. https://doi.org/10.1007/BF01386390

Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). Exploring network structure, dynamics, and function using NetworkX. In G. Varoquaux, T. Vaught, & J. Millman (Eds.), *Proceedings of the 7th python in science conference* (pp. 11–15). http://conference.scipy.org/proceedings/SciPy2008/paper_2/

Hewitt, D., Kanagawa, Y., Kim, N., Grunwald, D., Niederbühl, A., messense, Kolenbrander, B., Brandl, G., & Ganssle, P. (2021). PyO3. In *GitHub repository*. GitHub. https://github.com/PyO3/pyo3

Holth, D. (2012). *The wheel binary package format 1.0* (PEP No. 427). https://www.python.org/dev/peps/pep-0427/

Jha, S., Chen, J., Householder, A., & Mi, A. (2021). Qtcodes. In *GitHub repository*. GitHub. https://github.com/yaleqc/qtcodes

Johnson, D. B. (1977). Efficient algorithms for shortest paths in sparse networks. *J. ACM*, *24*(1), 1–13. https://doi.org/10.1145/321992.321993

Jüttner, A., & Madarasi, P. (2018). VF2++—an improved subgraph isomorphism algorithm. *Discrete Applied Mathematics*, *242*, 69–81. https://doi.org/10.1016/j.dam.2018.02.018

Leskovec, J., & Sosič, R. (2016). SNAP: A general-purpose network analysis and graph-mining library. *ACM Trans. Intell. Syst. Technol.*, *8*(1). https://doi.org/10.1145/2898361

Matsakis, N. D., & Klock, F. S. (2014). The rust language. *Proceedings of the 2014 ACM SIGAda Annual Conference on High Integrity Language Technology*, 103–104. https://doi.org/10.1145/2663171.2663188

Peixoto, T. P. (2014). The graph-tool python library. *Figshare*. https://doi.org/10.6084/m9.figshare.1164194

Treinish, M., Gambetta, J., Rodríguez, D. M., Marques, M., Bello, L., Wood, C. J., Gomez, J., Nation, P., Chen, R., Winston, E., Gacon, J., Cross, A., Krsulich, K., Sertage, I. F., Wood, S., Alexander, T., Capelluto, L., Puente González, S. de la, Rubio, J., … Woerner, S. (2021). *Qiskit/qiskit-terra: Qiskit terra 0.18.3* (Version 0.18.3) [Computer software]. Zenodo. https://doi.org/10.5281/zenodo.2583252

Ullberg, S. (2021). Atompack: A flexible python library for atomic structure generation. In *GitHub repository*. GitHub. https://github.com/seatonullberg/atompack