

Rustworkx

Matthew Treinish

Senior Software Engineer - IBM Research

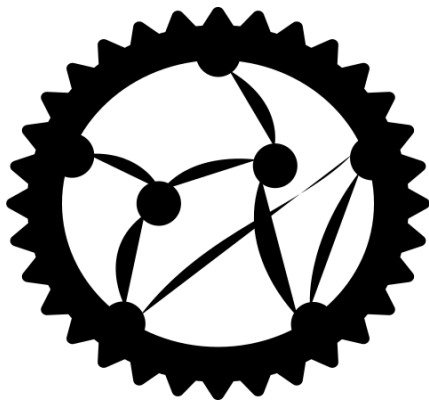
mtreinish@kortar.org

<https://github.com/mtreinish/rustwork-presentation>

June 18, 2024

What is Rustworkx?

- ▶ Open source graph library written in Rust
- ▶ Primarily a library for Python
- ▶ Designed for high performance and flexibility
- ▶ Apache 2.0 Licensed

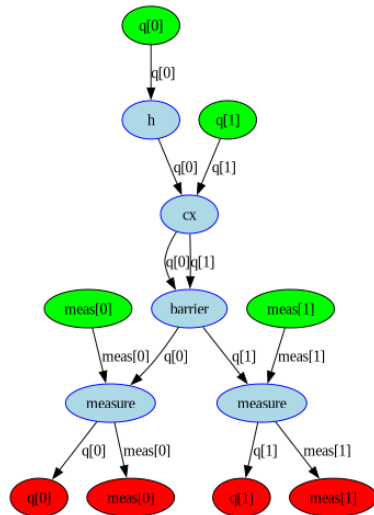


Unofficial logo, see:

<https://github.com/Qiskit/rustworkx/issues/824>

Origin of the project

- ▶ Started as a library to replace NetworkX usage for the Qiskit¹ compiler's DAG IR.
- ▶ Use case where a graph is built, analyzed, mutated, and analyzed again as part of compiler transforms.
- ▶ Designed to be flexible so that any Python object can be attached to a graph
- ▶ Expanded to cover other graph functionality



¹Javadi-Abhari, A., Treinish, M., Krsulich, K., Wood, C. J., Lishman, J., Gacon, J., ... & Gambetta, J. M. (2024). Quantum computing with Qiskit. arXiv preprint arXiv:2405.08810. <https://doi.org/10.48550/arXiv.2405.08810>

Why Rust?

- ▶ Rust is designed for safety with no runtime
- ▶ Compiler checks to make sure it's memory safe and has safe concurrency
- ▶ Performance of Rust code is equivalent to C or C++ but has the safety guarantees
- ▶ Rust packaging and build tooling make integration with Python simple



¹Matsakis, N. D., & Klock, F. S. (2014). The rust language. ACM SIGAda Ada Letters, 34(3), 103-104. <https://doi.org/10.1145/2692956.2663188>

Two Language Bindings

Python Library

- ▶ Primary Interface
- ▶ Python library for creating graphs
- ▶ Enables using any Python object for node and edge payloads
- ▶ Visualization functions for using [Matplotlib](#) and [Graphviz](#)
- ▶ Package available on [PyPI](#) for Linux (x86_64, i686, ppc64le, s390x, aarch64), Windows (32bit and 64bit), MacOS (x86_64, arm64).

Rust Library

- ▶ Rust language library that provides rust implementation of graph algorithms
- ▶ Currently built as an extension to [petgraph](#) library
- ▶ Pure rust library (no Python needed)
- ▶ Designed to be generic to work with with arbitrary data types
- ▶ Used to build the Python library

Rust interface

```
use rustworkx_core::petgraph;
use rustworkx_core::max_weight_matching::max_weight_matching;
use rustworkx_core::Result;

// Create a path graph
let g = petgraph::graph::UnGraph::<i32, i128>::from_edges(&[
    (1, 2, 5), (2, 3, 11), (3, 4, 5)
]);
// Run max weight matching with max cardinality set to true
let matching = max_weight_matching(&g, true, |e| Ok(*e.weight()), true);
```

Python Interface

```
import rustworkx as rx

# Create a path graph
g = rx.PyGraph()
g.extend_from_weighted_edge_list([(0, 1, 5), (1, 2, 11), (2, 3, 5)])
# Run max weight matching with max cardinality set to True
matching = rx.max_weight_matching(g, max_cardinality=True, weight_fn=int)
```

Using Rustworkx

- ▶ Two classes `PyGraph` and `PyDiGraph` for undirected and directed graphs respectively

```
import rustworkx as rx

graph = rx.PyGraph()
graph.extend_from_edge_list([(0, 1), (1, 2)])
assert graph.has_edge(1, 0)
digraph = rx.PyDiGraph()
digraph.extend_from_edge_list([(0, 1), (1, 2)])
assert not digraph.has_edge(1, 0)
```


Using Rustworkx

- ▶ Two classes PyGraph and PyDiGraph for undirected and directed graphs respectively
- ▶ All nodes and edges are addressed with integer indices and data payloads can be any Python object

```
import rustworkx as rx

graph = rx.PyDiGraph()
node_a = graph.add_node("my_node_a")
node_b = graph.add_node("my_node_b")
assert node_a == 0
assert node_b == 1
graph.add_edge(node_a, node_b, None)
assert "my_node_a" == graph[node_a]
assert "my_node_b" == graph[node_b]
```

Using Rustworkx

- ▶ Two classes `PyGraph` and `PyDiGraph` for undirected and directed graphs respectively
- ▶ All nodes and edges are addressed with integer indices and data payloads can be any Python object
- ▶ Graphs can be multigraphs (the default)

```
import rustworkx as rx

simple_graph = rx.PyGraph(multigraph=False)
multi_graph = rx.PyGraph()

simple_graph.extend_from_edge_list(
    [(0, 1), (1, 0)]
)
multi_graph.extend_from_edge_list(
    [(0, 1), (1, 0)]
)
assert simple_graph.num_edges() == 1
assert multi_graph.num_edges() == 2
```

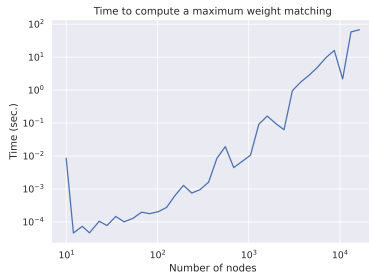
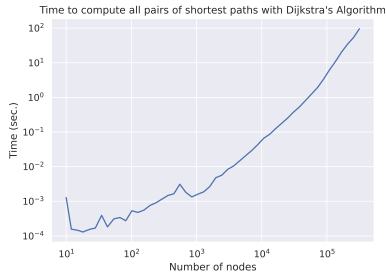
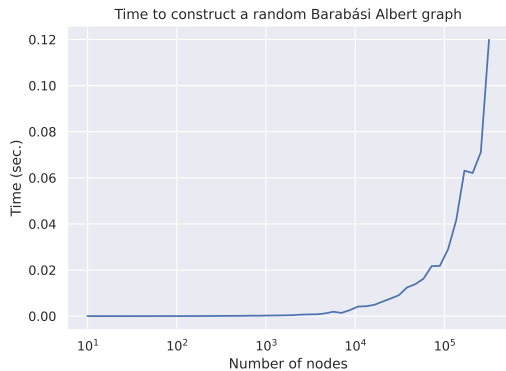
Using Rustworkx

- ▶ Two classes `PyGraph` and `PyDiGraph` for undirected and directed graphs respectively
- ▶ All nodes and edges are addressed with integer indices and data payloads can be any Python object
- ▶ Graphs can be multigraphs (the default)
- ▶ Callbacks are typically used to convert data payloads to static types for algorithms

```
import rustworkx as rx

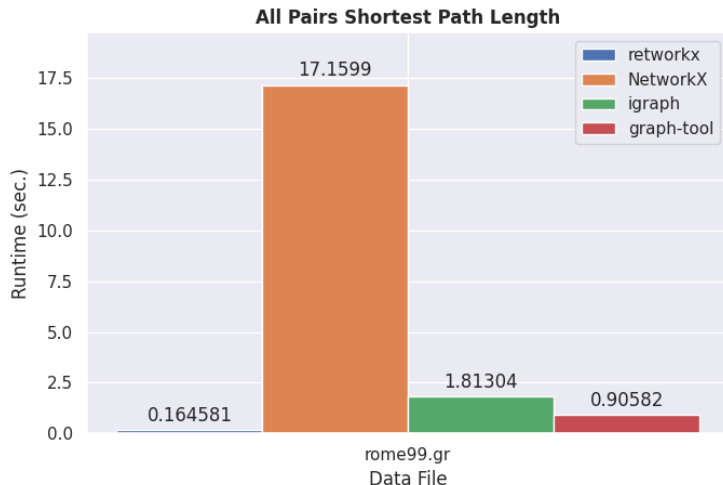
graph = rx.PyDiGraph()
graph.extend_from_weighted_edge_list(
    [
        (0, 1, {"weight": 1}),
        (0, 2, {"weight": 2}),
        (1, 3, {"weight": 2}),
        (3, 0, {"weight": 3}),
    ]
)
# Callback weight_fn to return float weight
dist_matrix = rx.floyd_warshall_numpy(
    graph, weight_fn=lambda edge: edge[weight]
)
```

Benchmarks with Random Barabási-Albert graphs²



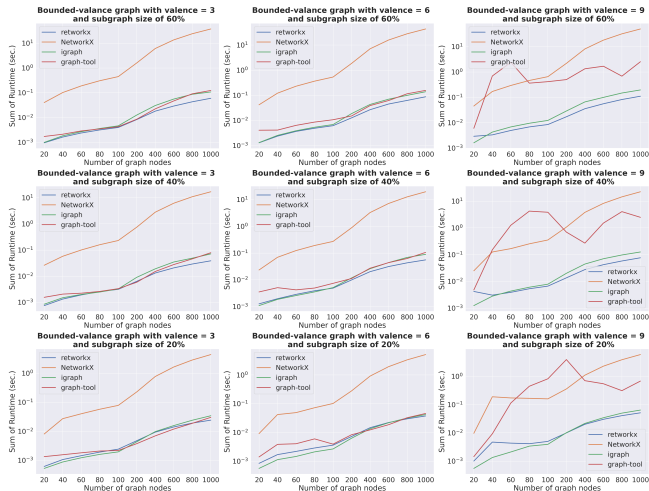
²Barabási, A. L., & Albert, R. (1999). Emergence of scaling in random networks. science,

All Pairs shortest path for Road Network of Rome, Italy in 1999³



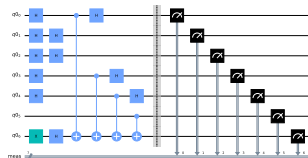
³Demetrescu, C., Goldberg, A., & Johnson, D. The Shortest Path Problem: Ninth DIMACS Implementation Challenge. <https://doi.org/10.1090/dimacs/074>

Subgraph isomorphism benchmarks using ARG database⁴

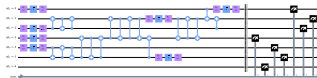


⁴Santo, M. D., Foggia, P., Sansone, C., & Vento, M. (2003). A large database of graphs and its use for benchmarking graph isomorphism algorithms. *Pattern Recognition Letters*, 24(8), 1067–1079. [https://doi.org/10.1016/S0167-8655\(02\)00253-2](https://doi.org/10.1016/S0167-8655(02)00253-2)

Subgraph Isomorphism based Quantum Circuit Layout⁵

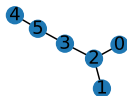


(a)

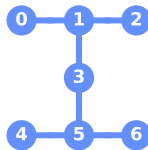


(b)

6



(c)



(d)

⁵Nation, P. D., & Treinish, M. (2023). Suppressing quantum circuit errors due to system variability. PRX Quantum, 4(1), 010327. <https://doi.org/10.1103/PRXQuantum.4.010327>

Subgraph Isomorphism based Quantum Circuit Layout⁵

```
import rustworkx as rx
# Build connectivity graph that models constraints of QPU
connectivity_graph = rx.generators.directed_heavy_hex_graph(9)
for index in connectivity_graph.node_indices():
    connectivity_graph[index] = {"rz", "sx", "x"}
for index in connectivity_graph.edge_indices():
    connectivity_graph.update_edge_by_index(index, {"cx"})
# Build interaction graph of circuit
interaction_graph = rx.PyDiGraph()
interaction_graph.add_nodes_from([set() for _ in range(dag.num_qubits)])
for node in dag.topological_sort():
    if len(node) == 2:
        if interaction_graph.has_edge(*node.qubits):
            interaction_graph.get_edge_data(*node.qubits).add(node.name)
        else:
            interaction_graph.add_edge(*node.qubits, {node.name,})
    else:
        interaction_graph[*node.qubits].add(node.name)
```

⁵Nation, P. D., & Treinish, M. (2023). Suppressing quantum circuit errors due to system variability. PRX Quantum, 4(1), 010327. <https://doi.org/10.1103/PRXQuantum.4.010327>

Subgraph Isomorphism based Quantum Circuit Layout⁶

```
def _target_match(a, b):  
    return a.issuperset(b)  
# Return an iterator of subgraphs mappings:  
subgraphs = vf2_mapping(  
    connectivity_graph,  
    interaction_graph,  
    node_matcher=_target_match,  
    edge_matcher=_target_match,  
    subgraph=True,  
    id_order=False,  
    induced=False,  
)  
best_layout = min(subgraphs, key=scoring_function)
```

Find best subgraph using VF2 with ordering heuristic from VF2++⁵

⁵Nation, P. D., & Treinish, M. (2023). Suppressing quantum circuit errors due to system variability. PRX Quantum, 4(1), 010327. <https://doi.org/10.1103/PRXQuantum.4.010327>

⁶Jüttner, 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>

Uses Outside of Quantum Computing

- ▶ Rautila, O. S., Kaivola, K., Rautila, H., Hokkanen, L., Launes, J., Strandberg, T. E., ... & Tienari, P. J. (2024). The shared ancestry between the C9orf72 hexanucleotide repeat expansion and intermediate-length alleles using haplotype sharing trees and HAPTK. *The American Journal of Human Genetics*, 111(2), 383-392. <https://doi.org/10.1016/j.ajhg.2023.12.019>
- ▶ Chen, R., Ding, Z., Zheng, S., Zhang, C., Leng, J., Liu, X., & Liang, Y. (2024, April). Magis: Memory optimization via coordinated graph transformation and scheduling for dnn. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, Volume 3 (pp. 607-621). <https://doi.org/10.1145/3620666.3651330>
- ▶ Tantos A, Kosmidis K. From Discourse Relations to Network Edges: A Network Theory Approach to Discourse Analysis. *Applied Sciences*. 2023; 13(12):6902. <https://doi.org/10.3390/app13126902>
- ▶ Caetano, J., Carriço, N., Figueira, J. R., & Covas, D. (2023). A novel methodology for pipe grouping and rehabilitation interventions scheduling in water distribution networks. *Urban Water Journal*, 20(7), 769–781. <https://doi.org/10.1080/1573062X.2023.2209560>
- ▶ Setiawan, T. H., Beltsazar, F., Aden, A., Gunawan, G., & Zarista, R. H. (2023). Graph coloring for determining courier frequency. *Desimal: Jurnal Matematika*, 6(3), 273 - 284.

Where to get more information

- ▶ These Slides: <https://github.com/mtreinish/rustwork-presentation>
- ▶ Rustworkx Documentation: <https://www.rustworkx.org>
- ▶ Tutorial for NetworkX users: <https://www.rustworkx.org/networkx.html>
- ▶ Rustworkx on Github: <https://github.com/Qiskit/rustworkx>
- ▶ JOSS Paper: <https://joss.theoj.org/papers/10.21105/joss.03968>