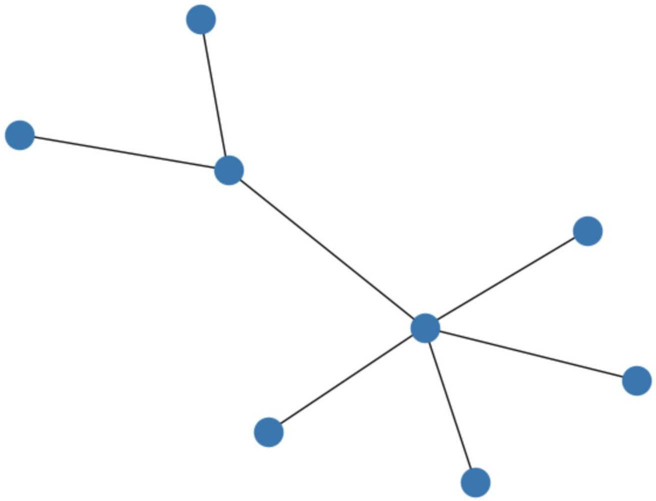
Accelerating NetworkX:
The Future of Easy Graph Analytics

Mridul Seth - Core Developer, NetworkX
Erik Welch - Sr. System Software Engineer, NVIDIA
Rick Ratzel - Sr. System Software Engineer, NVIDIA
NetworkX and backend developers

Why do we care about Graphs?



Do you have data?

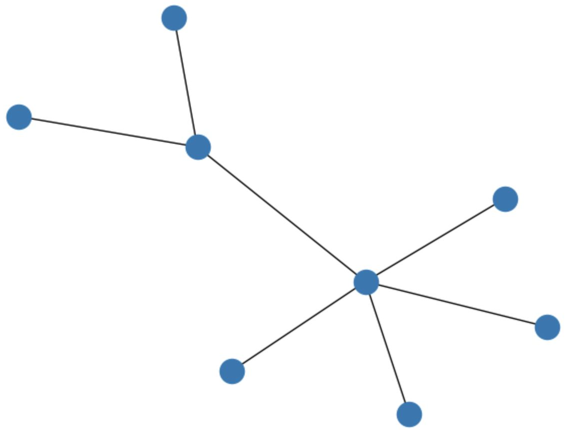
Does your data have *relationships*? (hint: probably!)

Do you want to gain insights from your data?

- Most “important”, “central” or “unusual” entities
- Detect communities and cliques
- Fraud detection
- Recommendation systems
  - e.g., movie, product, or LinkedIn recommendation
- Connectivity, shortest paths, route planning
- Much, much more (including GNNs)
- Used by virtually all science and engineering fields!

What do you need to work with graphs?

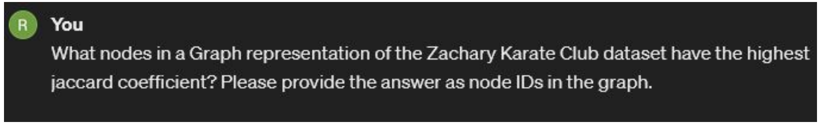
- A good graph analytics library should be:
    - Easy to use
    - Well-documented
    - Robust, reliable, and well-tested
    - Popular, with a large, friendly community backing it
    - Able to support a wide breadth of algorithms
        - ...and plotting would also be very nice
    - Easy to install without exotic dependencies
    - Permissive to use with and open-source license
    - Performant



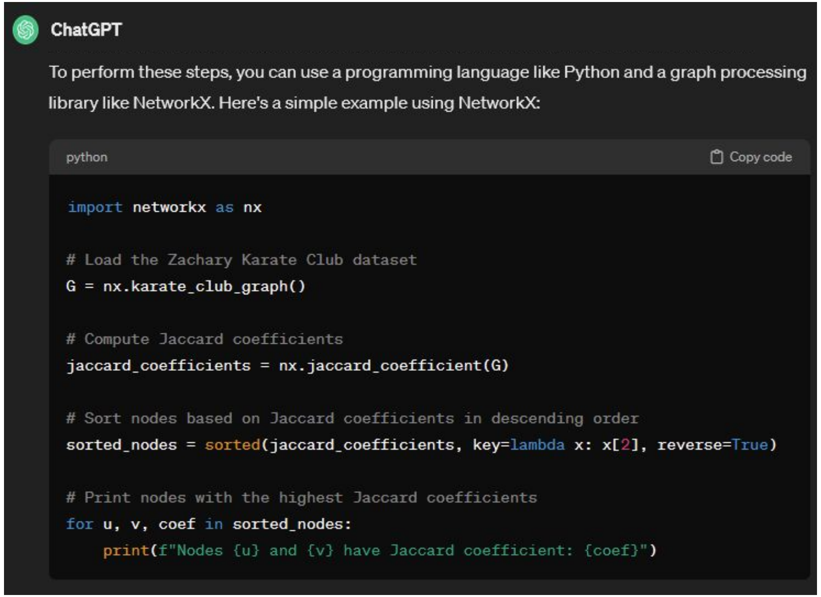
…this describes NetworkX

NetworkX
Network Analysis in Python

- Considered the de facto standard



You
What nodes in a Graph representation of the Zachary Karate Club dataset have the highest jaccard coefficient? Please provide the answer as node IDs in the graph.



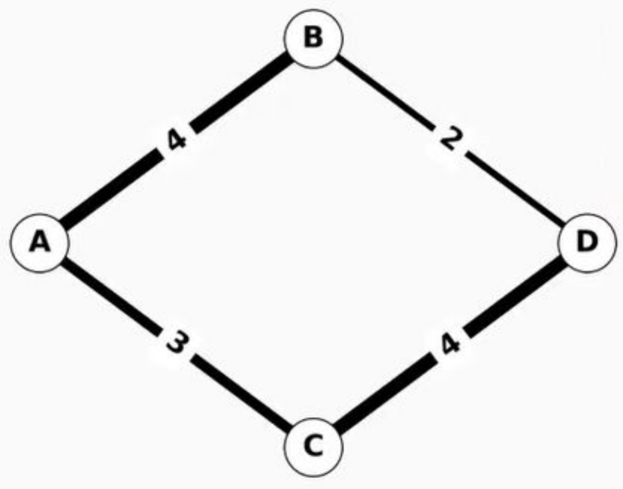
ChatGPT
To perform these steps, you can use a programming language like Python and a graph processing library like NetworkX. Here's a simple example using NetworkX:

```python
import networkx as nx

# Load the Zachary Karate Club dataset
G = nx.karate_club_graph()

# Compute Jaccard coefficients
jaccard_coefficients = nx.jaccard_coefficient(G)

# Sort nodes based on Jaccard coefficients in descending order
sorted_nodes = sorted(jaccard_coefficients, key=lambda x: x[2], reverse=True)

# Print nodes with the highest Jaccard coefficients
for u, v, coef in sorted_nodes:
    print(f"Nodes {u} and {v} have Jaccard coefficient: {coef}")
```

- Easy to use - API supports familiar usage and flexible types to accommodate real-world data
- Pure python, runs anywhere python is supported
- Easy install provided by all major package managers
- Well documented: excellent API docs, examples, notebooks
- Permissive open source (BSD license) and community governed
- Mature, battle-tested, and bug-vetted
  - First public release was 19 years ago!
- Very popular with a large, friendly community
  - ~14K stars and 3K forks on GitHub
  - ~47M PyPI and ~360k conda downloads per month
- Hundreds of graph algorithms and functions
  - Exploring rich graph data is rarely limited by the algorithms available

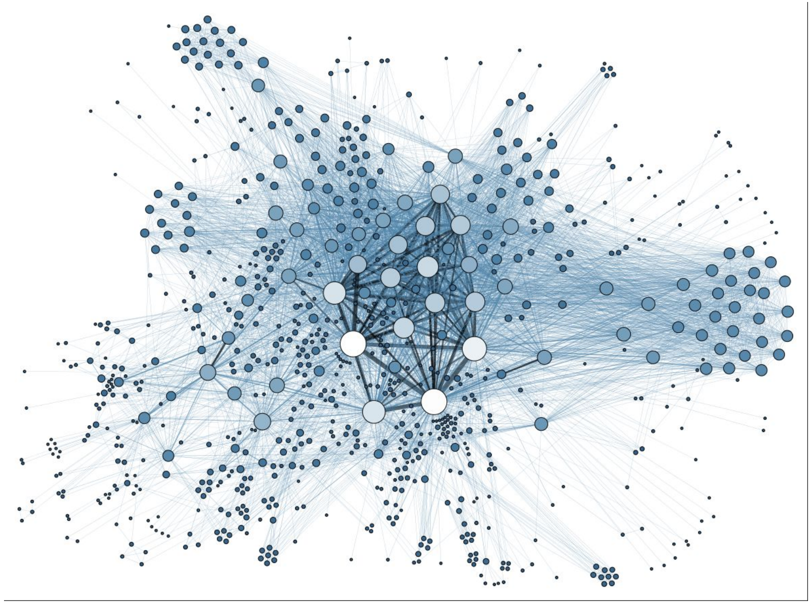
In NetworkX, simple things stay simple

```
>>> import networkx as nx

>>> G = nx.Graph()
>>> G.add_edge("A", "B", my_cost=4)
>>> G.add_edge("B", "D", my_cost=2)
>>> G.add_edge("A", "C", my_cost=3)
>>> G.add_edge("C", "D", my_cost=4)

>>> nx.shortest_path(G, "A", "D", weight="my_cost")
['A', 'B', 'D']
```



…but what about *less simple* things?

"Um, did you say 'pure python'"?



- NetworkX is a data tool… and data is getting bigger!
- Pure python dictionary-of-dictionaries implementation does not scale well to larger problems
  - Maximum reasonable graph size varies by algo and options used
    - Eg. `nx.betweenness_centrality()` on 3.7M nodes, 16.5M edge graph, k=500, takes approx. 80 minutes*
  - Single Processor bound to CPU

* NetworkX 3.2 on Intel(R) Xeon(R) Gold 6128 CPU @ 3.40GHz, 45GB RAM

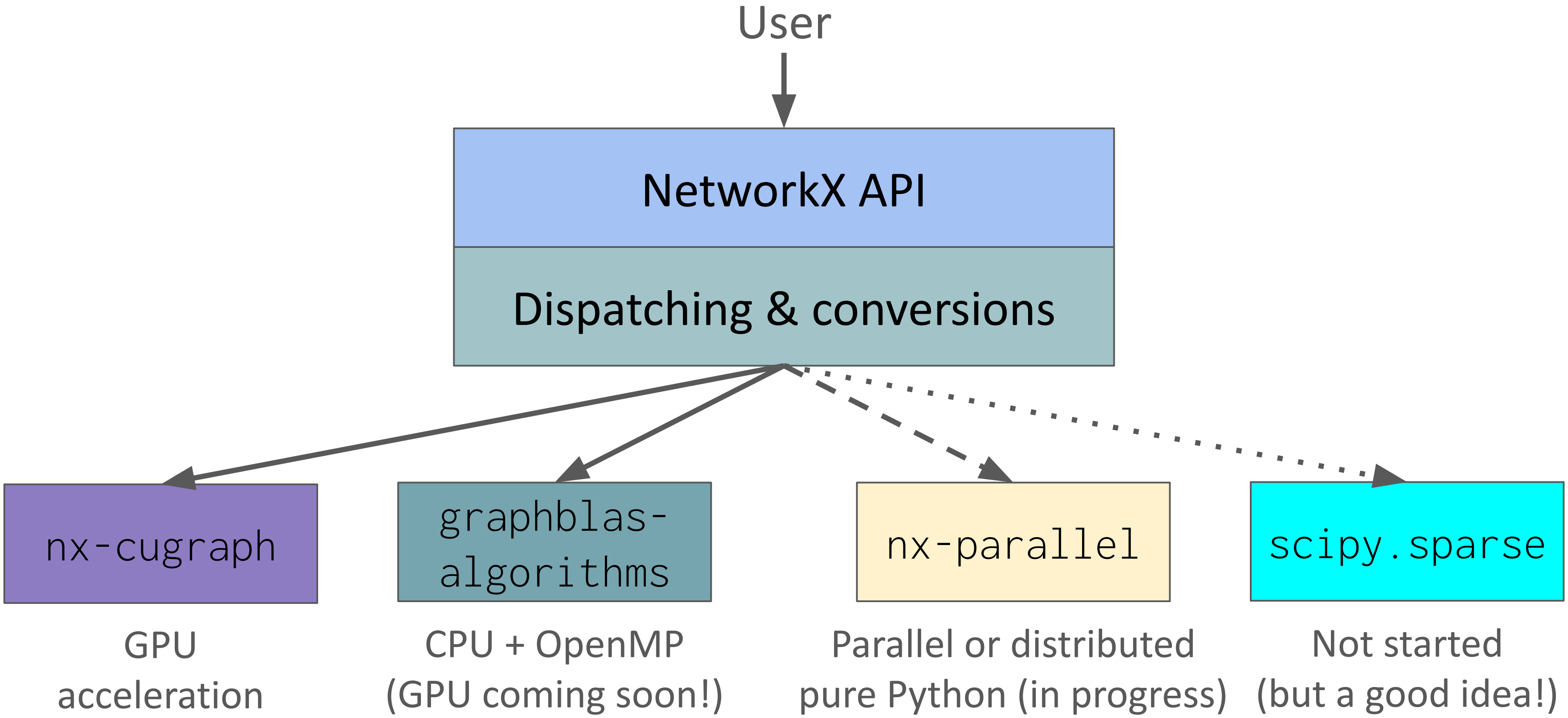
How do we fix this without losing all the good things about NetworkX?

Let’s rewrite everything in Rust.

~~Let’s rewrite everything in Rust.~~

Dispatching.

NetworkX as a user-facing API + pluggable backends



User

NetworkX API

Dispatching & conversions

`nx-cugraph`

GPU acceleration

`graphblas-algorithms`

CPU + OpenMP (GPU coming soon!)

`nx-parallel`

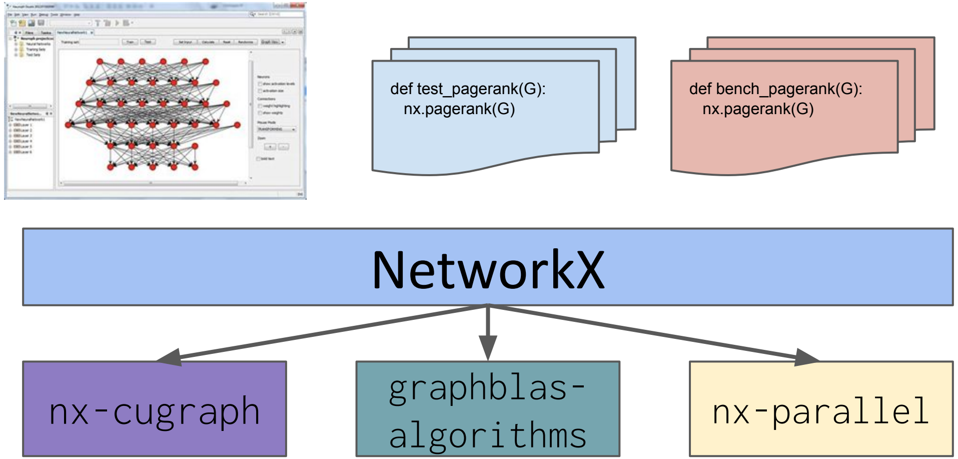
Parallel or distributed pure Python (in progress)

`scipy.sparse`

Not started (but a good idea!)

Some other benefits of dispatching to backends

- NetworkX becomes a “standard” graph API
  - Learn a single library, scale as needed with backends.
- Share your graph algorithms with the NetworkX community - write your own backend



```
def test_pagerank(G):
  nx.pagerank(G)
```

```
def bench_pagerank(G):
  nx.pagerank(G)
```

NetworkX

nx-cugraph

graphblas-algorithms

nx-parallel

  - Don’t pay the open-source tax
    - Good OSS needs good docs, examples, benchmarks, etc. - these aren’t cheap!
  - Backends leverage the NetworkX API and everything written to it
    - Run against the NetworkX test suite - tests for free!
    - Reference your backend in the NetworkX docs - docs for free!
    - Run with NetworkX benchmarks - benchmarks for free!

This is GTC! Tell me more about that GPU-accelerated NetworkX backend!

- **nx-cugraph** : NetworkX backend using RAPIDS cuGraph
- No-code-change GPU acceleration
- Run on GPU if available, fallback to CPU if not

bc_demo.ipy

```
import pandas as pd
import networkx as nx

url = "https://data.rapids.ai/cugraph/datasets/cit-Patents.csv"
df = pd.read_csv(url, sep=" ", names=["src", "dst"], dtype="int32")
G = nx.from_pandas_edgelist(df, source="src", target="dst")

%time result = nx.betweenness_centrality(G, k=10)
```

```
user@machine:/# ipython bc_demo.ipy
CPU times: user 7min 38s, sys: 5.6 s, total: 7min 44s
Wall time: 7min 44s

user@machine:/# NETWORKX_BACKEND_PRIORITY=cugraph ipython bc_demo.ipy
CPU times: user 18.4 s, sys: 1.44 s, total: 19.9 s
Wall time: 20 s
```

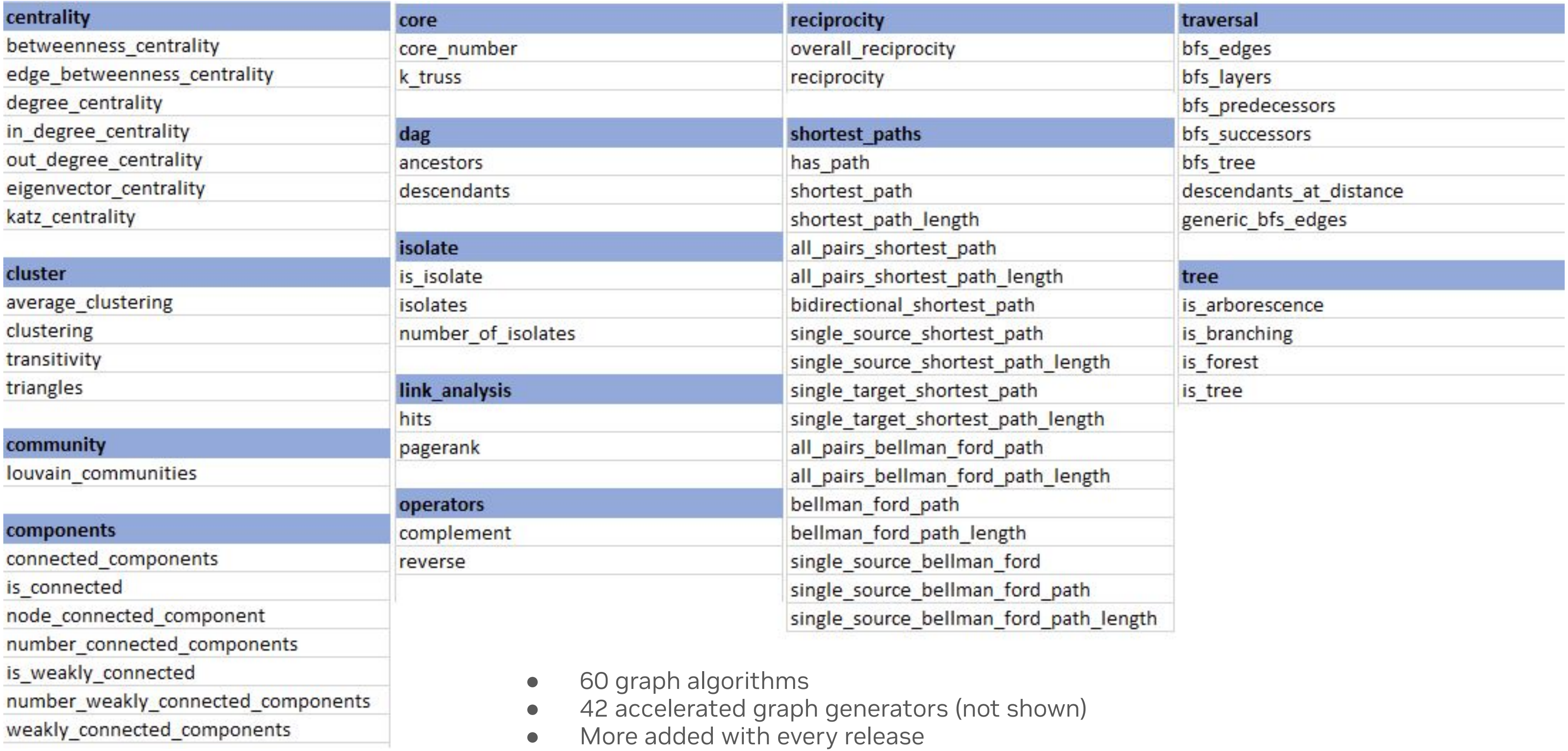
NetworkX 3.2, CPU: Intel(R) Xeon(R) Gold 6128 CPU @ 3.40GHz 45GB RAM, GPU: NVIDIA Quadro RTX 8000 80GB RAM

nx-cugraph – GPU-accelerated NetworkX

- Included in the RAPIDS cuGraph github repo
  - https://github.com/rapidsai/cugraph/tree/branch-24.04/python/nx-cugraph
  - Leverages RAPIDS' established release process and CI infrastructure
  - **nx-cugraph CI runs NetworkX unit tests using the nx-cugraph backend**

- Install instructions
  - conda: `conda install -c rapidsai -c conda-forge -c nvidia nx-cugraph`
  - pip: `pip install --extra-index-url=https://pypi.nvidia.com nx-cugraph-cu12`
  - from source: https://github.com/rapidsai/cugraph
  - for more information: https://rapids.ai/#quick-start

nx-cugraph – supported algorithms



centrality
- betweenness_centrality
- edge_betweenness_centrality
- degree_centrality
- in_degree_centrality
- out_degree_centrality
- eigenvector_centrality
- katz_centrality

cluster
- average_clustering
- clustering
- transitivity
- triangles

community
- louvain_communities

components
- connected_components
- is_connected
- node_connected_component
- number_connected_components
- is_weakly_connected
- number_weakly_connected_components
- weakly_connected_components

core
- core_number
- k_truss

dag
- ancestors
- descendants

isolate
- is_isolate
- isolates
- number_of_isolates

link_analysis
- hits
- pagerank

operators
- complement
- reverse

reciprocity
- overall_reciprocity
- reciprocity

shortest_paths
- has_path
- shortest_path
- shortest_path_length
- all_pairs_shortest_path
- all_pairs_shortest_path_length
- bidirectional_shortest_path
- single_source_shortest_path
- single_source_shortest_path_length
- single_target_shortest_path
- single_target_shortest_path_length
- all_pairs_bellman_ford_path
- all_pairs_bellman_ford_path_length
- bellman_ford_path
- bellman_ford_path_length
- single_source_bellman_ford
- single_source_bellman_ford_path
- single_source_bellman_ford_path_length

traversal
- bfs_edges
- bfs_layers
- bfs_predecessors
- bfs_successors
- bfs_tree
- descendants_at_distance
- generic_bfs_edges

tree
- is_arborescence
- is_branching
- is_forest
- is_tree

- 60 graph algorithms
- 42 accelerated graph generators (not shown)
- More added with every release

nx-cugraph benchmarks - betweenness_centrality

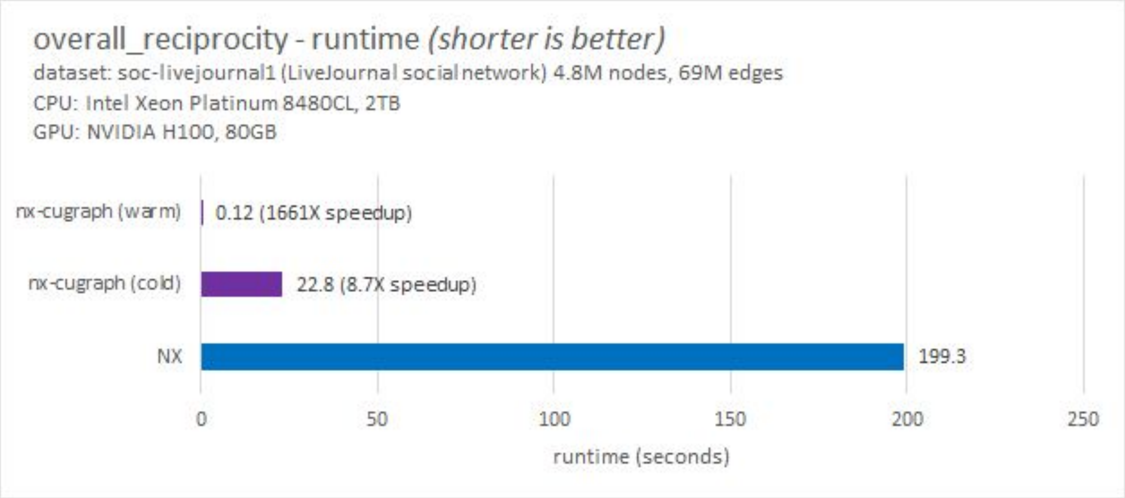
Dataset: U.S. patent dataset, National Bureau of Economic Research (https://snap.stanford.edu/data/cit-Patents.html) : directed graph, **3.7M nodes, 16.5M edges**
CPU: Intel(R) Xeon(R) Gold 6128 CPU @ 3.40GHz, 45GB
GPU: NVIDIA Quadro RTX 8000, 48GB

```
bash $> NETWORKX_BACKEND_PRIORITY=”cugraph” python
>>> import networkx as nx
>>> G = nx.DiGraph()
…
>>> nx.betweenness_centrality(G, k=k)
```

	k=10	k=20	k=50	k=100	k=500	k=1000
NetworkX	100.6 s	206.01 s	494.78 s	1007.71 s	5132.08 s	10233.25 s
NetworkX + nx-cugraph (**cold** graph conversion cache)	10.22 s	10.28 s	10.81 s	11.45 s	16.43 s	23.07 s
speedup	*9.84 X*	*20.04 X*	*45.77 X*	*88.0 X*	*312.36 X*	*443.57 X*
NetworkX + nx-cugraph (**warm** graph conversion cache)	1.16 s	1.19 s	1.65 s	2.19 s	7.24 s	13.57 s
speedup	*86.7 X*	*173.11 X*	*299.87 X*	*460.14 X*	*708.85 X*	*754.11 X*

https://github.com/rapidsai/cugraph/blob/branch-24.04/benchmarks/nx-cugraph/pytest-based/bench_algos.py

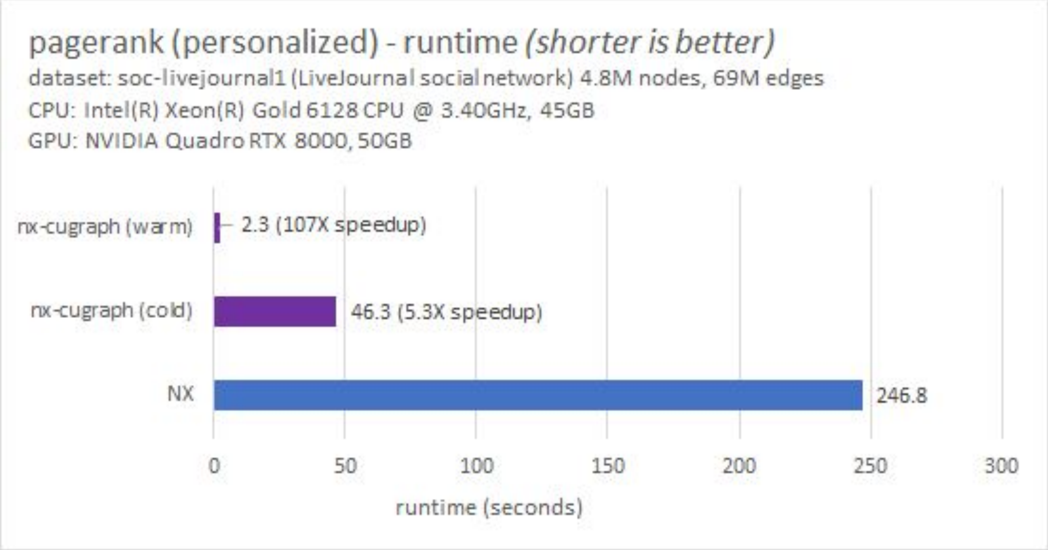
nx-cugraph benchmarks


overall_reciprocity - runtime *(shorter is better)*

dataset: soc-livejournal1 (LiveJournal social network) 4.8M nodes, 69M edges

CPU: Intel Xeon Platinum 8480CL, 2TB

GPU: NVIDIA H100, 80GB

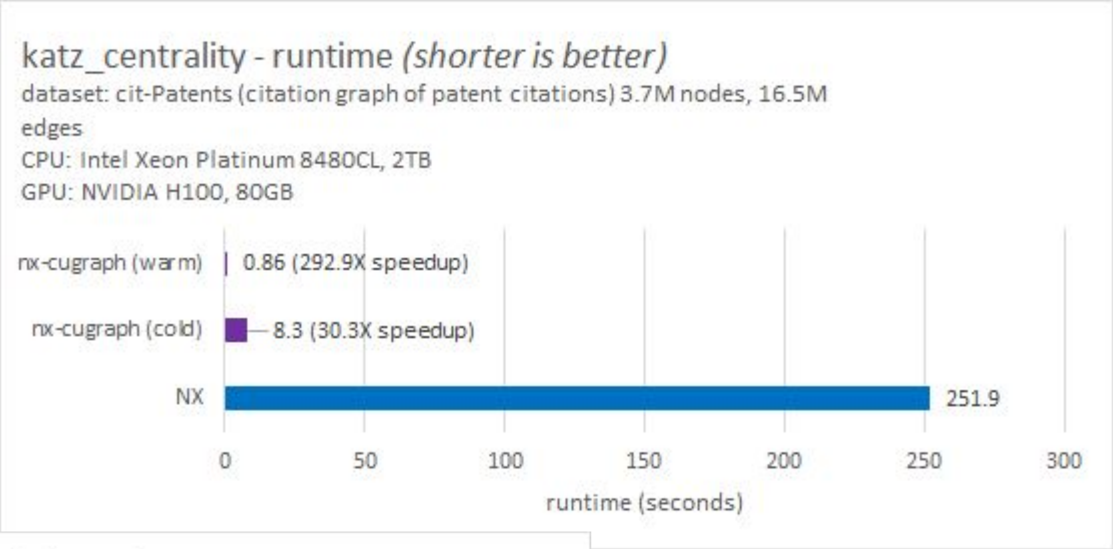

pagerank (personalized) - runtime *(shorter is better)*

dataset: soc-livejournal1 (LiveJournal social network) 4.8M nodes, 69M edges

CPU: Intel(R) Xeon(R) Gold 6128 CPU @ 3.40GHz, 45GB

GPU: NVIDIA Quadro RTX 8000, 50GB

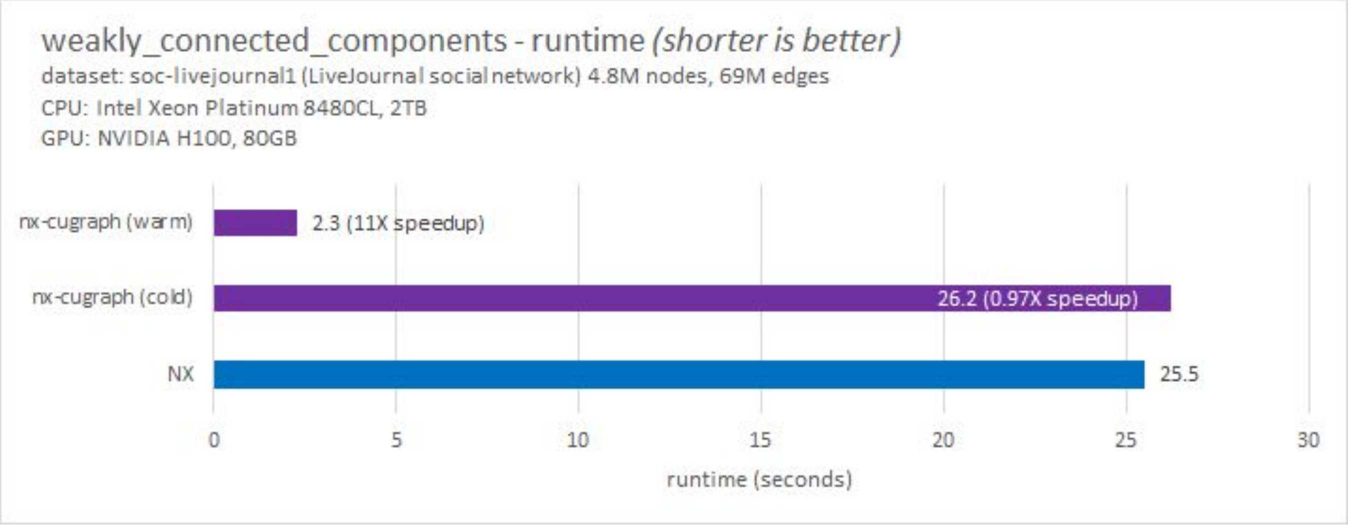
nx-cugraph benchmarks


katz_centrality - runtime *(shorter is better)*

dataset: cit-Patents (citation graph of patent citations) 3.7M nodes, 16.5M edges

CPU: Intel Xeon Platinum 8480CL, 2TB

GPU: NVIDIA H100, 80GB


weakly_connected_components - runtime *(shorter is better)*

dataset: soc-livejournal1 (LiveJournal social network) 4.8M nodes, 69M edges

CPU: Intel Xeon Platinum 8480CL, 2TB

GPU: NVIDIA H100, 80GB

Next Steps

Something not here you'd like to see?

Leave a suggestion…or a pull-request!

https://github.com/networkx/networkx
https://github.com/rapidsai/cugraph

NetworkX dispatching

- Configuration API
- Introspection and Logging
- More dispatchable APIs (plotting? 3rd-party-only APIs?)

nx-cugraph

- Many more algorithms *(suggestions welcome)*
- Multi-GPU support

Thank You!

BEGIN EXTRA SLIDES

nx-cugraph benchmarks - betweenness_centrality

Dataset: U.S. patent dataset maintained by the National Bureau of Economic Research (https://snap.stanford.edu/data/cit-Patents.html) : directed graph, 3.7M nodes, 16.5M edges

CPU: Intel(R) Xeon(R) Platinum 8480CL, 2TB

GPU: NVIDIA H100, 80GB

```
bash $> NETWORKX_AUTOMATIC_BACKENDS=”cugraph” python
>>> import networkx as nx
>>> G = nx.DiGraph()


...

>>> nx.betweenness_centrality(G, k=k)
```

	k=10	k=20	k=50	k=100	k=500	k=1000
NetworkX	97.28 s	184.77 s	463.15 s	915.84 s	4,585.96 s	9,125.48 s
nx-cugraph	8.71 s	8.26 s	8.91 s	8.67 s	11.31 s	14.37 s
speedup	11.17 X	22.37 X	51.96 X	105.58 X	405.59 X	634.99 X

https://github.com/rapidsai/cugraph/blob/branch-24.04/benchmarks/nx-cugraph/pytest-based/bench_algos.py