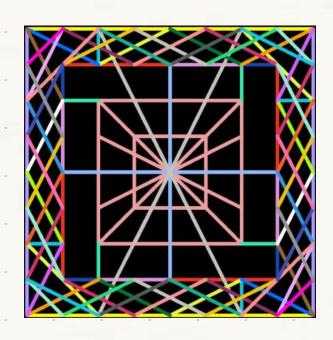


Understanding API Dispatching in NetworkX



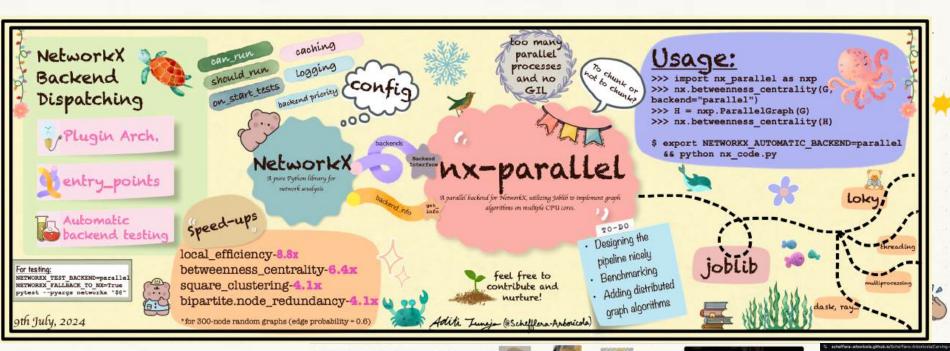
About Me

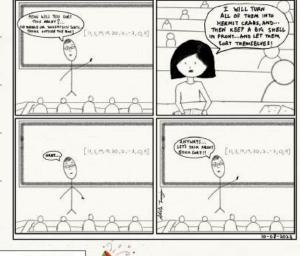
- · Aditi Juneja (@Schefflera-Arboricola)
- Worked mostly around API Dispatching stuff and the nx-parallel backend
- · Network X's core developer



And...here is some of my stuff...







Google Summer of Code 2024 Final Report

Project Title - Beyinting and expanding me-paratiel Organisation - NumPCCUS(NotworkX: nx-paratiel) Mentors - Dan Schult, Mrisul Seth Contributor - Adb Junea.

1.Abstra

This report distalls the anche conceptioned 2005, Society on the re-peated project, a preside inversigne (2005). For sourcing grash algorithms are project inversional residentially fair facilities and compared to the control of t

The contributions made studing this perior and this report, along with my blogs, will serve as sondraying this work.

2.Background

The margar allied package is a backened the execution of graph algorithms through parallel and a supply of the control of the package.



Carving out a path in a garden

Pull/listed on Says 11, 2023

There is a beautiful garden composed of several smaller flower gardens. It's soor going to open to the general public. Before that, the gardener wants to carve out path and pove it with concrete, allowing visitors to strall around the garden. He w to put in the least effort and time, while also minimizing the damage to the groun and flowers. He also wants all the flower gardens' centers to be accessible from nonther.









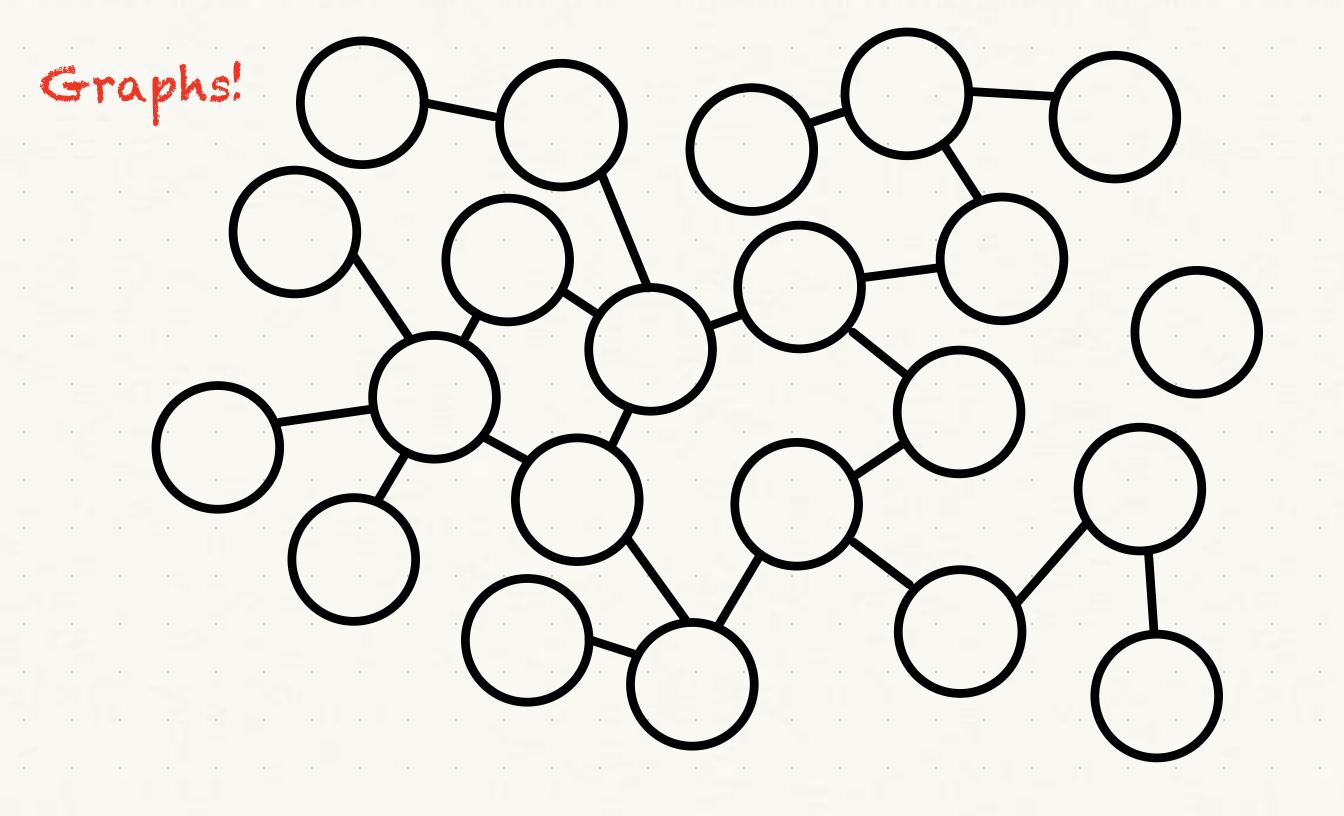
Understanding API Dispatching in NetworkX

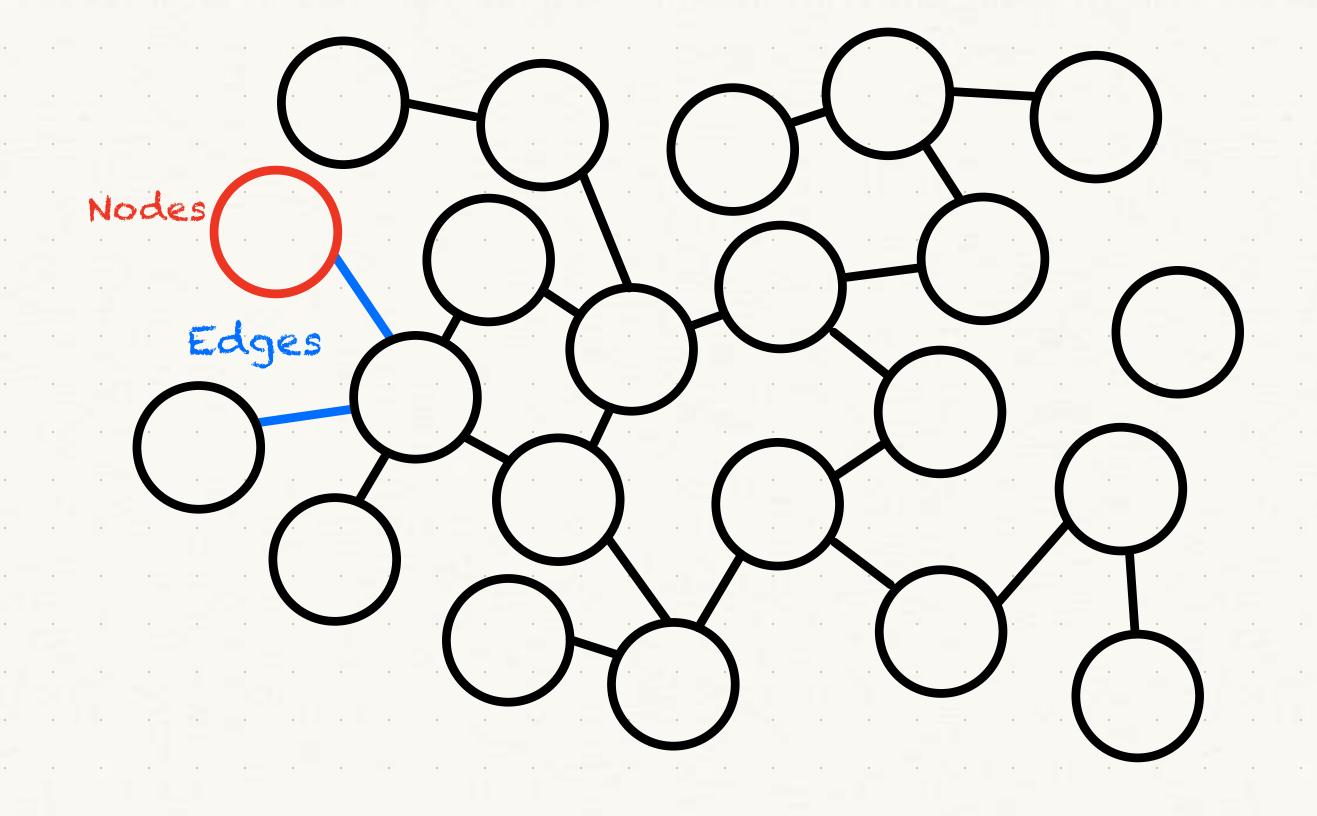


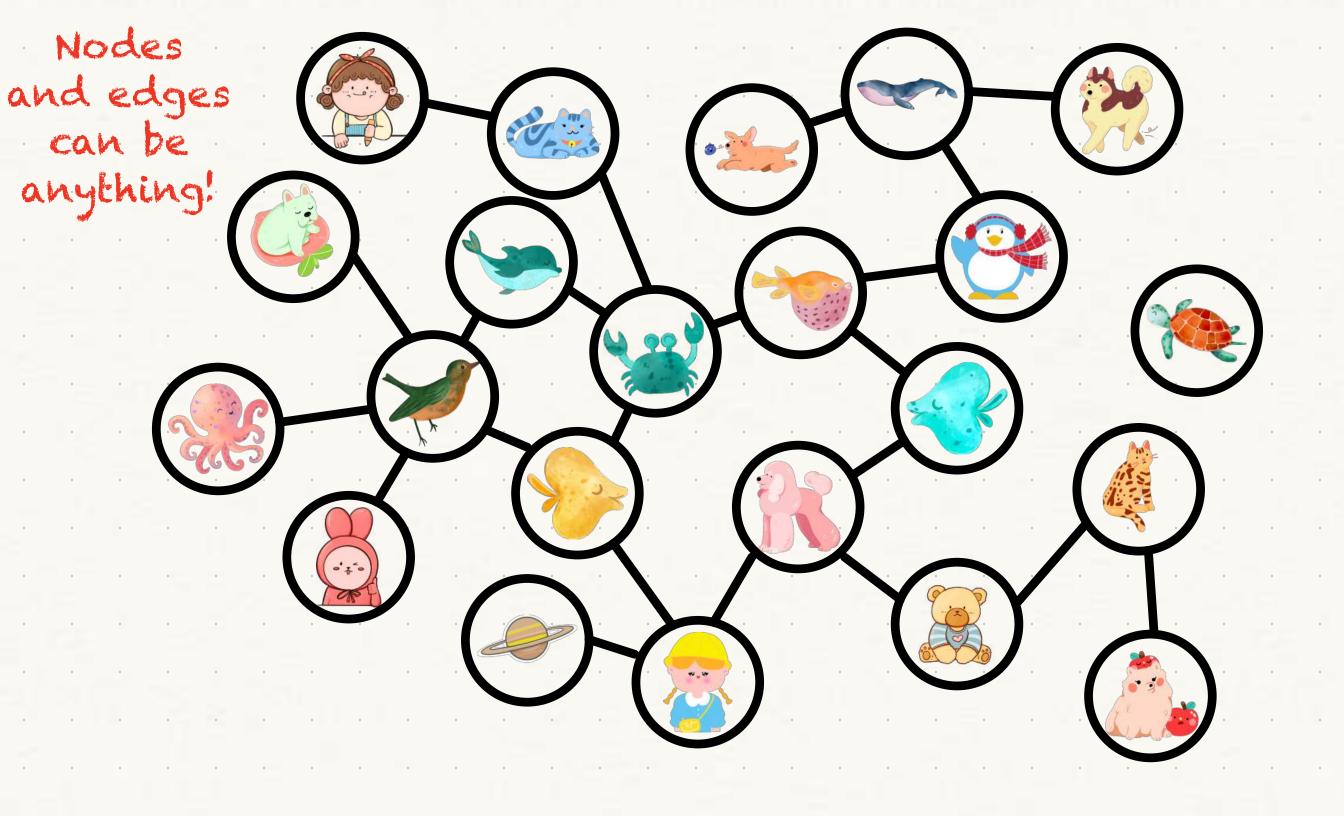
NetworkX is a **graph** (aka network) analysis
Python library

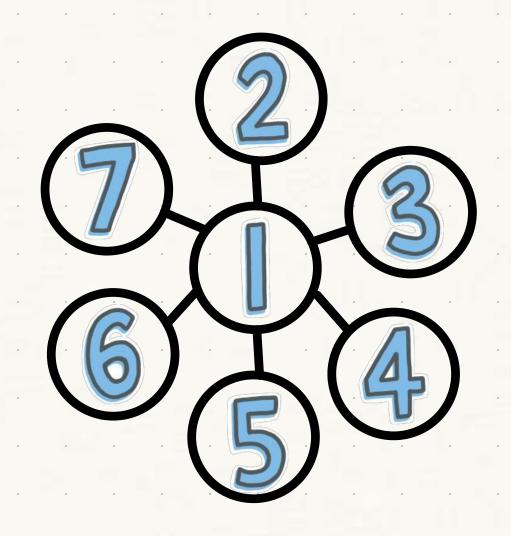


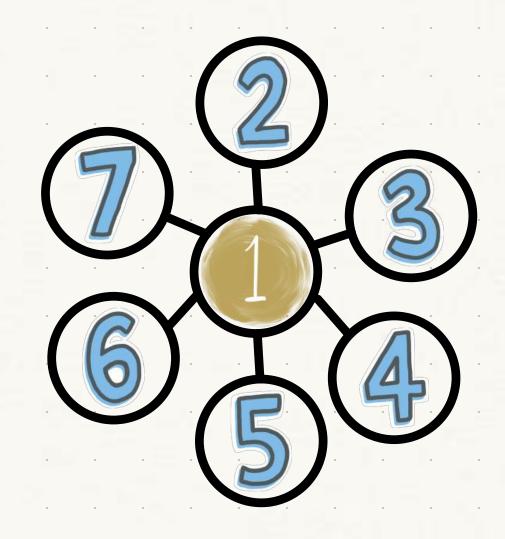
NOT these kind of graphs!

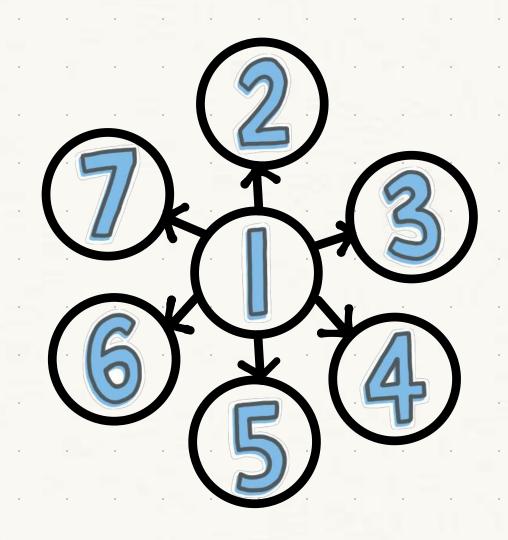


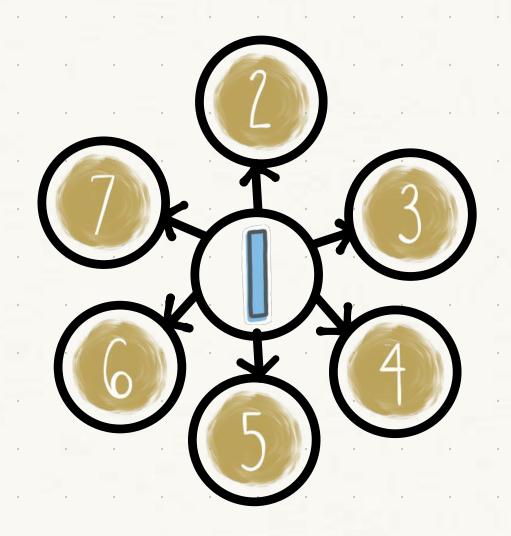


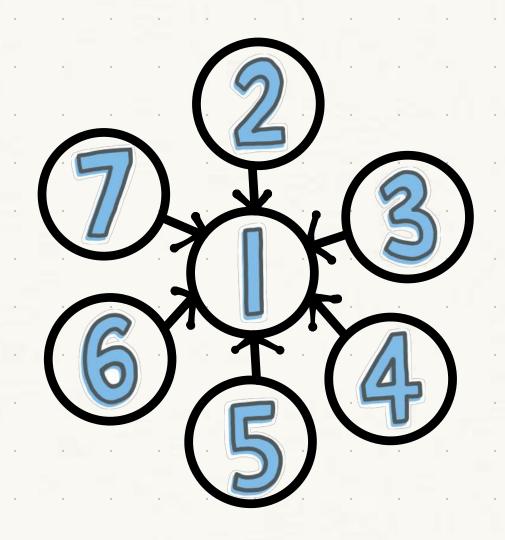


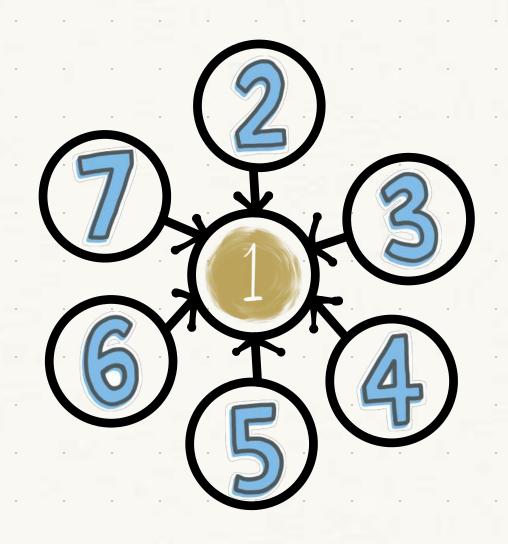












But, what is this "importance"?

betweenness_centrality

pagerank

degree

closeness_centrality

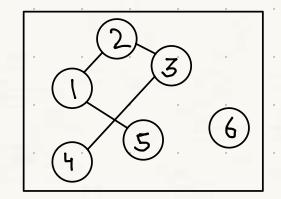
eigenvector_centrality

square_clustering

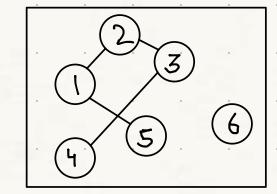
But, graph analysis is more than this...



```
>>> import networkx as nx
>>> G = nx.Graph()
>>> G.add_edges_from([(1,2), (3,4), (2,3), (5,1)])
>>> G.add_node(6)
>>> nx.betweenness_centrality(G)
```



```
>>> import networkx as nx
>>> G = nx.Graph()
>>> G.add_edges_from([(1,2), (3,4), (2,3),
 (5,1)])
>>> G.add_node(6)
>>> nx.betweenness_centrality(G)
{1: 0.3000000000000004, 2: 0.4, 3:
 0.300000000000004, 4: 0.0, 5: 0.0, 6: 0.0}
```



Lets try this...



```
>>> import networkx as nx
>>> G = nx.Graph()
>>> G.add_edges_from([(1,2), (3,4), (2,3),
 (5,1)])
>>> G.add_node(6)
>>> nx.betweenness_centrality(G)
>>> G = nx.fast_gnp_random_graph(1000000, 0.5)
>>> nx.betweenness_centrality(G)
```

```
>>> import networkx as nx
>>> G = nx.Graph()
>>> G.add_edges_from([(1,2), (3,4), (2,3),
 (5,1)])
>>> G.add_node(6)
>>> nx.betweenness_centrality(G)
>>> G = nx.fast_gnp_random_graph(1000000, 0.5)
>>> nx.betweenness_centrality(G)
 ...takes forever 😩 😩 😩 😭 😭
```

```
>>> import networkx as nx
>>> G = nx.Graph()
>>> G.add_edges_from([(1,2), (3,4), (2,3),
 (5,1)])
>>> G.add_node(6)
>>> nx.betweenness_centrality(G)
>>> G = nx.fast_gnp_random_graph(1000000, 0.5)
>>> nx.betweenness_centrality(G)
 ...takes forever 答 答 答 😭 😭
```





Understanding API Dispatching in NetworkX



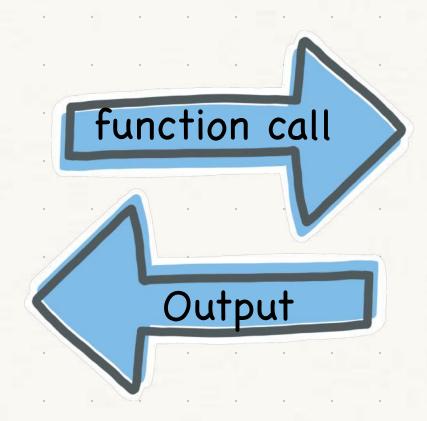
Understanding API Dispatching in NetworkX



What is Dispatching?

Usually...



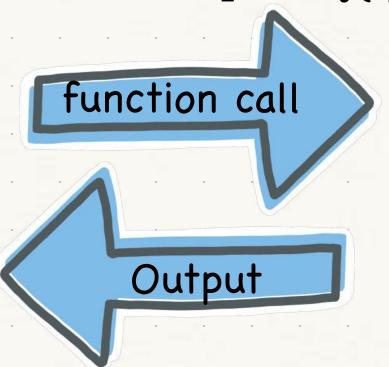


The main Library

Usually...



nx.betweenness_centrality(G)

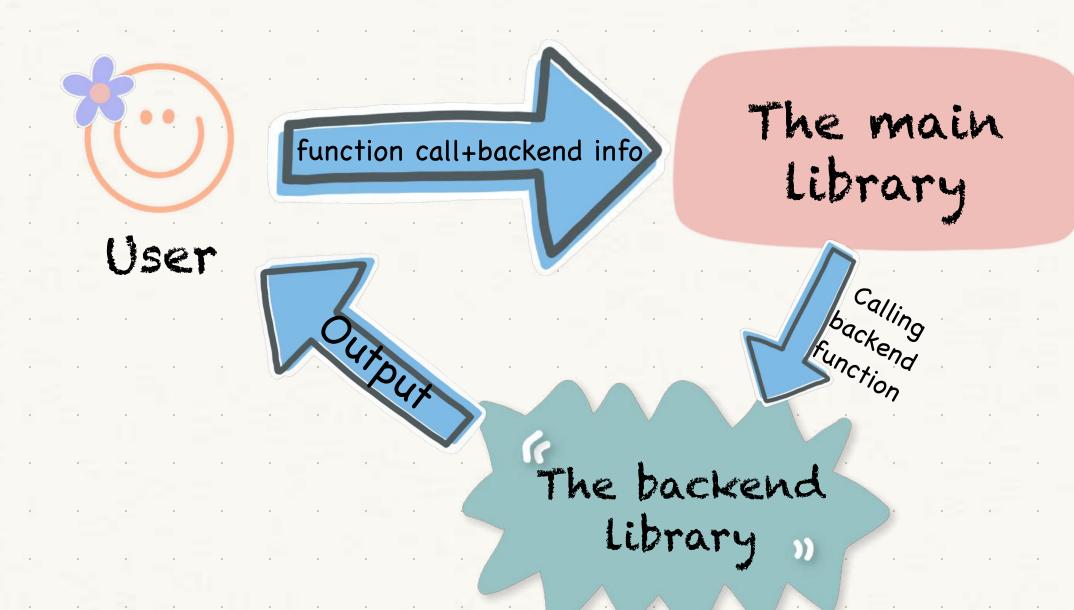


The main Library

{1: 0.3000000000000004, 2: 0.4, 3:

0.3000000000000004, 4: 0.0, 5: 0.0, 6: 0.0}

Dispatching



Dispatching



function call+backend info

The main library

Calling backend Function

8

Here, backend info can be inside the function call or stored globally as an environment variable, or in the Graph object itself. The backend library

Dispatching in NetworkX

nx-parallel

def betweenness_centrality(G, ...):

nx.betweenness_centrality(G, backend="parallel")-

nx.betweenness_centrality(cug)

with nx.config(backend_priority=[graphblas,]):
nx.betweenness_centrality(G) = = = =

NetworkX

@_dispatchable 🛥🗗

def betweenness_centrality(G, ...):

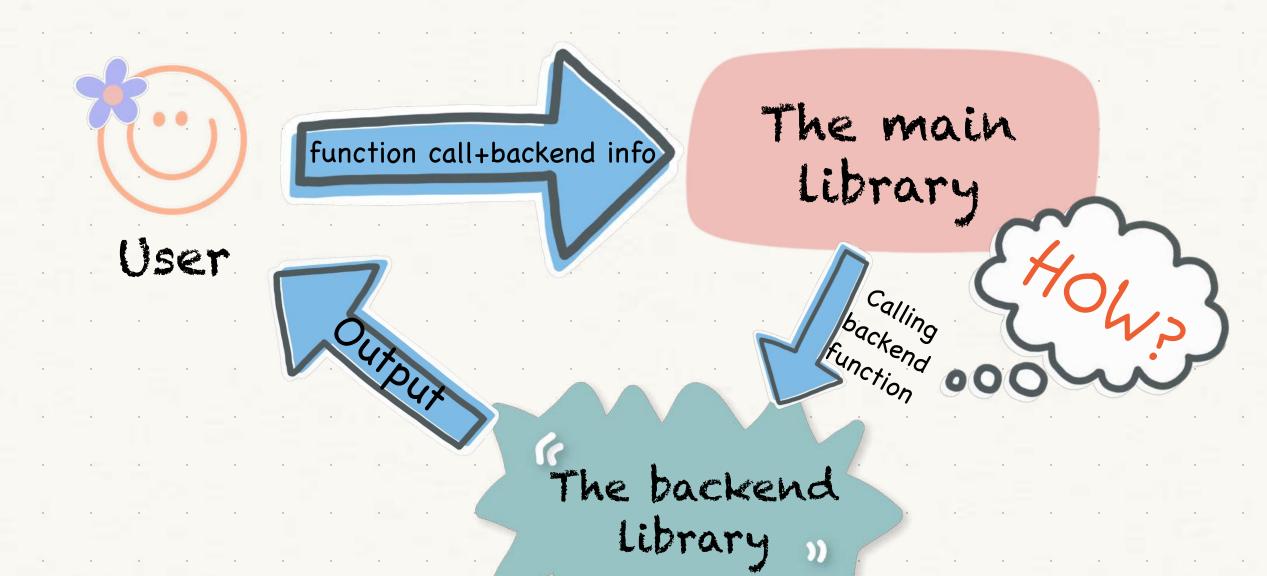
nx-cugraph

def betweenness_centrality(G, ...):

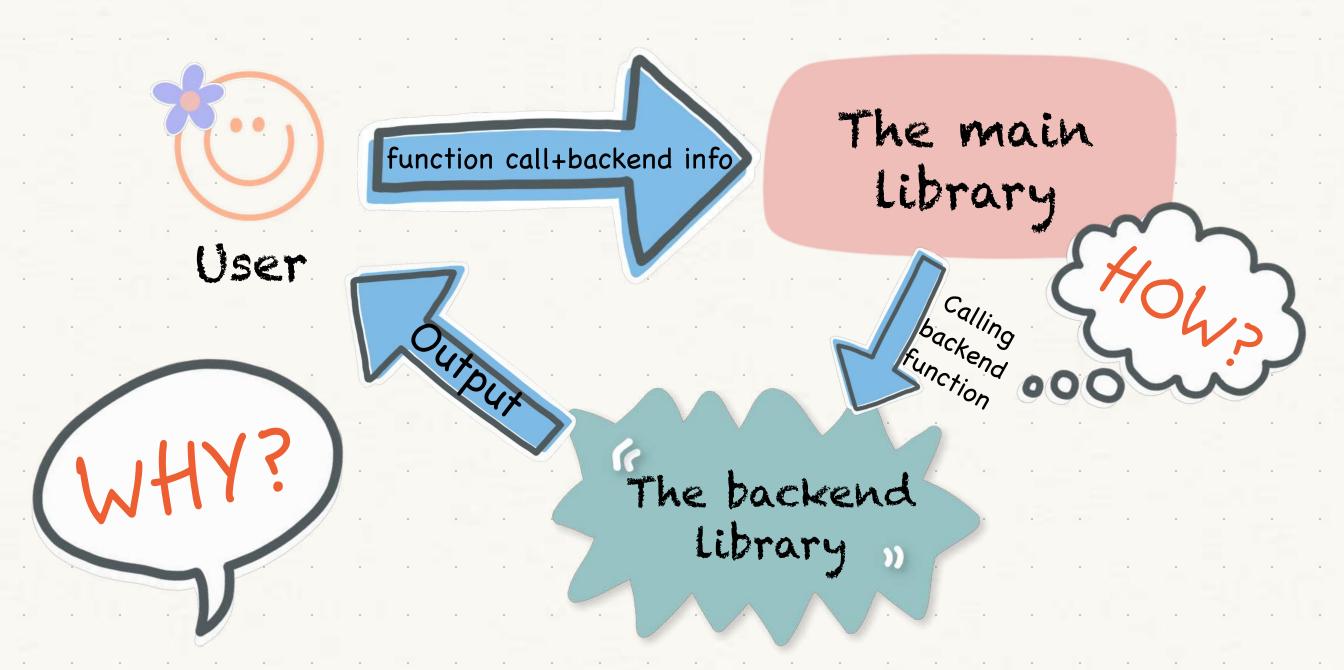
python-graphblas

def betweenness_centrality(G, ...):

Dispatching



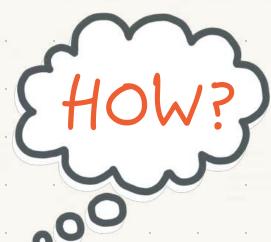
Dispatching



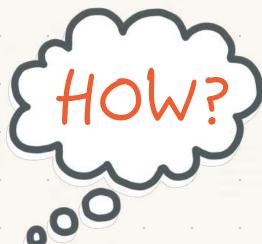




- NetworkX is a big and old project...
 - Consistency
 - Maintenance
- The User-API remains the same!
- Will this API dispatching be good for your project? IDK...



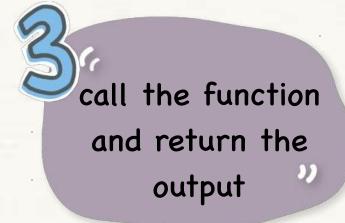
does the main library calls the function in the backend library?



does the main library calls the function in the backend library?







There may be more or less steps depending on the nature of the library!!

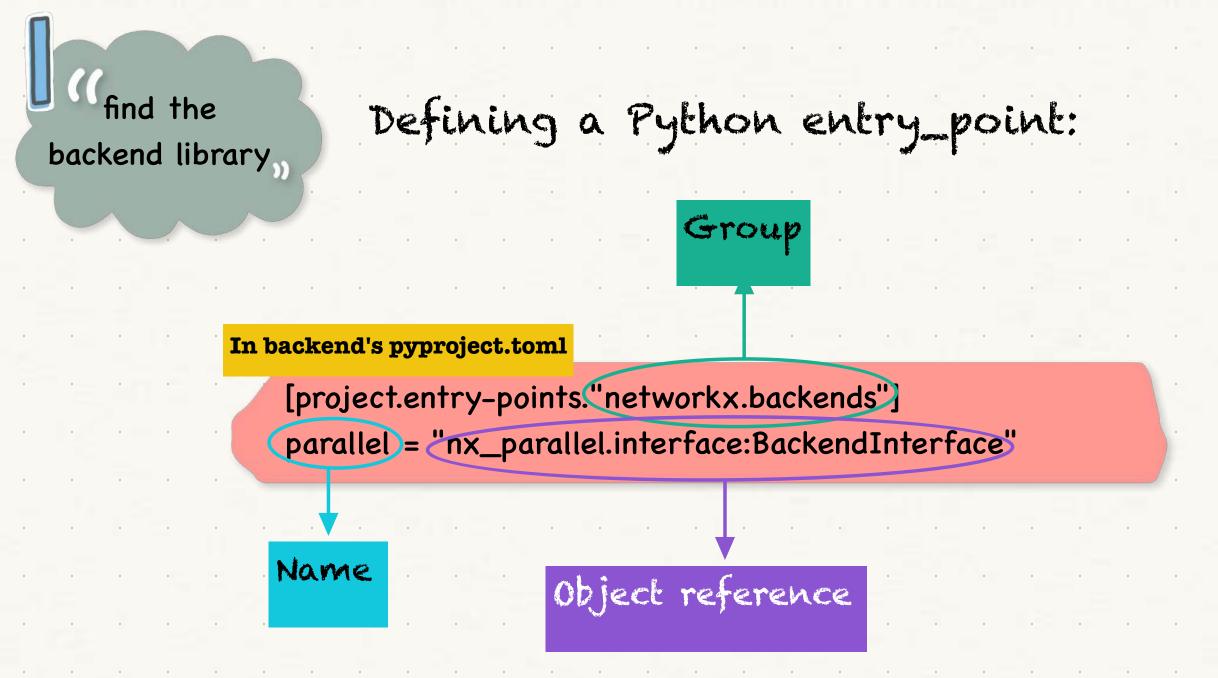


does NetworkX calls the function in the backend library?

find the backend library,

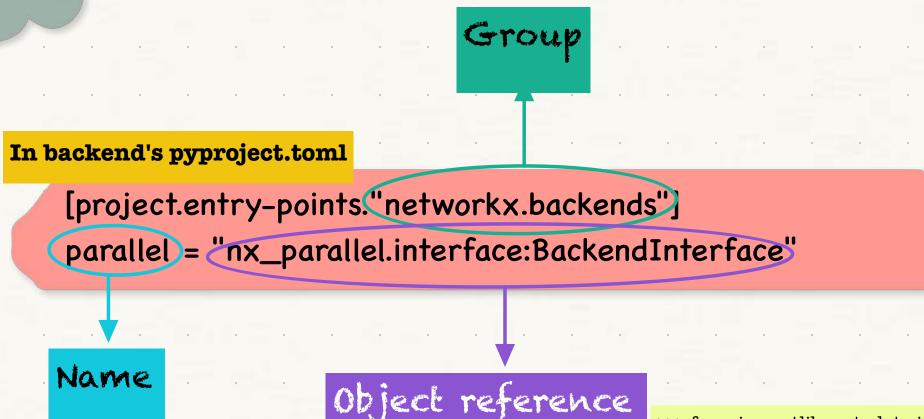
convert the args to backend apt. arg type find the function in the backend

call the function and return the output



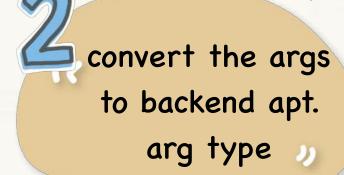


Defining a Python entry_point:



ref. https://packaging.python.org/en/latest/specifications/entry-points/

>>> from importlib.metadata import entry_points
>>> entry_points(group="networkx.backends")
(EntryPoint(name='parallel',
value='nx_parallel.interface:BackendInterface',
group='networkx.backends'),
EntryPoint(name='nx_loopback',
value='networkx.classes.tests.dispatch_interface:b
ackend_interface', group='networkx.backends'))

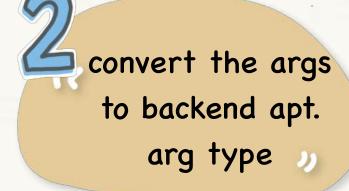


Object reference

BackendInterface object

`convert_from_nx`

`convert_to_nx`



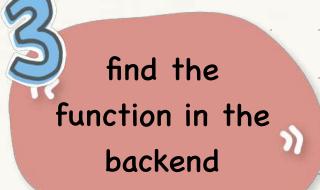
Object reference

BackendInterface object

`convert_from_nx`

`convert_to_nx`

All the backend algorithms



What makes up a NetworkX backend?

`networkx.backends` entry_point referring
to a `BackendInterface` object with
attributes - `convert_from_nx`,
`convert_to_nx`, and all the algorithms
implemented in the backend. And a backend
graph object with `__networkx_backend_`
attribute.

Some more dispatching-related stuff:

logging

Automatic testing

NETWORKX_TEST_BACKEND=parallel NETWORKX_FALLBACK_TO_NX=True pytest --pyargs networkx

Configurations

nx.config.backends.parallel.n_jobs = 8
nx.config.backends.parallel.verbose = 10

Additional backend args

>>> nx.betweenness_centrality(G, backend="parallel", get_chunks=get_chunks)

2nd entry point

[project.entrypoints."networkx.backend_info"] parallel = "_nx_parallel:get_info"

Additional backends implement this function

cugraph: GPU-accelerated backend.

Negative cycles are not yet supported. NotImplementedError will be raised if there are negative edge weights. We plan to support negative edge weights soon. Also, callable weight argument is not supported.

Additional parameters:

dtype: dtype or None, optional

The data type (np.float32, np.float64, or None) to use for the edge weights in the algorithm. If None, then dtype is determined by the edge

graphblas: OpenMP-enabled sparse linear algebra backend.

Additional parameters:

chunksize: int or str, optional

Split the computation into chunks; may specify size as string or number of rows. Default "10 MiB"

parallel: Parallel backend for NetworkX algorithms

The parallel implementation first divides the nodes into chunks and then creates a generator to lazily compute shortest paths lengths for each node in node_chunk, and then employs joblib's Parallel function to execute these

fallback option

Caching conversion

Is dispatching just to run algorithms faster?

Is dispatching just to run algorithms faster?

No...

- database backend: nx-arangodb
- visualisation backend: ??
- more to come...

Dispatching and other projects

- scikit-image
- sklearn
- numpy
- ?

Dispatching and other projects

- scikit-image
- sklearn
- numpy
- ?

what are your challenges?

Dispatching discussions:

- Weekly dispatch meetings: https://scientific-python.org/calendars/networkx.ics
- Dispatching channel in Scientific Python discord: https://discord.com/invite/vur45CbwMz
- `spatch` repository: https://github.com/scientificpython/spatch
- SPEC-2 (API Dispatching) document: https://scientific-python.org/specs/spec-0002/
- Scientific Python discuss: https://discuss.scientific-python.org/t/spec-2-api-dispatch/173



