Communities and more

About me

- PhD student at CEU's Center for Network Science, Budapest, Hungary
- Python enthusiast
- Research interest: temporal networks and team dynamics

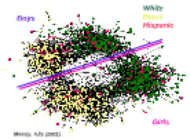
Network = Graph +Data

Why networks?

a picture is worth a thousand words

understanding complexity

hidden correlations



Pretty... so what?

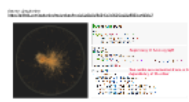
Community detection

Visualization

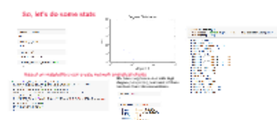
I suppose I have network data... so?

- Get data
- Figure out where the connections are
- Build up the network --- networkx, igraph
- Think again what is ur aim
- Do some preliminary analysis -- stats
- Visualize (if u can)
- Analyze the results

Data - Python Libraries Dependency Network



So, let's do some stats

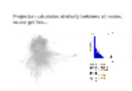


What kind of clusters can be found based on the similarity of packages?

"Bipartite network"

Newman Projection

Hairball :(



Backbone Filtering

Let's find communities

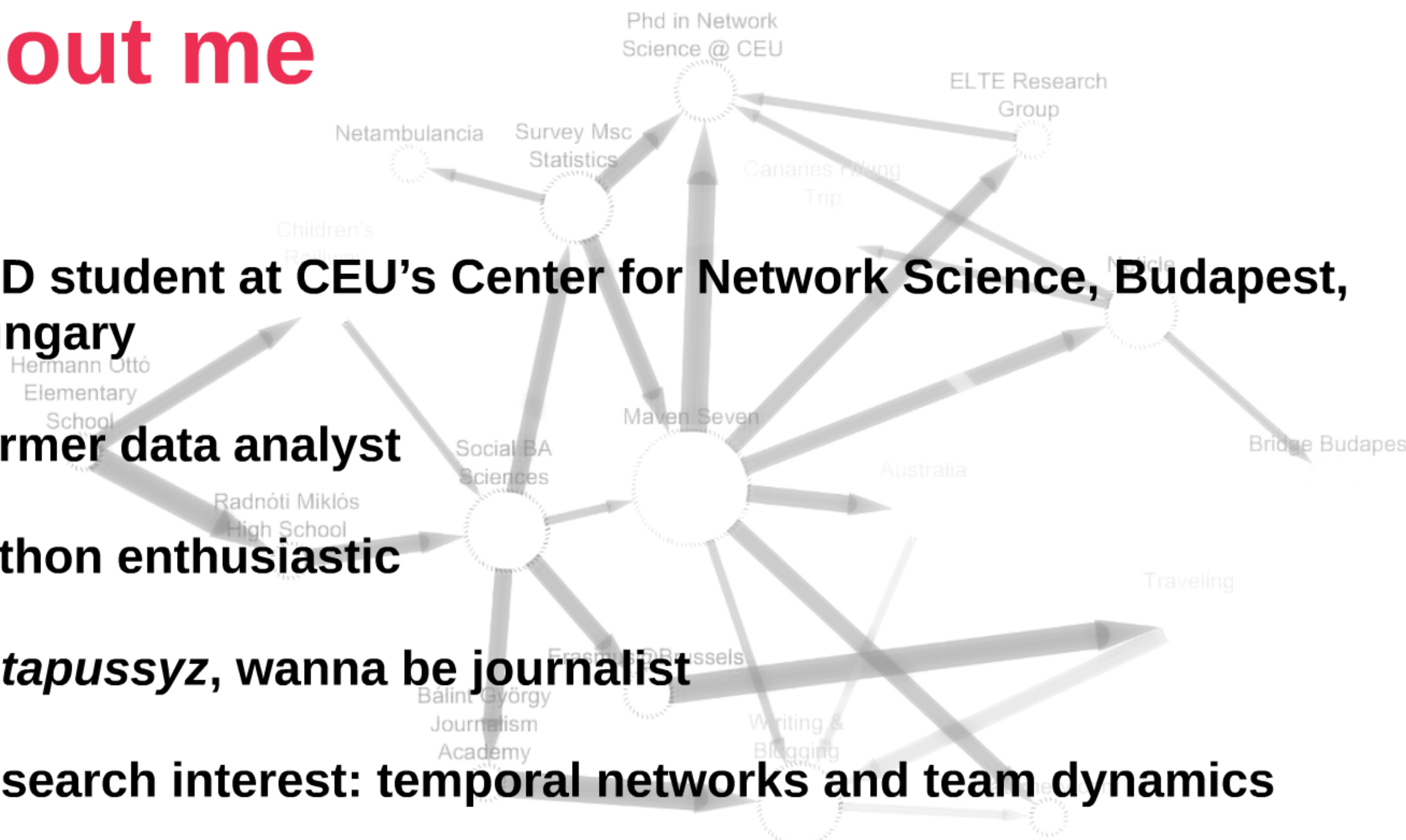
Small enough to visualize in Python

But still better with an other program

Communities and more

i Vasarhelyi
ter For Network Science, CEU, Budapest

About me



PhD student at CEU's Center for Network Science, Budapest, Hungary

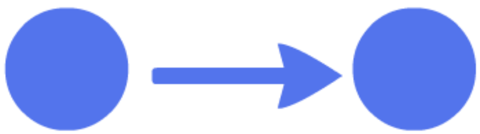
Former data analyst

Python enthusiastic

***Datapussyz*, wanna be journalist**

Research interest: temporal networks and team dynamics

Network = Graph +Data



directed

Who follows whom on Twitter

udirected

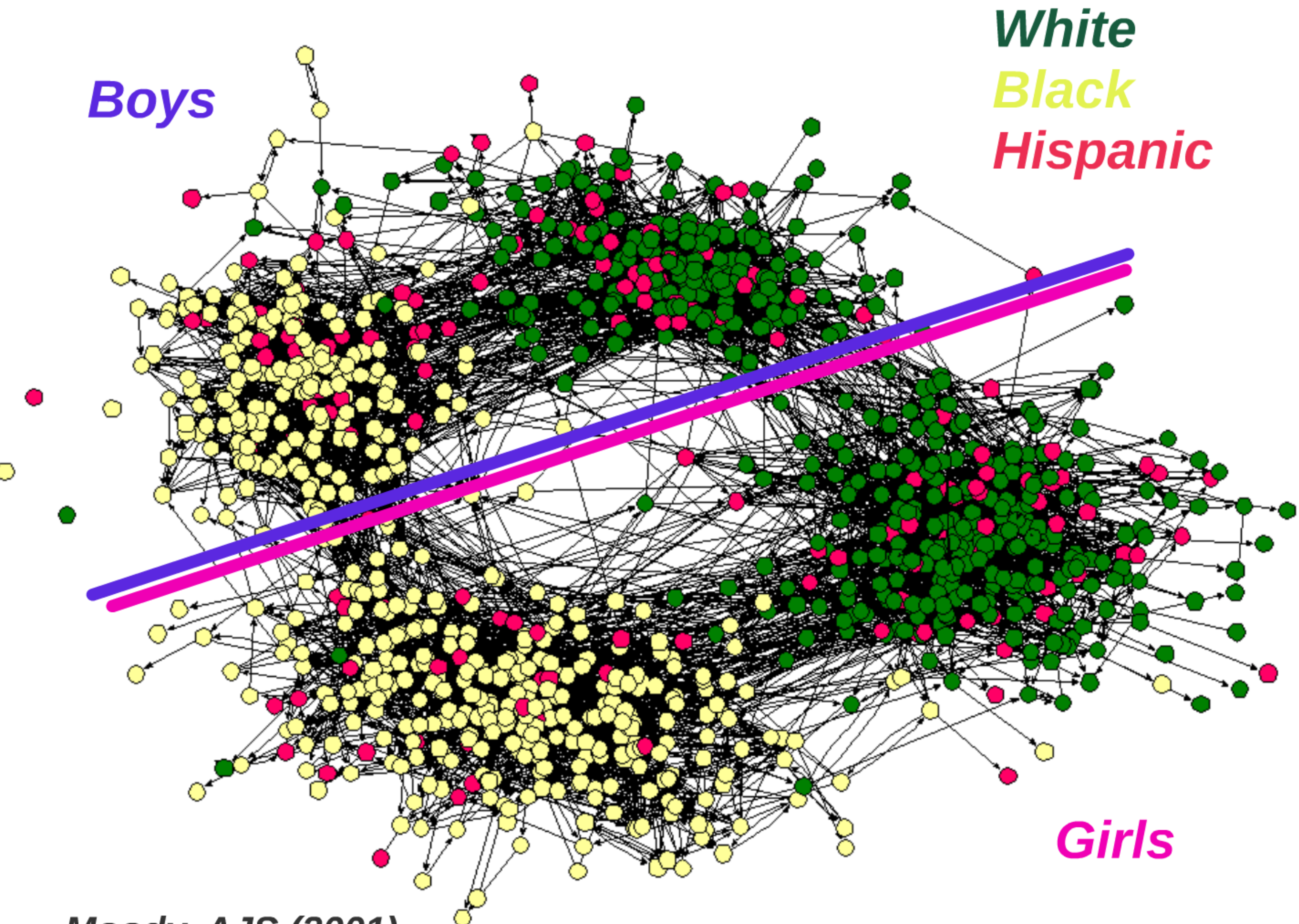
Friendship network on Facebook

Why networks?

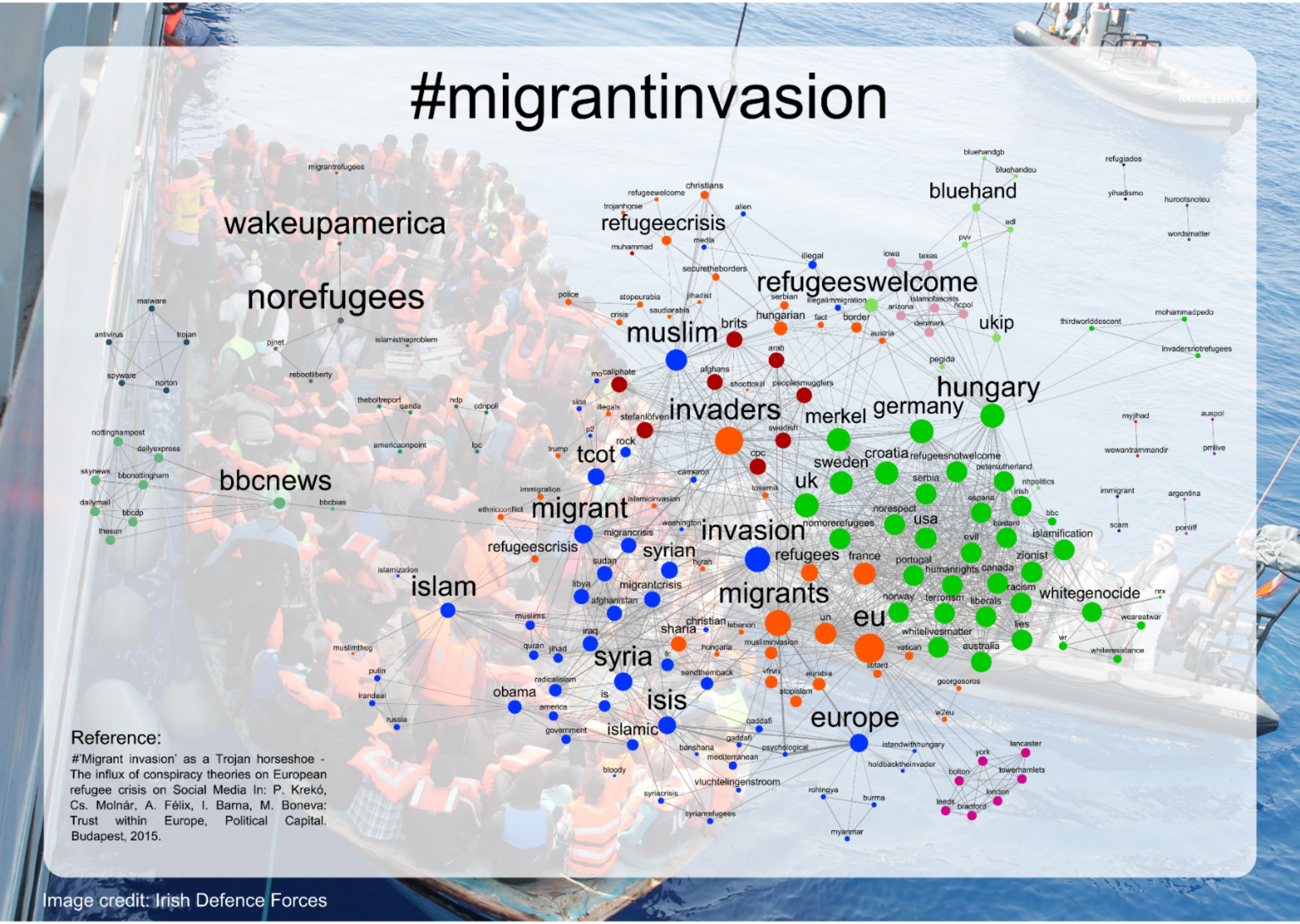
a picture is worth a thousand words

understanding complexity

hidden correlations



Moody, AJS (2001).


#migrantinvasion
wakeupamerica
norefugees
bbcnews
refugeecrisis
refugeeswelcome
bluehand
ukip
muslim
brits
hungarian
invaders
merkel
germany
hungary
tcot
migrant
invasion
syrian
refugees
france
sweden
croatia
uk
usa
islam
migrants
eu
syria
isis
islamic
obama
europe
refugeescrisis
whitegenocide
islamification
sharia
zionist
australia
Reference:
#'Migrant invasion' as a Trojan horseshoe - The influx of conspiracy theories on European refugee crisis on Social Media In: P. Krekó, Cs. Molnár, A. Félix, I. Barna, M. Boneva: Trust within Europe, Political Capital. Budapest, 2015.
Image credit: Irish Defence Forces

Pretty... so what?

Community detection

built-in function:

Communities

K-Clique

`k_clique_communities` (G, k[, cliques])	Find k-clique communities in graph using the percolation method.

http://perso.crans.org/aynaud/communities/

http://igraph.org/python/doc/igraph.Graph-class.html

Visualization

http://matplotlib.org/

http://pygraphviz.github.io/

https://github.com/erocarrera/pydot

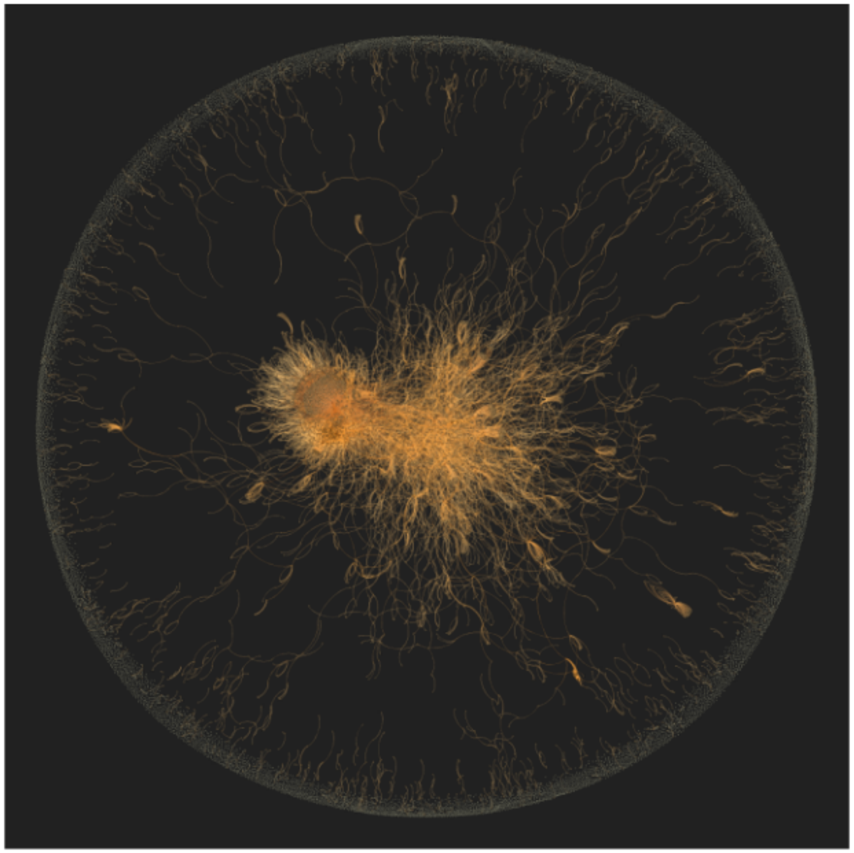
http://lightning-viz.org/

I suppose I have network data... so?

- Get data
- Figure out where the connections are
- Build up the network --- networkx, igraph
- Think again what is ur aim
- Do some preliminary analysis -- stats
- Visualize (if u can)
- Analyze the results

Data - Python Libaries Dependency Network

Source: @ogirardot
https://github.com/ogirardot/meta-deps/tree/a61d2c0ef8d246c3e666f2a191df303ea416da7



```
import networkx as nx, json
from base64 import b64decode

data = []
with open('pypi-deps.csv', 'r') as file:
    for line in file:
        name, version, deps = line.split('\t')
        deps = json.loads(b64decode(deps))
        data+= [(name, version, deps)]
```

```
G=nx.Graph()
edges_dict={}
for ex in data:
    name, version, deps = ex
    G.add_node("%s-%s" % (name, version))
    for dep in deps:
        if not '#' in dep: G.add_edge("%s-%s" % (name, version), dep.replace("\"", ""))

nx.write_edgelist(G,'Edges')
```

Super easy to have a graph

```
G.edges()[:10]
```

```
[(u'', 'simplemail-0.3'),
 (u'requests>=0.13', 'pathod-0.3.0'),
 ('zope.component-4.0.2', u'zope.interface>=3.8.0'),
 ('zope.component-4.0.2', u'zope.event'),
 ('zope.component-4.0.2', u'setuptools'),
 (u'Products.CMFDynamicViewFTI', 'Products.CMFPlone-4.1.1'),
 (u'Products.CMFDynamicViewFTI', 'Products.CMFPlone-4.1.3'),
 (u'Products.CMFDynamicViewFTI', 'Products.CMFPlone-4.1.2'),
 (u'Products.CMFDynamicViewFTI', 'Products.CMFPlone-4.1.4'),
 (u'Products.CMFDynamicViewFTI', 'plone.app.layout-2.2.2')]
```

Two nodes are connected if one is the dependency of the other

So, let's do some stats

```
G.number_of_nodes()
```

```
20693
```

```
G.number_of_edges()
```

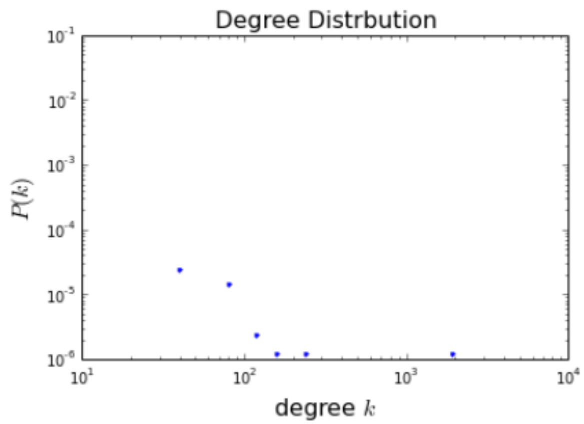
```
14071
```

```
nx.density(G)
```

```
6.5724763374737l2e-05
```

```
nx.number_connected_components(G)
```

```
11776
```




```
degrees=nx.degree(G)
sorted_degree = sorted(degrees.items(), key=operator.itemgetter(1))
sorted_degree[-20::]
```

```
[(u'ZODB3', 71),
 (u'argparse', 72),
 (u'zope.i18nmessageid', 76),
 (u'zope.publisher', 79),
 (u'fanstatic', 92),
 (u'Flask', 98),
 ('Products.CMFPlone-4.1.2', 100),
 ('Products.CMFPlone-4.1.1', 100),
 ('Products.CMFPlone-4.1.3', 100),
 ('Products.CMFPlone-4.1.4', 100),
 (u'zope.schema', 102),
 (u'distribute', 106),
 (u'zc.buildout', 107),
 (u'lxml', 108),
 (u'django', 115),
 (u'simplejson', 133),
 (u'requests', 143),
 (u'zope.component', 190),
 (u'zope.interface', 239),
 (u'setuptools', 1931)]
```

Based on matplotlib u can create network analytical charts

```
def make_a_plot(deg_list, title1, title2,title3, n):
    bin_edges = np.logspace(np.log10(min(deg_list)), np.log10(max(deg_list)), num=n)
    density, _ = np.histogram(deg_list, bins=bin_edges, density=True)

    bin_edges_lin = np.linspace(min(deg_list), max(deg_list))
    density_lin, _ = np.histogram(deg_list, bins=bin_edges_lin, density=True)

    fig = plt.figure(figsize=(6,4))
    plt.loglog(bin_edges_lin[:-1], density_lin, marker='.', linestyle='none',color='b')
    plt.loglog(bin_edges[:-1], density, marker='o', linestyle='none',color='r',markeredgecolor='0.6')

    plt.xlabel(title1, fontsize=16)
    plt.ylabel(title2, fontsize=16)
    plt.title(title3,fontsize=16)
    plt.show()
```

We have very few nodes with high degree *(setuptools)*, but most of them has less than 100 connections.

```
degrees['pandas']
```

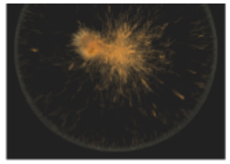
```
5
```

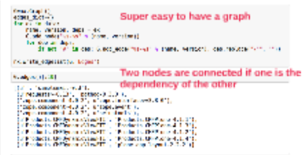
```
nx.neighbors(G,'pandas')
```

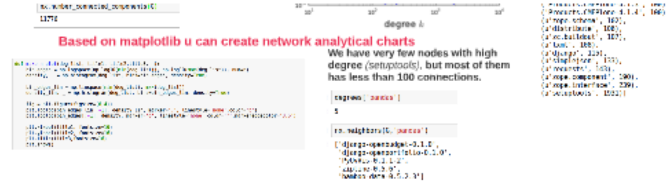
```
['django-openbudget-0.1.0',
 'django-openportfolio-0.1.0',
 'PyUvVis-0.1.1-2',
 'zipline-0.5.6',
 'bamboo-data-0.5.2.3']
```

n the similarity of packages?

- Think again what is ur aim
- Do some preliminary analysis -- stats
- Visualize (if u can)
- Analyze the results

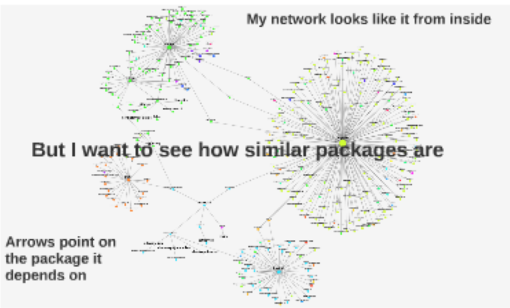




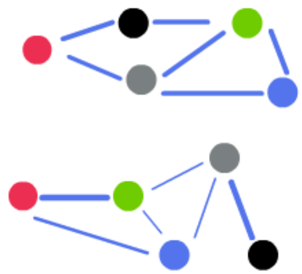


What kind of clusters can be found based on the similarity of packages?

"Bipartite network"



Newman Projection



original network

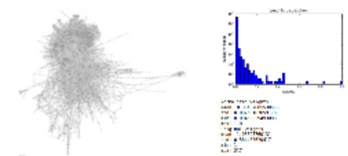
Newman Projection (2002)

More about projections: https://github.com/velf/Network_Analytical_Notebooks.git

projected network
based on common neighbors

Hairball :(

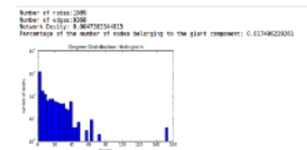
Projection calculates similarity between all nodes, so we get this...



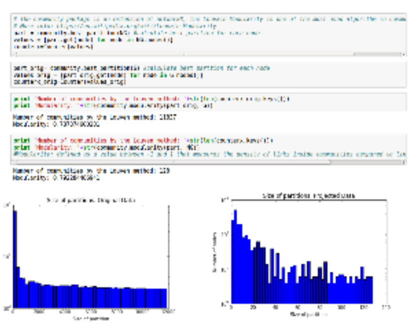
But based on weights (similarity) we can filter, but how?

Backbone Filtering

Disparity filter is a network reduction algorithm to extract the backbone structure of undirected weighted network . I create a threshold based on the Bonferroni Correction for each edge and decide is it significantly important for the certain node or not.

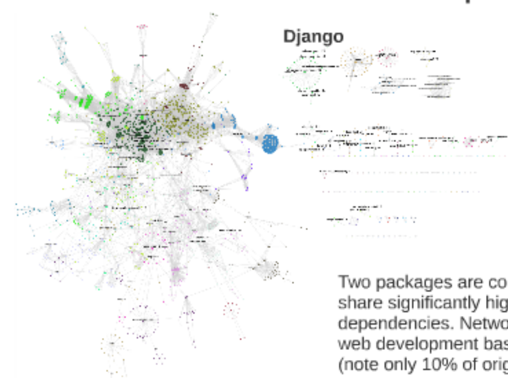


Let's find communities



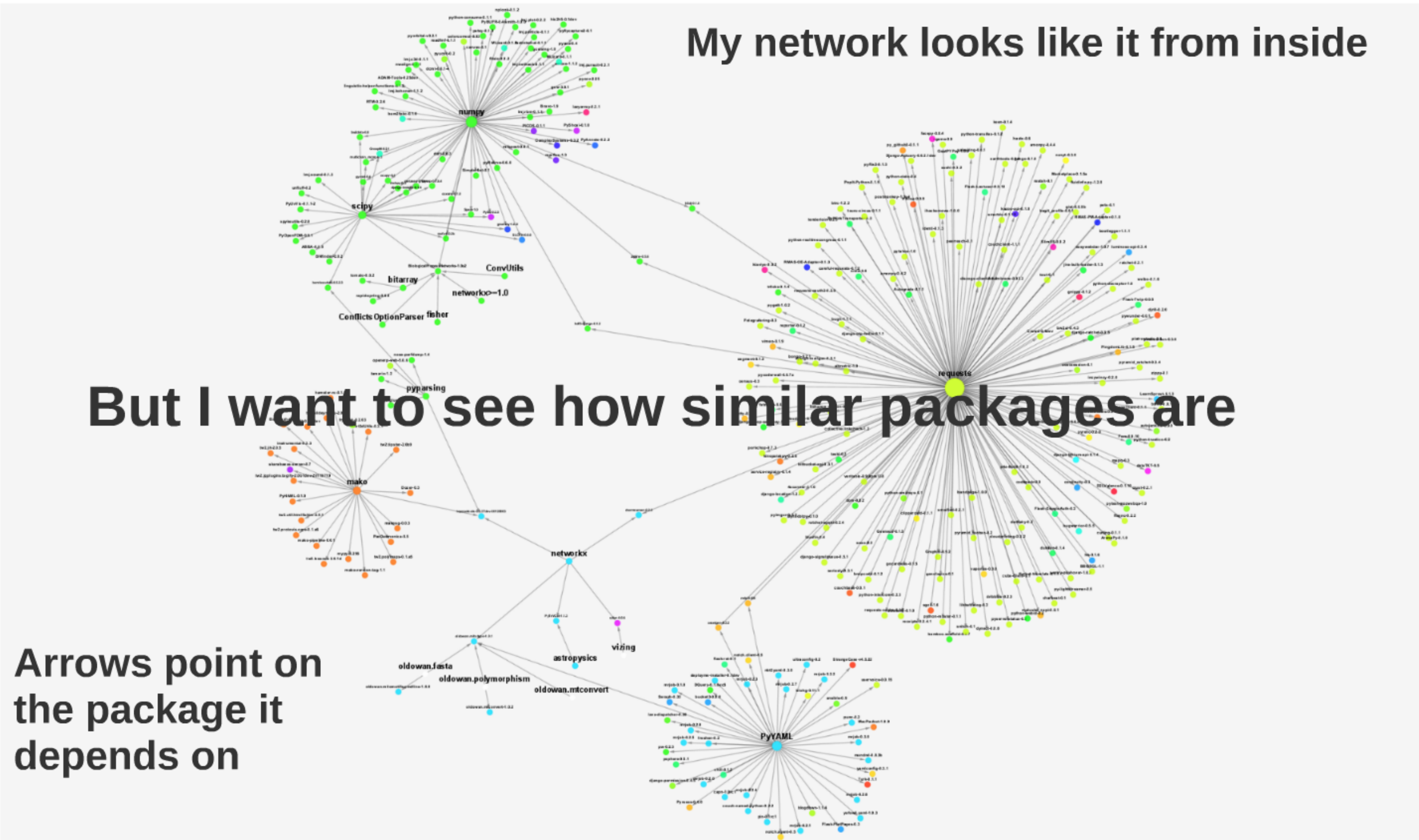
Small enough to visualize in Python

But still better with an other program



Two packages are connected if they share significantly high level of dependencies. Network kept mainly web development based packages (note only 10% of original data in it).

"Bipartite network"

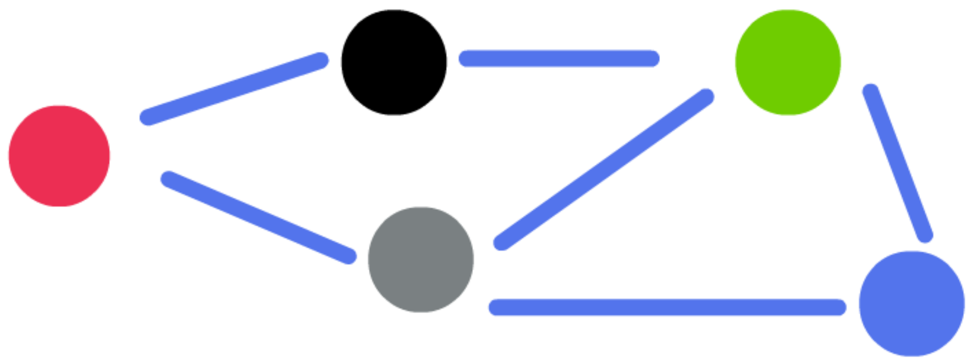


My network looks like it from inside

But I want to see how similar packages are

Arrows point on the package it depends on

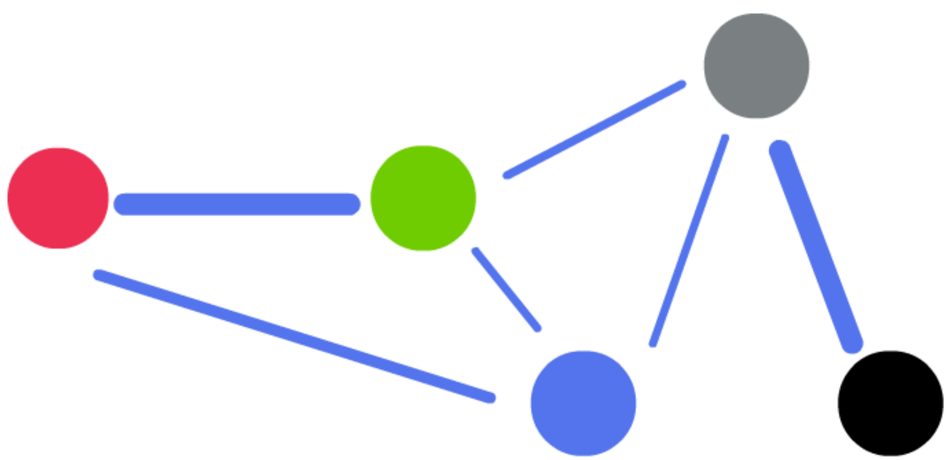
Newman Projection



original network

Newman Projection (2001)
Wij= sum(1/Np - 1) where Np is the number of authors on paper p.
In the context of scientific collaboration networks, it means that if two scientists only wrote a single paper together with no other co-authors get a weight of 1. If two scientists have written two papers together without any co-author, the weight of their tie would be 2 and so on. This method finally takes care about the different set sizes.

More about projections: https://github.com/velf/Network_Analytical_Notebooks.git



projected network
based on common neighbors

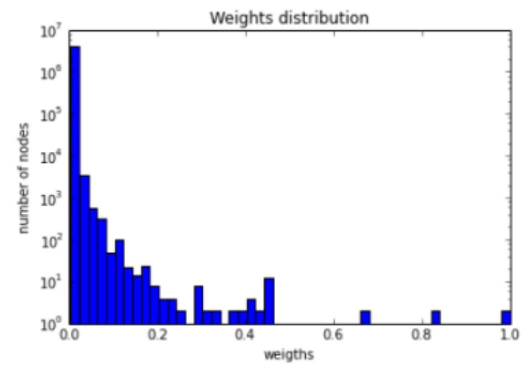
Hairball :(

Ba

Projection calculates similarity between all nodes, so we get this...

Dis
the
cre
eac
cer

wrote a
e written
n. This



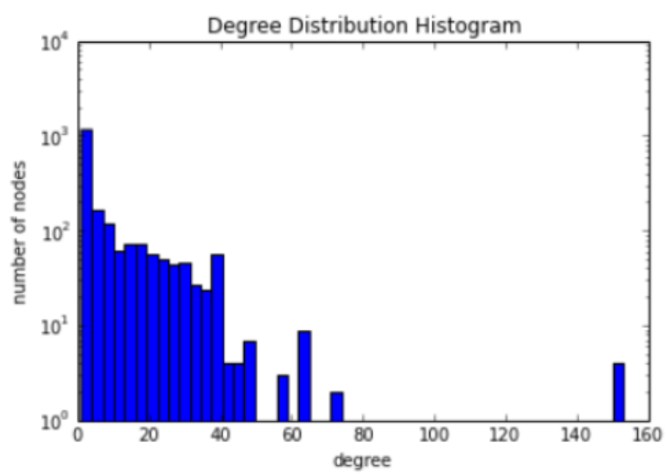

```
Normalized weights
mean: 0.00487685530556
std: 0.00260107659178
min: 0.00460829493088
max: 1.0
Original weights
mean: 1.05827760131
std: 0.564433620417
min: 1
max: 217
```

But based on weights (similarity) we can filter, but how?

Backbone Filtering

Disparity filter is a network reduction algorithm to extract the backbone structure of undirected weighted network . I create a threshold based on the Bonferroni Correction for each edge and decide is it significantly important for the certain node or not.

```
Number of nodes:1989
Number of edges:9368
Network Desity: 0.0047383344815
Percentage of the number of nodes belonging to the giant component: 0.817496229261
```



Let's find communities

```
# The community package is an extension of networkX, the Louvain Modularity is one of the most used algorithm in commu
# More info: https://en.wikipedia.org/wiki/Louvain_Modularity
part = community.best_partition(NG) #calculate best parttion for each node
values = [part.get(node) for node in NG.nodes()]
counterx=Counter(values)
```

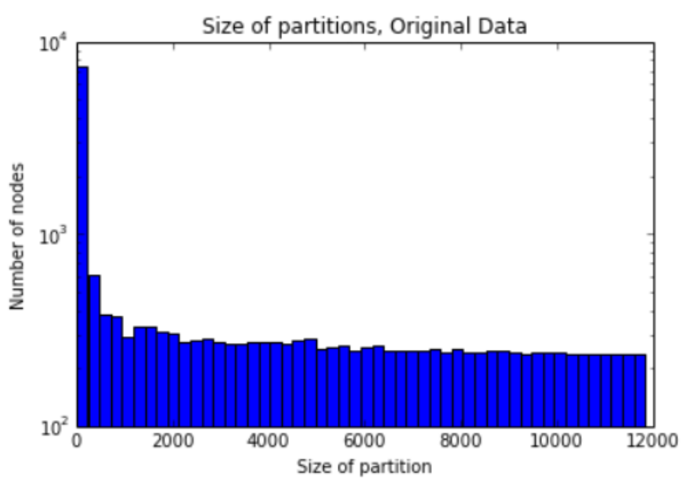
```
part_orig= community.best_partition(G) #calculate best parttion for each node
values_orig = [part_orig.get(node) for node in G.nodes()]
counterx_orig=Counter(values_orig)
```

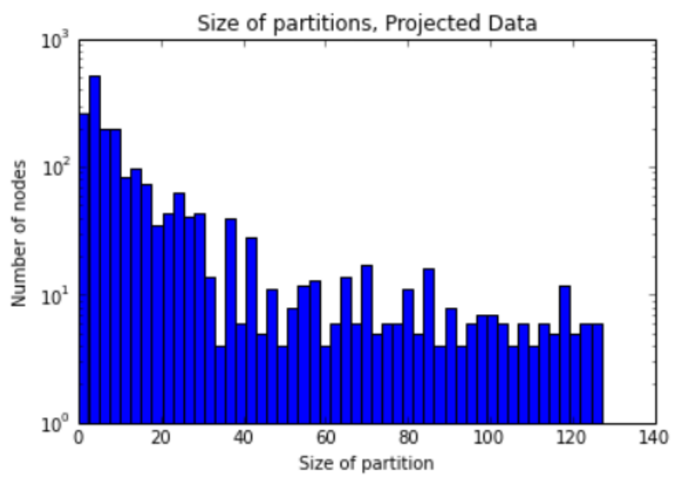
```
print 'Number of communities by the Louven method: '+str(len(counterx_orig.keys()))
print 'Modularity: '+str(community.modularity(part_orig, G))
```

```
Number of communities by the Louven method: 11837
Modularity: 0.787374600201
```

```
print 'Number of communities by the Louven method: '+str(len(counterx.keys()))
print 'Modularity: '+str(community.modularity(part, NG))
#Modularity: defined as a value between -1 and 1 that measures the density of links inside communities compared to lin
```

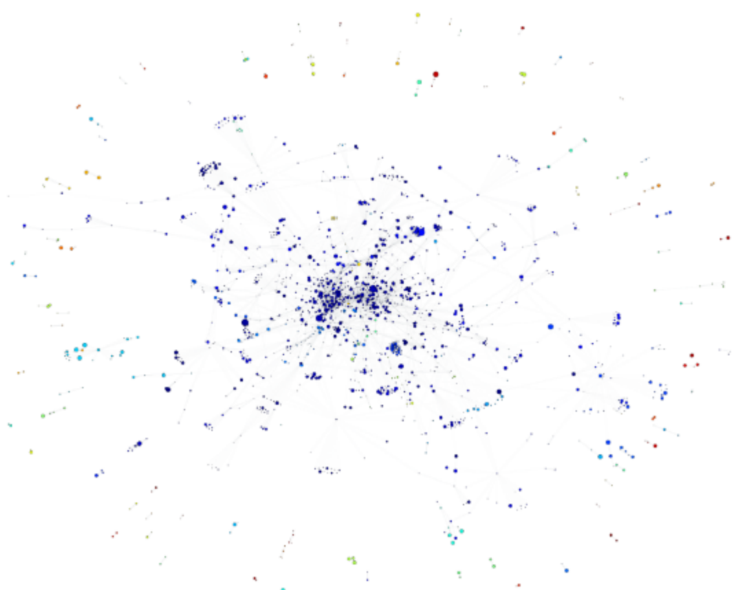
```
Number of communities by the Louven method: 128
Modularity: 0.792284486941
```

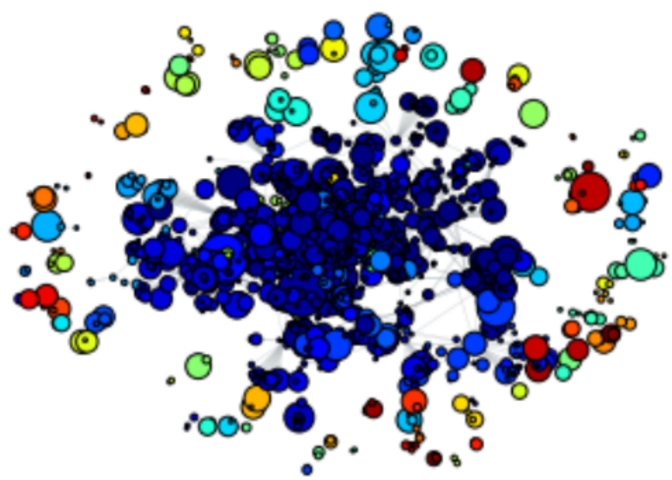




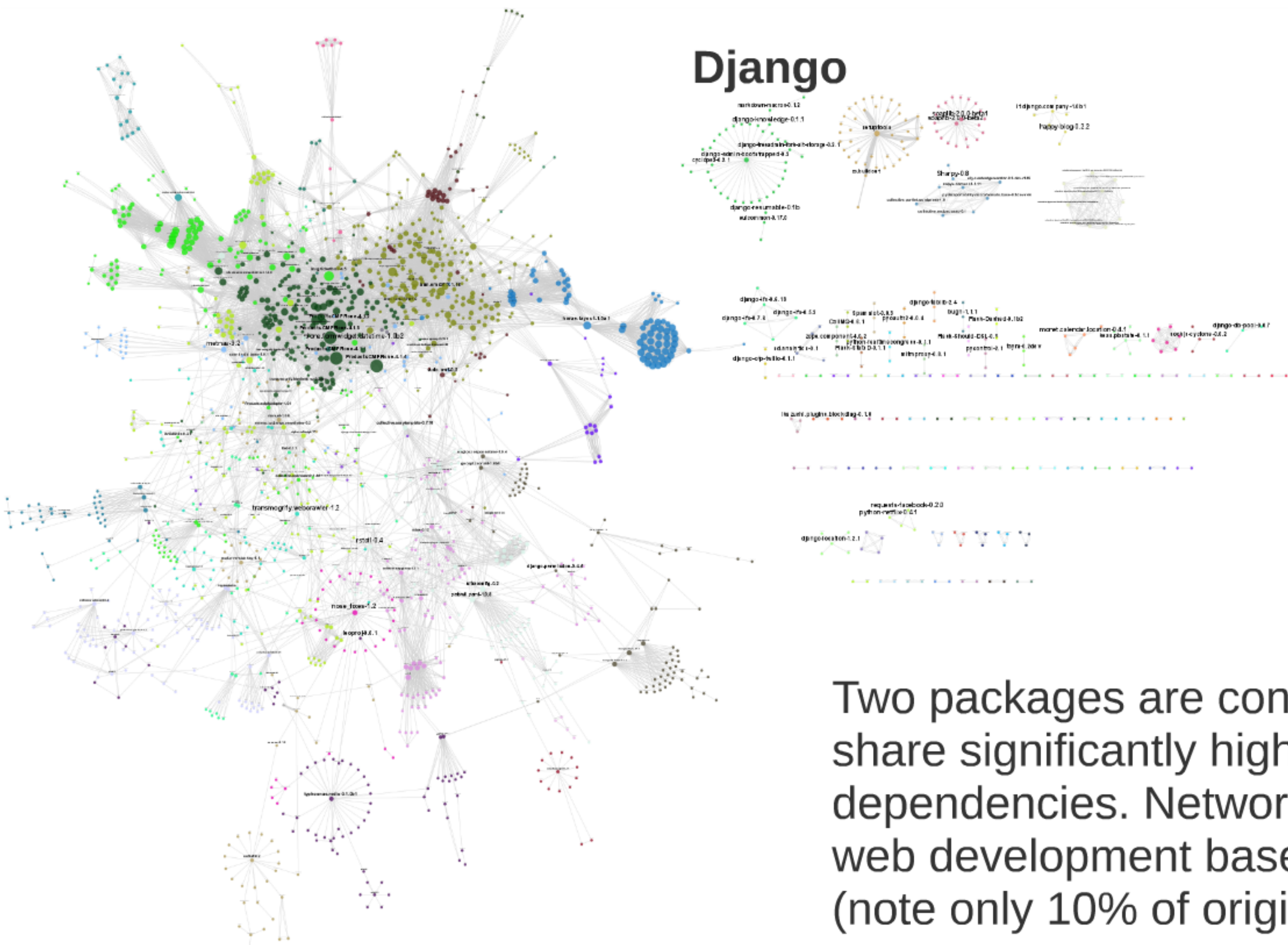
Small enough to visualize in Python

```
pos=nx.fruchterman_reingold_layout(NG, scale=20) #Fruchterman-Reingold force-directed algorithm
#See more: https://en.wikipedia.org/wiki/Force-directed_graph_drawing
nx.draw_networkx_nodes(NG,pos,node_size=[i*6 for i in NG.degree().values()],cmap=plt.get_cmap('jet'),
                       node_color=values )
nx.draw_networkx_edges(NG,pos,edgelist=weights,width=weights_to_plot,alpha=0.5,edge_color='#A6AFB4')
fig = plt.gcf()
fig.set_size_inches(90, 70)
fig.savefig('Python_Dependency_Network_Clustered_final.png', dpi=120)
plt.close()
```



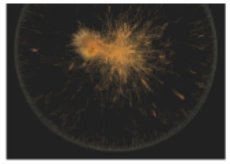


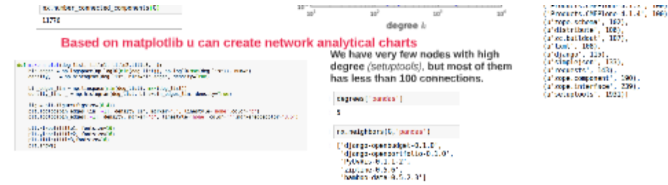
But still better with an other program



Two packages are connected if they share significantly high level of dependencies. Network kept mainly web development based packages (note only 10% of original data in it).

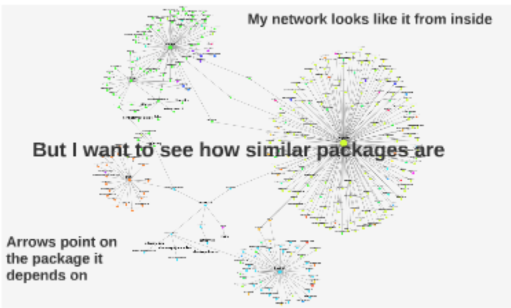
- Think again what is ur aim
- Do some preliminary analysis -- stats
- Visualize (if u can)
- Analyze the results



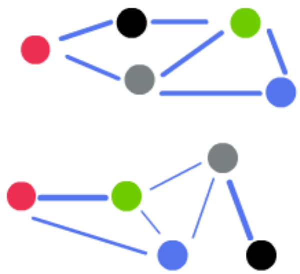


What kind of clusters can be found based on the similarity of packages?

"Bipartite network"



Newman Projection



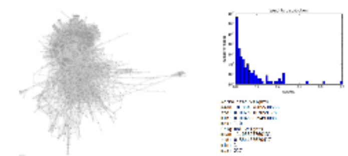
original network

More about projections: https://github.com/vEll/Network_Analytical_Notebooks.git

projected network based on common neighbors

Hairball :(

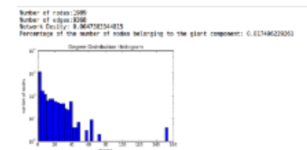
Projection calculates similarity between all nodes, so we get this...



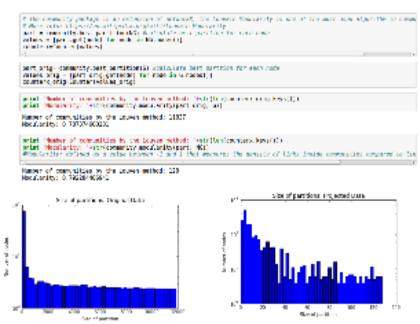
But based on weights (similarity) we can filter, but how?

Backbone Filtering

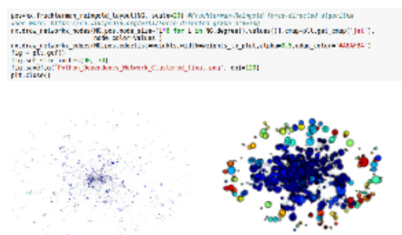
Disparity filter is a network reduction algorithm to extract the backbone structure of undirected weighted network . I create a threshold based on the Bonferroni Correction for each edge and decide is it significantly important for the certain node or not.



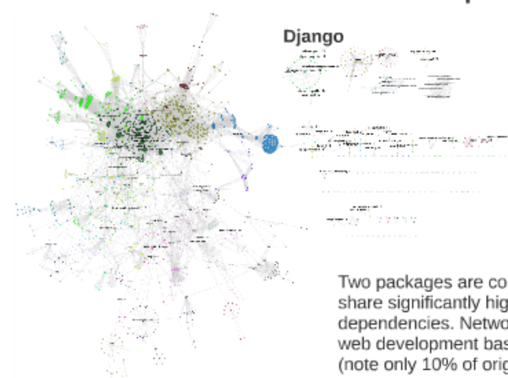
Let's find communities



Small enough to visualize in Python



But still better with an other program



Two packages are connected if they share significantly high level of dependencies. Network kept mainly web development based packages (note only 10% of original data in it).

Little Extra for the Web

Interactive graphs wth Lightning

```
import os
from lightning import Lightning
from numpy import random, asarray, linspace, corrcoef
from colorsys import hsv_to_rgb
from sklearn import datasets
```

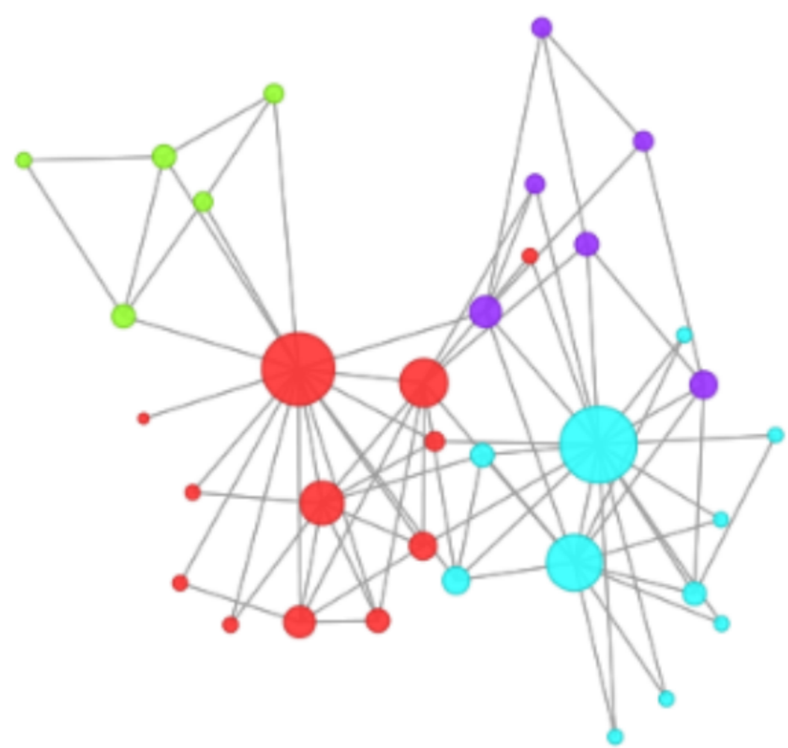
U can zoom in and play with it!

```
lgn = Lightning(ipython=True, host='http://public.lightning-viz.org')
```

Lightning initialized

Connected to server at http://public.lightning-viz.org

```
mat = nx.adjacency_matrix(G).todense()
n=G.number_of_nodes()
c = [list(asarray(hsv_to_rgb(float(y)/4, 0.8, 1.0))*255) for x,y in part.iteritems()]
g = G.degree().values()
lgn.force(mat, color=c, size=(asarray(g) + 1.5))
```



Summary

NetworkX is a good tool, with huge support from the scientific community

Still lack of the newest results/methods of network science (Come and work on it!

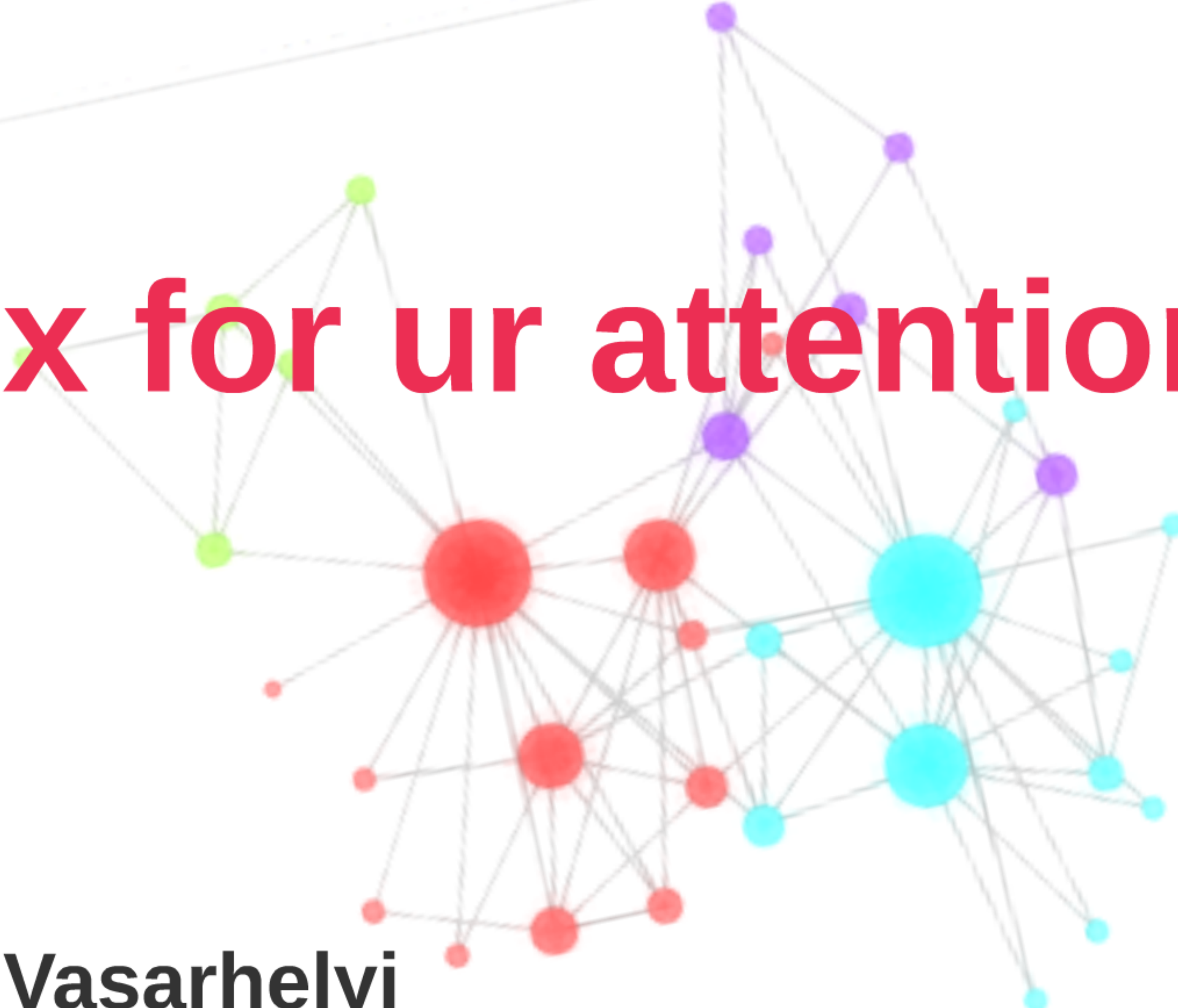
Not efficient in every case (large and very dense networks, visualization)

Think twice if u want network visualizations (not always worth the effort)

Other good python packages (iGraph, wrtten in C/C++ -faster, but not the best documentation)

For visualization: Gephi, Cytoscape

Thx for ur attention:)



Orsi Vasarhelyi
CEU, Center for Network Science