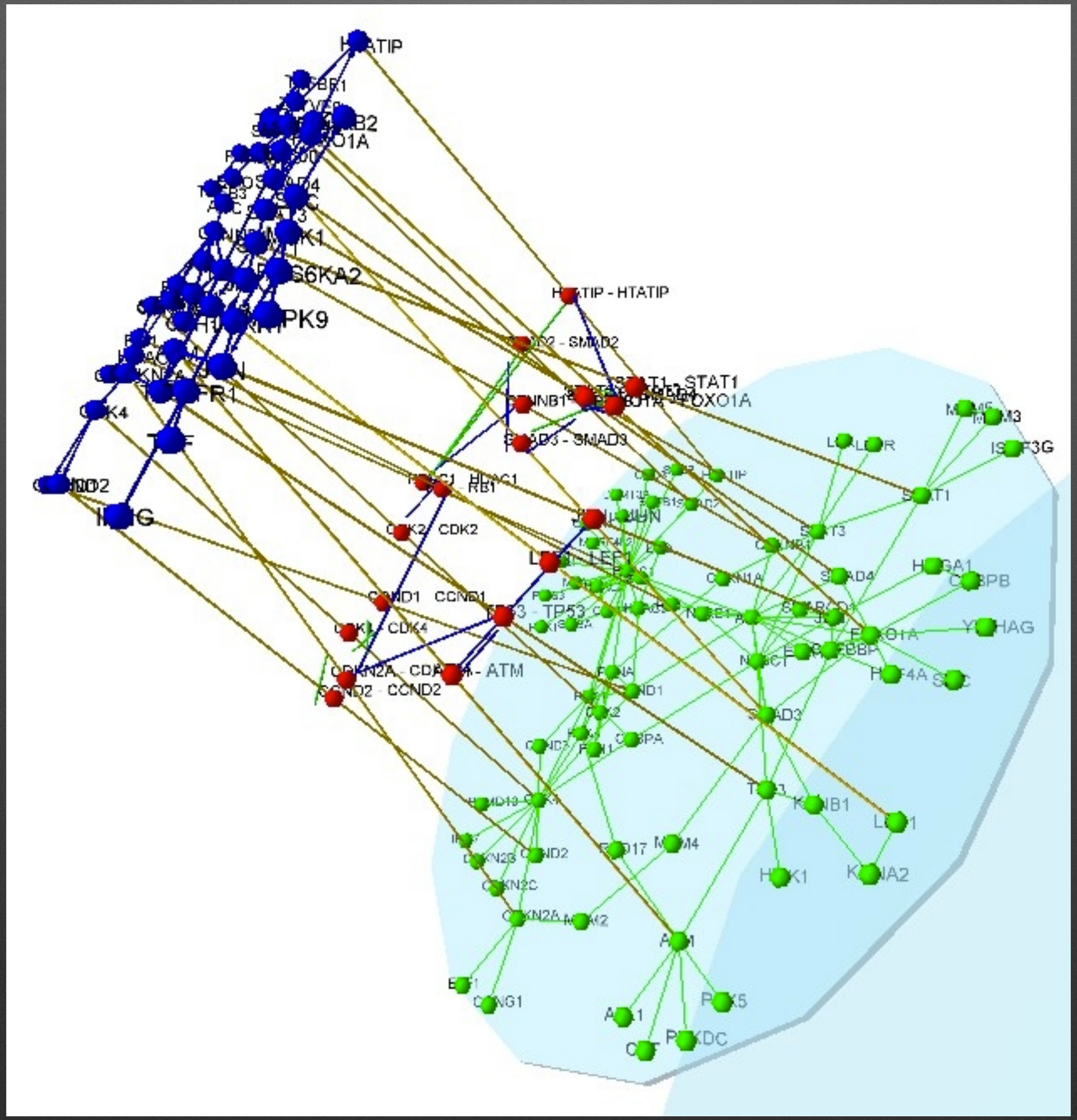
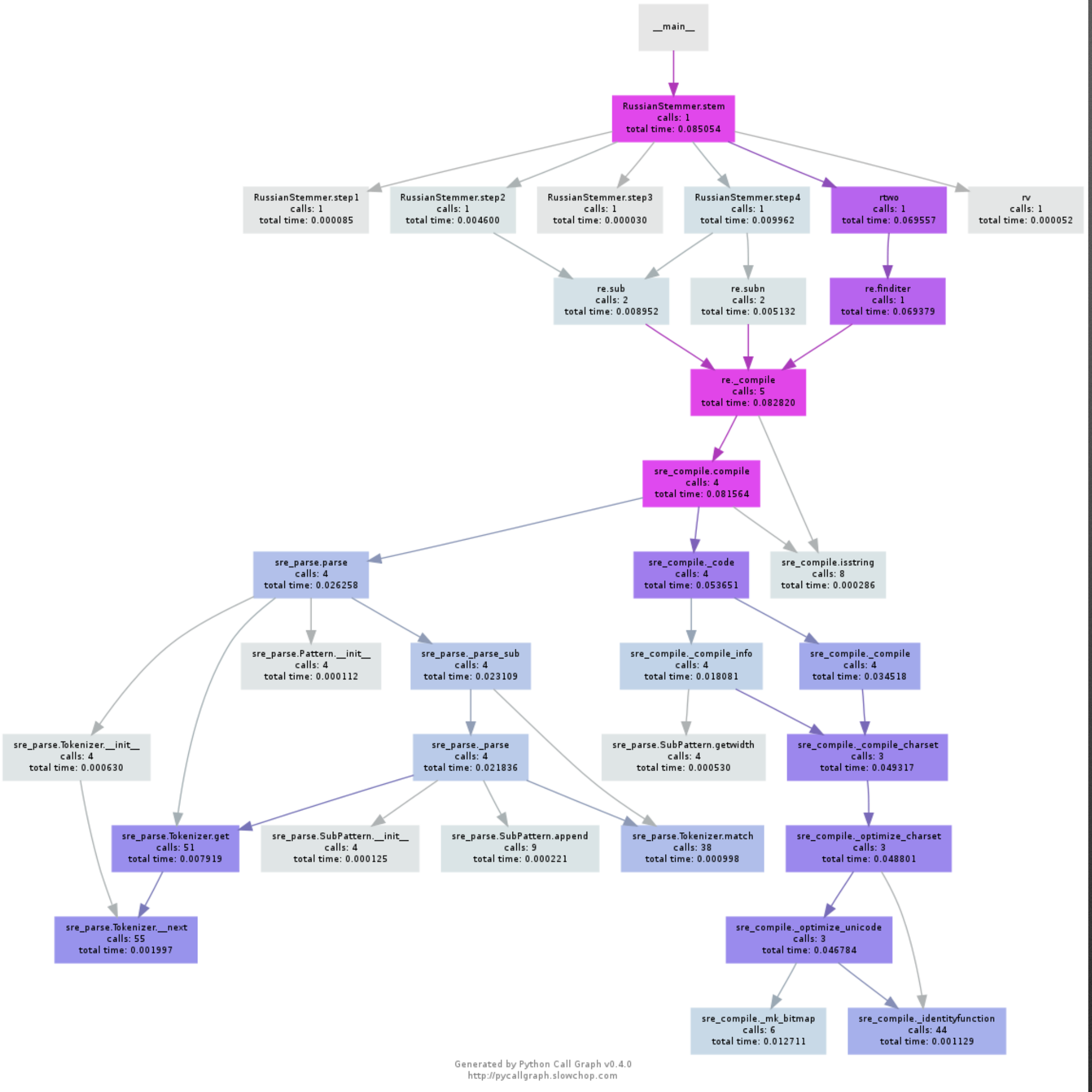
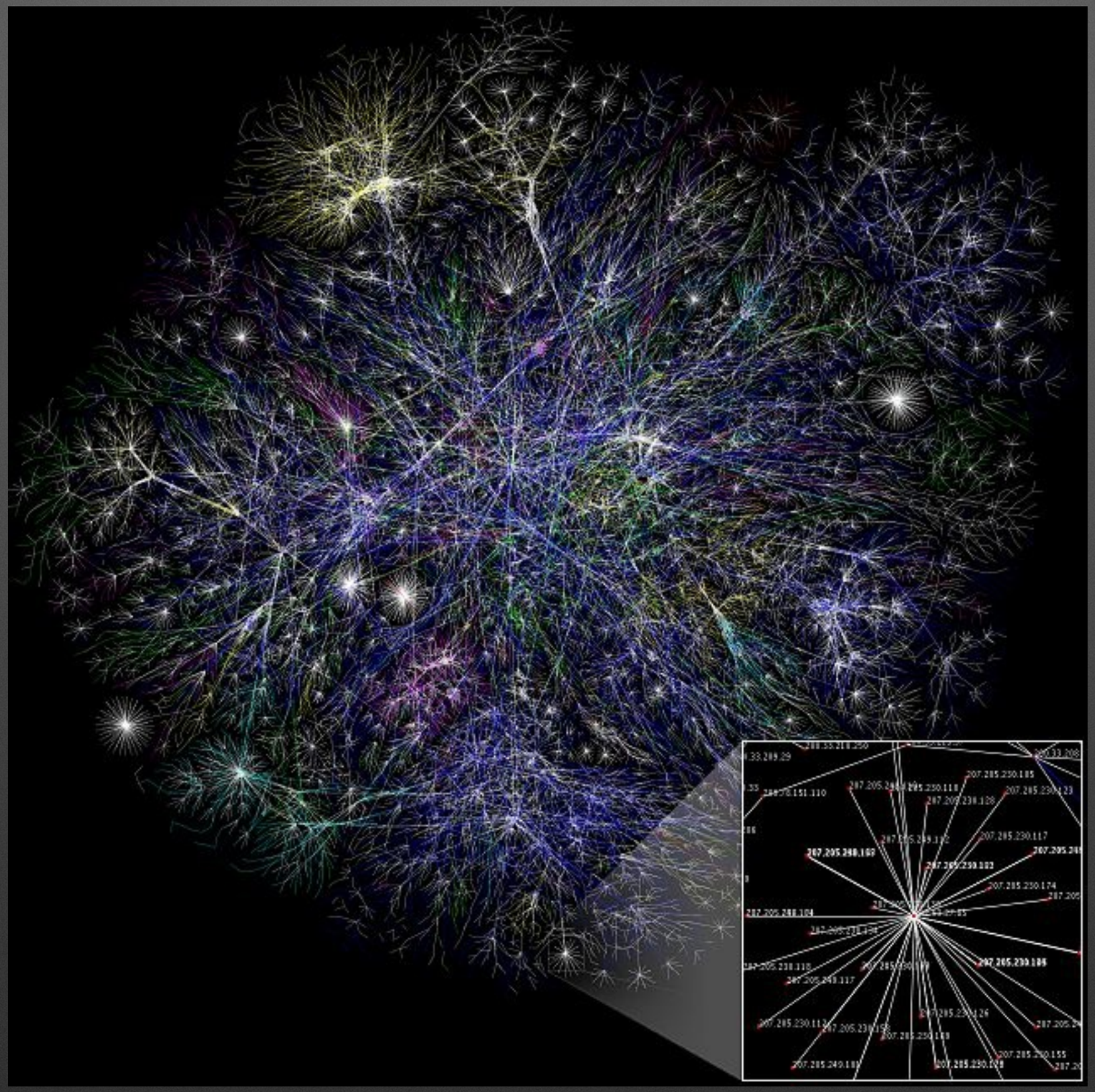
Graphs, Networks and Python: The Power of Interconnection

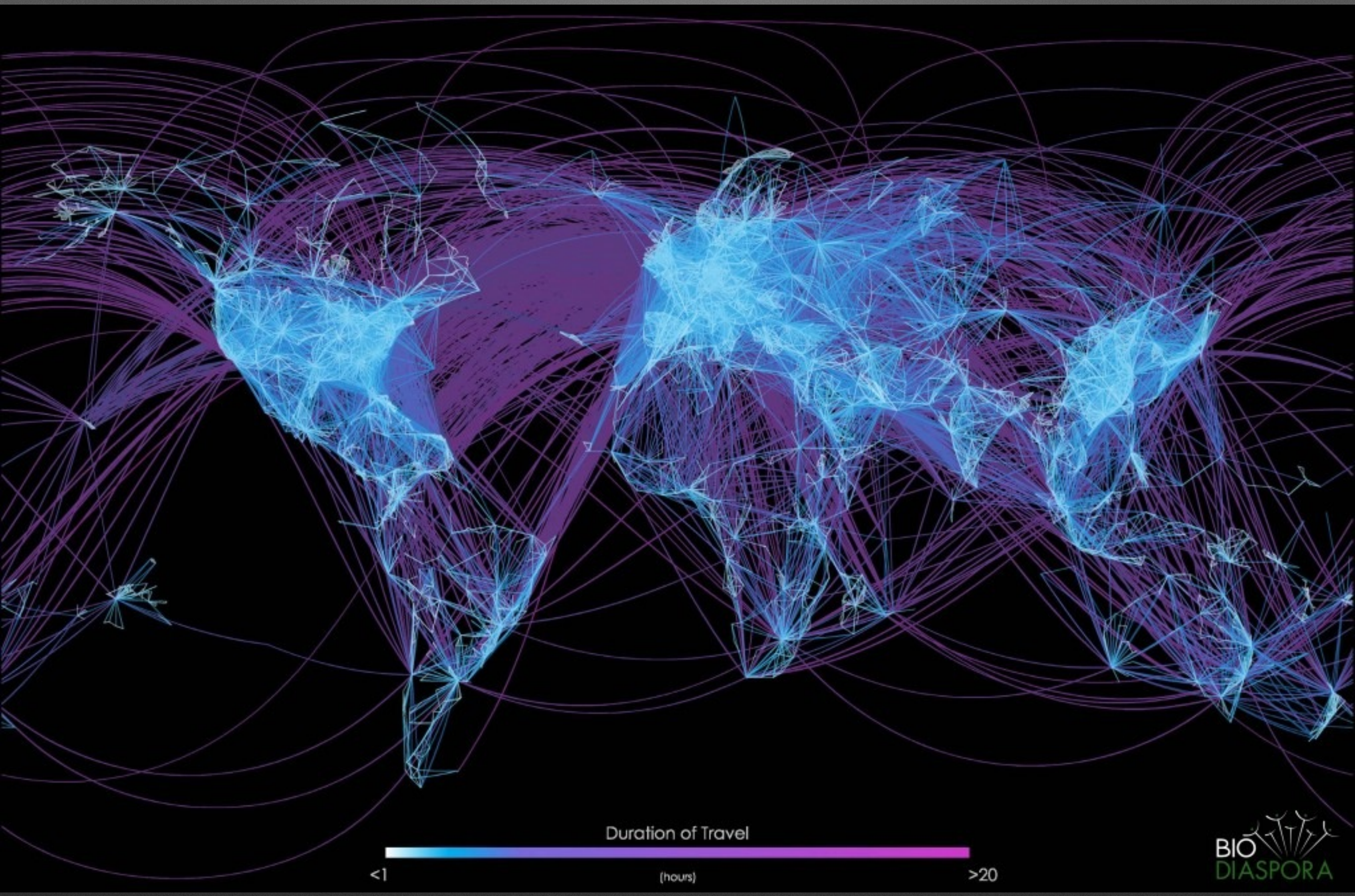
Lachlan Blackhall - lachlan@repositpower.com

A little about me…


HTATIP - HTATIP
STAT1 - STAT1
FOXO1A
SMAD3 - SMAD3
HDAC1
RB1
CDK2
CCND1
CCND2
TP53 - TP53
CDK4
ATM
S6KA2
PK9
TGFR1
LEF1
STAT1
STAT3
SMAD4
SMAD3
CEBPB
YWHAG
CCNB1
CCNA2
CDK1
CCNG1
PRKDC


__main__
RussianStemmer.stem
calls: 1
total time: 0.085054
RussianStemmer.step1
calls: 1
total time: 0.000085
RussianStemmer.step2
calls: 1
total time: 0.004600
RussianStemmer.step3
calls: 1
total time: 0.000030
RussianStemmer.step4
calls: 1
total time: 0.009962
rtwo
calls: 1
total time: 0.069557
rv
calls: 1
total time: 0.000052
re.sub
calls: 2
total time: 0.008952
re.subn
calls: 2
total time: 0.005132
re.finditer
calls: 1
total time: 0.069379
re._compile
calls: 5
total time: 0.082820
sre_compile.compile
calls: 4
total time: 0.081564
sre_parse.parse
calls: 4
total time: 0.026258
sre_compile._code
calls: 4
total time: 0.053651
sre_compile.isstring
calls: 8
total time: 0.000286
sre_parse.Pattern.__init__
calls: 4
total time: 0.000112
sre_parse._parse_sub
calls: 4
total time: 0.023109
sre_compile._compile_info
calls: 4
total time: 0.018081
sre_compile._compile
calls: 4
total time: 0.034518
sre_parse.Tokenizer.__init__
calls: 4
total time: 0.000630
sre_parse._parse
calls: 4
total time: 0.021836
sre_parse.SubPattern.getwidth
calls: 4
total time: 0.000530
sre_compile._compile_charset
calls: 3
total time: 0.049317
sre_parse.Tokenizer.get
calls: 51
total time: 0.007919
sre_parse.SubPattern.__init__
calls: 4
total time: 0.000125
sre_parse.SubPattern.append
calls: 9
total time: 0.000221
sre_parse.Tokenizer.match
calls: 38
total time: 0.000998
sre_compile._optimize_charset
calls: 3
total time: 0.048801
sre_parse.Tokenizer.__next
calls: 55
total time: 0.001997
sre_compile._optimize_unicode
calls: 3
total time: 0.046784
sre_compile._mk_bitmap
calls: 6
total time: 0.012711
sre_compile._identityfunction
calls: 44
total time: 0.001129
Generated by Python Call Graph v0.4.0
http://pycallgraph.slowchop.com

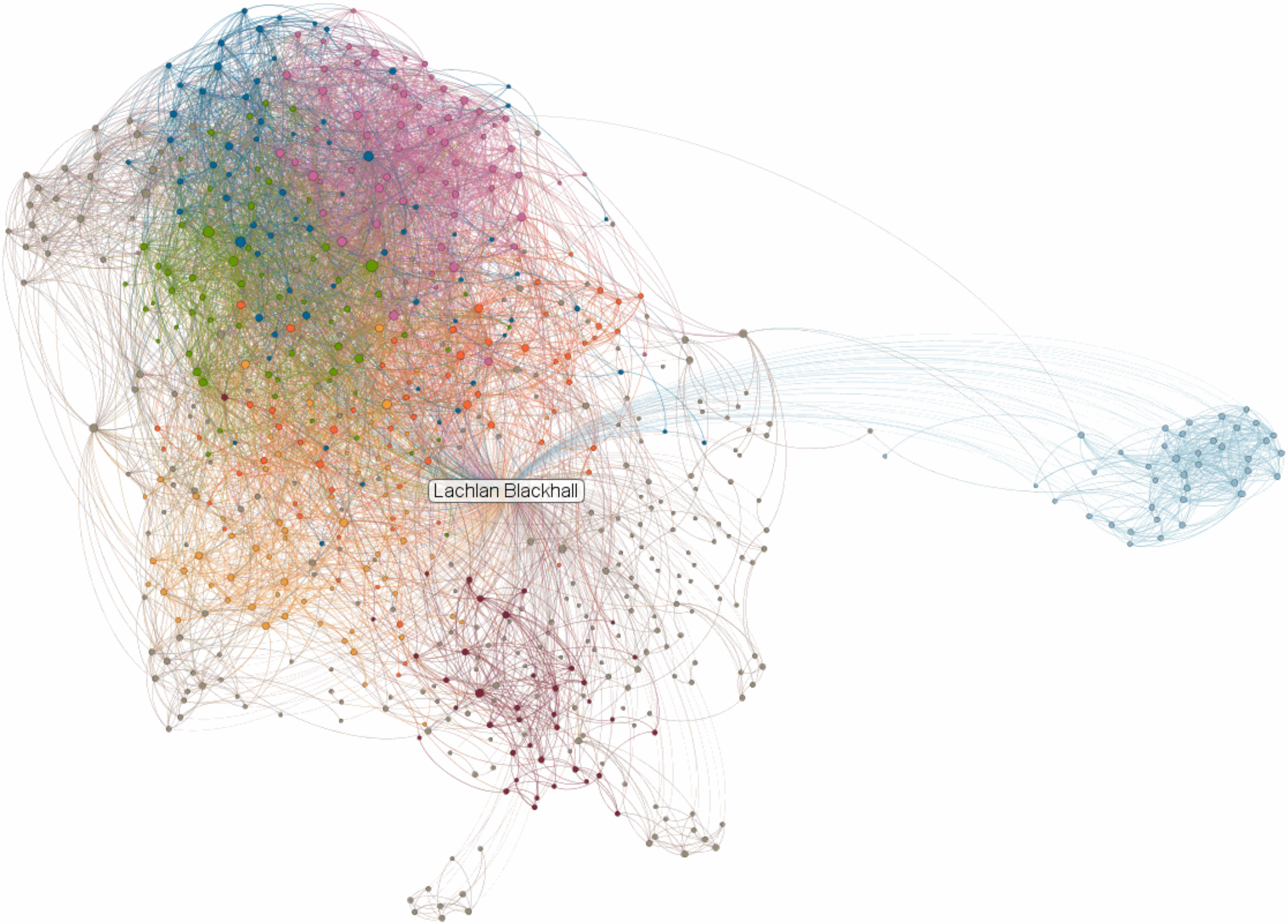



Duration of Travel
<1
(hours)
>20
BIO
DIASPORA

LinkedIn Maps

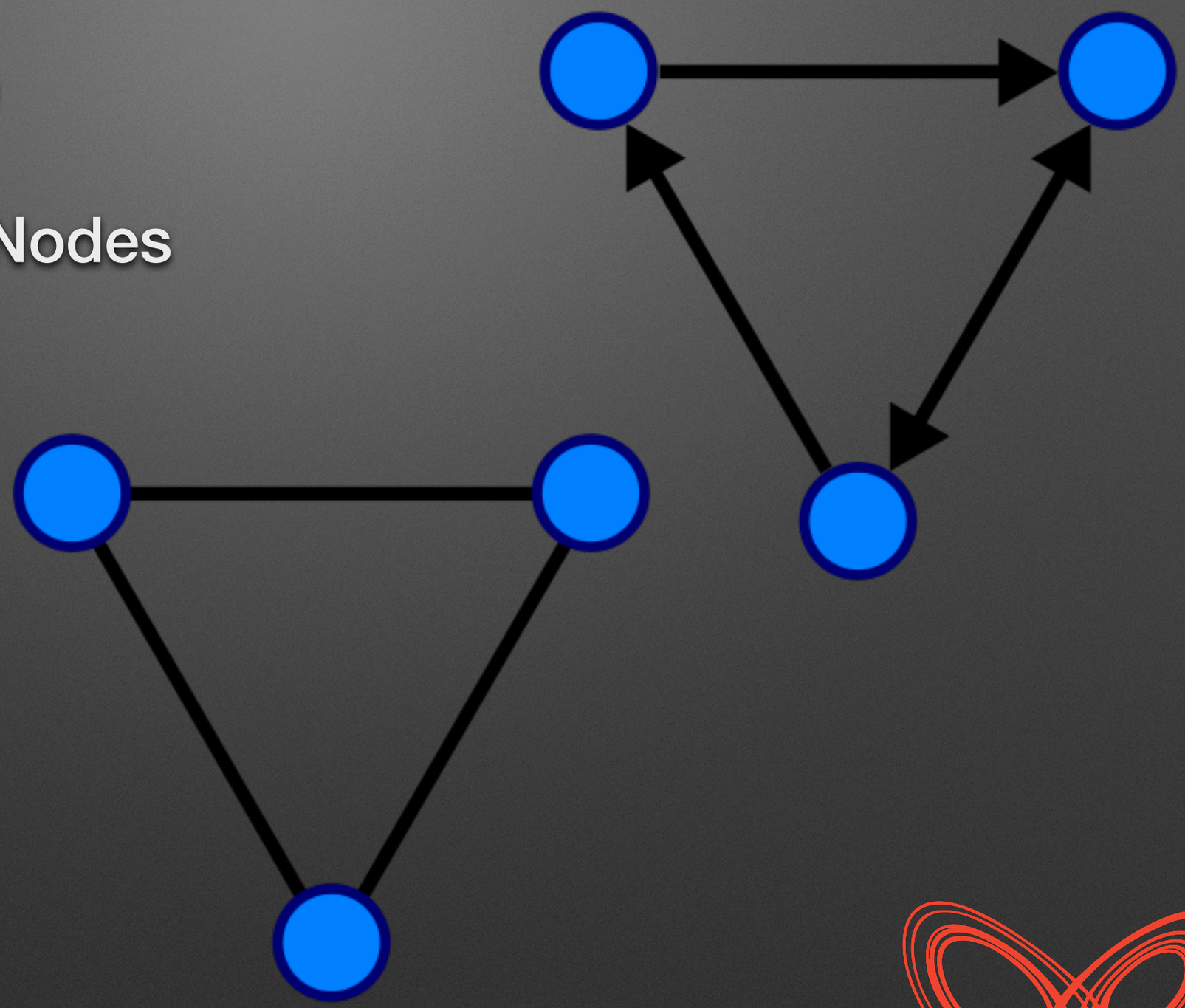
Lachlan Blackhall's Professional Network

as of July 30, 2014



Graphs

- Graph, G = (V, E)
  - V = Vertices / Nodes
  - E = Edges



NetworkX

- Native graph structures for Python.
  - Nodes can be any hashable object.
  - Edges are tuples of nodes with optional edge data which is stored in a dictionary.
  - Well maintained package

Lets make a Graph

```
import networkx as nx
import matplotlib.pyplot as plt

G = nx.Graph()
```

Lets add some nodes

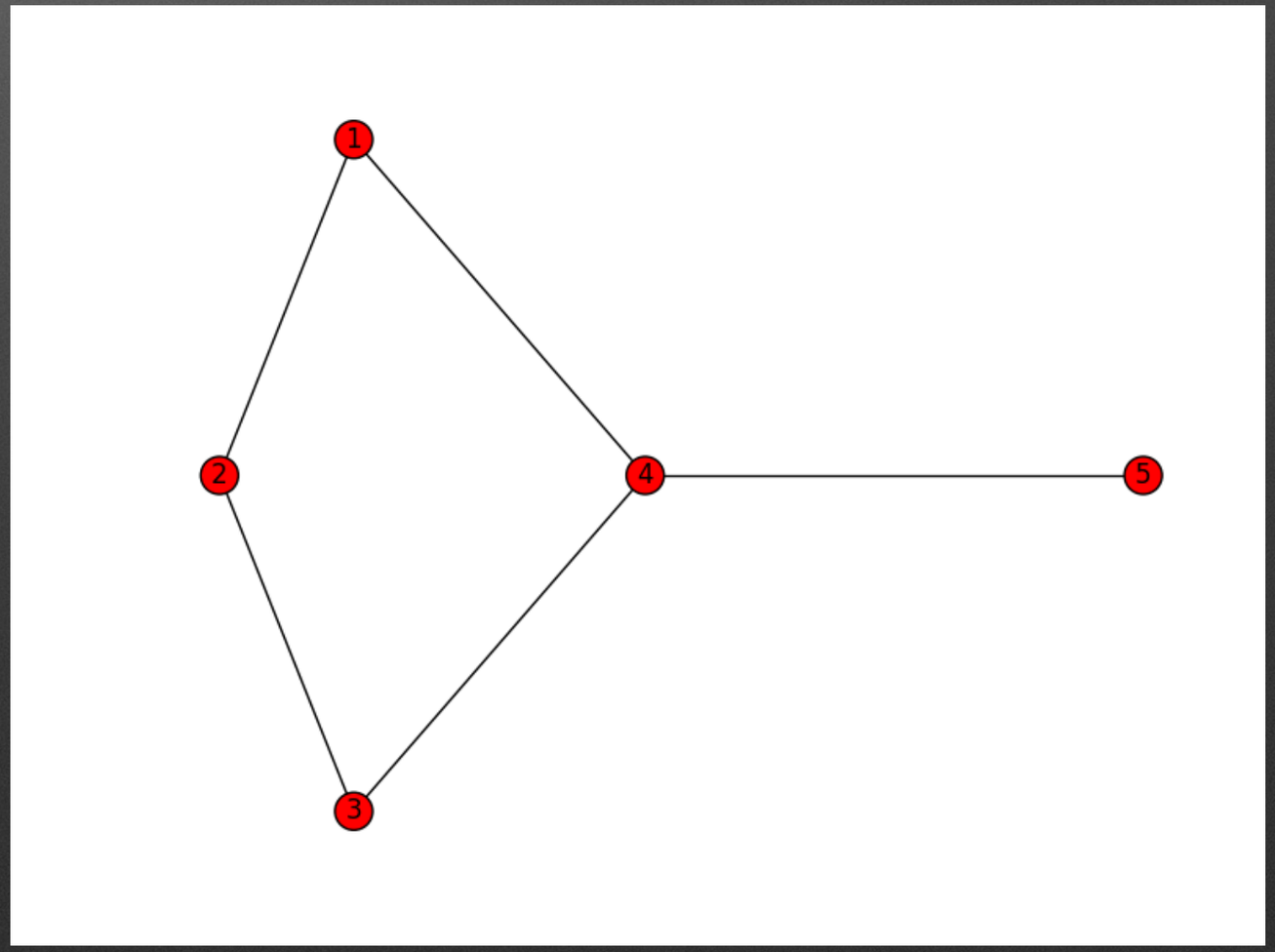
```
import networkx as nx
import matplotlib.pyplot as plt

G = nx.Graph()
G.add_node('1')
G.add_node('2')
G.add_node('3')
G.add_node('4')
G.add_node('5')
```

Lets add some edges

```
import networkx as nx
import matplotlib.pyplot as plt

G = nx.Graph()
G.add_node('1')
G.add_node('2')
G.add_node('3')
G.add_node('4')
G.add_node('5')
G.add_edge('1', '2')
G.add_edge('2', '3')
G.add_edge('3', '4')
G.add_edge('4', '1')
G.add_edge('4', '5')
```

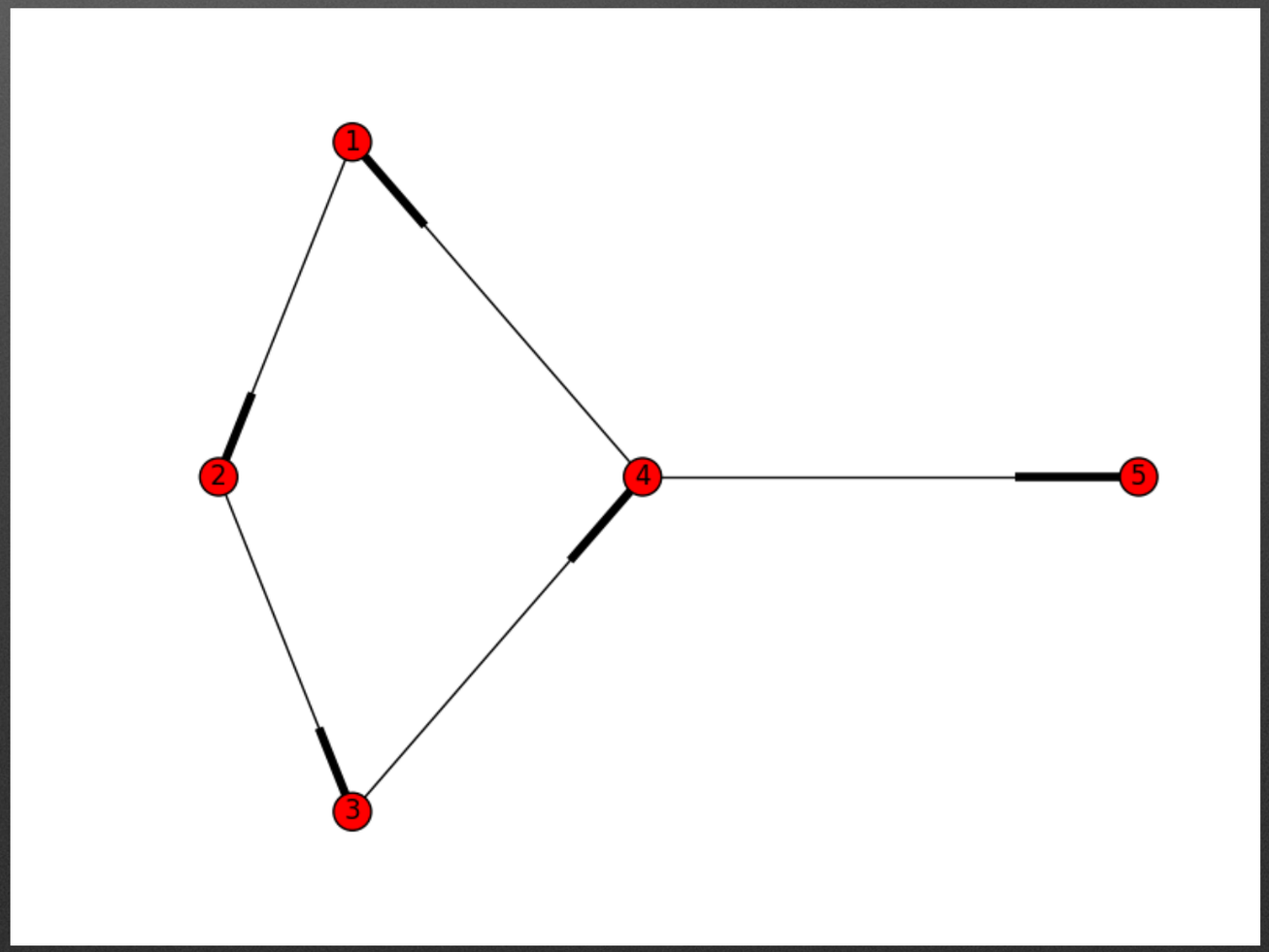
Lets Visualise it

```
import networkx as nx
import matplotlib.pyplot as plt

G = nx.Graph()
G.add_node('1')
G.add_node('2')
G.add_node('3')
G.add_node('4')
G.add_node('5')
G.add_edge('1', '2')
G.add_edge('2', '3')
G.add_edge('3', '4')
G.add_edge('4', '1')
G.add_edge('4', '5')
nx.draw_spectral(G)
plt.show()
```



Now a directed graph

```
import networkx as nx
import matplotlib.pyplot as plt

G = nx.DiGraph()
G.add_nodes_from(['1', '2', '3', '4', '5'])
G.add_edge('1', '2')
G.add_edge('2', '3')
G.add_edge('3', '4')
G.add_edge('4', '1')
G.add_edge('4', '5')
nx.draw_spectral(G)
plt.show()
```



Look Ma - No Nodes

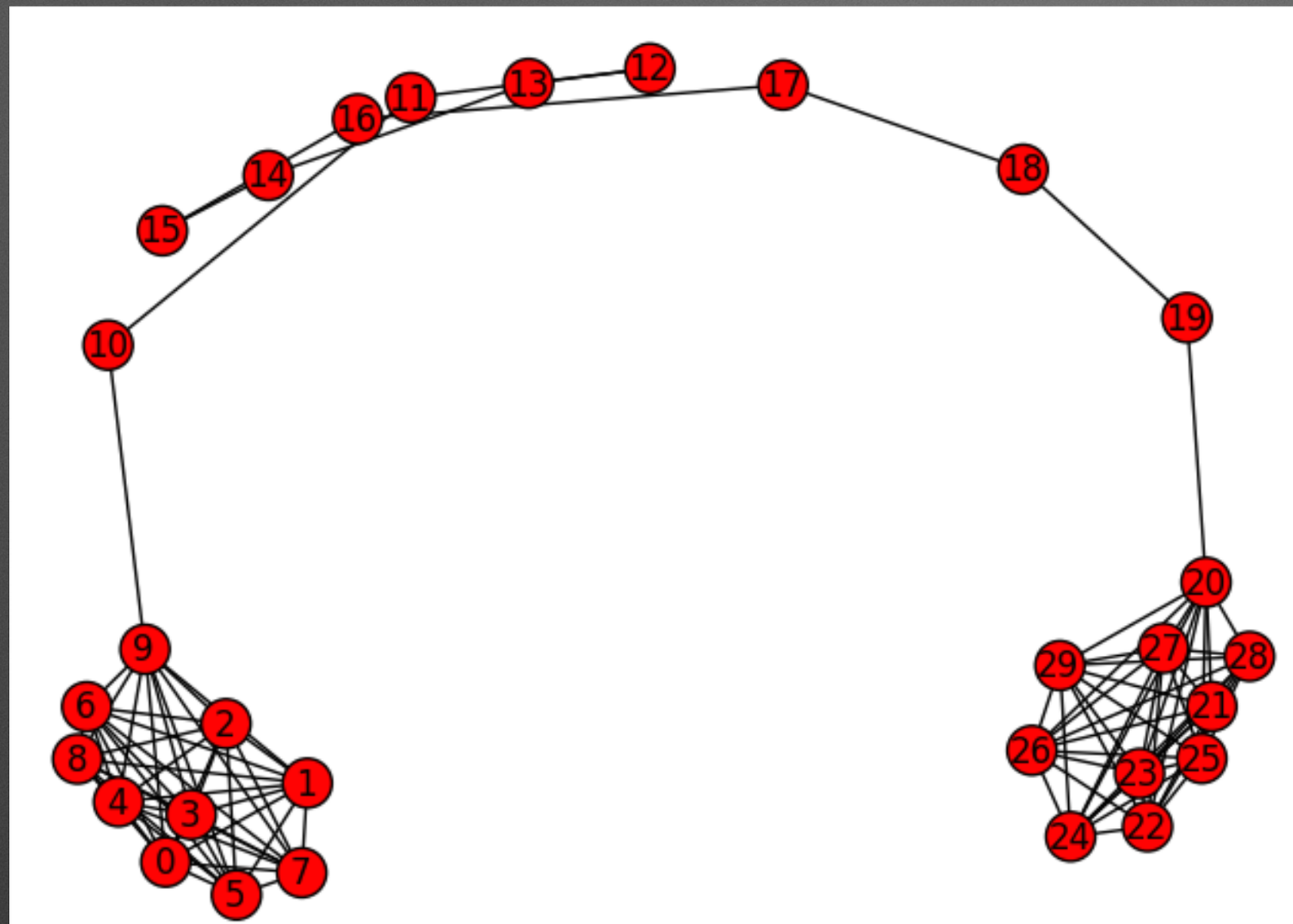
```
import networkx as nx

G = nx.DiGraph()
G.add_edge('1', '2')
G.add_edge('2', '3')
G.add_edge('3', '4')
G.add_edge('4', '1')
G.add_edge('4', '5')
```

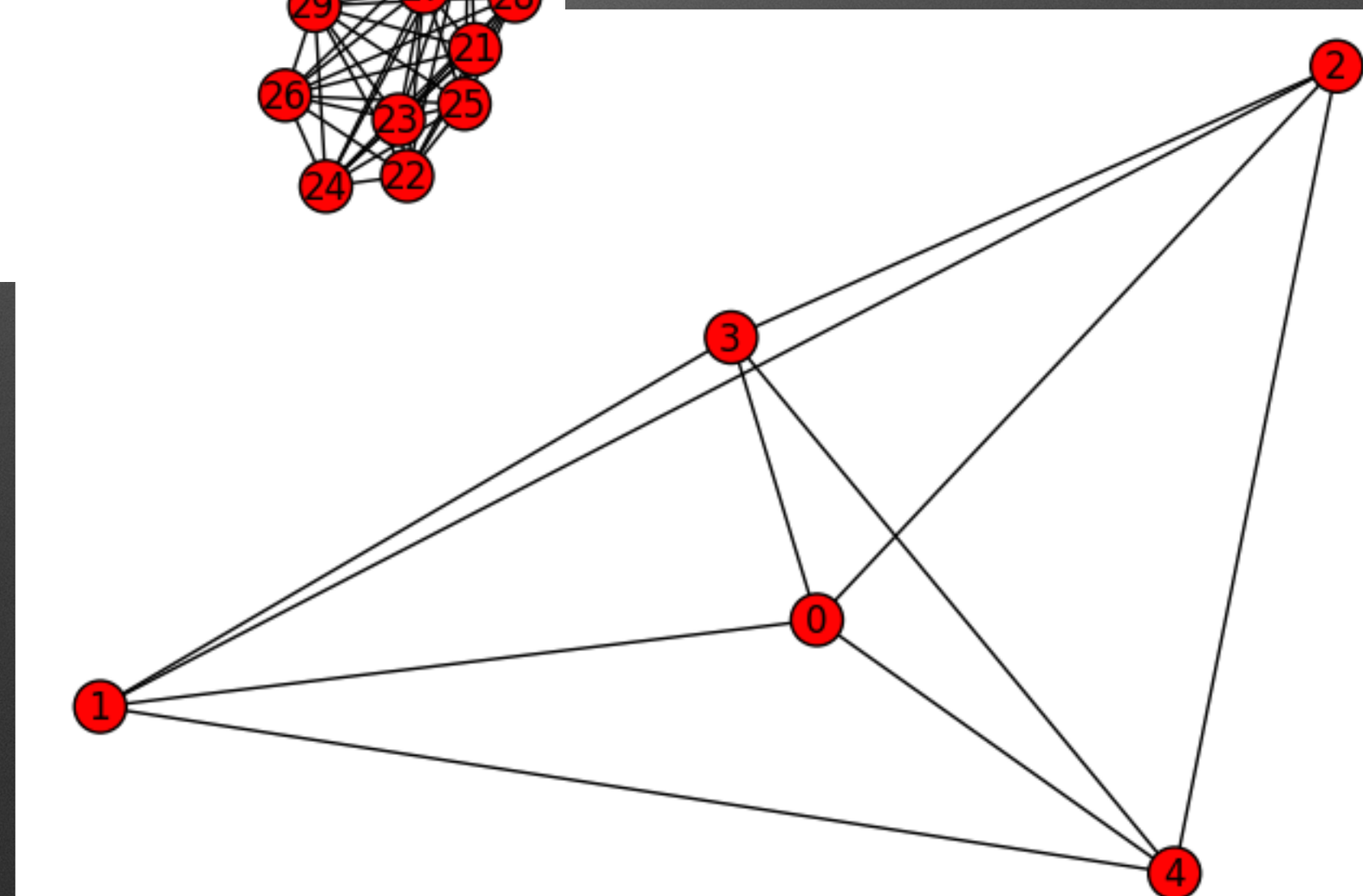
More Graphs

- Networkx can generate lots of interesting graphs to experiment with.
- Lets have a look at a few of them.

More Graphs

```
import networkx as nx
import matplotlib.pyplot as plt

K_5 = nx.complete_graph(5)
nx.draw(K_5)
plt.show()


barbell = nx.barbell_graph(10, 10)
nx.draw(barbell)
plt.show()
```


12
13
11
16
17
14
18
15
10
19
20
9
27
28
29
6
2
21
8
1
26
25
23
4
3
24
22
0
7
5

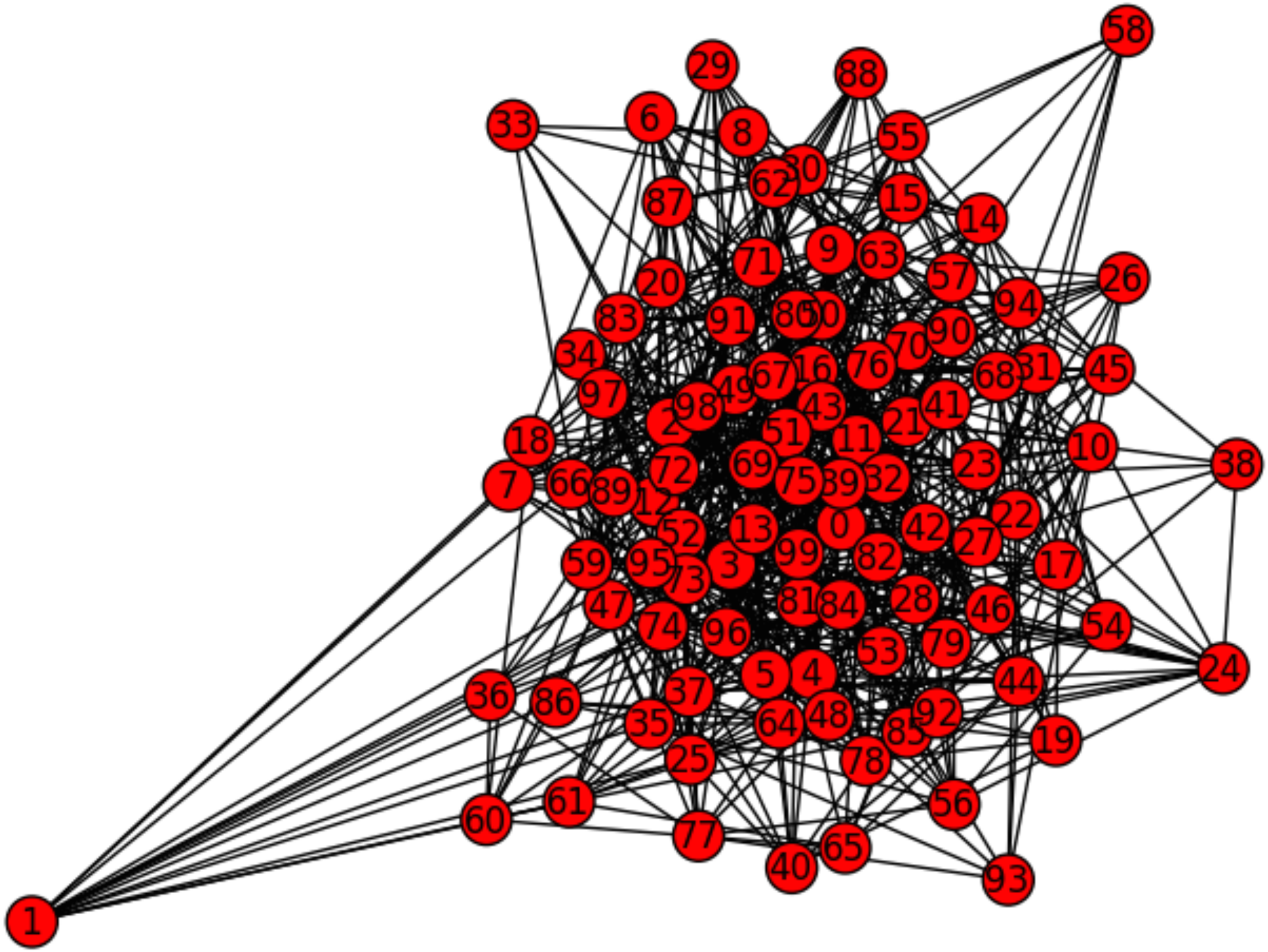

2
3
0
1
4

Erdos-Renyi

- Named for Paul Erdős and Alfréd Rényi.
- The Erdős–Rényi model is a model for generating random graphs.
- The model sets an edge between each pair of nodes with equal probability.

Erdos-Renyi

```
import networkx as nx
import matplotlib.pyplot as plt

er = nx.erdos_renyi_graph(100, 0.15)
nx.draw(er)
plt.show()
```

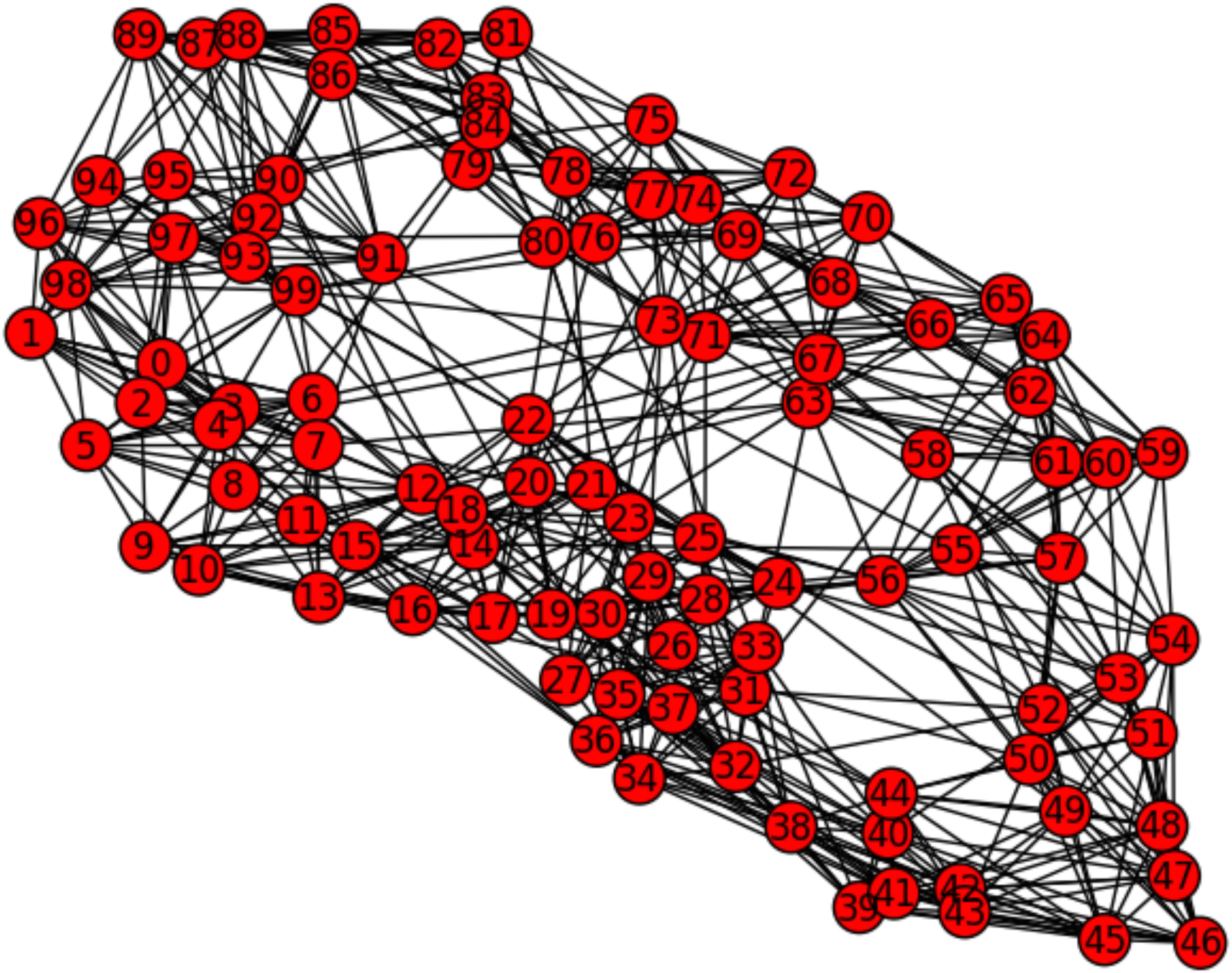

58
29
88
33
6
8
55
30
62
87
15
14
9
63
71
57
26
20
94
83
91
86
50
90
70
34
76
68
31
45
67
16
97
49
98
43
41
21
2
51
11
18
10
38
23
69
72
75
39
32
7
66
89
12
22
52
13
0
42
27
99
82
59
95
3
17
73
81
84
28
46
47
74
96
54
53
79
5
4
24
44
36
86
37
35
64
48
92
83
19
25
78
61
56
60
77
65
40
93
1

Watts-Strogatz

- Named for Duncan J. Watts and Steven Strogatz.
- The Watts–Strogatz model produces graphs with small-world properties, including short average path lengths and high clustering.
- Most nodes are not neighbours but can be reached from every other node by a small number of hops or steps.

Watts-Strogatz

```
import networkx as nx
import matplotlib.pyplot as plt

ws = nx.watts_strogatz_graph(100, 15, 0.1)
nx.draw(ws)
plt.show()
```

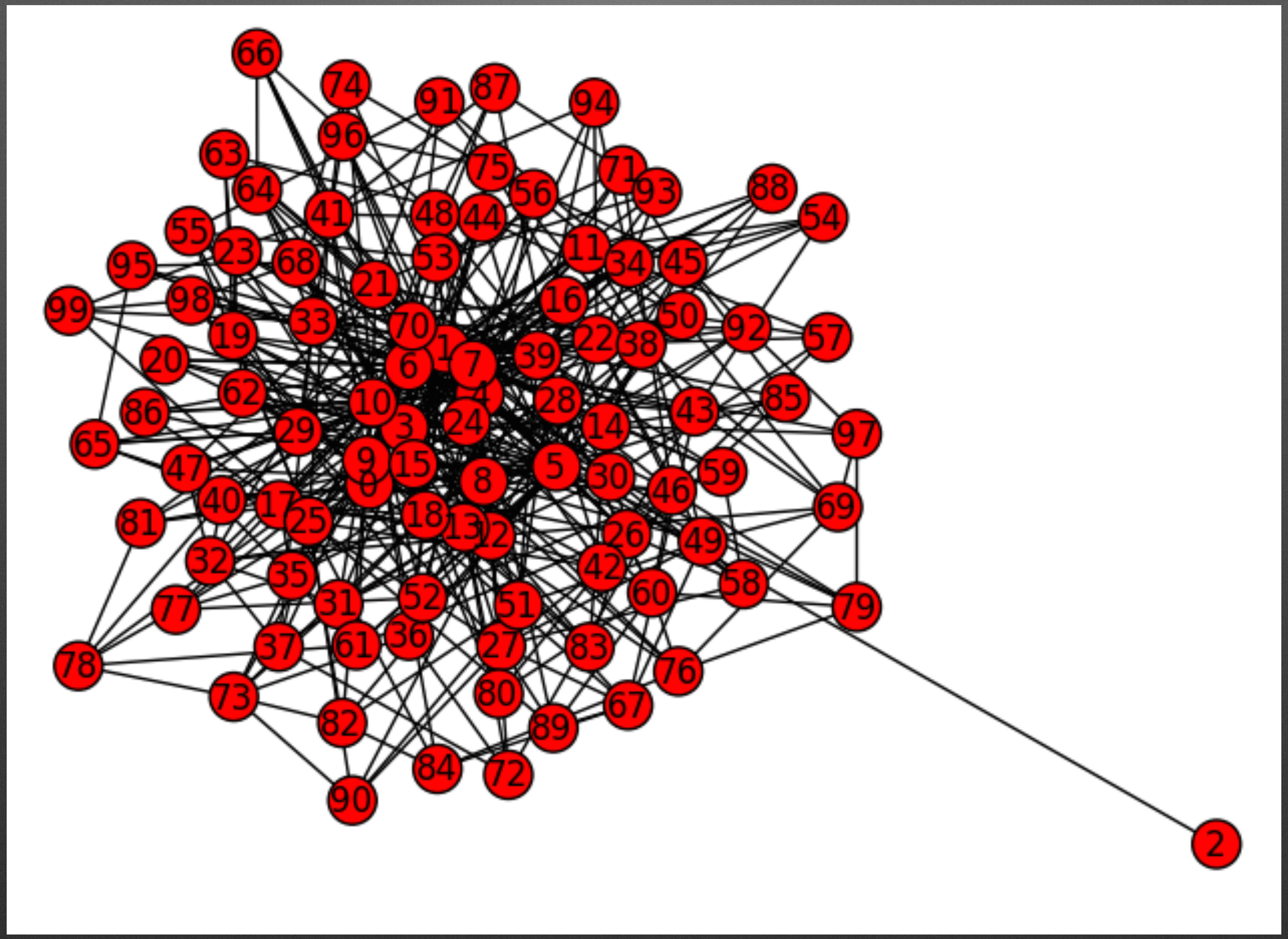

89
87
88
85
82
81
86
83
84
75
79
78
72
94
95
90
77
74
70
96
92
97
93
91
80
76
69
98
99
68
65
1
73
71
66
64
0
67
2
3
6
63
62
4
22
5
7
58
61
60
59
8
12
20
21
18
23
11
25
9
15
14
55
57
10
29
24
56
13
16
17
19
30
28
54
26
33
27
35
31
53
37
52
36
51
50
34
32
44
49
38
40
48
47
41
42
39
43
45
46

Barabasi - Albert

- Named for Albert-László Barabási and Réka Albert.
- The Barabási–Albert model is an algorithm for generating random scale-free networks.
- Scale free describes the distribution of node degrees of the network.

Barabasi - Albert

```
import networkx as nx
import matplotlib.pyplot as plt

ba = nx.barabasi_albert_graph(100, 5)
nx.draw(ba)
plt.show()
```


66
74
91
87
94
96
63
75
71
93
88
64
56
41
48
44
54
55
23
68
53
11
34
45
95
21
16
99
98
33
70
50
92
57
19
22
38
20
6
1
7
39
62
10
4
28
85
86
3
24
14
43
97
29
65
9
15
5
30
59
47
0
8
46
40
17
18
69
81
25
13
12
26
49
32
42
35
58
60
79
77
31
52
51
37
61
36
27
83
78
76
73
80
67
82
89
84
72
90
2

Social Network Analyis

- One major area of interest in network analysis.
- Networkx is well suited to this type of analysis.
- Interested in understanding graph properties that explain the social interaction.

Enron Data

- Enron declared bankruptcy in 2001
- Enron email corpus contains data from about 150 users, mostly senior management of Enron.
- Converted the corpus to a TSV file of the form

SENDER	RECIPIENTS	EMAIL DATA …
str	str,…,str	

A Graph

```
import csv
import networkx as nx

G = nx.Graph()
```

Load the email data

```
import csv
import networkx as nx

G = nx.Graph()

with open('enron.tsv', 'rb') as tsvin:
    tsvin = csv.reader(tsvin, delimiter='\t')
```

Build the graph

```
import csv
import networkx as nx

G = nx.Graph()

with open('enron.tsv', 'rb') as tsvin:
    tsvin = csv.reader(tsvin, delimiter='\t')

    for row in tsvin:
        sender = row[0]
        recipients = row[1].split(',')
        for recipient in recipients:
            G.add_edge(sender, recipient)
```

And visualise!!!

```
import csv
import networkx as nx

G = nx.Graph()

with open('enron.tsv', 'rb') as tsvin:
    tsvin = csv.reader(tsvin, delimiter='\t')

    for row in tsvin:
        sender = row[0]
        recipients = row[1].split(',')
        for recipient in recipients:
            G.add_edge(sender, recipient)

nx.write_gexf(G, 'enron.gexf')
```

So what?

- Visualisations are pretty but…
  - Networkx can help us do better
  - Lets dive into some graph analysis

Node Degree

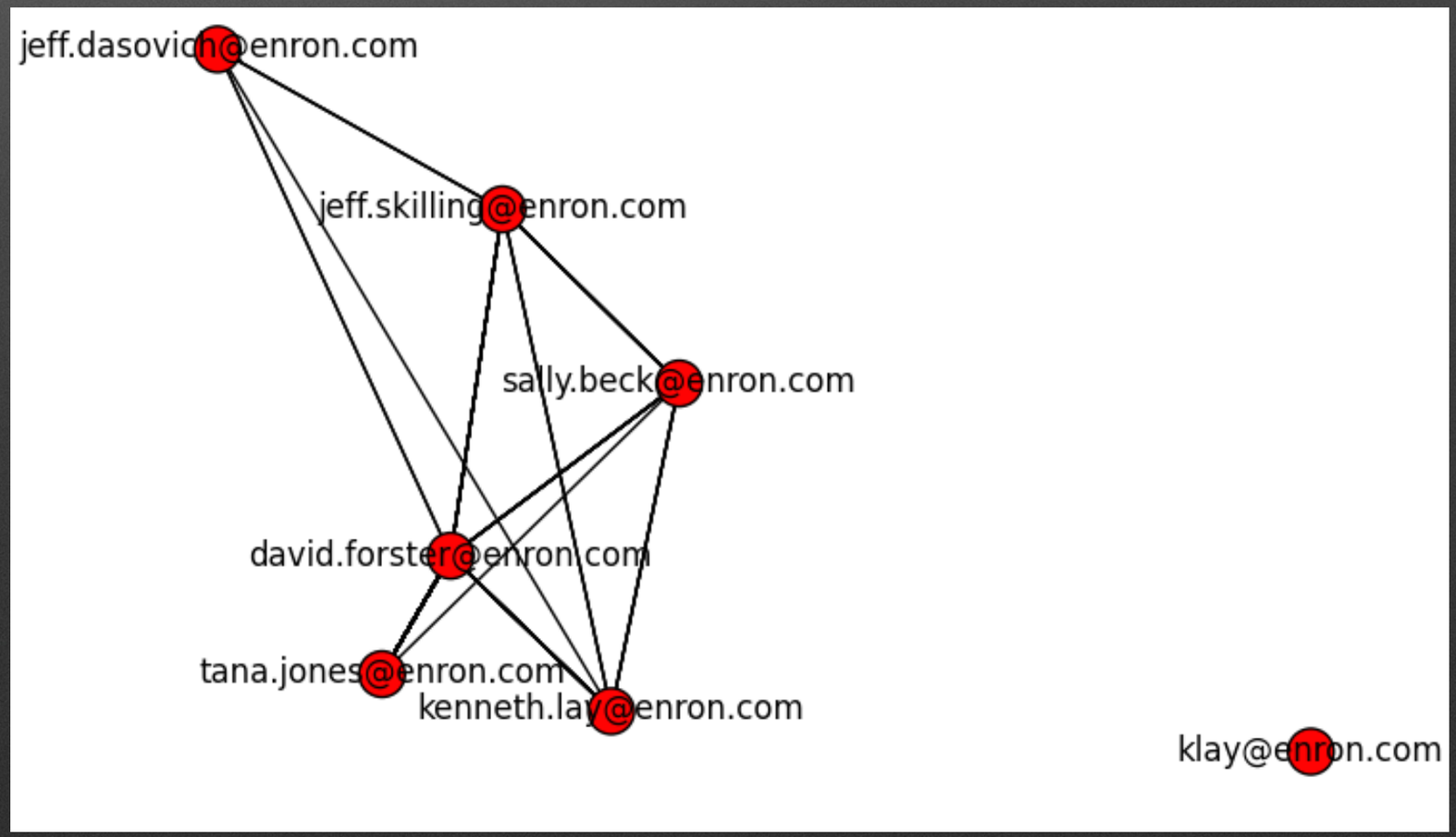
```
node_degree = nx.degree(G).items()
sorted_degrees = sorted(node_degree, key=lambda tup: tup[1])
poi =  [email for (email, degree) in sorted_degrees[-10:]]

>>> ['tana.jones@enron.com', 'technology.enron@enron.com', '',
'klay@enron.com', 'jeff.skilling@enron.com',
'david.forster@enron.com', 'jeff.dasovich@enron.com',
'outlook.team@enron.com', 'sally.beck@enron.com',
'kenneth.lay@enron.com']
```

The People

- Kenneth Lay - Chairman
- Jeff Skilling - CEO
- Tana Jones - Senior Legal Specialist
- Sally Beck - COO
- David Forster - Vice President
- Jeff Dasovich - Government Relation Executive

Subgraphs

```
real_poi = ['tana.jones@enron.com', 'klay@enron.com',
'jeff.skilling@enron.com', 'david.forster@enron.com',
'jeff.dasovich@enron.com',  'sally.beck@enron.com',
'kenneth.lay@enron.com']

sub_G = G.subgraph(real_poi)
nx.draw(sub_G)
plt.show()
```



MultiGraphs

- Allows you to have multiple edges between nodes. i.e. one edge per email.

- Instead of

```
G = nx.Graph()
```

- Use

```
G = nx.MultiGraph()
```

Relationships?

- Neighbours

```
G.neighbors('kenneth.lay@enron.com')
G.neighbors_iter('kenneth.lay@enron.com')
```

- Cliques

```
nx.find_cliques(G)
```

Networkx Functionality

NetworkX

NetworkX Home | Documentation | Download | Developer (Github)

High-productivity software for complex networks

NetworkX is a Python language software package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.

Documentation
all documentation

Examples
using the library

Reference
all functions and methods

Features

- Python language data structures for graphs, digraphs, and multigraphs.
- Nodes can be "anything" (e.g. text, images, XML records)

Versions

Latest Release

1.9 - 21 June 2014
downloads | docs | pdf

Development

2.0dev
github | docs | pdf
build passing
coverage 90%

Contact

Mailing list
Issue tracker
Developer guide

Other useful graph Packages

- Gephi (http://gephi.github.io/)
- Pajek (http://pajek.imfm.si/doku.php)

Conclusion

- Graphs are good
- Networkx is awesome
- Have fun :)

Questions

???