Michael Uftring
Indiana University
Network Science, Spring 2018
Week#8 Assignment - Link Communities

Link communities

For this assignment we will take a look at link communities and how they differ from node communities. In order to do this we will use the algorithm discussed in the reading ("Link communities reveal multiscale complexity in networks") and link community video from canvas.

A small python module has been prepared that will allow you to use the link community algorithm with Anaconda and Python 3.5. To install the module you want to open a terminal or shell and use:

```
pip install git+https://github.com/Nathaniel-Rodriguez/linkcom.git
```

This will install the package locally in your Anaconda site-packages directory (the same place where the `conda` command would install new packages). Make sure you have git (https://git-scm.com/) installed first. If you are using Windows you will need to use the Anaconda command prompt when using pip, so that it adds the package to Anaconda. If you have trouble installing the package you can just unpack the zip file from the github repository (https://github.com/Nathaniel-Rodriguez/linkcom/tree/master/linkcom) and put the linkcom folder in your working directory.

To use the package you can do:

In [3]:

```
import linkcom
```

The code has been adapted so that it will work with networkx graphs. The link communities algorithm requires simple undirected graphs to use. This means there can't be any self-loops or parallel edges. However, you can use weighted graphs.

In [4]:

```
# First lets import networkx
import networkx as nx

# And generate a new graph
my_graph = nx.erdos_renyi_graph(100, 0.1)

# We need to make sure this is a graph of type Graph
print(type(my_graph).__name__)
```

```
Graph
```

If the graph you load in isn't of type Graph (it maybe a Multigraph or DiGraph), it is easy to convert it to one:

In [5]:

```
my_graph = nx.Graph(my_graph)
```

Using linkcom

Now lets call the `cluster` method in `linkcom` to cluster the links of the graph. The `cluster` method takes several optional arguments:

```
linkcom.cluster(nx_graph, threshold=None, is_weighted=False, weight_key='w
eight', dendro_flag=False, to_file=False, basename="clustering", delimiter
='\t')
```

`Threshold`: sets the cut-off for the dendrogram.

`is_weighted`: can be `True` or `False` depending upon whether the graph has weights or not. Set this to `True` if the graph is weighted.

`weight_key`: specifies what attribute the edges have that has weight values. In networkx it is convention that this key be set to `weight`. Most functions in networkx will assume this is the key. This is also the default value for the cluster method.

`dendro_flag`: specifies whether to return the dendrogram (only applicable if the graph is unweighted and no threshold is given).

`to_file`: specifies whether to write the outputs to file. Several files will be written and given names based on `basename` with elements separated by `delimiter`.

These outputs will be written to file:

Three text files with extensions .edge2comm.txt, .comm2edges.txt,
and .comm2nodes.txt store the communities.

edge2comm, an edge on each line followed by the community
id (cid) of the edge's link comm:
node_i <delimiter> node_j <delimiter> cid <newline>

comm2edges, a list of edges representing one community per line:
cid <delimiter> ni,nj <delimiter> nx,ny [...] <newline>

comm2nodes, a list of nodes representing one community per line:
cid <delimiter> ni <delimiter> nj [...] <newline>

The output filename contains the threshold at which the dendrogram
was cut, if applicable, or the threshold where the maximum
partition density was found, and the value of the partition
density.

If no threshold was given to cut the dendrogram, a file ending with
'_thr_D.txt' is generated, containing the partition density as a
function of clustering threshold.

If the dendrogram option was given, two files are generated. One with
'.cid2edge.txt' records the id of each edge and the other one with
'.linkage.txt' stores the linkage structure of the hierarchical
clustering. In the linkage file, the edge in the first column is
merged with the one in the second at the similarity value in the
third column.

The cluster method will return a tuple with different elements:

If no threshold is given, then a tuple is returned with: (dict) dictionary
with keys=edges and values=community membership, (float) best similarity,
(float) best partition density, (list) partition density list.

If dendro_flag is given (only applicable if no threshold), then a tuple is
returned with: (dict) dictionary with keys=edges and values=community memb
ership, (float) best similarity, (float) best partition density, (list) pa
rtition density list, (dict) keys=edges and values=community membership fo
r original, (list) dendrogram.

If threshold is given a tuple is returned with: (dict) dictionary with key
s=edges and values=community membership, partition density at threshold.

You will mostly just be interested in using the dictionary which contains the community assignment data, which is always the first element of the tuple. Lets do a short example:

In [6]:

```
e2c, S, D, Dlist = linkcom.cluster(my_graph)
```

```
clustering...
computing similarities...
# D_max = 0.104996
# S_max = 0.192308
```

If we print `e2c` we will see that each edge has a community membership:

In [7]:

```
print(e2c)
```

```
{(25, 82): 40, (6, 28): 1, (36, 68): 2, (2, 84): 3, (6, 54): 4, (73,
82): 43, (42, 62): 337, (55, 80): 7, (40, 66): 268, (43, 63): 9, (30
, 66): 268, (7, 68): 11, (42, 88): 12, (55, 78): 13, (6, 98): 4, (1,
28): 15, (65, 96): 205, (33, 95): 17, (71, 78): 18, (22, 48): 19, (5
6, 58): 20, (23, 53): 142, (8, 38): 22, (48, 77): 23, (2, 73): 24, (
7, 44): 398, (13, 89): 276, (6, 67): 268, (12, 50): 28, (18, 62): 25
4, (50, 56): 30, (2, 12): 31, (46, 52): 32, (3, 17): 33, (68, 97): 3
4, (1, 33): 35, (45, 97): 36, (58, 59): 37, (7, 15): 448, (32, 58):
39, (39, 82): 40, (4, 48): 41, (12, 41): 42, (49, 82): 43, (65, 88):
12, (18, 57): 45, (77, 88): 200, (47, 99): 267, (16, 66): 268, (20,
71): 35, (58, 83): 50, (24, 40): 202, (14, 95): 276, (70, 72): 424,
(0, 81): 54, (74, 97): 366, (13, 37): 126, (31, 79): 57, (65, 81): 5
8, (18, 48): 111, (24, 99): 267, (55, 85): 61, (12, 62): 62, (10, 26
): 63, (31, 85): 124, (19, 58): 65, (23, 64): 448, (21, 26): 276, (8
2, 99): 40, (37, 80): 126, (36, 44): 398, (54, 82): 43, (2, 76): 24,
(49, 76): 249, (52, 70): 108, (18, 43): 75, (42, 60): 76, (43, 49):
75, (39, 47): 267, (64, 90): 79, (7, 70): 80, (17, 88): 81, (67, 82)
: 82, (19, 67): 83, (44, 80): 84, (7, 92): 337, (0, 9): 86, (24, 54)
: 87, (47, 69): 88, (83, 87): 50, (10, 73): 249, (58, 79): 39, (39,
99): 267, (35, 67): 93, (32, 79): 267, (5, 6): 95, (2, 45): 96, (34,
77): 225, (27, 95): 276, (6, 47): 99, (2, 71): 100, (49, 69): 101, (
31, 61): 124, (8, 20): 324, (65, 69): 379, (76, 81): 105, (9, 17): 1
06, (42, 65): 337, (48, 52): 108, (19, 72): 337, (2, 16): 110, (14,
48): 111, (10, 90): 354, (6, 12): 113, (48, 70): 108, (9, 46): 115,
(27, 84): 116, (15, 64): 448, (38, 88): 28, (12, 43): 119, (24, 30):
267, (5, 68): 121, (42, 72): 337, (22, 81): 123, (12, 31): 124, (19,
55): 125, (37, 97): 126, (59, 87): 37, (1, 56): 128, (6, 7): 129, (1
9, 29): 130, (37, 75): 126, (0, 83): 132, (11, 67): 268, (9, 87): 13
4, (15, 73): 254, (32, 41): 276, (18, 20): 137, (6, 51): 95, (37, 38
): 126, (6, 85): 113, (38, 53): 126, (13, 23): 142, (68, 81): 143, (
60, 86): 192, (45, 81): 145, (22, 61): 146, (73, 76): 249, (2, 80):
148, (5, 37): 149, (11, 94): 150, (29, 40): 151, (30, 55): 267, (73,
86): 254, (51, 78): 154, (64, 92): 337, (22, 99): 146, (7, 64): 448,
(19, 69): 379, (50, 61): 28, (80, 82): 160, (32, 65): 202, (91, 96):
162, (6, 11): 268, (72, 97): 126, (82, 88): 328, (6, 45): 95, (4, 53
```

): 167, (51, 87): 168, (8, 48): 398, (9, 13): 276, (18, 32): 276, (1
0, 20): 172, (41, 89): 173, (67, 99): 174, (43, 95): 367, (9, 74): 2
84, (0, 94): 177, (25, 74): 146, (39, 78): 267, (5, 49): 180, (36, 5
8): 208, (17, 54): 182, (7, 63): 448, (65, 92): 337, (37, 59): 185,
(17, 72): 186, (39, 53): 450, (0, 25): 188, (40, 96): 205, (1, 4): 1
90, (16, 92): 205, (15, 60): 192, (17, 98): 182, (69, 88): 12, (24,
44): 398, (33, 71): 196, (0, 85): 197, (29, 51): 448, (37, 83): 126,
(61, 77): 200, (9, 63): 201, (24, 65): 202, (39, 93): 2, (50, 87): 2
76, (40, 65): 205, (43, 60): 206, (46, 96): 205, (48, 58): 208, (88,
97): 366, (43, 77): 225, (18, 89): 276, (80, 93): 126, (52, 62): 213
, (22, 51): 443, (9, 94): 86, (2, 46): 110, (34, 78): 225, (7, 23):
448, (26, 88): 28, (6, 40): 268, (54, 86): 254, (2, 72): 222, (49, 6
4): 254, (78, 88): 200, (39, 43): 225, (28, 76): 226, (48, 49): 111,
(17, 92): 228, (58, 97): 50, (33, 81): 230, (1, 8): 324, (0, 35): 23
2, (24, 32): 267, (3, 46): 234, (68, 96): 408, (8, 44): 398, (29, 63
): 448, (24, 79): 267, (4, 55): 239, (1, 83): 240, (13, 83): 126, (1
1, 47): 99, (16, 50): 243, (8, 24): 398, (17, 51): 245, (28, 75): 37
9, (18, 56): 276, (14, 18): 276, (10, 18): 249, (16, 65): 205, (11,
27): 268, (7, 81): 252, (18, 75): 276, (15, 49): 254, (21, 78): 255,
(61, 88): 124, (59, 72): 185, (33, 61): 258, (81, 91): 259, (22, 25)
: 146, (79, 88): 261, (4, 68): 262, (49, 84): 263, (13, 38): 126, (3
7, 57): 126, (12, 61): 124, (39, 55): 267, (11, 16): 268, (20, 77):
269, (0, 27): 270, (68, 72): 271, (21, 87): 272, (85, 99): 61, (46,
65): 205, (57, 79): 275, (14, 75): 276, (23, 63): 448, (21, 27): 276
, (73, 79): 279, (2, 85): 31, (31, 47): 281, (63, 79): 282, (26, 95)
: 276, (9, 61): 284, (16, 32): 285, (1, 71): 35, (50, 85): 28, (28,
89): 276, (18, 42): 254, (55, 83): 290, (6, 73): 4, (15, 25): 339, (
0, 18): 293, (66, 67): 268, (68, 93): 2, (51, 63): 448, (57, 80): 12
6, (39, 96): 408, (27, 38): 276, (75, 96): 300, (48, 78): 23, (39, 8
4): 302, (88, 94): 387, (50, 76): 304, (11, 40): 268, (79, 84): 306,
(3, 94): 307, (36, 48): 398, (17, 56): 106, (9, 18): 276, (43, 68):
225, (27, 63): 356, (60, 71): 313, (48, 69): 314, (77, 99): 267, (16
, 30): 268, (39, 77): 267, (9, 49): 276, (18, 28): 276, (75, 85): 32
0, (28, 69): 379, (16, 67): 268, (41, 93): 323, (1, 7): 324, (22, 74
): 146, (14, 92): 326, (59, 84): 327, (12, 82): 328, (62, 92): 337,
(34, 56): 330, (10, 49): 249, (51, 70): 332, (50, 88): 28, (20, 97):
334, (20, 79): 335, (44, 60): 336, (42, 92): 337, (18, 64): 254, (4,
25): 339, (9, 89): 276, (5, 38): 149, (6, 53): 342, (72, 95): 455, (
44, 59): 344, (1, 17): 128, (78, 97): 346, (24, 55): 267, (46, 78):
348, (71, 75): 349, (52, 58): 208, (32, 64): 254, (12, 64): 62, (33,
39): 353, (10, 35): 354, (28, 82): 355, (16, 63): 356, (36, 50): 357
, (45, 51): 358, (7, 84): 359, (47, 77): 267, (12, 97): 366, (18, 76
): 249, (58, 87): 37, (4, 15): 364, (32, 71): 365, (61, 97): 366, (2
1, 43): 367, (26, 38): 276, (7, 8): 398, (22, 32): 370, (2, 95): 371
, (6, 57): 372, (3, 73): 373, (38, 51): 149, (41, 67): 375, (60, 98)
: 376, (10, 66): 377, (57, 74): 378, (19, 28): 379, (52, 54): 380, (
11, 66): 268, (9, 86): 382, (27, 44): 276, (39, 68): 2, (17, 22): 38
5, (48, 84): 386, (11, 88): 387, (9, 56): 106, (18, 21): 276, (17, 6
6): 390, (65, 72): 337, (47, 83): 392, (20, 56): 128, (57, 83): 126,
(25, 85): 61, (86, 95): 396, (37, 93): 126, (8, 36): 398, (7, 20): 3
24, (61, 71): 258, (38, 75): 126, (34, 43): 225, (26, 69): 403, (18,
38): 276, (35, 50): 405, (53, 75): 126, (30, 71): 407, (39, 40): 408
, (1, 23): 324, (69, 75): 379, (17, 93): 411, (41, 87): 412, (33, 82

): 413, (14, 44): 276, (47, 70): 415, (55, 99): 267, (68, 89): 417,
(83, 88): 366, (25, 52): 419, (36, 98): 420, (19, 42): 337, (21, 32)
: 276, (82, 89): 355, (37, 70): 424, (2, 40): 110, (11, 86): 426, (1
6, 27): 268, (75, 80): 126, (77, 90): 438, (7, 86): 448, (32, 99): 2
67, (50, 53): 432, (83, 97): 126, (3, 14): 434, (25, 61): 146, (58,
85): 436, (21, 79): 276, (35, 77): 438, (32, 89): 276, (24, 38): 22,
(60, 67): 441, (25, 39): 267, (5, 22): 443, (36, 93): 2, (13, 77): 4
50, (26, 87): 276, (42, 45): 447, (23, 29): 448, (38, 95): 276, (13,
39): 450, (28, 65): 379, (24, 97): 452, (66, 81): 453, (12, 26): 28,
(21, 72): 455, (44, 95): 276, (32, 80): 457, (14, 32): 276, (18, 95)
: 276, (21, 50): 276, (37, 72): 461}

Since this is a random graph, we expect there not to be any meaningful communities, and indeed they are all labelled `56` (the community ID), so there doesn't appear to be any link communities in the graph.

We can now readily take our results and put them back into our graph so that it can be saved and viewed in Gephi. We can do this using the `set_edge_attributes` (https://networkx.github.io/documentation/stable/reference/generated/networkx.classes.function.set_edge_att highlight=set_edge_attributes#networkx.classes.function.set_edge_attributes) function in networkx. It works just like the `set_node_attributes` function from previuos assignments, but with edges instead:

In [8]:

```
# Put the link communities into the graph
# Note this function's syntax depends on the networkx version.
# If you use networkx 1.9 and below this line should be
# nx.set_edge_attributes(my_graph, "linkcom", e2c)
nx.set_edge_attributes(my_graph, e2c, "linkcom")

# Save the graph to file
nx.write_gexf(my_graph, "my_graph.gexf")
```

```
---------------------------------------------------------------------
-------
TypeError                                 Traceback (most recent cal
l last)
<ipython-input-8-aa4c726efb91> in <module>()
      3 # If you use networkx 1.9 and below this line should be
      4 # nx.set_edge_attributes(my_graph, "linkcom", e2c)
----> 5 nx.set_edge_attributes(my_graph, e2c, "linkcom")
      6
      7 # Save the graph to file

/anaconda/lib/python3.6/site-packages/networkx/classes/function.py
in set_edge_attributes(G, name, values)
    400     else:
    401         for (u, v), value in values.items():
--> 402             G[u][v][name] = value
    403
    404

TypeError: unhashable type: 'dict'
```

In [9]:

```
nx.set_edge_attributes(my_graph, "linkcom", e2c)
```

In [10]:

```
my_graph[0]
```

Out[10]:

```
{9: {'linkcom': 86},
 18: {'linkcom': 293},
 25: {'linkcom': 188},
 27: {'linkcom': 270},
 35: {'linkcom': 232},
 81: {'linkcom': 54},
 83: {'linkcom': 132},
 85: {'linkcom': 197},
 94: {'linkcom': 177}}
```

In [11]:

```
# Save the graph to file
nx.write_gexf(my_graph, "my_graph.gexf")
```

Now if we were to open the graph in Gephi we should be able to use the link communities to color the graph edges. In Gephi you may need to click the little attribute type button so that edge attributes are set to ranked rather than numeric (https://gephi.org/tutorials/gephi-tutorial-quick_start.pdf). This is because we want to color the edges according to their membership and not with a gradient. Additionally, in order to keep the edge colors when saving the graph you will need to make sure the edge color in the Preview tab is set to `original`. Lastly, since you will be looking at link communities (which determine the node membership in link clustering) it will be helpful to increase the size of the edges in Gephi so the colors are more visible and so you can detect nodes that belong to multiple communities.

The network science collaboration graph

You will be using the NetSci collaboration graph for your assignment. The nodes of the graph are people and links are formed between people who co-author a scientific paper together in network science. You can download it from here (http://vlado.fmf.uni-lj.si/pub/networks/data/collab/netscience.htm). Make sure to read the graph description on that page. The graph has ~1500 nodes and is partly disconnected. If you have difficulty working with the full graph or trouble loading it into Gephi, you can use the largest connected component (which only has about 350 nodes). The largest connected component of a graph can be returned from networkx using:

```
largest_component=max(nx.connected_component_subgraphs(my_graph), key=len)
```

Most of the interesting stuff is happening around this component anyway. So you don't lose much except for the scientists and groups that decided to work alone.

Follow these steps for the assignment:

1. Load the Netsci graph and run the link communities clustering algorithm on it.
2. Save the link communities to the graph and save the graph to file.
3. In Gephi choose a good layout for the graph.
4. Run the modularity command to generate communities for the nodes.
5. Color the edges according to the link communities and the nodes according to the communities found by Gephi. Remember to take care in choosing the resolution parameter.
6. How well does link clustering do at detecting community structure? How well does Gephi's node modularity do at detecting community structure? What do you think the communities represent?
7. What are the similarities and differences between the communities detected by either algorithm?
8. Which authors have a prominent position in multiple communities? What do you think these author's roles are?
9. What other features do you notice about the graph that are captured with overlaping communities?
10. Save your visualization to file.
11. Once complete, submit a PDF document to Canvas that contains your responses and your graph visualization (since this is a larger graph feel free to crop the figure so that it only includes parts relevant to your responses).

Answers

Collaboration network in science of networks
M.E.J. Newman, Finding community structure in networks using the eigenvectors of matrices, Preprint physics/0605087 (2006).
http://vlado.fmf.uni-lj.si/pub/networks/data/collab/netscience.htm (http://vlado.fmf.uni-lj.si/pub/networks/data/collab/netscience.htm)

1. Load the Netsci graph and run the link communities clustering algorithm on it

In [12]:

```
graph = nx.read_pajek("netsience.net")
```

In [13]:

```
print(nx.info(graph))
```

```
Name:
Type: MultiGraph
Number of nodes: 1589
Number of edges: 2742
Average degree:   3.4512
```

In [14]:

```
graph = nx.Graph(graph)
```

In [15]:

```
print(nx.info(graph))
```

```
Name:
Type: Graph
Number of nodes: 1589
Number of edges: 2742
Average degree:   3.4512
```

In [16]:

```
e2c, S, D, Dlist = linkcom.cluster(graph)
```

```
clustering...
computing similarities...
# D_max = 0.693791
# S_max = 0.368421
```

2. Save the link communities to the graph and save the graph to file.

In [17]:

```
nx.set_edge_attributes(graph, "linkcom", e2c)
```

In [18]:

```
nx.write_gexf(graph, "network-science-collaboration-network.gexf")
```

3. In Gephi choose a good layout for the graph.

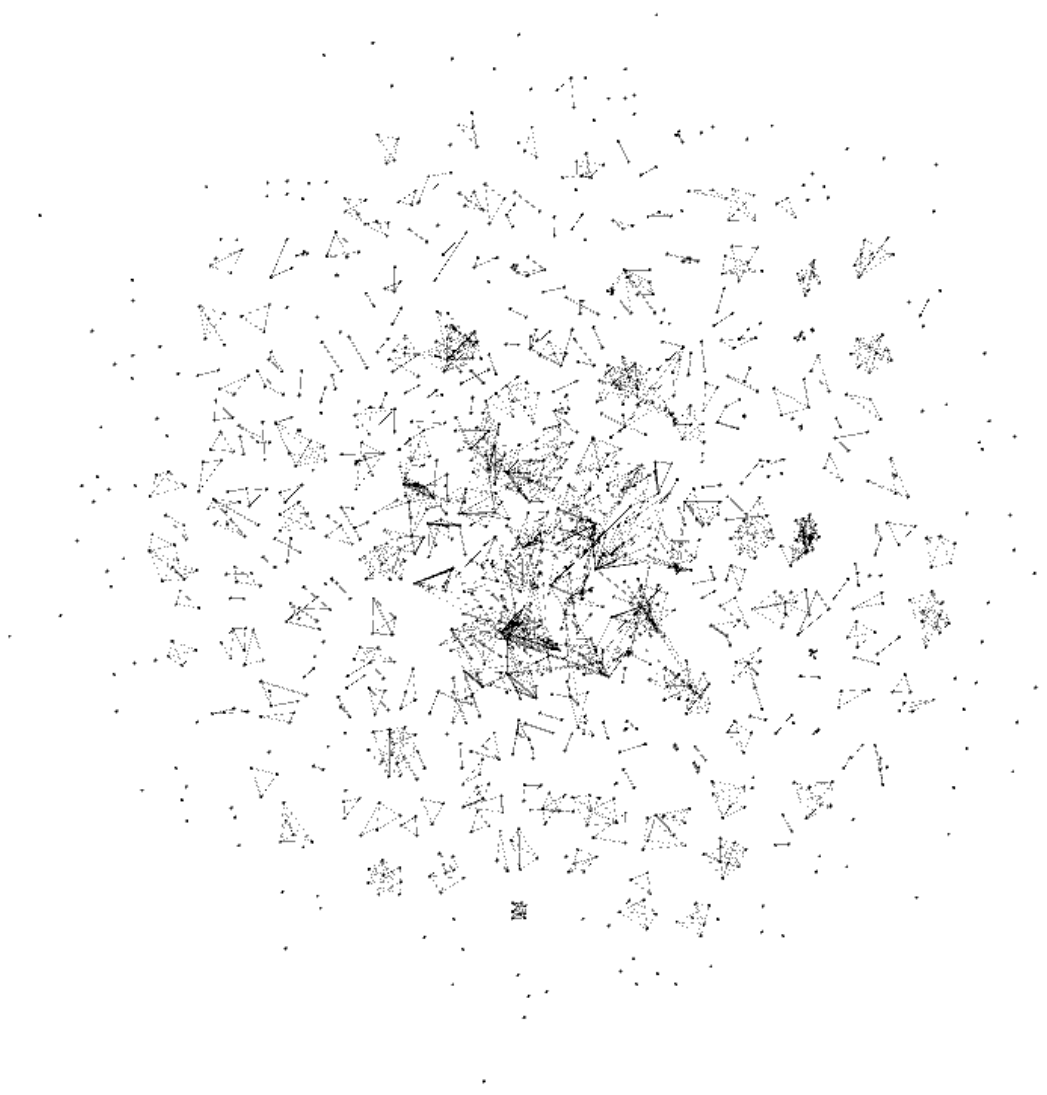
Layout

- *OpenOrd*
- default settings

In [19]:

```
from IPython.display import Image
Image(filename="screenshot_100616.png")
```

Out[19]:



4. Run the modularity command to generate communities for the nodes.

Running **Modularity** with the default *Resolution* produced 406 communities. The objective is to try to reduce the number of communities to something more "manageable," so we try again with increasing *Resolution* values. Running with 10, 100, and 1000 all produced about the same result: the smallest number of communities found was 396. This is probably the best result we can expect given that there are a lot of nodes with very small degree (4 or below, as observed in the Degree Report), and from the visualization (above) we can see many subgraphs disconnected from what must be the largest connected component.

Modularity Report

Parameters:

- Randomize: On
- Use edge weights: On
- Resolution: 1000.0

Results:

- Modularity: 0.825
- Modularity with resolution: 999.825
- Number of Communities: 396

The Modularity Class partitioning shows that there are five classes with a portion greater than 1.0%, and an additional three when looking at portion sizes greater than 0.8%. These are the classes that receive node coloring other than gray. Note that the edges are colored by the source node, which helps to visualize the community structures (which would otherwise be very difficult to perceive with just node coloring).

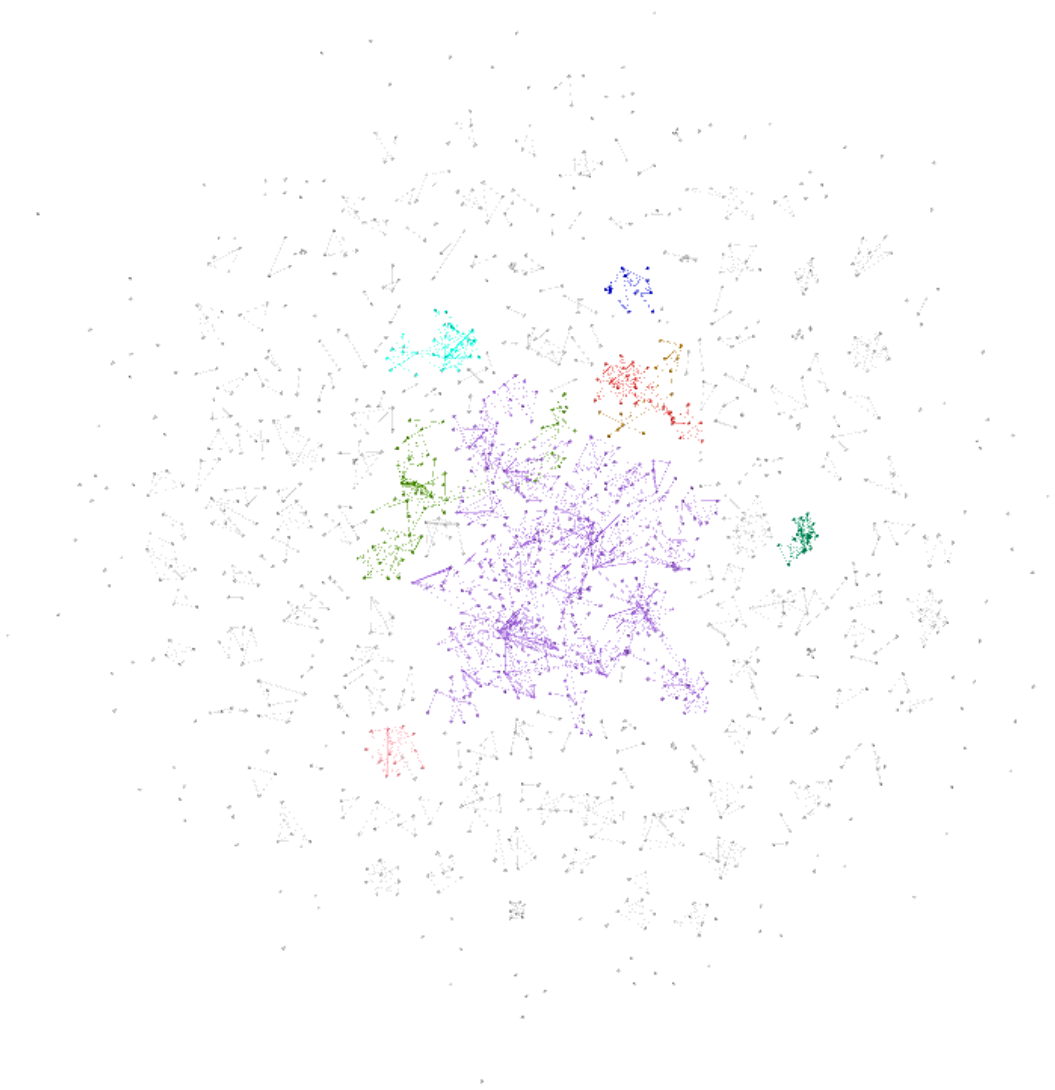
Community	**% of Network**	**Numbner of Nodes**	**Color**	**Description / Location**
1	23.85%	379	purple	massive **large component** in the center
2	3.59%	57	green	mostly to the left of center, some fringing on the top of the large component
3	1.95%	31	light blue	above and left of center
4	1.76%	28	red	above and right of center
5	1.32%	21	blue-green	to the right of center
6	0.88%	14	brown	above and right of center
7	0.88%	14	dark blue	far above the center
8	0.82%	13	pink (salmon)	below and to the left of center

Grouping the nodes by Modularity would be very beneficial, but Gephi does not have a layout algorithm for this. A manual approach for Clustering layout by modularity (http://parklize.blogspot.com/2014/12/gephi-clustering-layout-by-modularity.html) was considered, and experimented with on the random graph, but this network (as a whole) is too large to attempt this operation.

In [20]:

```
Image(filename="screenshot_101820.png")
```

Out[20]:



5. Color the edges according to the link communities and the nodes according to the communities found by Gephi. Remember to take care in choosing the resolution parameter.

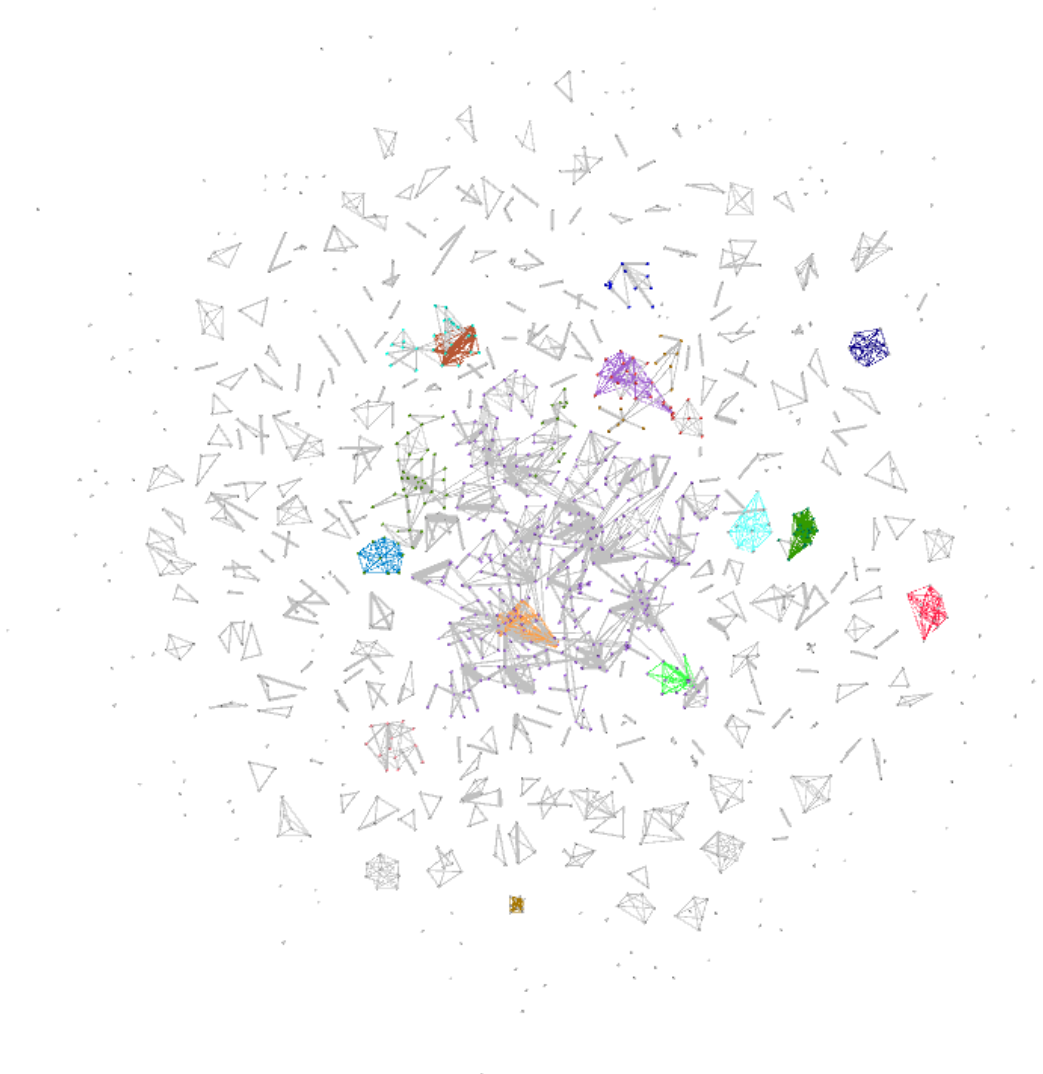
The link community clustering algorithm detected 618 communities. The largest community is just 6.93% of the total network, and the remaining top 10 are all above 1.0%.

Community	% of Network	Numbner of Nodes	Color	Description / Location
1	6.93%	190	green	tightly grouped to the right of center
2	2.26%	62	purple	above and slight to the right of center
3	1.64%	45	red	far right
4	1.64%	45	light blue	to the right and cloeser to the center
5	1.64%	45	brown	far below, straight down from center
6	1.6%	44	fluorescent green	below and to the right
7	1.57%	43	orange	in the lower part of the center region
8	1.31%	36	medium blue	to the left of center
9	1.31%	36	dark blue	to the right, and above center
10	1.28%	35	rust	above center, and slightly to the left

In [21]:

```
Image(filename="screenshot_102706.png")
```

Out[21]:



Repeat steps 1-5, looking at just the largest component.

In [22]:

```
largest_component=max(nx.connected_component_subgraphs(graph), key=len)
```

In [23]:

```
print(nx.info(largest_component))
```

```
Name:
Type: Graph
Number of nodes: 379
Number of edges: 914
Average degree:   4.8232
```

In [24]:

```
e2c, S, D, Dlist = linkcom.cluster(largest_component)
```

```
clustering...
computing similarities...
# D_max = 0.463159
# S_max = 0.416667
```

In [25]:

```
nx.set_edge_attributes(largest_component, "linkcom", e2c)
```

In [26]:

```
nx.write_gexf(largest_component, "network-science-collaboration-network-largest-component.gexf")
```

In Gephi choose a good layout for the graph.

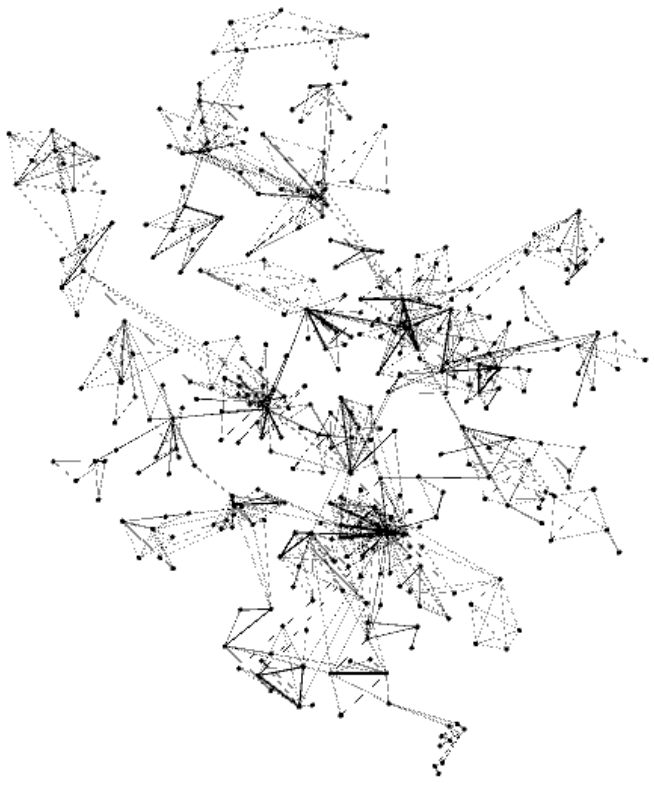
Again, used *OpenOrd* with default parameters to produce the layout.

Here we see a much smaller network, with the same look of the center of the larger network. A subgraph of connected nodes, with some well-connected nodes more towards the center.

In [28]:

```
Image(filename="screenshot_104718.png")
```

Out[28]:



Run the **Modularity** command to generate communities for the nodes.

Modularity Report

Parameters:

- Randomize: On
- Use edge weights: Off
- Resolution: 2.0

Results:

- Modularity: 0.827
- Modularity with resolution: 1.782
- Number of Communities: 11

In [30]:

```
Image(filename="screenshot_105125.png")
```

Out[30]:

Color the edges according to the link communities and the nodes according to the communities found by Gephi.

There are 243 link communities, we will color just the top 10.

In [31]:

```
Image(filename="screenshot_110427.png")
```

Out[31]:

6. How well does link clustering do at detecting community structure? How well does Gephi's node modularity do at detecting community structure? What do you think the communities represent?

As expected, Link Clustering found many smaller groups. These communities probably represent research and publication in the same area (the same or closely related topics) either by direct collaboration (a team) or "competing" groups.

7. What are the similarities and differences between the communities detected by either algorithm?

Link Clustering does a better job at finding smaller communities, which is expected. In order to detect even close to the same number of communities with Modularity, the Resolution had to be reduced to 0.001; at this point the community size was 2 or 3 members, which is not very meaningful. Visualy we can see that some of these pairs found by Modularity were in the same communities identified by Link Clustering.

8. Which authors have a prominent position in multiple communities? What do you think these author's roles are?

The visualization produced (below) does not show any authors in multiple communities, which seems rather unusual.

However, looking closer at the purple and brown communities (above the center, near the top of the visualization) we find Kurths, J (purple) and Mancini, H (brown). Each have a strong relationship with others in their community - judging by the number of links, and they have a relationship with each other. Perhaps this should have been a dual-community membership for both.

9. What other features do you notice about the graph that are captured with overlaping communities?

There were no overlapping communities found.

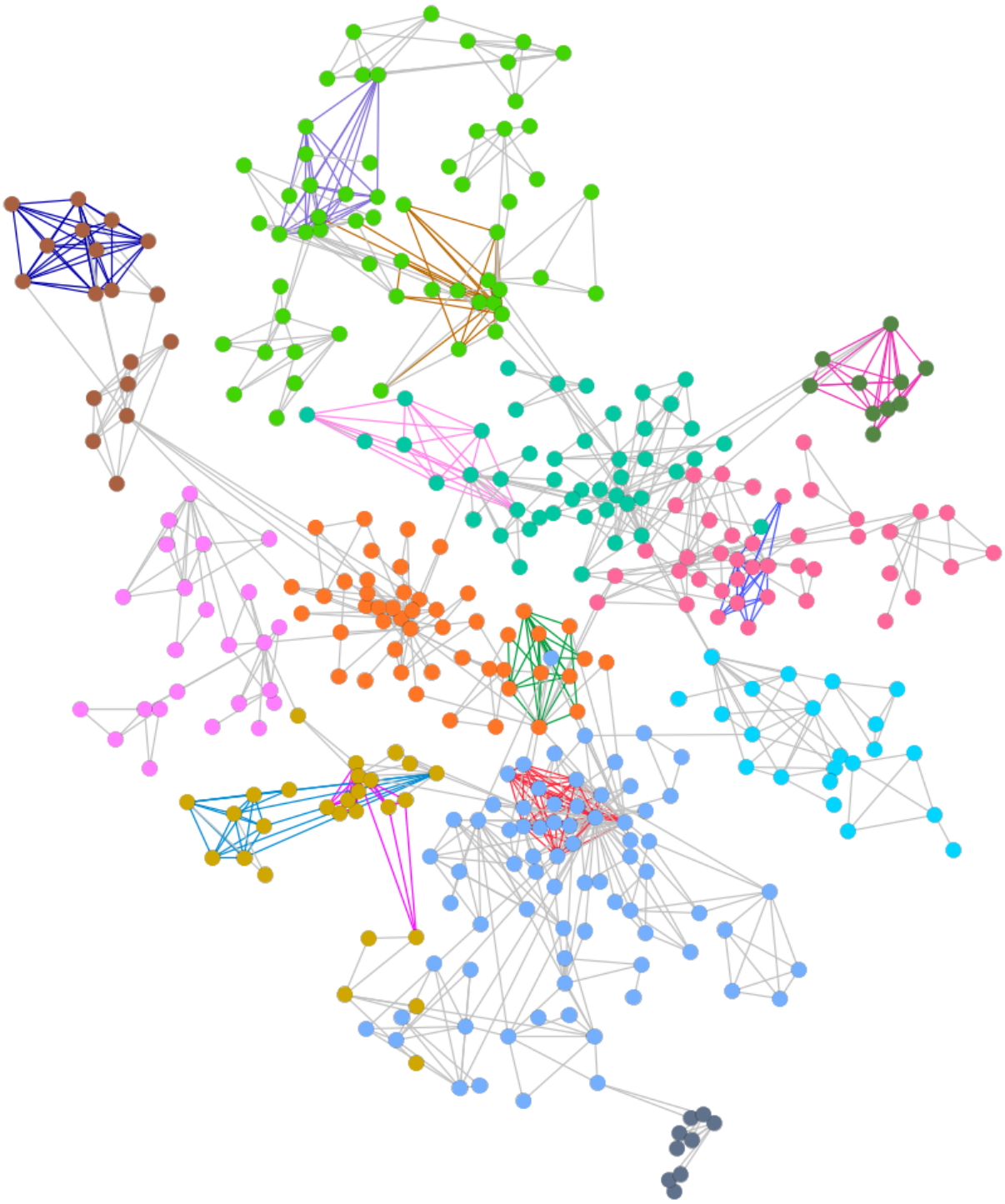
10. Save your visualization to file.

Some additional tweaking of colors, node and link size, and "making it look nice" in the Preview tab produced the visualization below.

In [32]:

```
Image(filename="link-communities.png")
```

Out[32]:



11. Once complete, submit a PDF document to Canvas that contains your responses and your graph visualization (since this is a larger graph feel free to crop the figure so that it only includes parts relevant to your responses).