Social Network Analysis using PatentsView and NetworkX

By Jesse Frumkin and Amanda Myers

August 28, 2017

The InventorAnalyze package is a powerful new tool for bibliometric (and other) researchers studying the social network of inventors, i.e., the community of inventors who collaborated on jointly invented patents. The InventorAnalyze package combines disambiguated patent data from the United States Patent and Trademark Office's PatentsView[1] project with social network analysis tools from the Los Alamos National Laboratory's NetworkX[2,3] library. PatentsView uses a statistical algorithm[4] for disambiguating patent inventor names, so that multiple variants of a name are assigned a common identifier and distinct inventors having a similar name are assigned separate identifiers. Such entity resolution is critical to identifying inventors and their collaborators over millions of distinct patents.

By importing disambiguated inventor data from PatentsView into NetworkX, InventorAnalyze enables researchers to more accurately study the entire inventor social network. Moreover, InventorAnalyze facilitates analysis of specific inventor network subgraphs generated via PatentsView API-based queries.[5] InventorAnalyze then leverages NetworkX functionality to study the structure, dynamics, and functions of these inventor subnetworks.[6]

In this document, we provide two examples of the type of inventor social network analysis facilitated by InventorAnalyze. The first example demonstrates some of the network-level analysis capabilities that InventorAnalyze facilitates. The second example shows the potential information that network-based metrics can convey about individual inventors and their influence.

Example 1: Quantifying Collaboration Degree across an Inventor Subnetwork

PatentsView data essentially forms a multipartite graph in which patent nodes are one mode, inventor nodes are a second mode, assignee nodes are a third, etc.[7] For this example, we require unimodal co-inventor networks. Accordingly, we use the InventorAnalyze package to extract bipartite graphs consisting of patents and inventors from PatentsView and project each into a unimodal graph, consisting of inventors linked if they collaborate on at least one patent.[8] Variants of this bipartite projection method involve various weighting schemes, such as weighting by the frequency of co-invented patents.[9]

We then use the InventorAnalyze package to generate a network-level measure of the density of collaboration, known as degree assortativity, among inventors in the unimodal graph. Degree assortativity

---

[1] http://www.patentsview.org/.

[2] Schult, Daniel A., and P. Swart. "Exploring network structure, dynamics, and function using NetworkX." In *Proceedings of the 7th Python in Science Conferences (SciPy 2008)*, vol. 2008, pp. 11-16. 2008.

[3] NetworkX does not use matrices as the primary network representation; hence, it is possible to study the entire PatentsView inventor social network because NetworkX efficiently structures networks using Python's hashmaps, called "dictionaries". *See* https://networkx.github.io/.

[4] N. Monath, A. Mccallum, Discriminative Hierarchical Coreference for Inventor Disambiguation. USPTO Inventor Disambiguation Workshop on September 24, 2015, available at http://www.patentsview.org/data/presentations/UMassInventorDisambiguation.pdf.

[5] http://www.patentsview.org/api/doc.html .

[6] https://networkx.readthedocs.io/en/stable/reference/introduction.html.

[7] Interestingly, several properties of a graph and its nodes are most accurately computed by taking into account the bipartite structure of two modes of the network. *See*, for example, the NetworkX functions for density of the graph, degrees of the nodes, and clustering, coefficients of the nodes that are specifically defined for bipartite structured networks.

[8] To project a bipartite graph to a unimodal graph, see the functions "projected_graph" in "networkx.algorithms.bipartite"

[9] For example, collaboration-based weighting entails weights edges between inventor nodes based on the count of common patents. See the functions "weighted_projected_graph", "collaboration_weighted_projected_graph", "overlap_weighted_projected_graph", and "generic_weighted_projected_graph" in "networkx.algorithms.bipartite"

is calculated based on individual node degree, i.e., the number of edges (co-inventors) connected to an individual node (a specified inventor). Node degree is a longstanding measure used to describe individual nodes within a network. In a co-inventor network, high degree nodes represent researchers that have invented with a myriad of collaborators. Low degree nodes represent researchers that co-invented with only a few other inventors. Degree assortativity[10] indicates the extent to which node degree is correlated across the entire network. It is a network level metric that ranges from -1 to +1. Assortativity is positive when there is a positive correlation between node degrees, meaning that high (low) degree nodes tend to be linked to other high (low) degree inventors.[11] Assortativity is negative when there is a negative correlation between node degrees, indicating that high degree nodes tend to be connected to low degree inventors. Where there is no correlation between node degree, assortativity is zero, meaning nodes are equally likely to be high or low degree. A randomly generated graph, for example, would have a degree assortativity of zero.

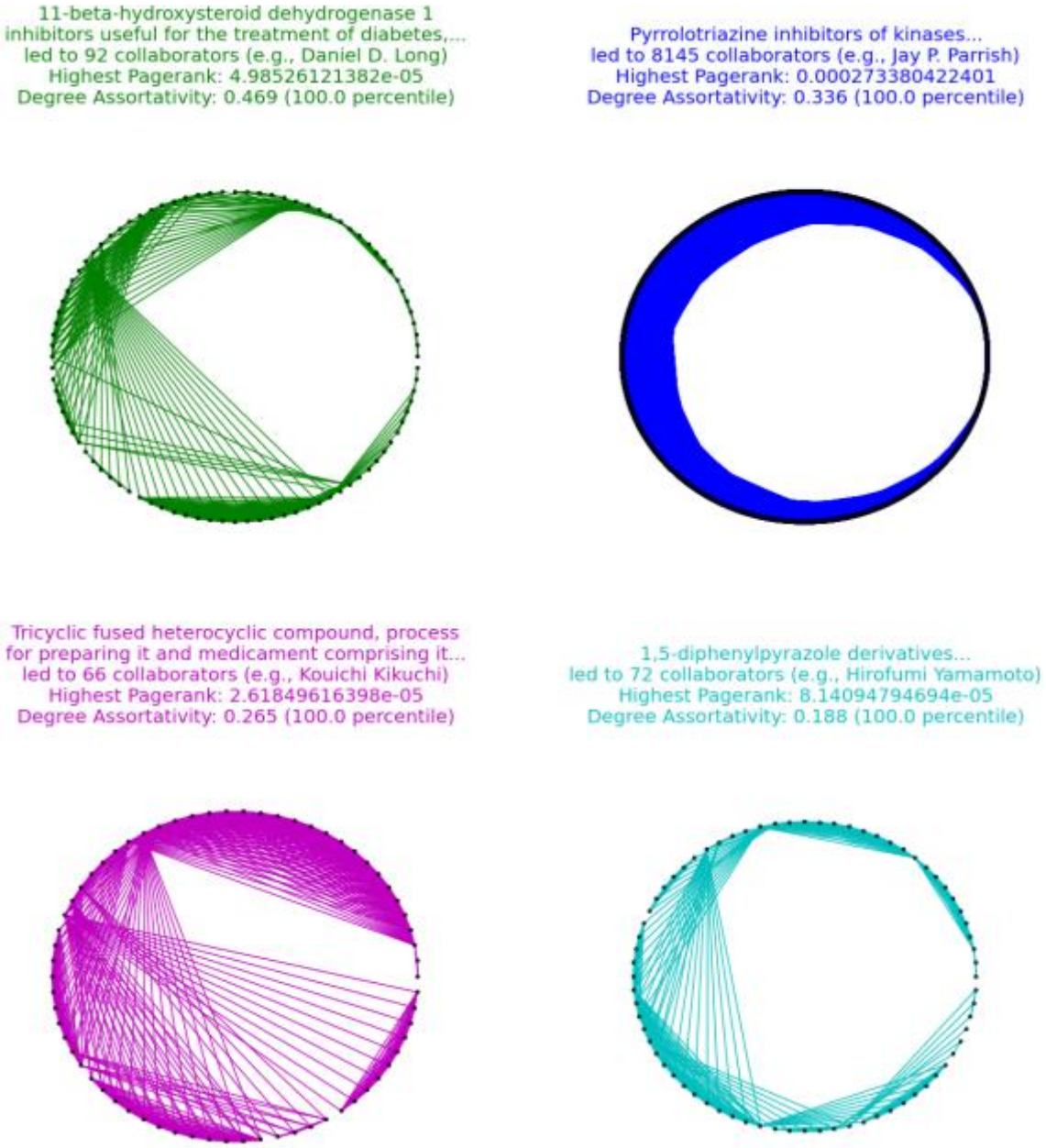
Degree assortativity quantifies the structure of a collaborative network. Positive degree assortativity occurs in a co-inventor network when inventors tend to collaborate with others having equally high or equally low number of patented inventions. Productive inventors tend to work together frequently, disparate individuals or small teams work in isolation, or both occur simultaneously. A new inventor to an assortative network is, on average, more likely to work in a small team of other novice inventors. Negative degree assortativity occurs in a co-inventor network when highly connected inventors collaborate with others that themselves have few co-invented patents. A disassortive co-inventor network has a more hierarchical structure where productive inventors work with different teams of less experienced or novice inventors, and those less experienced inventors do not co-invent among themselves. A new inventor to a disassortive co-inventor network is, on average, more likely to work on a team with an experienced patent inventor.

The InventorAnalyze package calculates degree assortativity for a subnetwork of co-inventors generated based on a PatentsView API query. For the subnetwork(s) matching the search criteria, InventorAnalyze calculates degree assortativity and produces network plot images. Figure 1 shows a sample of the results and images returned by InventorAnalyze for the following query of patents in International Patent Classification (IPC) subclass A61P (specific therapeutic activity of chemical compounds or medicinal preparations):

```
python InventorAnalyze.py
'http://www.patentsview.org/api/patents/query?q={"_and":[{"ipc_section
":"A"},{"ipc_class":"61"},{"ipc_subclass":"P"}]}&o={"per_page":9999}'
```

[10] Newman, Mark EJ. "Mixing patterns in networks." *Physical Review E* 67, no. 2 (2003): 026126.
[11] Degree Assortativity can be computed using the degree_assortativity_coefficient in "networkx.algorithms"

Figure 1. Positivity assortativity in the largest connected components of therapeutic drug co-inventor network.



Notes: Figure shows the four largest connected components of the co-inventor network for patents classified in IPC subclass A61P (specific therapeutic activity of chemical compounds or medicinal preparations). Nodes represent unique inventors and edges are their co-invented U.S. patents. The assortativity value for each connected component is significantly positive.[12] Each component label includes a patent title and inventor name, chosen based on highest pagerank.[13]

[12] Based on the NetworkX Conditional Uniform Graph test, which tests observed assortativity against randomly generated graphs having identical node and edge counts. Assortativity values were significant more positive than that of 200 generated random graphs ($p < 0.005$).
[13] See example 2 for more on pagerank.

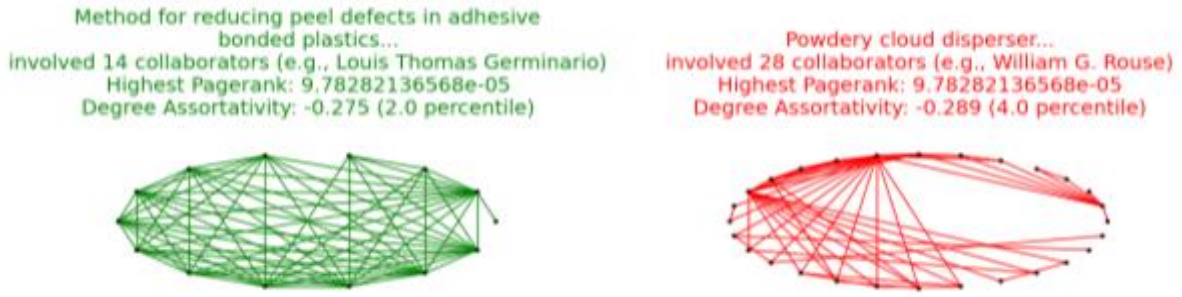
We limit Figure 1 to the four largest connected components that meet the search criteria. These largest components have positive assortativity, meaning that inventors that are the frequent collaborators are generally not collaborating with inventors who collaborate less frequently. Hence, there is a separation between two types of inventors within this research community.

By contrast, Figure 2 depicts an inventor collaboration subnetwork with negative assortativity. Figure 2 shows a sample of the results and images returned by InventorAnalyze for the following PatentsView API query for "Defensive Publications" and "Statutory Invention Registrations":

```
python InventorAnalyze.py
'http://www.patentsview.org/api/patents/query?q={"_or":[{"_contains":{
"patent_number":"H"}},{"_contains":{"patent_number":"T"}}]}&o={"per_pa
ge":9999}'
```

Defensive publications and invention registrations disclose an innovation but do not claim patent rights over that invention. Historically, inventors used these types of disclosures to block others from patenting similar inventions. Although such documents are no longer filed, we can construct the social network of inventors on such disclosures from PatentsView data. Figure 2 depicts the two largest connected components with the most negative assortativity returned from the query. Overall, in Figure 2, we see more negative assortativity values for the connected components than those of Figure 1.

Figure 2. Defensive Publications and Statutory Invention Registrations connect a network of co-inventors who are not claiming intellectual property using their disclosures.



Notes: Figure shows the two largest connected components having the most negative assortativity values out of 16 such networks of co-inventors on "Defensive Publications" or "Statutory Invention Registrations". Nodes represent unique inventors and edges are their co-invented U.S. patents. The assortativity value for each connected component is significantly negative.[14] Each separate connected component includes a representative patent and inventor, chosen based on highest pagerank.

Example 2: Filtering the Cancer Network Based on Patent Importance

In this example, we apply the InventorAnalyze package to co-inventor and patent citation networks in PatentsView to identify the most influential oncology researchers and their patented innovations. PatentsView embodies multiple networks, each formed through different processes. The co-inventor network is formed by collaboration on a common patent where nodes represent unique inventors and edges denote co-patents. The patent citation network is a directed graph formed as Examiners or

[14] Based on the NetworkX Conditional Uniform Graph test, which tests observed assortativity against randomly generated graphs having identical node and edge counts. Assortativity values were significant more negative than that of 200 generated random graphs (left: *p=0.02 & right: p=0.04*).

Applicants cite prior patents during prosecution of a patent application. For these two networks, we use the InventorAnalyze package to construct centrality measures that indicate the influence of patents and their inventors.

In graph theory, there are numerous centrality measures for quantifying the importance or influence of a particular node in the network. Degree centrality quantifies the local influence of a particular node based on its direct links to other nodes. Thus, the node with the highest degree is the most important. In the patent citation network, degree centrality would emphasize patents with the most forward and backward citations. Eigenvector centrality[15] measures importance beyond the local influence to include the "edges of the edges". Eigenvector centrality assigns a score to each node based on the influence of its connected nodes, so that nodes connected to highly-connected nodes have higher eigenvector centrality relative to nodes connected to low-connected nodes. A variant of eigenvector centrality, known as pagerank[16], quantifies influence in directed graphs. In the patent citation network, pagerank is calculated for each patent based on its forward citations. Pagerank is higher for patents that are cited in patents that are also highly cited.

In this example, we follow Bruck et al. (2016)[17] and Yan et al. (2011)[18] by using pagerank to identify the most influential patents in the citation network of cancer-related patents.[19] We modify the code of the InventorAnalyze package to calculate pagerank for the subnetwork of patents in the USPTO Cancer Moonshot Patent Dataset.[20] The Cancer Moonshot Dataset contains detailed information on granted patents relevant to cancer research and development. We query PatentsView for these cancer-related patents and build four oncology patent citation networks (one for each decade from 1976 to 2015). We then apply the NetworkX pagerank function to compute individual patent's pagerank in each of the four cohorts. We rank cancer-related patents by pagerank and find the emerging technologies in each decade to score highest. Broken down by decade, the most influential oncology patents based on citation network pagerank included patents in the following emerging technological areas:

- **From 1976 to 1985:** Method of producing antibodies, method of producing tumor antibodies, specific classes of chemotherapy drugs
- **From 1986 to 1995:** Proteins that bind to antibodies, chemotherapy, radio-sensitizers, Polymerase chain reaction (PCR) DNA amplification, hyperthermia (heat-based treatment)
- **From 1996 to 2005:** Hormonal therapy for cancer, immunotherapy by cytokine modulation, genetic algorithm for the evolution of genes, DNA mutagenesis for making diverse proteins and antibodies, a type of immunotherapy by stimulating a local immune response
- **From 2006 to 2015:** MicroRNAs in genetics, antibody purification

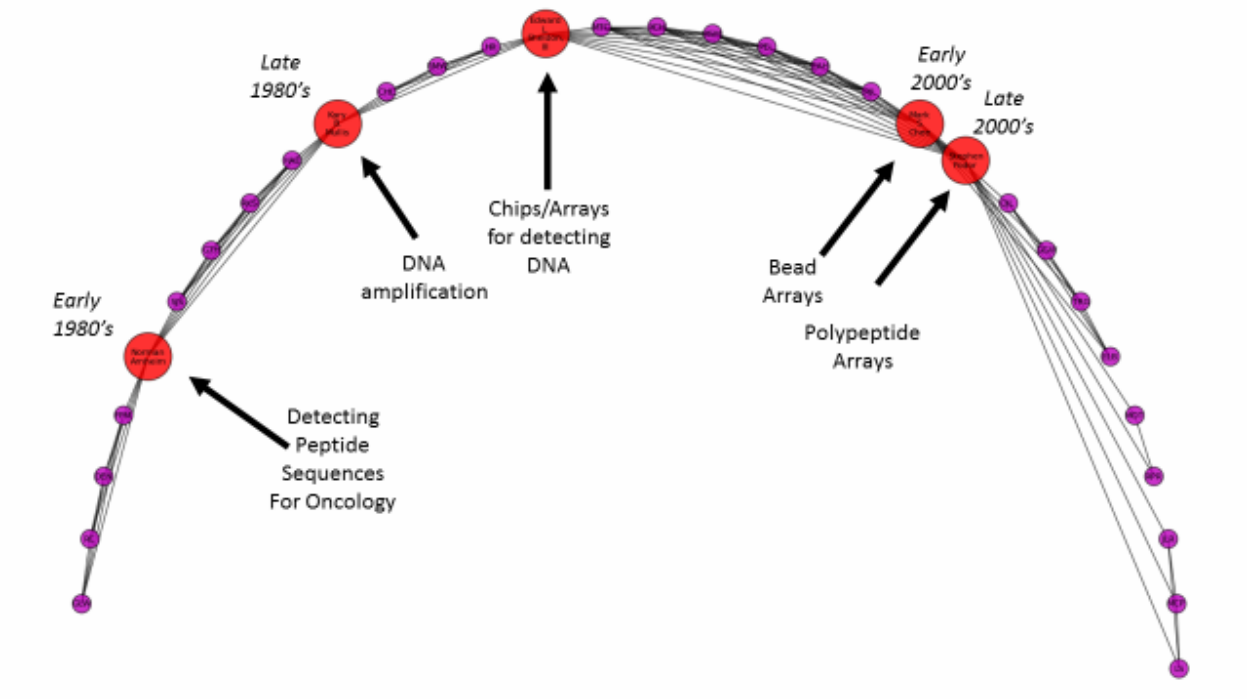
We then filter the oncology co-inventor network to include only the edges (co-invented patents) that exceed a specified pagerank score in the cancer-related patent citation network.[21] Thus, we are aiming to identify and analyze the collaborators on only the most influential patents in the cancer research space.

---

[15] Bonacich, Phillip. "Power and centrality: A family of measures." *American journal of sociology* 92, no. 5 (1987): 1170-1182.
[16] Page, Lawrence, Sergey Brin, Rajeev Motwani, and Terry Winograd. *The PageRank citation ranking: Bringing order to the web*. Stanford InfoLab, 1999.
[17] Bruck, Péter, István Réthy, Judit Szente, Jan Tobochnik, and Péter Érdi. "Recognition of emerging technology trends: class-selective study of citations in the US Patent Citation Network." *Scientometrics* 107, no. 3 (2016): 1465-1475.
[18] Yan, Erjia, and Ying Ding. "Discovering author impact: A PageRank perspective." *Information processing & management* 47, no. 1 (2011): 125-134.
[19] To view of the pageranks for a set of patents, call InventorAnalyze.py passing the "-p" command line parameter. To view of the eigenvector centralities pass the "-e" command line parameter. Eigenvector centralities in InventorAnalyze are calculated using patent pageranks to compute the edge weights of the co-inventor network, to ensure that mostly important patents contribute to the eigenvector centrality of the inventors.
[20] https://www.uspto.gov/learning-and-resources/electronic-data-products/cancer-moonshot-patent-data
[21] To analyze pagerank, the "pagerank" function was called using the "networkx" package.

We compute the pagerank of patents using the entire oncology citation network as a whole, not just for decade-cohorts. We then generate a co-inventor network that includes only the inventors who collaborate on an oncology patent exceeding a specified pagerank threshold. We use a variant of the InventorAnalyze package to identify, and visualize in Figure 2, the co-inventor subnetwork that generated the most influential cancer-related patents. This subnetwork includes inventors and patents critical to the development of genomics.

Figure 3. Inventors at articulation points bridge together teams in a subnetwork of co-inventors.



Notes: Figure shows the co-inventor subnetwork of oncology researchers that generated the most influential cancer-related patents (based on highest pagerank in the patent citation network). This subnetwork includes inventors and patents critical to the development of genomics.

We use the package to trace the evolution of this critical technology through individual inventors, revealing the trajectory of development over time. The InventorAnalyze package presorts the graph using the Cuthill-McKee algorithm for sorting sparse adjacency matrices.[22] This effectively minimizes the large cross-over distances between inventors in the visualization. It also reveals key articulation points (larger nodes),[23] indicating inventors that bridged teams and, thereby, likely transferred significant knowledge and ability from one collaborative team to another. For example, Nobel Prize winner Kary B. Mullis bridged together groups detecting peptide sequences with teams detecting DNA. Other key inventors linked collaborators using large-scale DNA chips and arrays.

Conclusion

Readers are encouraged to modify the source code of InventorAnalyze to use the wide range of functionality in the NetworkX package for analysis of PatentsView networks. In addition to assortativity

[22] Cuthill, Elizabeth, and James McKee. "Reducing the bandwidth of sparse symmetric matrices." In *Proceedings of the 1969 24th national conference*, pp. 157-172. ACM, 1969.

[23] To identify articulation point, the "articulation_points" function was called using the "networkx" package.

and centrality measures, NetworkX includes analysis of cores, cycles, clusters, color, communities, boundaries, cliques, connectivity, independent sets, and more. The source code for InventorAnalyze.py is in Appendix 2 and can easily be adapted to exploit additional NetworkX functionality.

Appendix 1 – Output from the InventorAnalyze Package

The following is sample textual output from InventorAnalyze using the query for Figure 1:

```
21653 disambiguated inventors.
8496 patents.

The disambiguation of the names of joint inventors yields a social
network of network of inventors wherein the edges between the nodes
represent a joint inventorship.  Some of these inventors have a high
degree of collaboration because they have invented with a myriad of
co-inventors since 1976; others have co-invented with only a few other
inventor since 1976.  How correlated are the collaboration degrees of
the inventors in these co-inventor pairs?  For example, do high
collaborators work with high collaborators and low with low?  We
measure this correlation using the graph-level property called degree
assortativity, which ranges from -1 to +1.  The assortativity measure
is positive when there is a positive correlation between the
collaboration degrees across the graph.

The Degree Assortativity of this Subnetwork of Disambiguated
Inventors:  0.50078407173


Patents:  Top 10 pageranks based on citations (calculated by
disambiguating inventors to remove their self-citations.)  To see all
of the pageranks, run again using the -p parameter.

     Patent Number, Pagerank
     US6166063 , 0.000335220809785,
http://www.patentsview.org/web/#detail/patent/6166063
     US6982265 , 0.000273380422401,
http://www.patentsview.org/web/#detail/patent/6982265
     US7750007 , 0.000194663570003,
http://www.patentsview.org/web/#detail/patent/7750007
     US6180636 , 0.000189228776184,
http://www.patentsview.org/web/#detail/patent/6180636
     US6329381 , 0.000179273335895,
http://www.patentsview.org/web/#detail/patent/6329381
     US7074798 , 0.000174017812502,
http://www.patentsview.org/web/#detail/patent/7074798
     US7022698 , 0.00016819086463,
http://www.patentsview.org/web/#detail/patent/7022698
     US7115741 , 0.00016819086463,
http://www.patentsview.org/web/#detail/patent/7115741
```

```
     US7642350 , 0.000164246256214,
http://www.patentsview.org/web/#detail/patent/7642350
     US7265104 , 0.000160301647797,
http://www.patentsview.org/web/#detail/patent/7265104

Inventors:  Top 10 eigenvector centralities, calculated by first
disambiguating inventors and then weighing each of their
collaborations by the total of the pageranks of their co-invented
patents.  To see all of the centralities, run again using the -e
parameter.
     Inventor , Centrality, Inventor Link
     Jay P. Parrish , 0.203391812444 ,
http://www.patentsview.org/web/#detail/inventor/7368237-1
     Manoj C. Desai , 0.202208068538 ,
http://www.patentsview.org/web/#detail/inventor/5202440-1
     John O. Link , 0.201915752796 ,
http://www.patentsview.org/web/#detail/inventor/3976428-1
     Michael L. Mitchell , 0.199778636105 ,
http://www.patentsview.org/web/#detail/inventor/7649015-11
     Darryl Kato , 0.199778636105 ,
http://www.patentsview.org/web/#detail/inventor/8088368-2
     James G. Taylor, VI , 0.199778636105 ,
http://www.patentsview.org/web/#detail/inventor/4146634-3
     Hongyan Guo , 0.199778636105 ,
http://www.patentsview.org/web/#detail/inventor/7608723-3
     Thorsten A. Kirschberg , 0.199778636105 ,
http://www.patentsview.org/web/#detail/inventor/6593292-5
     Randall L. Halcomb , 0.197316219326 ,
http://www.patentsview.org/web/#detail/inventor/5104982-1
     Choung U. Kim , 0.196961739125 ,
http://www.patentsview.org/web/#detail/inventor/4071513-1
```

Appendix 2 – Open Source Python Script

Below is a script which downloads disambiguated inventor data from PatentsView, computes the degree assortativity of the subnetwork of disambiguated inventors. It computes the top ten patents by pagerank and the top ten inventors by eigenvector centrality, wherein the co-inventor network is weighted by the pagerank influence of the patents that were co-invented. The script also creates visualizations of connected components, and the top inventor and top patent for some of the connected components based on the user's query. For instructions on running the script, run the python script using the "-h" parameter.

```
# InventorAnalyze
# To construct a query using the PatentsView API, see: www.patentsview.org/api/query-language.html
# Be mindful of the default number of results returned by PatentsView

from argparse import ArgumentParser
from collections import defaultdict
from json import load as jsonload
from math import ceil
```

```
from textwrap import wrap
from urllib import urlopen
from warnings import catch_warnings, simplefilter

from matplotlib import pyplot
from networkx import Graph, DiGraph, degree_assortativity_coefficient, bipartite, eigenvector_centrality, pagerank, \
   gnm_random_graph, articulation_points, draw_networkx_edges, connected_components, density, draw_networkx, \
   shell_layout, connected_component_subgraphs
from networkx.utils.rcm import cuthill_mckee_ordering


_help_message = """
To analyze a network of patents and inventors, pass a URL as an argument using the PatentsView API:
See http://www.patentsview.org/api/patent.html for the API query structure.
""".strip()

_further_help = """
For example, enter the following using single quotes for the URL argument:
python InventorAnalyze.py
'http://www.patentsview.org/api/patents/query?q={"cpc_subgroup_id":"C12Q1/6886"}&o={"per_page":5000}'
""".strip()

_assortativity_explanation = """The disambiguation of the names of joint inventors yields a social
network of network of inventors wherein the edges between the nodes represent a joint inventorship.
Some of these inventors have a high degree of collaboration because they have
invented with a myriad of co-inventors since 1976; others have co-invented with
only a few other inventor since 1976.  How correlated are the collaboration degrees
of the inventors in these co-inventor pairs?  For example, do high collaborators work with
high collaborators and low with low?  We measure this correlation using the graph-level
property called degree assortativity, which ranges from -1 to +1.  The assortativity measure
is positive when there is a positive correlation between the collaboration degrees across the graph.""".strip()

_pagerank_explanation = """Patents:  Top 10 pageranks based on citations (calculated by disambiguating inventors
to remove their self-citations.)  To see all of the pageranks, run again using the -p parameter.""".strip()

_eigenvector_centrality_explanation = """Inventors:  Top 10 eigenvector centralities, calculated by first disambiguating
inventors
and then weighing each of their collaborations by the total of the pageranks of their co-invented
patents.  To see all of the centralities, run again using the -e parameter.""".strip()

def main():
   def the_weight(GP, u, v):
      '''
      This will be used to weights of the co-inventor graph in a way that is weighted by the pagerank.
      :param GP: the graph from which the weights are being computed
      :param u: a first node
      :param v: a second node
      :return: the weight for the edge between the nodes u and v
      '''
      uset = set(GP[u])
      vset = set(GP[v])
      x = sum([pr.get(patent, 0.0) for patent in
            uset.intersection(vset)])  # pr will be computed before this function is called
      return x

   def make_str(input_string):
      '''
```

```
    :param input_string: a string that potentially has unprintable characters
    :return: a string omitting unprintable characters
    '''
    result = "".join([letter for letter in input_string if ord(letter) < 128])
    return result

def output_assort(graphassort):
    '''

    :param graphassort: a graph for which assortativity should be calculated
    :return: a printable string indicating the assortativity of the graph and the
             significance of the assortativity compared to random graphs using
             the conditional uniform graph test.
    '''
    if density(graphassort) == 1.0:
        result = "Complete Graph"
    else:
        nbootstrap = 200
        this_assortativity = degree_assortativity_coefficient(graphassort)
        random_distributions_this_graph = [degree_assortativity_coefficient(
            gnm_random_graph(graphassort.number_of_nodes(), graphassort.number_of_edges())) for _ in
                                   xrange(nbootstrap)]
        this_p = float(
            len([datum for datum in random_distributions_this_graph if datum > this_assortativity])) / float(
            len(random_distributions_this_graph))
        result = "\nDegree Assortativity: " + str(float(round(1000 * this_assortativity)) / 1000.0) + " (" + str(
            100.0 - float(round((this_p * 100000.0) / 1000.0))) + " percentile)"
    return result

show_graphs = True  # show visualizations 1 and 2
n = 16  # number of graphs to display in a grid in the first visualization
top_patent_number = 250  # number of edges in the second visualization

help_message = _help_message
further_help = _further_help
parser = ArgumentParser(description=help_message)
parser.add_argument('url', metavar='URL', type=str, nargs='*', help=further_help)
parser.add_argument("-e", "--all_eigenvector_centralities", action="store_true")
parser.add_argument("-p", "--all_pageranks", action="store_true")
args = parser.parse_args()

if args.all_eigenvector_centralities:
    output = "e"
elif args.all_pageranks:
    output = "p"
else:
    output = "a"

if args.url == []:
    print help_message
    print further_help
else:
    query = args.url[
            0] + '&f=["patent_number","inventor_id","inventor_first_name","inventor_last_name",' \
                 '"cited_patent_number","patent_title"]'
    # retrieve these fields in addition to what was specified by the user's URL

    BP = Graph()  # A BiPartite Graph (BP) of applications and inventors
```

```
DG = DiGraph()  # A Directed Graph for the citation network
inventors = defaultdict(frozenset)
# a dictionary of applications mapping to their inventors to eliminate self-citations
titles = defaultdict(str)
try:
   data = jsonload(urlopen(query))
   assert not (data[u'patents'] is None)
   queryset = set([])  # for later filtering the network to include only queried patents
   allinventors = set([])  # for later computing the co-inventor network
   for patent in data[u'patents']:
      patent_number = patent[u'patent_number']
      if patent_number[0] != "D":  # exclude design patents
         queryset.add(patent_number)  # for later filtering the network to include only queried patents
         inventorset = set(
            [])  # for keeping track of an application's inventors to later remove self-citations
         for inventor in patent[u'inventors']:
            inventor_info = (
            inventor[u'inventor_id'], inventor[u'inventor_first_name'], inventor[u'inventor_last_name'])
            BP.add_edge(patent_number,
                        inventor_info)
            # A BiPartite Graph (BP) with patents and inventors, later to be projected into a co-inventor network
            inventorset.add(
               inventor_info)
            # for keeping track of an application's inventors to remove self-citations later
         inventorset = frozenset(inventorset)
         inventors[
            patent_number] = inventorset  # for later when removing self-citations from the citation network
         allinventors.update(inventorset)
         titles[patent_number] = patent[u'patent_title']
         for citation in patent[u'cited_patents']:  # for constructing the citation network
            citation_number = citation[u'cited_patent_number']
            if citation_number is not None:
               if citation[u'cited_patent_number'][0] != "D":  # exclude design patents
                  DG.add_edge(patent_number, citation[
                     u'cited_patent_number'])
                  # some of these citations are not in the original application query; these will later be deleted
   for patent, cited_patent in DG.edges():
      # to eliminate citations that were not in the original query and to eliminate self-citations
      ok = (patent in queryset) and (cited_patent in queryset) and (
      len(inventors[patent].intersection(inventors[cited_patent])) == 0)
      if not ok:
         DG.remove_edge(patent, cited_patent)
   pr = pagerank(DG)
   proutput = sorted(pr.items(), key=lambda x: x[1], reverse=True)
   CoInventors = bipartite.generic_weighted_projected_graph(BP, allinventors, weight_function=the_weight)
   try:
      e = sorted(eigenvector_centrality(CoInventors, weight='weight', max_iter=10000).items(),
               key=lambda x: x[1], reverse=True)
      eflag = True
   except:
      eflag = False
      pass
   if output == 'a':
      if len(CoInventors) > 1:
         print
         print str(CoInventors.number_of_nodes()) + " disambiguated inventors."
         print str(len(data[u'patents'])) + " patents."
         print
```

```
actual_assortativity = degree_assortativity_coefficient(CoInventors)
print _assortativity_explanation
print
print "The Degree Assortativity of this Subnetwork of Disambiguated Inventors:  " + str(
   actual_assortativity)
print

print _pagerank_explanation
print "Patent Number, Pagerank"
for patent, pagerank_value in proutput[0:10]:
   print "US" + str(patent) + "," + str(
      pagerank_value) + ", http://www.patentsview.org/web/#detail/patent/" + str(
      patent.encode('utf-8'))
print
if eflag:
   print _eigenvector_centrality_explanation
   print "Inventor , Centrality, Inventor Link"
   for inventor, eigenvector_centrality_value in e[0:10]:
      print inventor[1].encode('utf-8') + " " + inventor[2].encode(
         'utf-8') + " , " + str(
         eigenvector_centrality_value) + " , http://www.patentsview.org/web/#detail/inventor/" + str(
         inventor[0])
   print
else:
   print "Eigenvector Centrality could not be computed for this graph."
   print

# Visualization One
some_colors = ['g', 'y', 'c', 'm', 'b', 'g', 'r']
ord_overall = list(cuthill_mckee_ordering(CoInventors))  # ordering to use to arrange graphs
if show_graphs:
   rows = ceil(n ** 0.5)
   edict = dict(e)
   largest_subgraphs = sorted([a_graph for a_graph in connected_component_subgraphs(CoInventors) if
                       (density(a_graph) < 1.0 and a_graph.number_of_nodes() > 3)],
                      key=lambda x: len(x), reverse=True)[0:n]
   if len(largest_subgraphs) > 0:
      subgraphs = [a_subgraph for a_subgraph in
              sorted(largest_subgraphs, key=lambda x: degree_assortativity_coefficient(x),
                  reverse=True)][0:n]
      for subpl, this_graph in enumerate(subgraphs):
         this_ord = [list(cuthill_mckee_ordering(this_graph))]
         e_dict = dict([(node, ec) for node, ec in e if node in set(this_ord[0])])
         top_pos = shell_layout(this_graph, nlist=this_ord)
         pyplot.figure(num=n, figsize=(40, 15), facecolor='w')
         sp = pyplot.subplot(rows, rows, subpl + 1)
         the_color = some_colors[subpl % len(some_colors)]
         try:
            top_inventors = sorted(this_graph.nodes(), key=lambda x: edict[x], reverse=True)
            top_inventor = top_inventors[0]
            flatten_me = [BP[inv].keys() for inv in top_inventors if len(BP[inv].keys()) > 0]
            these_pageranks = [pr.get(pat, 0) for pat in
                       set([item for sublist in flatten_me for item in sublist])]
            top_group_patent = \
            sorted(list(set([item for sublist in flatten_me for item in sublist])),
                 key=lambda x: pr.get(x, 0), reverse=True)[0]
         except:
            raise
```

```
                top_inventor = ["", "", ""]
                top_group_patent = ""
            max_length = 48
            a_caption = "\n".join(
                wrap(titles[top_group_patent], max_length)[0:2]) + "...\ninvolved " + str(
                len(set(top_inventors))) + " collaborators (e.g., " + top_inventor[1].encode(
                'utf-8') + " " + top_inventor[2].encode(
                'utf-8') + ")\n" + "Highest Pagerank: " + str(max(these_pageranks)) + output_assort(
                this_graph)
            sp.set_title(a_caption, color=the_color)
            pyplot.axis('off')
            pyplot.tight_layout()
            threshold = .50
            with catch_warnings():
                simplefilter("ignore")
                draw_networkx(this_graph, node_size=5, nodelist=e_dict.keys(), node_color=the_color,
                              pos=top_pos, with_labels=False, edge_color=the_color)
        with catch_warnings():
            simplefilter("ignore")
            pyplot.show()

        # Visualization Two
        P = Graph()
        for node1, node2, weightdict in sorted(largest_subgraphs[0].edges(data=True),
                                               key=lambda x: x[2]["weight"], reverse=True)[
                                    :top_patent_number]:
            P.add_edge(node1, node2)

        ord1 = list(cuthill_mckee_ordering(P))
        if ord1[0] > ord1[-1]:
            ord1 = list(reversed(ord1))
        blank_nodes = ["blank_node"]
        visualized_graph = P.copy()
        visualized_graph.add_nodes_from(blank_nodes)
        the_pos = shell_layout(visualized_graph, nlist=[ord1 + blank_nodes])
        a = list(set(articulation_points(P)))
        non_a = (set(ord1) - set(a))
        the_labels = {
        (invid, first_name, last_name): (make_str(first_name) + " " + make_str(last_name)) for
        invid, first_name, last_name in P.nodes()}
        articulation_labels = {nodeid: "\n".join(name.split(" ")) for nodeid, name in
                               the_labels.items() if (nodeid in a)}
        non_articulation_labels = {nodeid: "".join([partname[0] for partname in name.split(" ")])
                                   for nodeid, name in the_labels.items() if not (nodeid in a)}
        pyplot.axis('off')
        pyplot.tight_layout()
        draw_networkx_edges(P, pos=the_pos, alpha=.8)
        draw_networkx(P, edgelist=[], nodelist=non_a, node_color='g', node_size=300, font_size=8,
                      pos=the_pos, labels=non_articulation_labels, alpha=.8)
        draw_networkx(P, edgelist=[], nodelist=a, node_color='c', node_size=2000, font_size=8,
                      pos=the_pos, labels=articulation_labels, alpha=.8)
        pyplot.show()
    else:
        print "The network is too small to visualize properly."
else:
    print "The network is too small."
elif output == 'e':
    for inventor, eigenvector_centrality_value in e:
```

```
            print inventor[1].encode('utf-8') + "," + inventor[2].encode('utf-8') + "," + str(
                eigenvector_centrality_value) + ", http://www.patentsview.org/web/#detail/inventor/" + str(
                inventor[0])
        elif output == 'p':
            for patent, pagerank_value in proutput:
                print patent.encode('utf-8') + "," + str(
                    pagerank_value) + ", http://www.patentsview.org/web/#detail/patent/" + str(
                    patent.encode('utf-8'))
    except (IOError, TypeError, ValueError):
        print "The URL that you entered probably has a typo."
        raise
    except AssertionError:
        print "The URL that you entered did not retrieve results."


if __name__ == "__main__":
    main()
```