Network Analasys

> https://colab.research.google.com/drive/12HpJOzGMUHpmMEK-_wEdnhXR-2JONI7A

Imports and getting the data

In [0]:

```
import pandas as pd
import matplotlib.pyplot as plt
import networkx as nx
import csv
import codecs
import re
import itertools
import copy
from collections import defaultdict
!pip install palettable
from palettable.colorbrewer.qualitative import Pastel1_3
```

```
Requirement already satisfied: palettable in /usr/local/lib/python3.6/dist-packages (3.3.
0)
```

In [0]:

```
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.filterwarnings("ignore", category=FutureWarning)
```

In [0]:

```
!wget https://www.dropbox.com/s/6wcx4z3akdmqy4f/wos_df_agile.csv?dl=0
```

```
--2019-10-24 18:27:00--  https://www.dropbox.com/s/6wcx4z3akdmqy4f/wos_df_agile.csv?dl=0
Resolving www.dropbox.com (www.dropbox.com)... 162.125.65.1, 2620:100:6021:1::a27d:4101
Connecting to www.dropbox.com (www.dropbox.com)|162.125.65.1|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: /s/raw/6wcx4z3akdmqy4f/wos_df_agile.csv [following]
--2019-10-24 18:27:01--  https://www.dropbox.com/s/raw/6wcx4z3akdmqy4f/wos_df_agile.csv
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://ucaf333ea7c7760407b65e3dbc9a.dl.dropboxusercontent.com/cd/0/inline/ArBZ
1Wc7dnZrWEAF7tZQCq9ArURKpe3H7G-p9r5nxawwF9CUUq43fW83YqVSOHk_muYiexLsGyOWqZoOeNcqV-SkBP57-
jx9iQMm0WQZgBXhQgt5jrx8WplqPkmvfMx5fIA/file# [following]
--2019-10-24 18:27:01--  https://ucaf333ea7c7760407b65e3dbc9a.dl.dropboxusercontent.com/c
d/0/inline/ArBZ1Wc7dnZrWEAF7tZQCq9ArURKpe3H7G-p9r5nxawwF9CUUq43fW83YqVSOHk_muYiexLsGyOWqZ
oOeNcqV-SkBP57-jx9iQMm0WQZgBXhQgt5jrx8WplqPkmvfMx5fIA/file
Resolving ucaf333ea7c7760407b65e3dbc9a.dl.dropboxusercontent.com (ucaf333ea7c7760407b65e3
dbc9a.dl.dropboxusercontent.com)... 162.125.65.6, 2620:100:6021:6::a27d:4106
Connecting to ucaf333ea7c7760407b65e3dbc9a.dl.dropboxusercontent.com (ucaf333ea7c7760407b
65e3dbc9a.dl.dropboxusercontent.com)|162.125.65.6|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5229225 (5.0M) [text/plain]
Saving to: 'wos_df_agile.csv?dl=0.2'

wos_df_agile.csv?dl 100%[===================>]   4.99M   709KB/s    in 6.8s

2019-10-24 18:27:08 (746 KB/s) - 'wos_df_agile.csv?dl=0.2' saved [5229225/5229225]
```

In [0]:

```
df = pd.read_csv('wos_df_agile.csv?dl=0',encoding='utf-8',sep='\t',index_col=0)
```

```
df.reset_index(drop=True,inplace=True)
```

DOI

So with the first network that we make, we want to examen how the articles reference each other. To do this we start by extracting the DOI numbers from the References collumn, and saving them as lists in a new collumn Ref_DOIs.

In [0]:

```
df['References'].isnull()
df.dropna(subset=['References'],inplace=True)
df.reset_index(drop=True,inplace=True)
```

In [0]:

```
ii=[]
for i in range(len(df['References'])):
  DOIs=re.findall(r'DOI[\s][-._()/:A-Za-z0-9]+',df['References'][i])
  doo=[re.findall(r'10[-._()/:A-Za-z0-9]+', dodo) for dodo in DOIs]
  ii.append(list(itertools.chain(*doo)))
df['Ref_DOIs']=ii
df.head(2)
```

Out[0]:

	Author	Title	Journal	Sub-field	Keywords	Abstract	References	City of pub	Year
0	Ettlie, JE	R&D and global manufacturing performance	MANAGEMENT SCIENCE	R&D; agility; process innovation; product inno...	LARGE MULTIPRODUCT FIRMS; D INTENSITY; DIVERSI...	R&D intensity and manufacturing performance we...	Allison P., 1990, SOCIOL METHODOL, V20, P93, D...	LINTHICUM HTS	1998.0
1	Ottaway, TA; Burns, JR	Adaptive, agile approaches to organizational a...	DECISION SCIENCES	NaN	FLEXIBLE MANUFACTURING SYSTEMS	Intelligent agent-based approaches to software...	Anthony R.N., 1965, PLANNING CONTROL SYS; BUZA...	ATLANTA	1997.0

Then we check if any of the mentioned refrencens' DOIs match any of the DOIs belonging to the articles in our dataframe.

In [0]:

```
edgelist_DOI = []
for i in range(len(df['DOI'])):
  for j in range(len(df['DOI'])):
    if df['DOI'][i] in df['Ref_DOIs'][j]:
      edgelist_DOI.append((j,i))
```

Here we make a directed graph from the edgelist, and then we plot it.

In [0]:

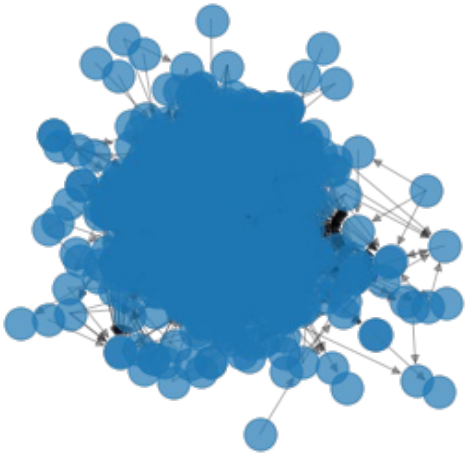
```
DOI_graph = nx.DiGraph()
for [u,v] in edgelist_DOI:
  if ~ DOI_graph.has_edge(u,v):
    DOI_graph.add_edge(u,v)
```

In [0]:

```
pos=nx.kamada_kawai_layout(DOI_graph)
plt.figure(figsize=(5, 5))
```

```
nx.draw_networkx_edges(DOI_graph, pos, alpha=0.3)
nx.draw_networkx_nodes(DOI_graph, pos, alpha=0.7)

plt.axis('off')
plt.show()
```



It is just a ball of yarn, so we do some more processing of the graph. We look at the in degree centrality as well as the eigenvector centrality.

In [0]:

```
print(sorted(dict(nx.in_degree_centrality(DOI_graph)).items(), key=lambda kv: kv[1],reve
rse=True)[:5])
print(sorted(dict(nx.eigenvector_centrality(DOI_graph)).items(), key=lambda kv: kv[1],re
verse=True)[:5])
```

```
[(427, 0.17744610281923714), (370, 0.13598673300165837), (299, 0.0845771144278607), (358,
0.0812603648424544), (359, 0.07131011608623555)]
[(427, 0.5773514454867709), (291, 0.5773495256921397), (290, 0.5773495255276603), (9, 0.0
0058173212033067 16), (3, 0.00013204818439356967)]
```

We see that only 3 articles has a high eigenvectorcentrality, while the in degree centrality is more evenly spaced. We then add the two centrality measures to the nodes as node attributes.

In [0]:

```
d_cent=dict(nx.in_degree_centrality(DOI_graph))
nx.set_node_attributes(DOI_graph, d_cent, 'd_cent')

e_cent=dict(nx.eigenvector_centrality(DOI_graph))
nx.set_node_attributes(DOI_graph, e_cent, 'e_cent')
```

Then we make a copy of the graph where we filter the nodes on teir in degree centrality, removing nodes with a in degree centrality less than 0.01.

In [0]:

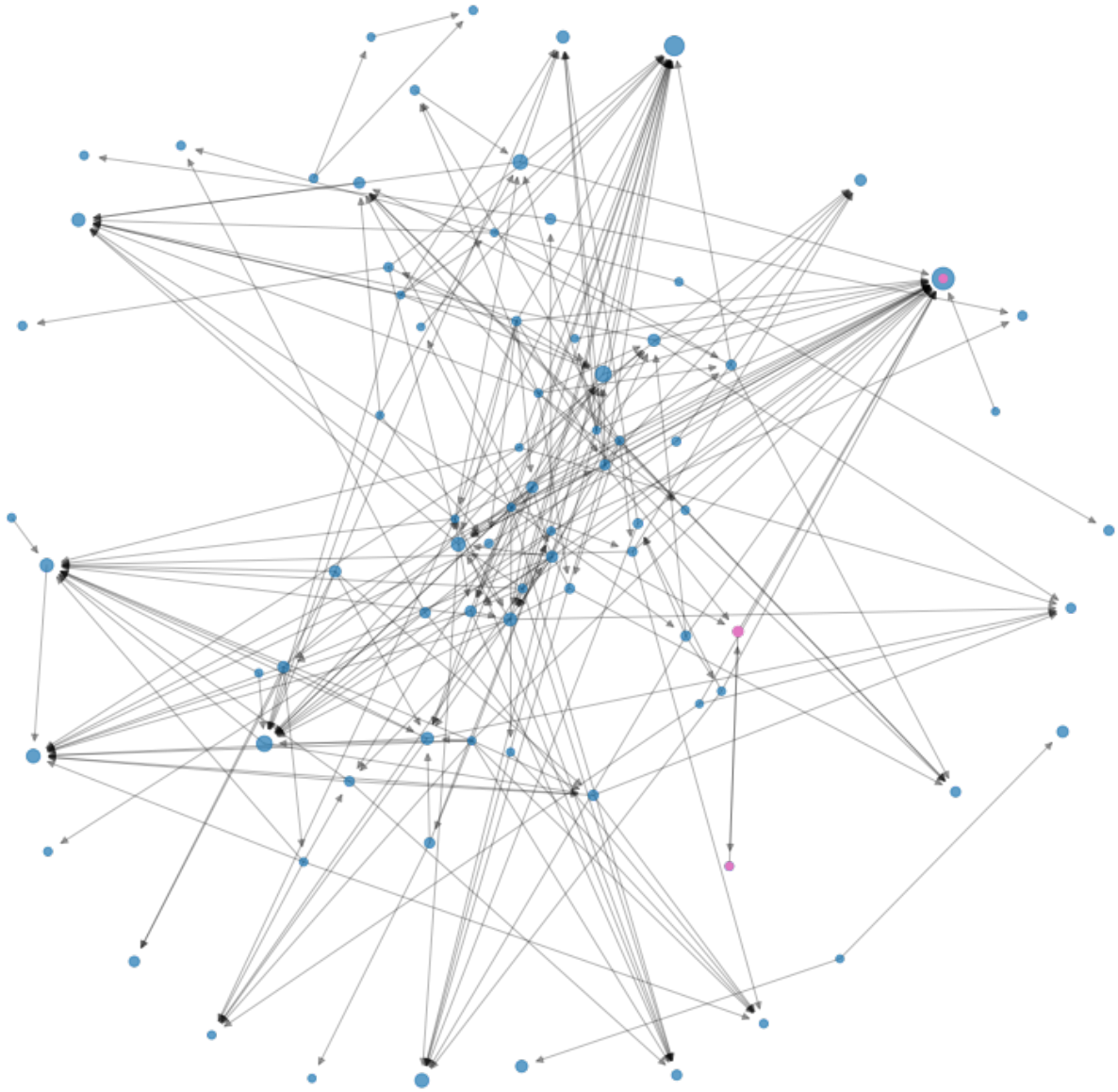
```
DOI_2 = copy.deepcopy(DOI_graph)
BadNodes=[]
for (n, d) in DOI_2.nodes(data=True):
    if d['d_cent'] < 0.01:
        BadNodes.append(n)
DOI_2.remove_nodes_from(BadNodes)
DOI_2.remove_nodes_from(list(nx.isolates(DOI_2)))
```

Here we plot the graph, the node size reprecent the in degree centrality, and the three pink nodes are the nodes with a high eigenvector centrality.

In [0]:

```
pos=nx.spring_layout(DOI_2, scale=2,k=0.4,seed=10)
plt.figure(figsize=(15, 15))
nx.draw_networkx_edges(DOI_2, pos, alpha=0.3)
nx.draw_networkx_nodes(DOI_2, pos, alpha=0.7, node_size=[DOI_2.nodes()[n]['d_cent']*1000
+ 10 for n in DOI_2.nodes()])
xx = [n for n,d in DOI_2.nodes(data=True) if d['e_cent']>0.5]
nx.draw_networkx_nodes(DOI_2, pos, nodelist=xx, alpha=1, node_size=[DOI_2.nodes()[n]['d_
cent']*1000 + 10 for n in DOI_2.nodes()],node_color='tab:pink')

plt.axis('off')
plt.show()
```



So we see some disconnected components, but mostly it seem to be one big network, where everyone reference each other.

Words in Title

For the second network we look at how the articles share words in the title. So first we tokenize the Titles and save the token lists as a new collumn in the dataframe. We use stemming, we remove stopwords and punctuation, and we lowercase.

```
In [0]:
```

```
import nltk
nltk.download('stopwords')
nltk.download('punkt')
from nltk.corpus import stopwords
from nltk import word_tokenize
import string
from nltk.stem import PorterStemmer

ps = PorterStemmer()
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

In [0]:

```
def stemSentence(sentence):
    token_words=word_tokenize(sentence)
    token_words
    stem_sentence=[]
    for word in token_words:
        stem_sentence.append(ps.stem(word))
        stem_sentence.append(" ")
    return "".join(stem_sentence)

df['Title_stem']=[stemSentence(phrase) for phrase in df['Title']]
```

In [0]:

```
stop = stopwords.words('english') + list(string.punctuation)
df['Title_tok']=[[i for i in word_tokenize(df['Title'][j].lower()) if i not in stop] for
j in range(len(df['Title_stem']))]
df.head(2)
```

Out[0]:

	Author	Title	Journal	Sub-field	Keywords	Abstract	References	City of pub	Year
0	Ettlie, JE	R&D and global manufacturing performance	MANAGEMENT SCIENCE	R&D; agility; process innovation; product inno...	LARGE MULTIPRODUCT FIRMS; D INTENSITY; DIVERSI...	R&D intensity and manufacturing performance we...	Allison P., 1990, SOCIOL METHODOL, V20, P93, D...	LINTHICUM HTS	1998.0
1	Ottaway, TA; Burns, JR	Adaptive, agile approaches to organizational a...	DECISION SCIENCES	NaN	FLEXIBLE MANUFACTURING SYSTEMS	Intelligent agent-based approaches to software...	Anthony R.N., 1965, PLANNING CONTROL SYS; BUZA...	ATLANTA	1997.0

Then we check if two Articles' Titles share any words, and how many.

In [0]:

```
T_edgelist = []
for i in range(len(df['Title'])):
  for aut in df['Title_tok'][i]:
    for j in range(len(df['Title'])):
      if aut in df['Title_tok'][j] and i!=j and (j,i) not in T_edgelist:
        T_edgelist.append((i,j))
#T_edgelist
```

Then we make a graph from the edgelist.

In [0]:

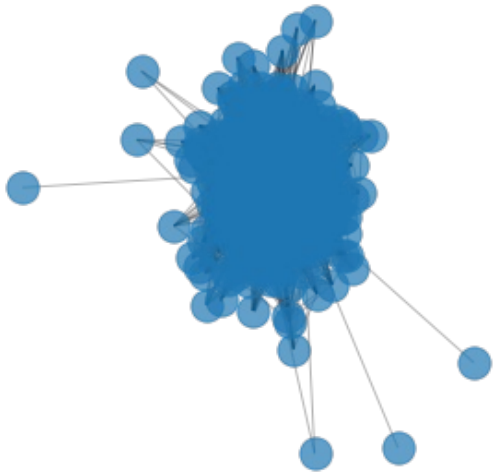
```
T = nx.Graph()
for [u,v] in T_edgelist:
  if T.has_edge(u,v):
    T[u][v]['weight'] += 1
  else:
    T.add_edge(u,v, weight=1)
```

And we plot it.

In [0]:

```
pos=nx.spring_layout(T)
plt.figure(figsize=(5, 5))
nx.draw_networkx_edges(T, pos, alpha=0.3, width=[T[u][v]['weight'] for u,v in T.edges()]
)
nx.draw_networkx_nodes(T, pos, alpha=0.7)


plt.axis('off')
plt.show()
```



The graph isn't verry informative, so we restrict us to looking at articles that share atleast 6 words in their Titles.

T6

Here we make a copy of T and call it T6, then we remove edges with a 'weight' less than 6, and remove any nodes that are isolated.

In [0]:

```
T6 = copy.deepcopy(T)
BadEdges=[]
for (u,v, d) in T6.edges(data=True):
    if d['weight'] < 6:
        BadEdges.append((u,v))
T6.remove_edges_from(BadEdges)
T6.remove_nodes_from(list(nx.isolates(T6)))
```

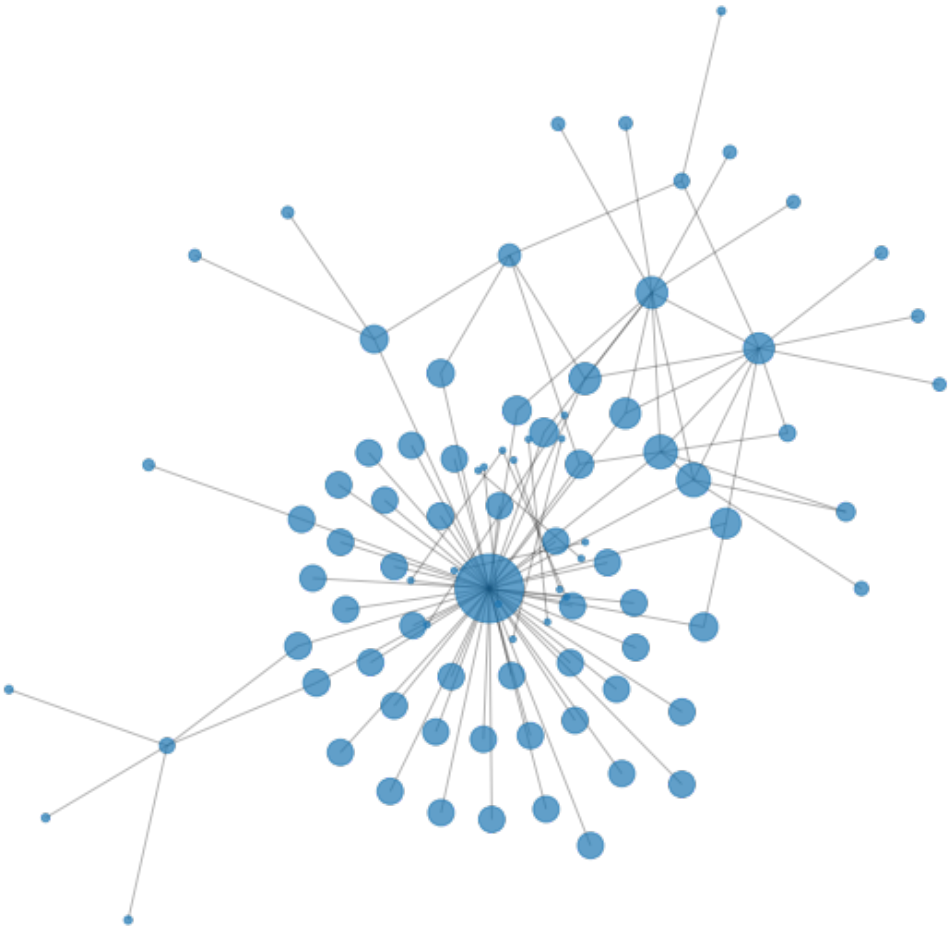
Here we plot the graph, we have added the eigenvector centrality as the size of the nodes this time.

In [0]:

```
eigen = nx.eigenvector_centrality(T6)

pos=nx.kamada_kawai_layout(T6)
plt.figure(figsize=(10, 10))
nx.draw_networkx_edges(T6, pos, alpha=0.3)
nx.draw_networkx_nodes(T6, pos, alpha=0.7,node_size=[v * 2000 + 10 for v in eigen.values
()])
```

```
plt.axis('off')
plt.show()
```



We clearly see one article that shares 6 words with many articles, and which has a high eigenvector centrality. We use the eigenvector centrality to identify this article.

In [0]:

```
print(sorted(nx.eigenvector_centrality(T6).items(), key=lambda x: x[1], reverse=True)[:1
])
```

```
[(641, 0.6732027568772992)]
```

And here we print the Title of the 3 Articles with the highest eigenvector centrality.

In [0]:

```
x=list(dict(sorted(nx.eigenvector_centrality(T6).items(), key=lambda x: x[1], reverse=Tr
ue)).keys())[:3]

for i in x:
  print(str(i)+':',df.iloc[i,1])
```

```
641: Dynamic supply chain capabilities: How market sensing, supply chain agility and adap
tability affect supply chain ambidexterity
726: The effects of strategic and manufacturing flexibilities and supply chain agility on
firm performance in the fashion industry
714: Developing Green Performance Through Supply Chain Agility in Manufacturing Industry:
A Case Study Approach
```

Not suprigsingly has the transitivity decressed from T to T6.

Finding Communities

Finding Communities

To help make the plot of T6 more informative we use networkx' community algorithm.

In [0]:

```
from networkx.algorithms import community

communities_generator = community.girvan_newman(T6)
top_level_communities = next(communities_generator)
next_level_communities = next(communities_generator)
List=sorted(map(sorted, next_level_communities))
```

In [0]:

```
#sorted(List,key=len,reverse=True) #Used this to see how many 'Good' communities the algo
rithm found.
```

So here we set a base color, gray, for all the nodes a an attribute, and then we color the nodes of the three bigest comunitis, using Pastel1_3.hex_colors.

In [0]:

```
nx.set_node_attributes(T6,'tab:gray','color')

for node in T6.nodes():
  for i in range(len(sorted(List,key=len,reverse=True)[0:3])):
    if node in sorted(List,key=len,reverse=True)[i]:
      T6.nodes()[node]['color']=Pastel1_3.hex_colors[i]
```

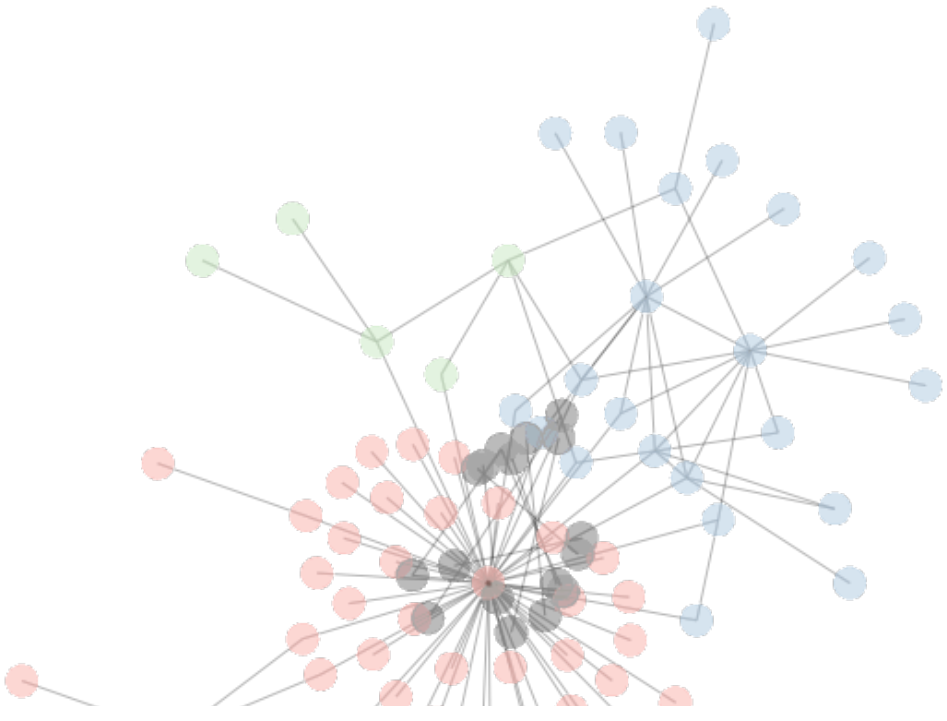
Then we plot the graph, using the colors.

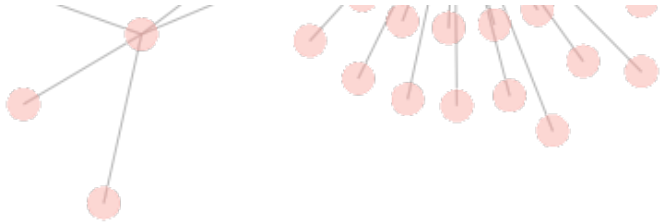
In [0]:

```
pos=nx.kamada_kawai_layout(T6)
plt.figure(figsize=(10, 10))
nx.draw_networkx_edges(T6, pos, alpha=0.3)

for i in T6.nodes:
  nx.draw_networkx_nodes(T6, pos, alpha=0.01, node_color=nx.get_node_attributes(T6,'colo
r').values())

plt.axis('off')
plt.show()
```





So the algorithm found 3 nice clusters (red, blue, green) and it made 8 clusters of 'the rest' that we just choose to color gray, since they seemed confused. So now we find the most central article in each of the three communities.

In [0]:

```
def most_central(liste,cent):
  for cen in cent:
    for num in liste:
        if num == cen[0]:
          return(cen[0],cen[1])
cent_sort = sorted(nx.degree_centrality(T6).items(), key=lambda x: x[1], reverse=True)

for li in sorted(List,key=len,reverse=True)[:3]:
  print(most_central(li,cent_sort), df['Title'][most_central(li,cent_sort)[0]])
```

```
(641, 0.5617977528089888) Dynamic supply chain capabilities: How market sensing, supply c
hain agility and adaptability affect supply chain ambidexterity
(461, 0.12359550561797752) The effect of supply chain agility on export performance The m
ediating roles of supply chain responsiveness and innovativeness
(518, 0.056179775280898875) The structural impact of supply chain management teams Supply
chain agility development in multidivisional firms
```

So in the first community we have something with: Capabilities, adaptability and ambidexterity. In the second:Export performance, responsiveness and innovativeness. And in the third: Structural impact.

Recomending colaborations between authors based on open triangles

Like first we make a new collumn where the Authors are stored in a list.

In [0]:

```
df['AuthorS']=[df['Author'][i].split(';') for i in range(len(df['Author']))]
```

Then we use itertool.combinations to make all to edges we need for the network, and we use networkx to create the Graph object.

In [0]:

```
Author_Combs = [list(itertools.combinations(df.AuthorS[i],2)) for i in df.index.values]
```

In [0]:

```
A2A = nx.Graph()
for [u,v] in list(itertools.chain(*Author_Combs)):
  if A2A.has_edge(u,v):
    A2A[u][v]['weight'] += 1
  else:
    A2A.add_edge(u,v, weight=1)
```

Since there are way too many edges and nodes in the full network, we first plot the graph where we only include authors that have worked together at least 2 times.

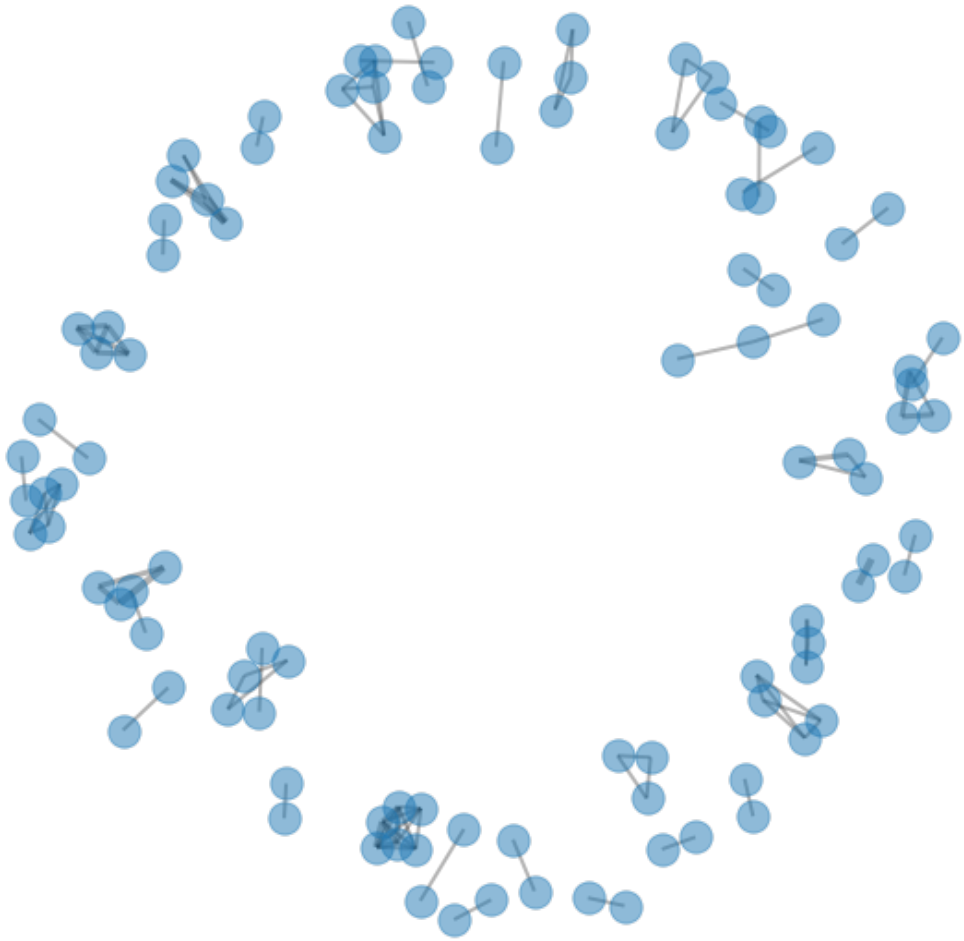
In [0]:

In [36]:

```
A2A_2 = copy.deepcopy(A2A)
BadEdges=[]
for (u,v, d) in A2A_2.edges(data=True):
    if d['weight'] < 2:
        BadEdges.append((u,v))
A2A_2.remove_edges_from(BadEdges)
A2A_2.remove_nodes_from(list(nx.isolates(A2A_2)))
```

In [37]:

```
pos=nx.spring_layout(A2A_2,k=0.3,seed=3)
plt.figure(figsize=(10, 10))
nx.draw_networkx_edges(A2A_2, pos, alpha=0.3, width=[A2A_2[u][v]['weight'] for u,v in A2
A_2.edges()])
nx.draw_networkx_nodes(A2A_2, pos, alpha=0.5)
plt.axis('off')
plt.show()
```



Here we see that pairs of authors, or trios, that work together multiple times are quit common. Now we want to explore the possibility to recommend future colaboraters based open triangles.

In [38]:

```
recommended = defaultdict(int)

for n, d in A2A.nodes(data=True):
    for n1, n2 in itertools.combinations(A2A.neighbors(n), 2):
      if not A2A.has_edge(n1, n2):
        recommended[(n1, n2)] += 1

recommended.items()
top_10 = sorted(recommended.items(), key=lambda kv: kv[1], reverse=True)[:10]
print(len(list(recommended.items())))
top_10
```

512

512

Out[38]:

```
[((' Dwivedi, Rajeev', ' Rigoni, Eduardo Henrique'), 4),
 ((' Dwivedi, Rajeev', ' Huang, Zhengwei (David)'), 4),
 ((' Rigoni, Eduardo', ' Rigoni, Eduardo Henrique'), 4),
 ((' Rigoni, Eduardo', ' Huang, Zhengwei (David)'), 4),
 ((' Rafiei, Farimah Mokhatab', ' Hisjam, Muhammad'), 3),
 ((' Liu, Jia', ' Wu, Qian'), 3),
 ((' Liu, Jia', ' Liu, Yang'), 3),
 ((' Chaudhry, Sohail S.', ' Wu, Qian'), 3),
 ((' Chaudhry, Sohail S.', ' Liu, Yang'), 3),
 ((' Wu, Qun', ' Goldsby, Thomas J.'), 2)]
```

So we found 512 open triangles that we can use to recommend future colaborators. We filter out the top 10 recommendations, based on how many neighbours they share, and use this for some visualization.

First we make a graph with only the authors we have recomendations for in the top 10, and their neighbours.

In [0]:

```
rec_auts = set(itertools.chain(*dict(top_10).keys()))
```

In [40]:

```
A_rec_nodes=[list(A2A.neighbors(node)) for node in rec_auts]
A_rec_nodes.append(rec_auts)
A_rec_nodes=list(itertools.chain(*A_rec_nodes))
A_rec=A2A.subgraph(A_rec_nodes)
A_rec.nodes()
```

Out[40]:

```
NodeView((' Goldsby, Thomas', ' Goldsby, Thomas J.', ' Huang, Zhengwei (David)', 'Xie, Ke
fan', ' Chaudhry, Sohail S.', ' Rafiei, Farimah Mokhatab', ' Liu, Jia', ' Helmi, Syed Ahm
ad', ' Chung, Chen', ' Derksen, Barry', ' Zadeh, Hossein S.', ' Hisjam, Muhammad', ' Wang
, Pan', ' Rigoni, Eduardo Henrique', ' Santana, Martin', ' Chen, Gang', 'Luftman, Jerry',
' Dwivedi, Rajeev', 'Li, Xun', ' Rigoni, Eduardo', ' Holsapple, Clyde W.', ' Wu, Qun', '
Liu, Yang', ' Rahim, Abd Rahman Abdul', 'Galankashi, Masoud Rahiminezhad', ' Wu, Qian'))
```

Then we make the same gaph again where we also add the recomndations.

- **This is done like this since you can't add edges to a graph created as a subgraph.**

In [0]:

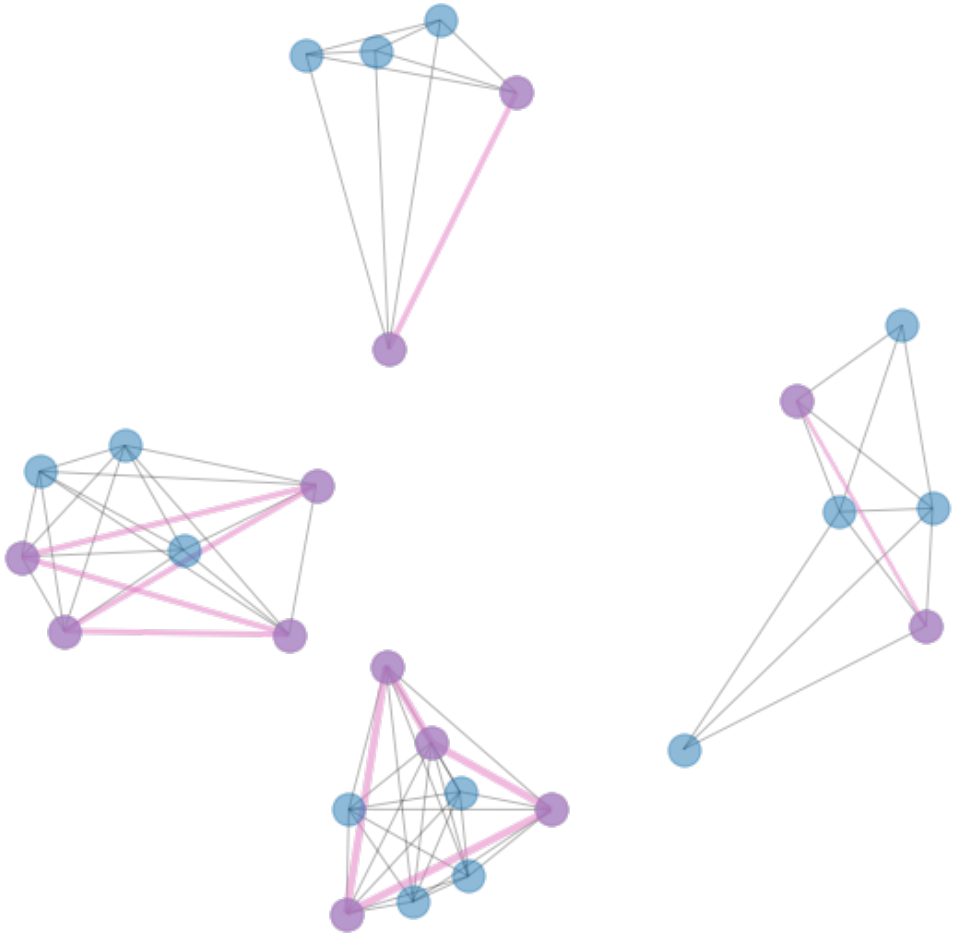
```
A2A_X=copy.deepcopy(A2A)
X_edges=list(dict(top_10).keys())
A2A_X.add_edges_from(X_edges)

A_recX=A2A_X.subgraph(A_rec_nodes)
```

And then we plot the graph, the pink nodes are authors we have recommendation for, and the pink edges are the recommendations. The wider the edge, the more neighbours to the two nodes share.

In [42]:

```
pos=nx.spring_layout(A_recX,scale=2,k=1,seed=2)
plt.figure(figsize=(10,10))
nx.draw_networkx_edges(A_recX, pos, edgelist=A_rec.edges(), alpha=0.3)
nx.draw_networkx_nodes(A_recX, pos, alpha=0.5)
nx.draw_networkx_nodes(A_recX, pos, nodelist=rec_auts , alpha=0.5,node_color='tab:pink')
nx.draw_networkx_edges(A_recX, pos, edgelist=list(dict(top_10).keys()), alpha=0.5,edge_co
lor='tab:pink',width=list(dict(top_10).values()))
plt.axis('off')
plt.show()
```



So we see four disconnected components, and the top 10 reccomendations.