se-networks

Unknown Author

February 10, 2014

1 Bipartite Networks with iPython

- Author: Stefan Kasberger
- Date: February 2014
- Course: SE Networks from Prof. Peter Csermely
- GitHub & openscienceASAP
- networkx bipartite module

1.1 Setup and Preprocessing

Import modules

In [78]:
```
import networkx as nx
from networkx.algorithms import bipartite
from networkx.algorithms import centrality
from networkx.algorithms import distance_measures
from networkx.algorithms import shortest_paths
from networkx.algorithms import components
from networkx.algorithms import isolates
from networkx.classes import function
import matplotlib.pyplot as plt
import math
import csv
```

Import config file with your working directory = dir_se_networks

In [79]:
```
from config import *

# CUSTOM WORKING DIRECTORY
# to add your working directory manually, uncomment the next line and add it instead o
# dir_se_networks = 'YOUR/FOLDER'
```

Read in filesRead out the data from the text files. The first column of the text file is the ID for the user, the second one is the ID for the Project. Save both columns as seperated lists for nodes.

In [80]:
```
users = []
projects = []

# read out text file and save every column in a list
with open(dir_se_networks + 'data/raw/github/out.github', 'rb') as csvfile:
    nodes = csv.reader(csvfile, delimiter=' ')
    for node in nodes:
        users.append('A' + node[0])
        projects.append('B' + node[1])
csvfile.close()
```

```
# delete head entries
del users[0]
del projects[0]
```

Read out the data from the text files. The first column is the ID for the entitites, the second the ID for the Countries. Save both columns as seperated lists for nodes.

In [81]:
```
entities = []
countries = []

# read out text file and save every column in a list
with open(dir_se_networks + 'data/raw/dbpedia-country/out.dbpedia-country', 'rb') as c
    nodes = csv.reader(csvfile, delimiter=' ')
    for node in nodes:
        entities.append('A' + node[0])
        countries.append('B' + node[1])
csvfile.close()

# delete head entries
del entities[0]
del countries[0]
```

Create Graphs

Create graph and add the the user and project lists as nodes to it.

In [82]:
```
B_Github = nx.Graph()
B_Github.add_nodes_from(users, bipartite=0)
B_Github.add_nodes_from(projects, bipartite=1)
print nx.function.info(B_Github)
```

```
Name:
Type: Graph
Number of nodes: 177386
Number of edges: 0
Average degree:   0.0000
```

Create graph and add the the entities and countries lists as nodes to it.

In [83]:
```
B_dbpedia = nx.Graph()
B_dbpedia.add_nodes_from(entities, bipartite=0)
B_dbpedia.add_nodes_from(countries, bipartite=1)
print nx.function.info(B_dbpedia)
```

```
Name:
Type: Graph
Number of nodes: 550522
Number of edges: 0
Average degree:   0.0000
```

Create subset for user and project nodes.

In [84]:
```
users_nodes = set(n for n,d in B_Github.nodes(data=True) if d['bipartite']==0)
projects_nodes = set(B_Github) - users_nodes
print 'User: ' + str(len(users_nodes)) + ' and Projects: ' + str(len(projects_nodes))
```

```
User: 56519 and Projects: 120867
```

Create subset for entities and countries nodes.

In [85]:
```
entities_nodes = set(n for n,d in B_dbpedia.nodes(data=True) if d['bipartite']==0)
countries_nodes = set(B_dbpedia) - entities_nodes
print 'Entities: ' + str(len(entities_nodes)) + ' and Countries: ' + str(len(countries
```

```
Entities: 548077 and Countries: 2445
```

Create a list of edge tuples and add them as edges to the graph.

In [86]:
```
edge_list = []
for i in range(len(users)):
    edge_list.append((user[i], projects[i]))

B_Github.add_edges_from(edge_list)
print nx.function.info(B_Github)
```

```
Name:
Type: Graph
Number of nodes: 177386
Number of edges: 440237
Average degree:   4.9636
```

In [87]:
```
edge_list = []
for i in range(len(entities)):
    edge_list.append((entities[i], countries[i]))

B_dbpedia.add_edges_from(edge_list)
print nx.function.info(B_dbpedia)
```

```
Name:
Type: Graph
Number of nodes: 550522
Number of edges: 584947
Average degree:   2.1251
```

Check if graph is bipartite

In [88]:
```
print(nx.bipartite.is_bipartite(B_Github))
print(nx.bipartite.is_bipartite(B_dbpedia))
```

```
True
True
```

Check Connectednes

In [89]:
```
print 'Connected: Github => ' + str(nx.is_connected(B_Github)) + ', dbpedia => ' + str
```

```
Connected: Github => False, dbpedia => False
```

In [90]:
```
print 'Isolation: Github => ' + str(nx.isolates(B_Github)) + ', dbpedia => ' + str(nx.
```

```
Isolation: Github => [], dbpedia => []
```

Get Connected Components

Cause the graph is not fully connected, and this is neccessary for further analysis, the giant component (largest connected component) gets calculated, then the second largest connected Component.

In [91]:
```
GC_Github = components.connected_component_subgraphs(B_Github)[0]
print nx.function.info(GC_Github)
```

```
Name:
Type: Graph
Number of nodes: 139752
Number of edges: 417361
Average degree:   5.9729
```

In [92]:
```
GC_dbpedia = components.connected_component_subgraphs(B_dbpedia)[0]
print nx.function.info(GC_dbpedia)
```

```
Name:
Type: Graph
Number of nodes: 544947
Number of edges: 580231
Average degree:   2.1295
```

In [93]:
```
GC_users_nodes = set(n for n,d in GC_Github.nodes(data=True) if d['bipartite']==0)
GC_projects_nodes = set(GC_Github) - GC_users_nodes
print 'Users: ' + str(len(GC_users_nodes)) + ' and Projects: ' + str(len(GC_projects_n
```

```
Users: 39845 and Projects: 99907
```

In [94]:
```
GC_entities_nodes = set(n for n,d in GC_dbpedia.nodes(data=True) if d['bipartite']==0)
GC_countries_nodes = set(GC_dbpedia) - GC_entities_nodes
print 'Entities: ' + str(len(GC_entities_nodes)) + ' and Countries: ' + str(len(GC_cou
```

```
Entities: 543589 and Countries: 1358
```

In [95]:
```
LCC2_Github = components.connected_component_subgraphs(B_Github)[1]
print nx.function.info(LCC2_Github)
```

```
Name:
Type: Graph
Number of nodes: 45
Number of edges: 44
Average degree:   1.9556
```

In [96]:
```
LCC2_dbpedia = components.connected_component_subgraphs(B_dbpedia)[1]
print nx.function.info(LCC2_dbpedia)
```

```
Name:
Type: Graph
Number of nodes: 338
Number of edges: 337
Average degree:   1.9941
```

In [118]:
```
LCC2_users_nodes = set(n for n,d in LCC2_Github.nodes(data=True) if d['bipartite']==0)
LCC2_projects_nodes = set(LCC2_Github) - LCC2_users_nodes
print 'User: ' + str(len(LCC2_users_nodes)) + ' and Projects: ' + str(len(LCC2_project
```

```
User: 3 and Projects: 42
```

In [98]:
```
LCC2_entities_nodes = set(n for n,d in LCC2_dbpedia.nodes(data=True) if d['bipartite']
LCC2_countries_nodes = set(LCC2_dbpedia) - LCC2_entities_nodes
print 'Entities: ' + str(len(LCC2_entities_nodes)) + ' and Countries: ' + str(len(LCC2
```

```
Entities: 337 and Countries: 1
```

Check Connectednes of Giant Component

In [99]:
```
print 'Connected, Giant Component: Github => ' + str(nx.is_connected(GC_Github)) + ',
print 'Connected, 2nd largest Connected Component: Github => ' + str(nx.is_connected(L
```

```
Connected, Giant Component: Github => True, dbpedia => True
Connected, 2nd largest Connected Component: Github => True, dbpedia =>
True
```

In [100]:
```
print 'Isolation: Github => ' + str(nx.isolates(GC_Github)) + ', dbpedia => ' + str(nx
print 'Isolation Reduced: Github => ' + str(nx.isolates(LCC2_Github)) + ', dbpedia =>
```

```
Isolation: Github => [], dbpedia => []
Isolation Reduced: Github => [], dbpedia => []
```

Weighted Projection

This can not be done cause of too expensive computation costs.

Weighted projection to user-user unipartite network.

In [115]:
```
# GC
# P_Github = nx.bipartite.weighted_projected_graph(GC_Github, GC_user_nodes, ratio=Fal
# print nx.classes.function.info(P_Github)

# LCC2
P_Github_LCC2 = nx.bipartite.weighted_projected_graph(LCC2_Github, LCC2_user_nodes, ra
print nx.classes.function.info(P_Github_LCC2)
```

```
Name:
Type: Graph
Number of nodes: 3
Number of edges: 3
Average degree:   2.0000
```

Weighted projection to entitie-entitie unipartite network.

In [116]:
```
# GC
# P_dbpedia = bipartite.weighted_projected_graph(GC_dbpedia_red, entities_nodes_red, r
# print function.info(P_dbpedia)

# LCC2
P_dbpedia_LCC2 = bipartite.weighted_projected_graph(LCC2_dbpedia, LCC2_entities_nodes,
print function.info(P_dbpedia_LCC2)
```

```
Name:
Type: Graph
Number of nodes: 337
Number of edges: 56616
Average degree: 336.0000
```

1.2 Analysis

Degree Centrality

Calculate the Degree Centrality.

In [76]:
```
DC_Github = bipartite.degree_centrality(GC_Github, users_nodes)
```

In [77]:
```
DC_dbpedia = bipartite.degree_centrality(GC_dbpedia, entities_nodes)
```

Betweeness Centralitiy

Calculate Betwenness Centrality

In [120]:
```
# GC
# BC_Github = bipartite.betweenness_centrality(GC_Github, GC_users_nodes)

# LCC2
LCC2_BC_Github = bipartite.betweenness_centrality(LCC2_Github, LCC2_users_nodes)
```

In [127]:
```
# GC
# BC_dbpedia = bipartite.betweenness_centrality(GC_dbpedia, GC_entities_nodes)

# LCC2 => ZeroDivisionError: float division by zero
# LCC2_BC_dbpedia = bipartite.betweenness_centrality(LCC2_dbpedia, LCC2_entities_nodes
```

```
        ------------------------------------------------------------
-------------
    ZeroDivisionError                         Traceback (most recent
call last)



        <ipython-input-127-c5cc5e0e64ab> in <module>()
          3
          4 # LCC2 => ERROR
    ----> 5 LCC2_BC_dbpedia =
bipartite.betweenness_centrality(LCC2_dbpedia, LCC2_entities_nodes)




        /usr/local/lib/python2.7/dist-
packages/networkx/algorithms/bipartite/centrality.pyc in
betweenness_centrality(G, nodes)
        164                                                 weight=None)
        165     for node in top:
    --> 166         betweenness[node]/=bet_max_top
        167     for node in bottom:
        168         betweenness[node]/=bet_max_bot



        ZeroDivisionError: float division by zero
```

Closeness Centrality

Calculate Closeness Centrality

In [130]:
```
# GC
# GC_CC_Github = nx.algorithms.closeness_centrality(GC_Github, GC_users_nodes)

# LCC2 => TypeError: unhashable type: 'set'
# LCC2_CC_Github = nx.algorithms.closeness_centrality(LCC2_Github, LCC2_users_nodes)
```

In [129]:
```
# GC
# GC_CC_dbpedia = nx.algorithms.closeness_centrality(GC_dbpedia, GC_entities_nodes)

# LCC2 => TypeError: unhashable type: 'set'
# LCC2_CC_dbpedia = nx.algorithms.closeness_centrality(LCC2_dbpedia, LCC2_entities_nod
```

Shortest Paths

In [132]:
```
# GC
# GC_SP_Github = shortest_paths.shortest_path_length(GC_Github)
# GC_SP_dbpedia = shortest_paths.shortest_path_length(GC_dbpedia)

# LCC2
```

```
LCC2_SP_Github = shortest_paths.shortest_path_length(LCC2_Github)
LCC2_SP_dbpedia = shortest_paths.shortest_path_length(LCC2_dbpedia)
```

In [134]:
```
# GC
# GC_ASP_Github = shortest_paths.average_shortest_path_length(GC_Github)
# print 'Avg. Shortest Path Github: ' + str(GC_ASP_Github)

# LCC2
LCC2_ASP_Github = shortest_paths.average_shortest_path_length(LCC2_Github)
print 'Avg. Shortest Path Github: ' + str(LCC2_ASP_Github)
```

```
Avg. Shortest Path Github: 2.11919191919
```

In [135]:
```
# GC
# GC_ASP_dbpedia = shortest_paths.average_shortest_path_length(GC_dbpedia)
# print 'Avg. Shortest Path Github: ' + str(GC_ASP_dbpedia)

# LCC2
LCC2_ASP_dbpedia = shortest_paths.average_shortest_path_length(LCC2_dbpedia)
print 'Avg. Shortest Path Github: ' + str(LCC2_ASP_dbpedia)
```

```
Avg. Shortest Path Github: 1.99408284024
```

Clustering

In [161]:
```
# GC
# GC_CC_Github = bipartite.cluster.clustering(B_Github, users_nodes)

# LCC2
LCC2_CC_Github = bipartite.cluster.clustering(LCC2_Github)
LCC2_CC_Github_users = bipartite.cluster.clustering(LCC2_Github, LCC2_users_nodes)
LCC2_CC_Github_projects = bipartite.cluster.clustering(LCC2_Github, LCC2_projects_node
```

In [163]:
```
# GC
# GC_CC_dbpedia = bipartite.cluster.clustering(B_dbpedia, entities_nodes)

# LCC2
LCC2_CC_dbpedia = bipartite.cluster.clustering(LCC2_dbpedia, LCC2_entities_nodes)
LCC2_CC_dbpedia_entities = bipartite.cluster.clustering(LCC2_dbpedia, LCC2_entities_no
LCC2_CC_dbpedia_countries = bipartite.cluster.clustering(LCC2_dbpedia, LCC2_countries_
```

Average Clustering Coefficient

In [194]:
```
# GC
# GC_ACC_Github = bipartite.cluster.average_clustering(B_Github, users_nodes)

# LCC2
LCC2_ACC_Github = bipartite.cluster.average_clustering(LCC2_Github)
LCC2_ACC_Github_users = bipartite.cluster.average_clustering(LCC2_Github, LCC2_users_n
LCC2_ACC_Github_projects = bipartite.cluster.average_clustering(LCC2_Github, LCC2_proj
ACC_P_Github_LCC2 = nx.cluster.average_clustering(P_Github_LCC2)
print 'Github Avg. Clustering Coefficient LCC2\nOverall: ' + str(LCC2_ACC_Github) + '\
```

```
Github Avg. Clustering Coefficient LCC2
Overall: 0.901071105949
Projection => 1.0
Users => 0.182733255904
Projects => 0.952380952381
```

In [196]:
```
# GC
# GC_ACC_dbpedia = bipartite.cluster.average_clustering(B_dbpedia, entities_nodes)

# LCC2
LCC2_ACC_dbpedia = bipartite.cluster.average_clustering(LCC2_dbpedia)
LCC2_ACC_dbpedia_entities = bipartite.cluster.average_clustering(LCC2_dbpedia, LCC2_en
```

```
LCC2_ACC_dbpedia_countries = bipartite.cluster.average_clustering(LCC2_dbpedia, LCC2_c
ACC_P_dbpedia_LCC2 = nx.cluster.average_clustering(P_dbpedia_LCC2)
print 'dbpedia Avg. Clustering Coefficient LCC2\nOverall: ' + str(LCC2_ACC_dbpedia) +
```

```
dbpedia Avg. Clustering Coefficient LCC2
Overall: 0.997041420118
Projection => 1.0
Entities => 1.0
Countries => 0.0
```

Density

In [101]:
```
print 'Density Github: User => ' + str(nx.bipartite.density(B_Github, user)) + ' Proje
```

```
Density Github: User => -3.80443673412e-06 Projects =>
-3.80443673412e-06
```

In [34]:
```
print 'Density dbpedia: Entities => ' + str(nx.bipartite.density(B_dbpedia, entities))
```

```
Density dbpedia: Entities => -2.12714710275e-05 Countries =>
-2.12714710275e-05
```

Diameter

In [145]:
```
# DURATION
Diam_Github = nx.algorithms.distance_measures.diameter(LCC2_Github)
print 'Diameter Github: ' + str(Diam_Github)
```

```
Diameter Github: 4
```

In [146]:
```
# DURATION
Diam_dbpedia = nx.algorithms.distance_measures.diameter(LCC2_dbpedia)
print 'Diameter dbpedia: ' + str(Diam_dbpedia)
```

```
Diameter dbpedia: 2
```

1.3 Plot Diagrams

matplotlib

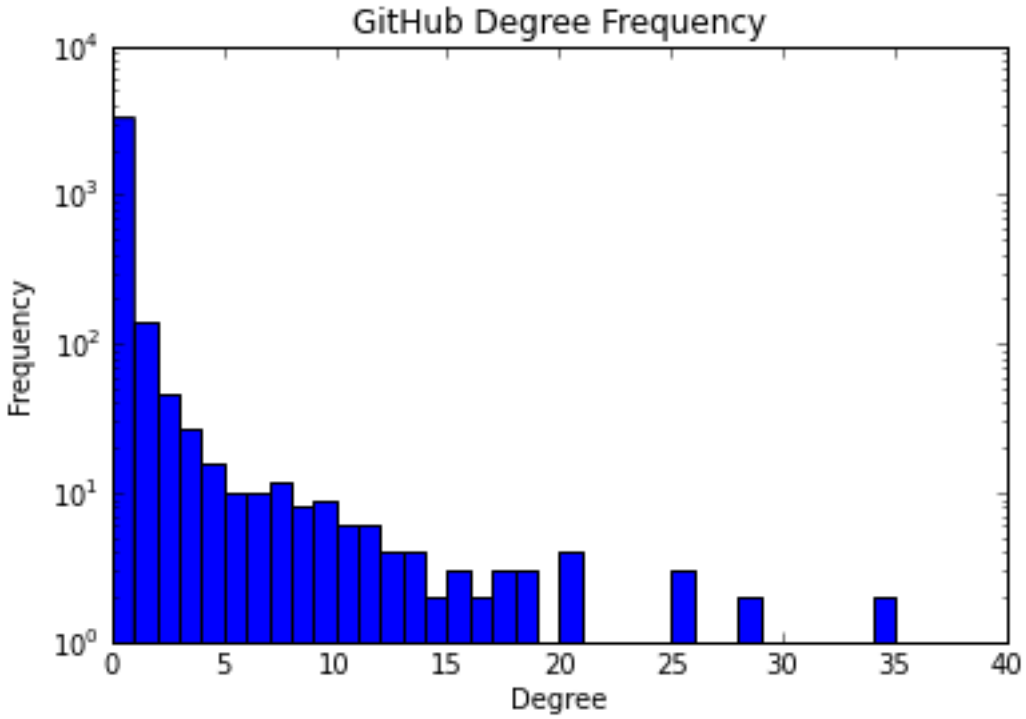
Degree

Degree Histogram returns a list of the frequency of each degree value.

In [105]:
```
DegFreq_Github = function.degree_histogram(B_Github)
print 'Degree Frequency\nLength: ' + str(len(DegFreq_Github)) + '\n' + 'Highest Freque
```

```
Length: 3676
Highest Frequency: 106430
```

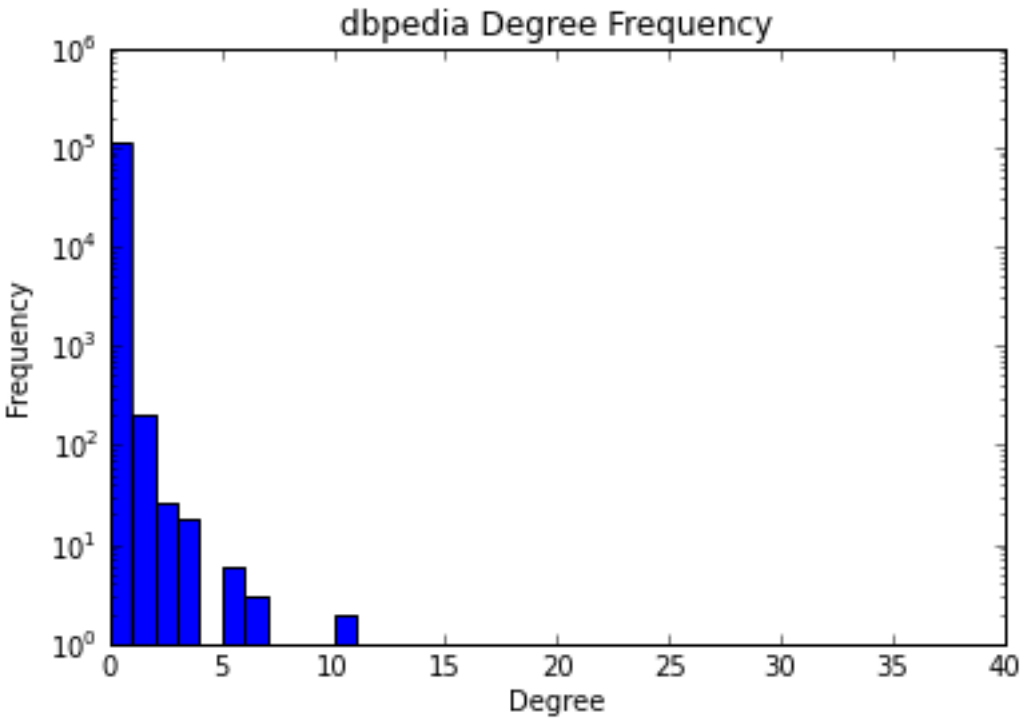
In [186]:
```
%matplotlib inline
plt.hist(DegFreq_Github, bins = 40, range = (0, 40), log=True)
plt.xlabel('Degree')
plt.ylabel('Frequency')
plt.title('GitHub Degree Frequency')
plt.savefig(dir_se_networks + 'figures/github_degfreq_hist.png')
```



In [107]:
```
DegFreq_dbpedia = function.degree_histogram(B_dbpedia)
print 'Degree Frequency\nLength: ' + str(len(DegFreq_dbpedia)) + '\n' + 'Highest Frequ
```

```
Length: 108686
Highest Frequency: 521213
```

In [113]:
```
%matplotlib inline
plt.hist(DegFreq_dbpedia, bins = 40, range = (0, 40), log=True)
plt.xlabel('Degree')
plt.ylabel('Frequency')
plt.title('dbpedia Degree Frequency')
plt.savefig(dir_se_networks + 'figures/dbpedia_degfreq_hist.png')
```

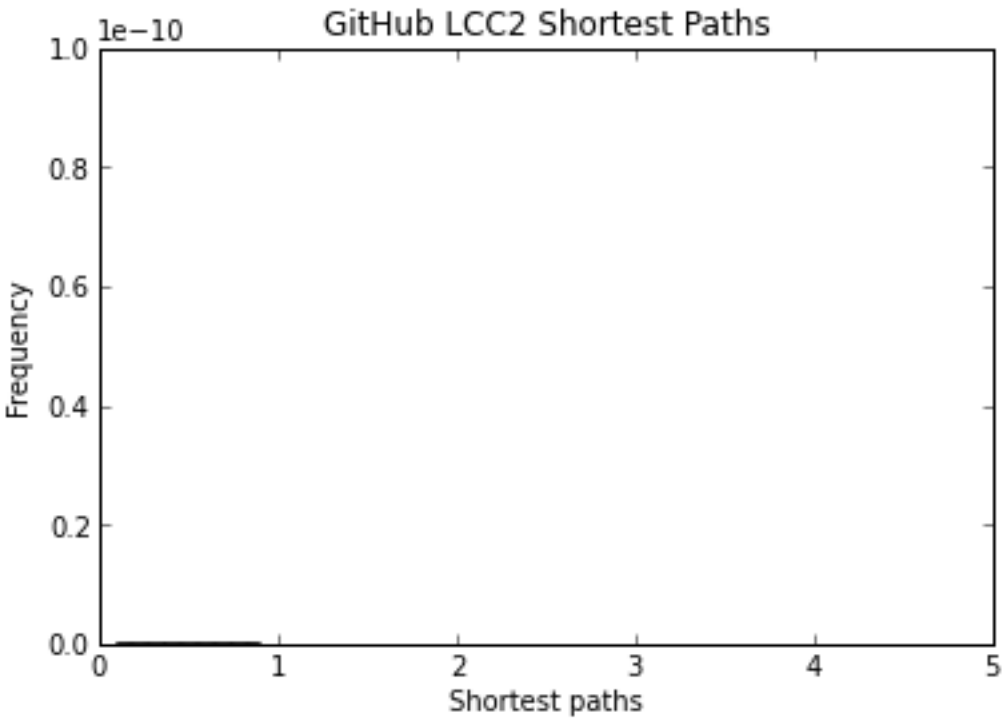


In [188]:
```
P_DegFreq_dbpedia = function.degree_histogram(P_dbpedia_LCC2)
print 'Projected dbpedia Degree Frequency \nLength: ' + str(len(P_DegFreq_dbpedia)) +
```

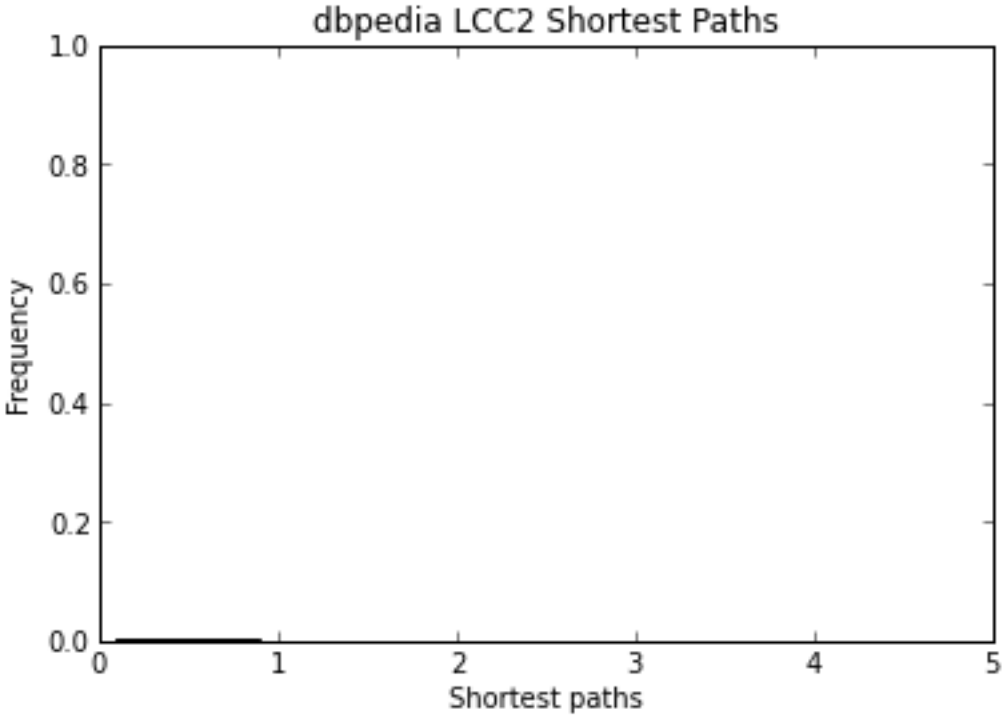
```
Projected dbpedia Degree Frequency
Length: 337
Highest Frequency: 0
```

Shortest Path Length

In [184]:
```
%matplotlib inline
plt.hist(LCC2_SP_Github.values(), bins = 5, range = (0, 5))
plt.xlabel('Shortest paths')
plt.ylabel('Frequency')
plt.ylim(ymax=0.0000000001)
plt.title('GitHub LCC2 Shortest Paths')
plt.savefig(dir_se_networks + 'figures/Github_LCC2_ShortPath_hist.png')
```



In [179]:
```
%matplotlib inline
plt.hist(LCC2_SP_dbpedia.values(), bins = 5, range = (0, 5))
plt.xlabel('Shortest paths')
plt.ylabel('Frequency')
plt.ylim(ymax=1)
plt.title('dbpedia LCC2 Shortest Paths')
plt.savefig(dir_se_networks + 'figures/dbpedia_LCC2_ShortPath_hist.png')
```

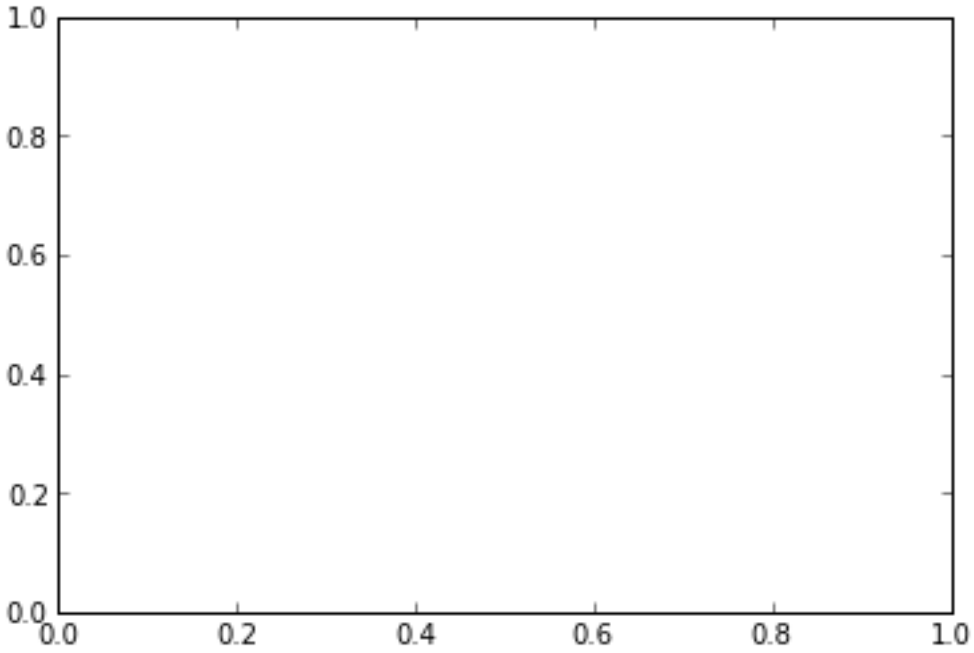


Clustering

In [180]:
```
%matplotlib inline
plt.hist(LCC2_CC_Github, bins = 5, range = (0, 5))
plt.xlabel('Clustering Coefficient')
plt.ylabel('Frequency')
plt.title('Github LCC2 Clustering Coefficient')
plt.savefig(dir_se_networks + 'figures/Github_ClustCoeff_hist.png')
```

```
---------------------------------------------------------------------
-------------
    KeyError                                  Traceback (most recent
call last)



        <ipython-input-180-438117286b5d> in <module>()
          1 get_ipython().magic(u'matplotlib inline')
    ----> 2 plt.hist(LCC2_CC_Github, bins = 5, range = (0, 5))
          3 plt.xlabel('Clustering Coefficient')
          4 plt.ylabel('Frequency')
          5 plt.title('Github LCC2 Clustering Coefficient')



        /usr/lib/pymodules/python2.7/matplotlib/pyplot.pyc in hist(x,
bins, range, normed, weights, cumulative, bottom, histtype, align,
orientation, rwidth, log, color, label, stacked, hold, **kwargs)
       2670                      histtype=histtype, align=align,
orientation=orientation,
       2671                      rwidth=rwidth, log=log, color=color,
label=label,
    -> 2672                      stacked=stacked, **kwargs)
       2673         draw_if_interactive()
       2674     finally:



        /usr/lib/pymodules/python2.7/matplotlib/axes.pyc in hist(self,
x, bins, range, normed, weights, cumulative, bottom, histtype, align,
orientation, rwidth, log, color, label, stacked, **kwargs)
       8024         # Massage 'x' for processing.
       8025         # NOTE: Be sure any changes here is also done
below to 'weights'
    -> 8026         if isinstance(x, np.ndarray) or not
iterable(x[0]):
       8027             # TODO: support masked arrays;
       8028             x = np.asarray(x)
```

```
KeyError: 0
```



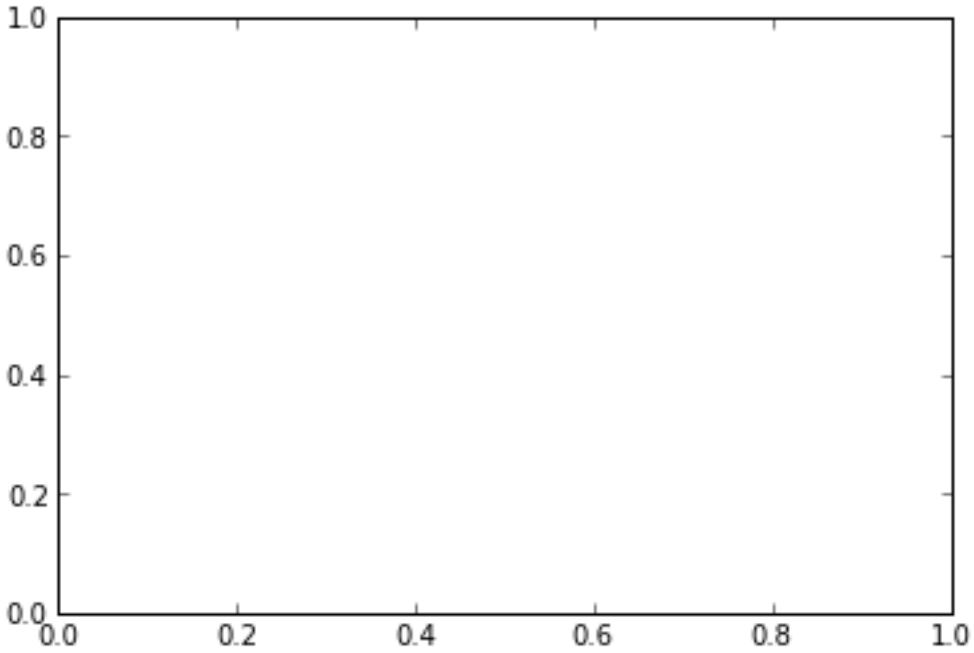
In [181]:
```
%matplotlib inline
plt.hist(LCC2_CC_dbpedia, bins = 5, range = (0, 5))
plt.xlabel('Clustering Coefficient')
plt.ylabel('Frequency')
plt.title('dbpedia LCC2 Clustering Coefficient')
plt.savefig(dir_se_networks + 'figures/dbpedia_ClustCoeff_hist.png')
# LCC2_CC_dbpedia_entities
# LCC2_CC_dbpedia_countries
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent
call last)

<ipython-input-181-40fb69c614fc> in <module>()
      1 get_ipython().magic(u'matplotlib inline')
----> 2 plt.hist(LCC2_CC_dbpedia, bins = 5, range = (0, 5))
      3 plt.xlabel('Clustering Coefficient')
      4 plt.ylabel('Frequency')
      5 plt.title('dbpedia LCC2 Clustering Coefficient')

/usr/lib/pymodules/python2.7/matplotlib/pyplot.pyc in hist(x,
bins, range, normed, weights, cumulative, bottom, histtype, align,
orientation, rwidth, log, color, label, stacked, hold, **kwargs)
   2670                         histtype=histtype, align=align,
orientation=orientation,
```

```
       2671                         rwidth=rwidth, log=log, color=color,
label=label,
    -> 2672                         stacked=stacked, **kwargs)
       2673         draw_if_interactive()
       2674     finally:
```

```
        /usr/lib/pymodules/python2.7/matplotlib/axes.pyc in hist(self,
x, bins, range, normed, weights, cumulative, bottom, histtype, align,
orientation, rwidth, log, color, label, stacked, **kwargs)
       8024         # Massage 'x' for processing.
       8025         # NOTE: Be sure any changes here is also done
below to 'weights'
    -> 8026         if isinstance(x, np.ndarray) or not
iterable(x[0]):
       8027             # TODO: support masked arrays;
       8028             x = np.asarray(x)
```

```
        KeyError: 0
```



In []:

In []:

In []:

In [ ]: