Link Partitioning by Partioning Around Medoids with Clarance Heuristic

In [1]:

```
import numpy as np
import random
random.seed = 108
from tqdm import tqdm_notebook as tqdm
import seaborn as sns
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import pandas as pd
%matplotlib inline
```

In [2]:

```
!pwd
```

```
/home/latna/aponom/clustering/Scripts
```

In [3]:

```
!java -jar ../my_git/out/artifacts/Clustering_jar/Clustering.jar
```

```
Missing required options: i, k
usage: PMPClustering <-i input_file> <-k number_of_clusters> [-ftb
cd]
 -a,--algorithm <arg>          Specifies algorithm that will be use
d to
                               find disjoint edge clusters. Possibl
e
                               values:
                               pmp (p-median exact algorithm).
                               kmd (k-medoids heuristic)
                               kmn (k-means heuristic).
                               If this option is omitted, the P-Med
ian
                               algorithm will be used.
 -b,--benchmark                use benchmark format
 -c,--communitiesFile <arg>    use communities file to validate ben
chmark
 -d,--distance <arg>           the type of function to measure the
distance
                               between nodes.
                               Possible values:
                               sp (shortest path)
                               gd (Generalize Degree)
                               cm (Commute Distance)
                               acm (Amplified Commute Distance).
                               If this option is omitted, the ampli
fied
                               commute distance function will be us
ed.
 -f,--force                    The previous founded solution by lp_
solver
                               will not be used. lp_solver will be
started
                               forcibly.
                               The flag effects only the PMP exact
edge
                               clustering algorithm.
                               By default algorithm will try to get
                               previous founded solution
 -i,--input <arg>              path to the input file in GML format
 -k <arg>                      Number of clusters to detect (int)
 -l,--linegraph                produce line graph as output
 -o,--outputDir <arg>          the name of the output file. If opti
on was
                               omitted, the name of the output file
                               will be composed automatically as
                               "{suffix}_{distanceName}_{clusterNum
ber}_out
                               .gexf"
 -t,--threshold <arg>          Vertex i belongs to cluster C, if no
de i has
                               fraction of edges in cluster C great
er then
                               this value
                               Default value is 0.


In [138]:
```

In [138]:

```
def lp_experiment(clustersNumber, algorithm, distance, inputFile, groundTruth,
params = {}, vertexNumerationShift=0, benchmarkFormat=False):
    datasetName = inputFile.split('/')[-2]
    sufix = inputFile.split('/')[-1].split('.')[0]
    outputDir = "lp_{}_{}_{}".format(algorithm, distance, datasetName)
    outputFile = outputDir +'/' +"pmp_{3}_{2}_{0}_{1}.dat".format(algorithm, c
lustersNumber, distance.upper(), sufix)
    print("Output dir name: {}".format(outputDir) )
    print("Output file name: {}".format(outputFile) )
    all_results = {}
    bestParam = "not found"
    nmi_best = 0;
    param_list = list(generate_params(params))
    for param in tqdm(param_list):
        if benchmarkFormat:
            tmp=!java -jar ../my_git/out/artifacts/Clustering_jar/Clustering.j
ar -b -a {algorithm} -o {outputDir} -i {inputFile} -k {clustersNumber} -d {dis
tance} {param}
            tmpFile = outputFile
        else:
            tmp=!java -jar ../my_git/out/artifacts/Clustering_jar/Clustering.j
ar -a {algorithm} -o {outputDir} -i {inputFile} -k {clustersNumber} -d {distan
ce} {param}
            lines=[]
            with open(outputFile) as f:
                lines = f.readlines()
            tmpFile="tmpFile.dat"
            with open(tmpFile, 'w') as the_file:
                for line in lines:
                    the_file.write(" ".join([str(int(a)+vertexNumerationShift)
for a in line.split()]) + "\n")

        output=!../pipeline/Overlapping-NMI/onmi {groundTruth} {tmpFile}
        nmi=float(output[0].split()[1])
        all_results[param] = nmi
        if nmi >  nmi_best:
            bestParam = param
            nmi_best = nmi
    #restorign solution for the best parameters
    if benchmarkFormat:
        tmp=!java -jar ../my_git/out/artifacts/Clustering_jar/Clustering.jar -
b -a {algorithm} -o {outputDir} -i {inputFile} -k {clustersNumber} -d {distanc
e} {bestParam}
        tmpFile = outputFile
    else:
        tmp=!java -jar ../my_git/out/artifacts/Clustering_jar/Clustering.jar -
a {algorithm} -o {outputDir} -i {inputFile} -k {clustersNumber} -d {distance}
{bestParam}
        lines=[]
        with open(outputFile) as f:
            lines = f.readlines()
        tmpFile="tmpFile.dat"
        with open(tmpFile, 'w') as the_file:
            for line in lines:
                the_file.write(" ".join([str(int(a)+vertexNumerationShift) for
a in line.split()]) + "\n")
```

```
    print("Best ONMI: {} params: '{}'".format(nmi_best, bestParam) )
    return all_results
```

In [4]:

```
def generate_params(params):
    keys = list(params.keys())
    if len(keys) == 1:
        for value in params[keys[0]]:
            yield ( keys[0] + " " + str(value) )
    if len( keys ) > 1:
        for value in params[keys[0]]:
            for remain_params in generate_params({k:params[k] for k in keys[1:
]}):
                yield ( keys[0] + " " + str(value) + " " + remain_params )
```

In [111]:

```
def plot_all_params(algorithm, dataset, all_results ):
    xdata=[]
    ydata=[]
    df = pd.DataFrame()
    for param, nmi in all_results.items():
        splited = param.split()
        xdata.append(float(splited[1]))
        ydata.append(nmi)
        df = df.append({'x': float(splited[1]), 'y': nmi}, ignore_index=True)

    plt.plot(xdata, ydata, 'C3', zorder=1, lw=3)
    # ax = plt.axes(projection='3d')
    plt.scatter(xdata, ydata,s=70,zorder=2)
    plt.xlabel('threshold')
    plt.ylabel('nmi value');
    plt.title('Algorithm: {}\nDataset: {}'.format(algorithm, dataset));
    plt.show()
```

School friendship network

In [120]:

```
params={}
params["-t"] = np.arange(0.05, 1.0, 0.05)
```

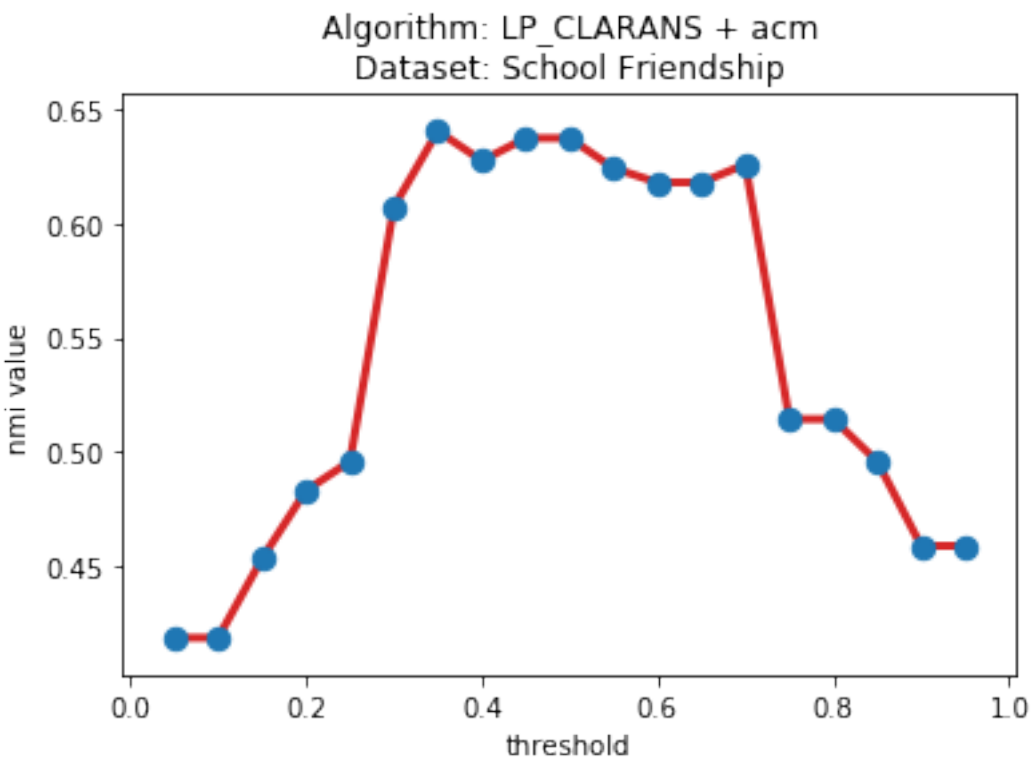
In [134]:

```
all_results = lp_experiment(clustersNumber=7,
              algorithm = "kmd",
              distance = "acm",
              inputFile =  "../datasets/school_friendship/school-2.gml",
              groundTruth = "../datasets/school_friendship/truth-school.dat",
              params = params,
              vertexNumerationShift=-1,
              benchmarkFormat=False)
```

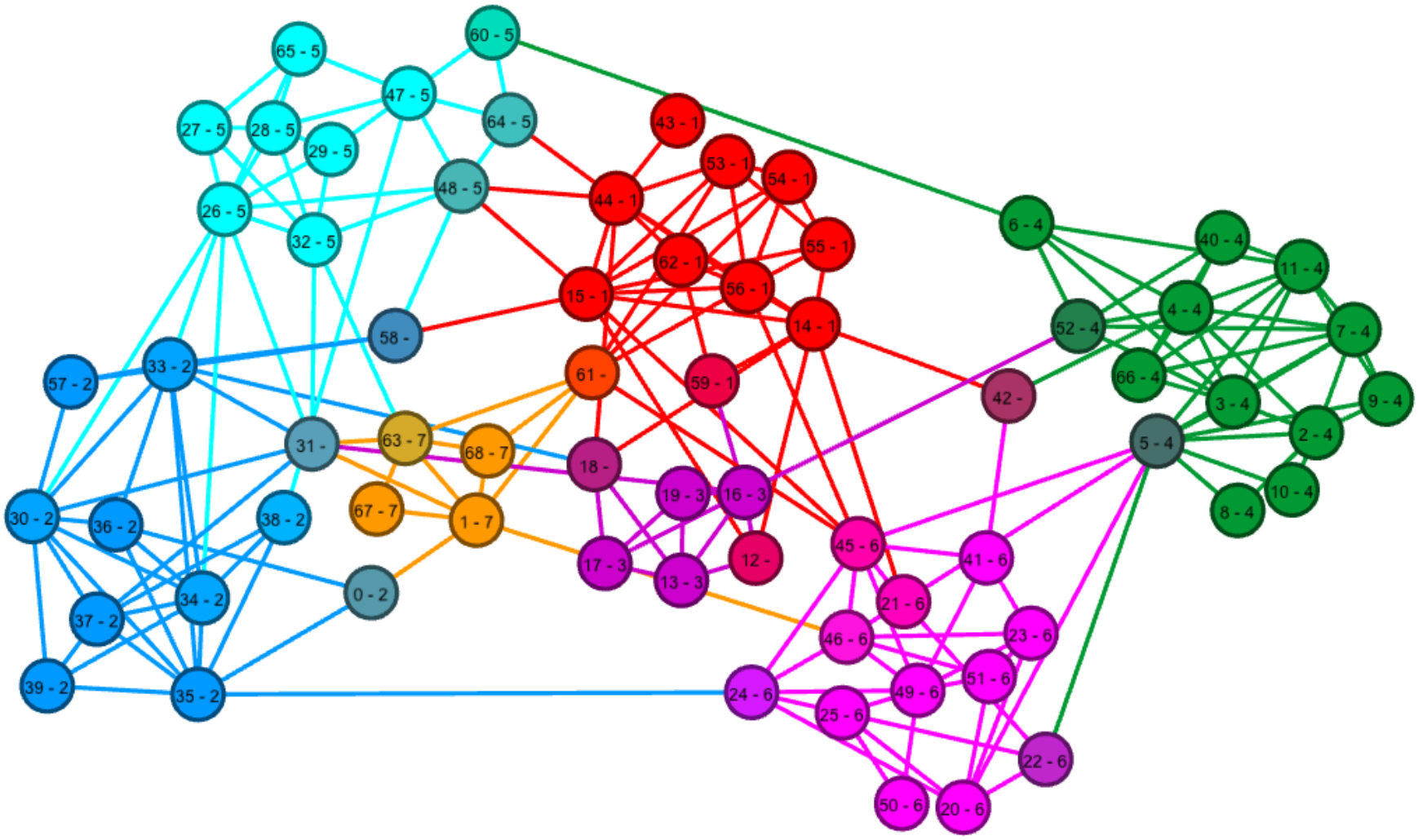
```
Output dir name: lp_kmd_acm_school_friendship
Output file name: lp_kmd_acm_school_friendship/pmp_school-2_ACM_km
d_7.dat


Best ONMI: 0.640249 params: '-t 0.35000000000000003'
```

In [129]:

```
plot_all_params(algorithm = "LP_CLARANS + acm", dataset = "School Friendship",
all_results = all_results)
```



Clustering results for School Frienship Netwrok with the best parameters



Karate Club

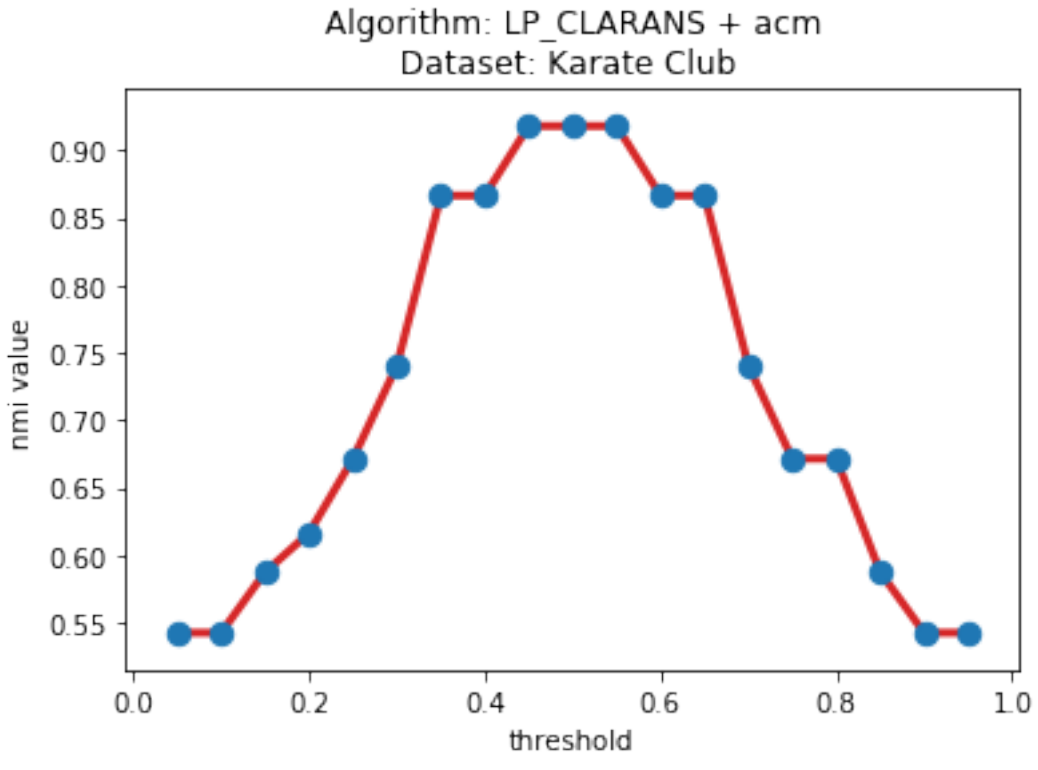
In [135]:

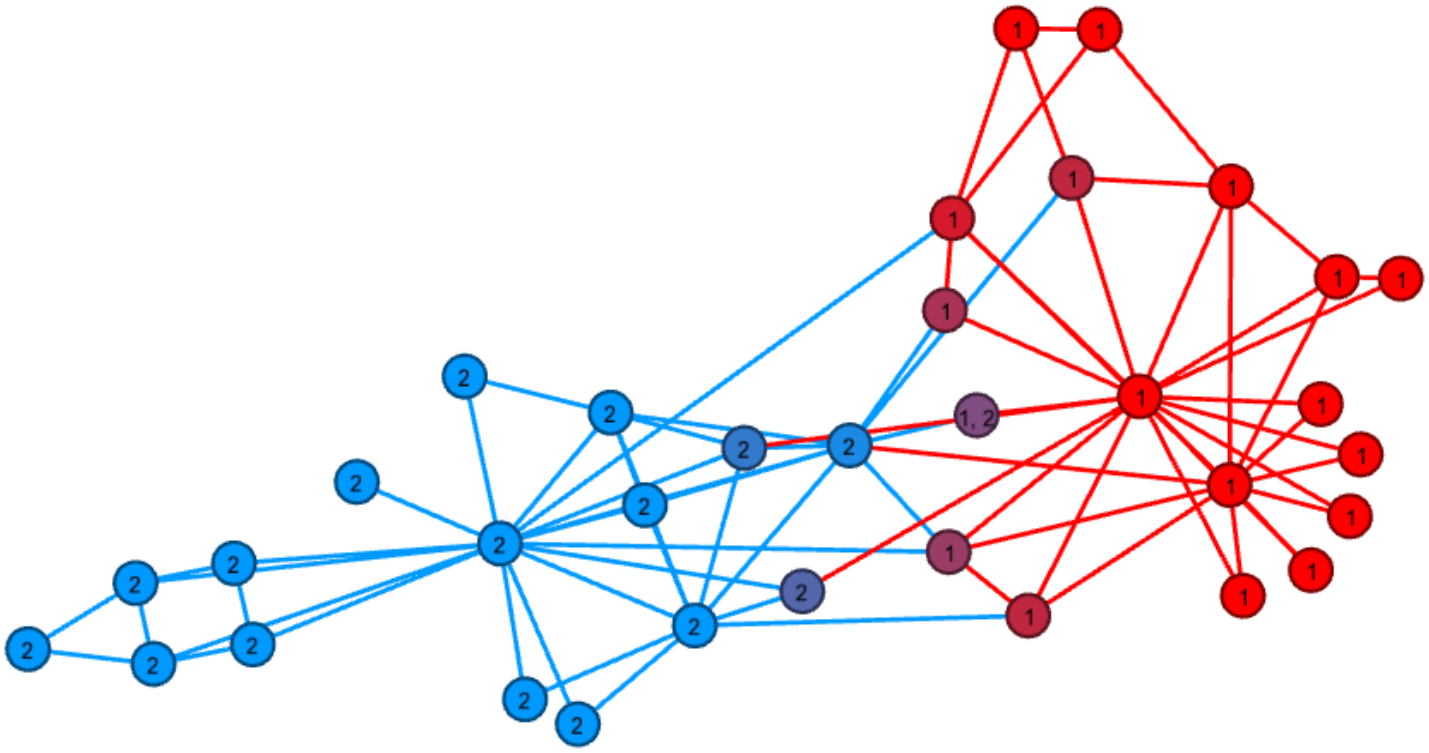
```
params={}
params["-t"] = np.arange(0.05, 1.0, 0.05)
all_results = lp_experiment(clustersNumber=2,
              algorithm = "kmd",
              distance = "acm",
              inputFile =  "../datasets/karate/karate.gml",
              groundTruth = "../datasets/karate/truth_karate.dat",
              params = params,
              vertexNumerationShift=0,
              benchmarkFormat=False)
```

```
Output dir name: lp_kmd_acm_karate
Output file name: lp_kmd_acm_karate/pmp_karate_ACM_kmd_2.dat


Best ONMI: 0.91796 params: '-t 0.45'
```

In [132]:

```
plot_all_params(algorithm = "LP_CLARANS + acm", dataset = "Karate Club ", all_
results = all_results)
```





American Football League c = 12

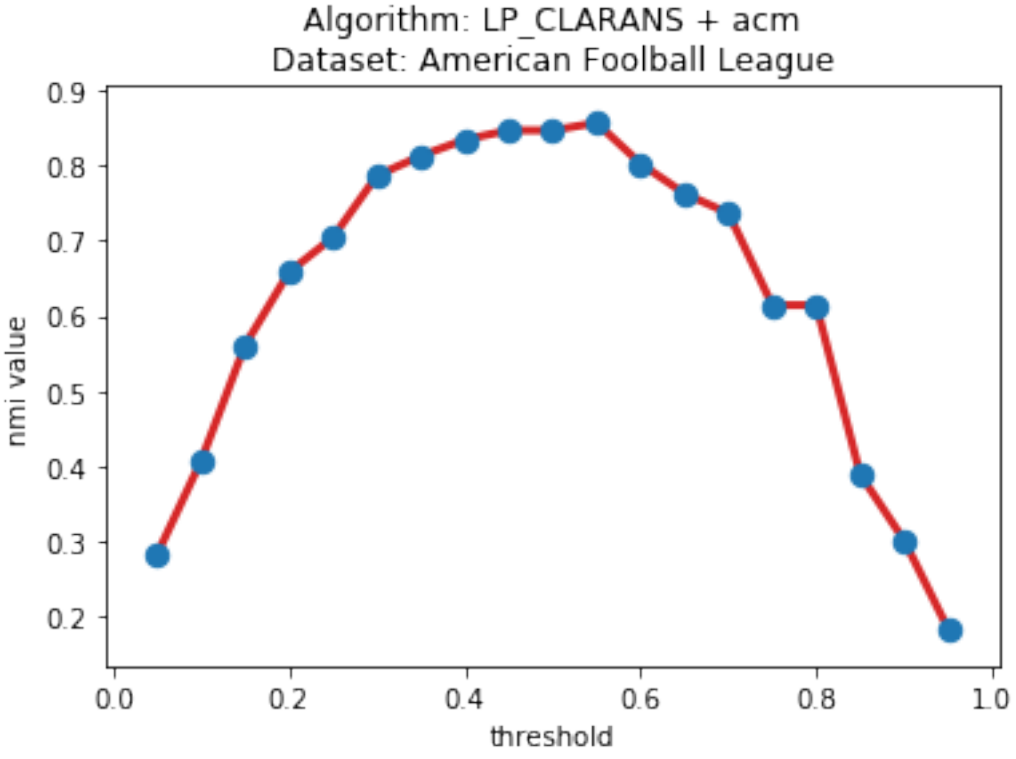
In [145]:

```
params={}
params["-t"] = np.arange(0.05, 1.0, 0.05)
all_results = lp_experiment(clustersNumber=12,
              algorithm = "kmd",
              distance = "acm",
              inputFile =  "../datasets/football/footballTSEinput.gml",
              groundTruth = "../datasets/football/truth_footballTSEinput.dat",
              params = params,
              vertexNumerationShift=-1,
              benchmarkFormat=False)
```

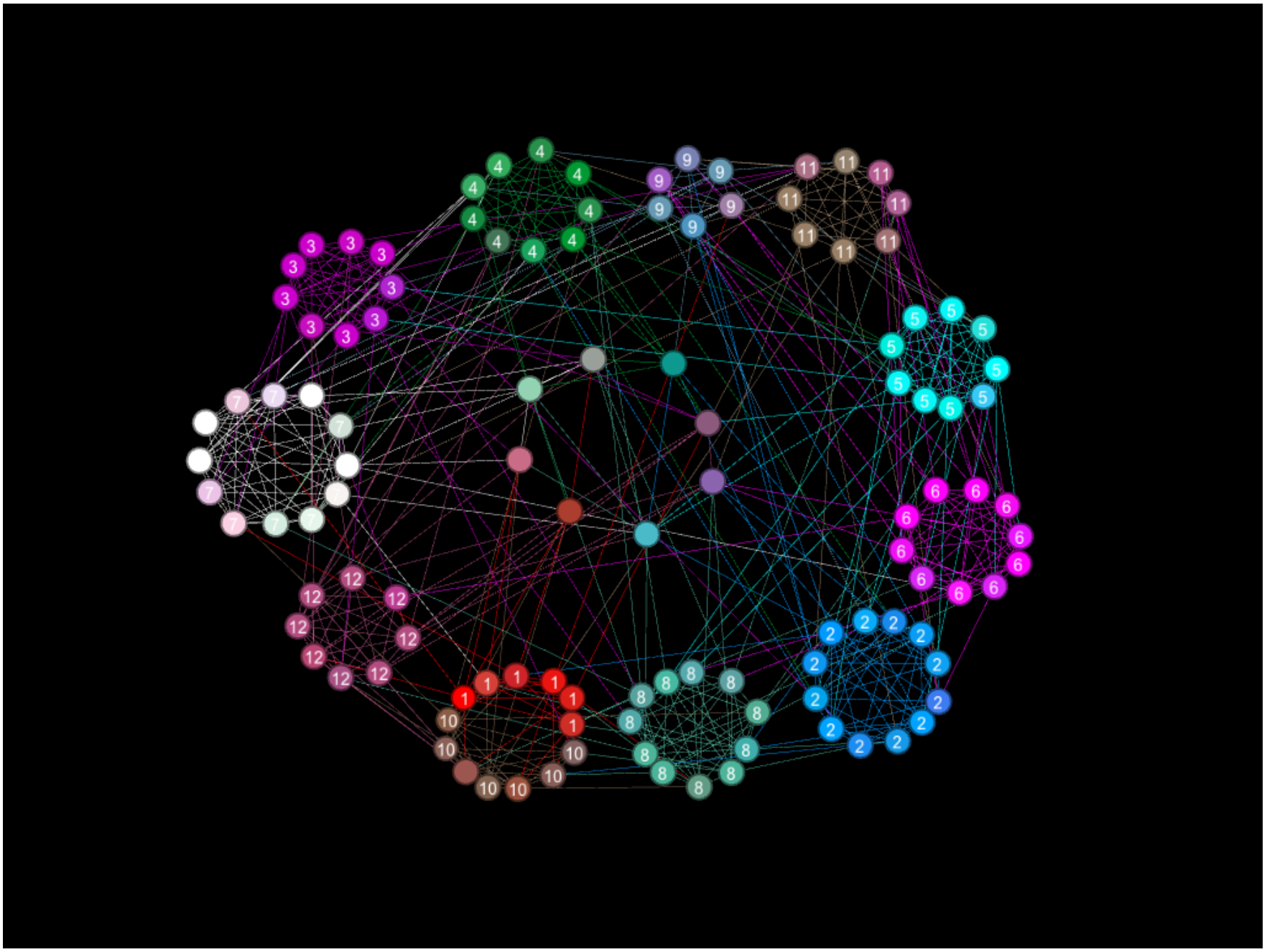
```
Output dir name: lp_kmd_acm_football
Output file name: lp_kmd_acm_football/pmp_footballTSEinput_ACM_kmd
_12.dat


Best ONMI: 0.856281 params: '-t 0.55'
```

In [146]:

```
plot_all_params(algorithm = "LP_CLARANS + acm", dataset = "American Foolball L
eague", all_results = all_results)
```



Americal Football League [onmi: 0.856281; thresould: 0.55]



Adj-noun

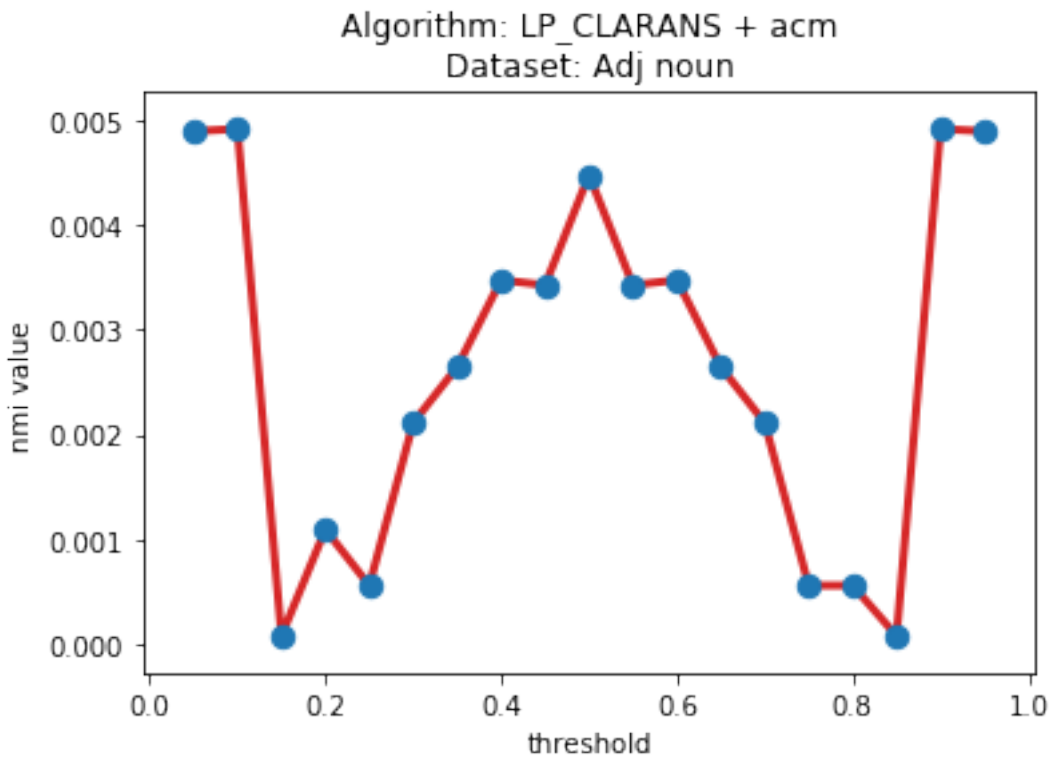
In [142]:

```
params={}
params["-t"] = np.arange(0.05, 1.0, 0.05)
all_results = lp_experiment(clustersNumber=2,
              algorithm = "kmd",
              distance = "acm",
              inputFile =  "../datasets/adjnoun/adjnoun.dat",
              groundTruth = "../datasets/adjnoun/truth_adjnoun.dat",
              params = params,
              vertexNumerationShift=0,
              benchmarkFormat=True)
```

```
Output dir name: lp_kmd_acm_adjnoun
Output file name: lp_kmd_acm_adjnoun/pmp_adjnoun_ACM_kmd_2.dat


Best ONMI: 0.00490969 params: '-t 0.1'
```

In [143]:

```
plot_all_params(algorithm = "LP_CLARANS + acm", dataset = "Adj noun", all_resu
lts = all_results)
```



Adj-noun [onmi: 0.00490969; thresould: 0.1]