Network Analytics with SNAP(.py)

Liu Jie

Overview of Network Analytics

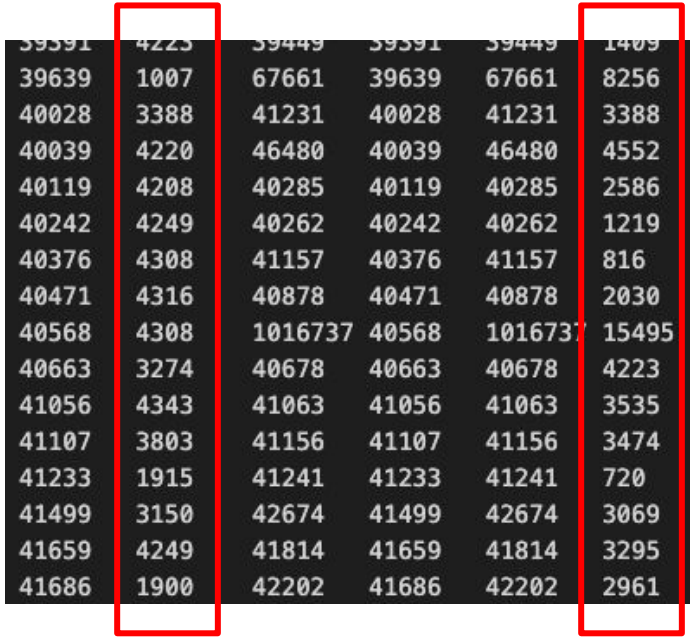
- How to get a network
  - From a real-world dataset (an existing network)
  - Generate a synthetic network
- Calculate network properties
  - Quick summary of network properties
  - Global connectivity: connected components
  - Local connectivity: node degrees
  - Key nodes in the network: node centrality
  - Neighborhood connectivity: triads, clustering coefficient
  - Graph traversal: breadth and depth first search
  - Groups of nodes: community detection
  - Global graph properties: spectral graph analysis
  - Core nodes: K-core decomposition

From an existing networks

```
G = snap.LoadEdgeList(snap.PNGraph, fname, col1, col2)
```

Loads a graph from a text file

- snap.PNGraph -> directed graph (type of the graph)
- col1 -> source vertex
- col2 -> destination vertex

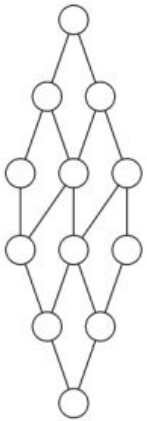


39639	1007	67661	39639	67661	8256
40028	3388	41231	40028	41231	3388
40039	4220	46480	40039	46480	4552
40119	4208	40285	40119	40285	2586
40242	4249	40262	40242	40262	1219
40376	4308	41157	40376	41157	816
40471	4316	40878	40471	40878	2030
40568	4308	1016737	40568	1016737	15495
40663	3274	40678	40663	40678	4223
41056	4343	41063	41056	41063	3535
41107	3803	41156	41107	41156	3474
41233	1915	41241	41233	41241	720
41499	3150	42674	41499	42674	3069
41659	4249	41814	41659	41814	3295
41686	1900	42202	41686	42202	2961

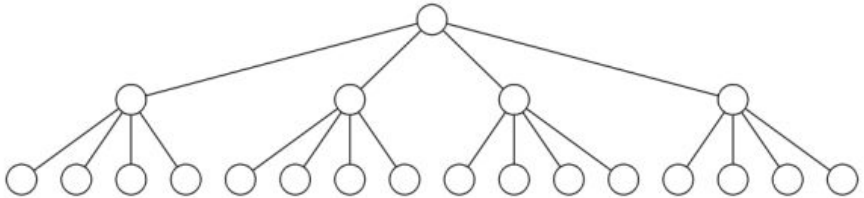
Generate graphs - Basic Graph Generators

```
GG = snap.GenGrid(snap.PUNGraph, 4, 3)
GT = snap.GenTree(snap.PUNGraph, 4, 2)
```

Complete, circle, grid, star, tree graphs



G-4-3



T-4-2

Generate graphs - Advanced Graph Generators

- Erdos-Renyi, Preferential attachment
- Forest Fire, Small-world, Configuration model
- Kronecker, RMat, Graph rewiring

```
GPA = snap.GenPrefAttach(30, 3, snap.TRnd())
GSW = snap.GenSmallWorld(10, 3, 0, snap.TRnd())
```

Quick summary of network properties

```
snap.PrintInfo(GG, "stats", "info.txt", False)
```

Prints basic Graph statistics to standard output or to a file

```
stats:
  Nodes:                     12
  Edges:                     17
  Zero Deg Nodes:            0
  Zero InDeg Nodes:          0
  Zero OutDeg Nodes:         0
  NonZero In-Out Deg Nodes:  12
  Unique directed edges:     34
  Unique undirected edges:   17
  Self Edges:                0
  BiDir Edges:               34
  Closed triangles:          0
  Open triangles:            34
  Frac. of closed triads:    0.000000
  Connected component size:  1.000000
  Strong conn. comp. size:   1.000000
  Approx. full diameter:     5
  90% effective diameter:    3.350000
```

Global connectivity: connected components

- Analyze graph connectedness
  - Strongly and Weakly connected components

```
MxWcc = snap.GetMxWcc(GG)    Get Largest Weakly Connected Component
print ("max wcc nodes %d, edges %d" % (MxWcc.GetNodes(), MxWcc.GetEdges()))
WccV = snap.TIntPrV()
snap.GetWccSzCnt(GG, WccV)
print ("# of connected component sizes : %d" % WccV.Len())
for comp in WccV:
    print ("size %d, number of components %d" %
        (comp.GetVal1(), comp.GetVal2()))
```

Output:

```
max wcc nodes 12, edges 17
# of connected component sizes : 1
size 12, number of components 1
```

Local connectivity: node degrees

- Analyze node connectivity
  - Find node degrees, maximum degree, degree distribution

```
NId = snap.GetMxDegNId(GG)
print ("max degree node: %d" % NId)
DegToCntV = snap.TIntPrV()
snap.GetDegCnt(GG, DegToCntV)
for item in DegToCntV:
    print( "%d nodes with degree %d" % (
        item.GetVal2(), item.GetVal1()))
```

Get node with max degree

Output:

```
max degree node: 7
4 nodes with degree 2
6 nodes with degree 3
2 nodes with degree 4
```

Key nodes in the network: node centrality

- Find “importance” of nodes in a graph
  - PageRank, Hubs and Authorities (HITS)
  - Degree-, betweenness-, closeness-, farness-, and eigen- centrality

```
PRankH = snap.TIntFltH()
snap.GetPageRank(G, PRankH)
for item in PRankH:
    print item, PRankH[item]
```

Output:

```
0 0.0625431402986
1 0.0892274169161
2 0.0625431402986
3 0.0874088240667
4 0.110868654353
5 0.0874088240667
6 0.0874088240667
7 0.110868654353
8 0.0874088240667
9 0.0625431402986
10 0.0892274169161
11 0.0625431402986
```

Neighborhood connectivity: triads, clustering coefficient

- Analyze connectivity among the neighbors
  - # of triads, fraction of closed triads
  - Fraction of connected neighbor pairs
  - Graph-based, node-based

```
Triads = snap.GetTriads(G)
print "triads", Triads
CC = snap.GetClustCf(G)
print "clustering coefficient", CC
```

Output:

```
triads 30
clustering coefficient 0.6
```

Graph traversal: breadth and depth first search

- Distances between nodes
  - Diameter, Effective diameter
  - Shortest path, Neighbors at distance d
  - Approximate neighborhood (not BFS based)

```
D = snap.GetBfsFullDiam(G, 100)
print "diameter", D
ED = snap.GetBfsEffDiam(G, 100)
print "effective diameter", ED
```

Output:

```
diameter 2
effective diameter 1.66666666667
```

Groups of nodes: community detection

- Identify communities of nodes
  - Clauset-Newman-Moore, Girvan-Newman
    - Can be compute time intensive
  - BigClam, CODA, Cesna (C++ only)

```
CmtyV = snap.TCnComV()
modularity = snap.CommunityCNM(G, CmtyV)
for Cmty in CmtyV:
    print "Community: "
    for NI in Cmty:
        print NI
```

Output:

```
Community:
0
1
7
8
9
Community:
2
Community:
3
Community:
4
Community:
5
Community:
6
```

Global graph properties: spectral graph analysis

- Calculations based on graph adjacency matrix
  - Get Eigenvalues, Eigenvectors
  - Get Singular values, leading singular vectors

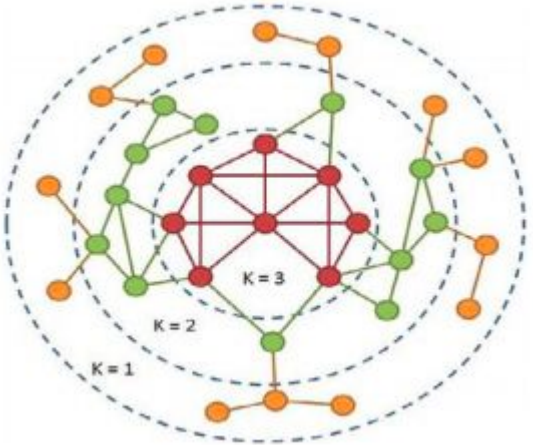
```
UGraph = snap.GenRndGnm(snap.PUNGraph, 20, 100)
EigVec =  snap.TFltV()
snap.GetEigVec(UGraph, EigVec)
for Val in EigVec:
    print(Val)
```

Get leading eigenvector

Output:

```
0.255508511266
0.197746945988
0.117178565664
0.222298288681
0.240047814483
0.255693477039
0.153849412996
0.178414586306
0.225737138978
0.242465857296
0.195446516717
0.183400146472
0.207045751862
0.222954206653
0.226630050343
0.306686516349
0.273669033535
0.253829290004
0.244181855752
0.188243556306
```

Core nodes: K-core decomposition



- Repeatedly remove nodes with low degrees
  - Calculate K-core

```
Core3 = snap.GetKCore(G, 3)
```

Calculate 3-core

References

- Snap.py Reference Manual
  - https://snap.stanford.edu/snappy/doc/reference/index-ref.html

- WWW-15 Tutorial Large Scale Network Analytics with SNAP
  - http://snap.stanford.edu/proj/snap-www/SNAP-WWW15-part3.pdf