



Tutorial: Large Scale Network Analytics with SNAP

<http://snap.stanford.edu/proj/snap-icwsm>

Rok Sosič, Jure Leskovec
Stanford University

ICWSM-14, Ann Arbor, MI

June, 2014





Network Analytics with SNAP

Rok Sosič, Jure Leskovec
Stanford University

Print Graph Information

```
G = snap.LoadEdgeList(snap.PNGraph, "qa.txt", 1, 5)
Snap.PrintInfo(G, "QA Stats", "qa-info.txt", False)
```

Output:

QA Stats: Directed

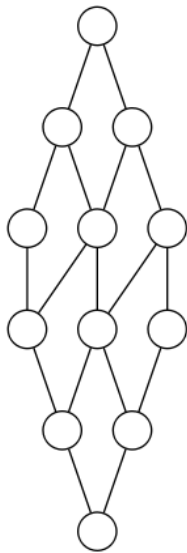
| | |
|---------------------------|----------|
| Nodes: | 146874 |
| Edges: | 333606 |
| Zero Deg Nodes: | 0 |
| Zero InDeg Nodes: | 83443 |
| Zero OutDeg Nodes: | 30963 |
| NonZero In-Out Deg Nodes: | 32468 |
| Unique directed edges: | 333606 |
| Unique undirected edges: | 333481 |
| Self Edges: | 20600 |
| BiDir Edges: | 20850 |
| Closed triangles: | 41389 |
| Open triangles: | 51597174 |
| Frac. of closed triads: | 0.000802 |
| Connected component size: | 0.893201 |
| Strong conn. comp. size: | 0.029433 |
| Approx. full diameter: | 14 |
| 90% effective diameter: | 5.588639 |

Basic Graph Generators

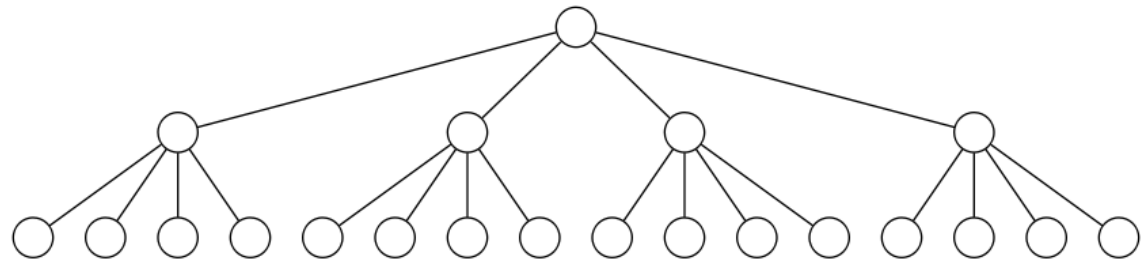
- Complete, circle, grid, star, tree graphs

GG = **snap.GenGrid(snap.PUNGraph, 4, 3)**

GT = **snap.GenTree(snap.PUNGraph, 4, 2)**



G-4-3

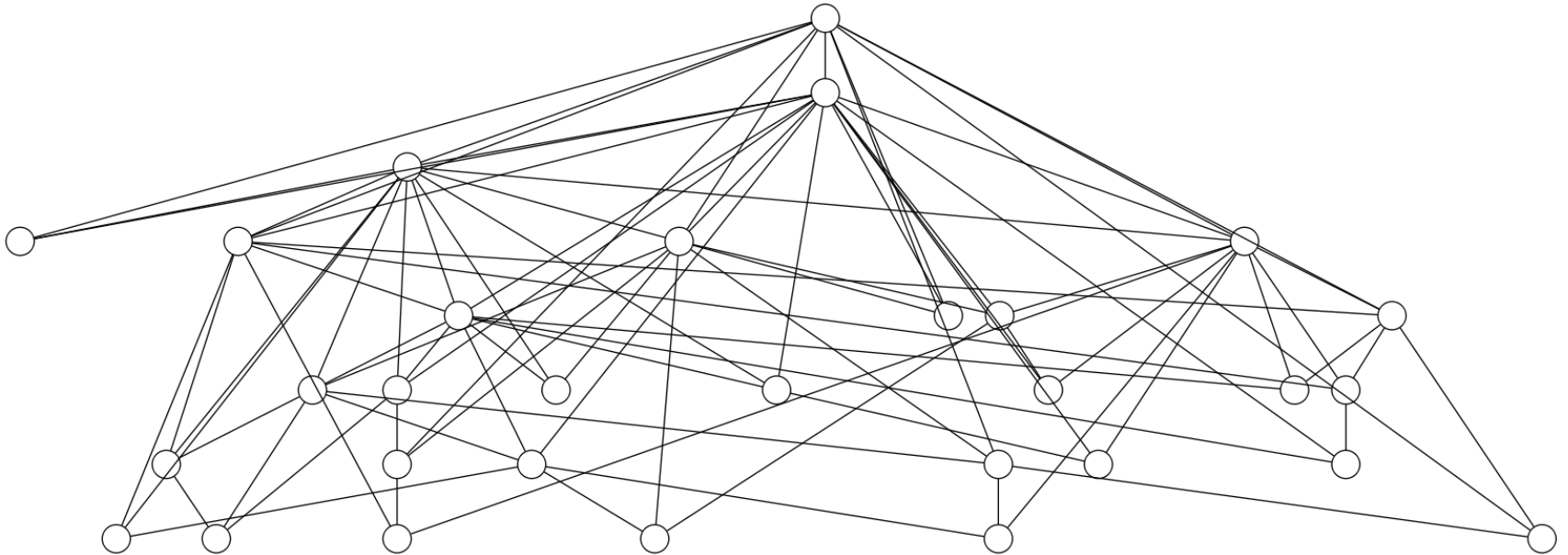


T-4-2

Advanced Graph Generators

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

GPA = snap.GenPrefAttach(30, 3, snap.TRnd())



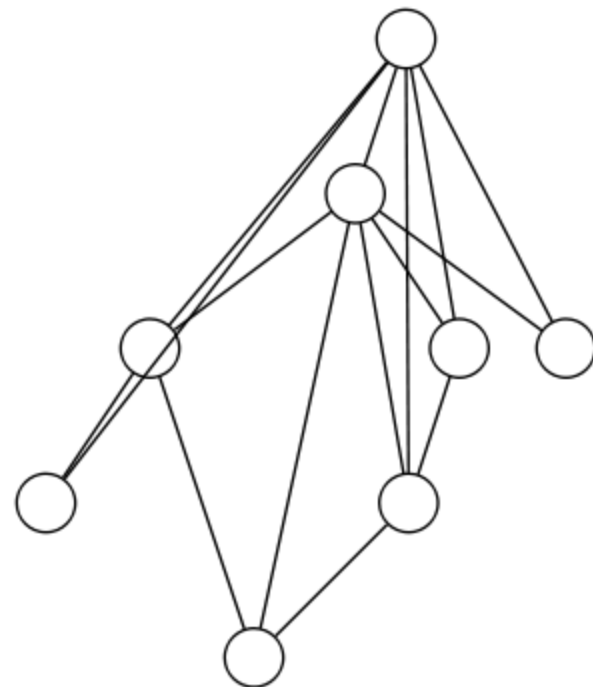
PA-30

Subgraphs and Conversions

- Extract subgraphs, convert from one graph type to another

Get an induced subgraph on a set of nodes **NIdV**:

```
NIdV = snap.TIntV()  
for i in range(1,9): NIdV.Add(i)  
  
SubGPA = snap.GetSubGraph(GPA, NIdV)
```



SPA-8

Connected Components

■ Analyze graph connectedness

- Strongly and Weakly connected components
 - Test connectivity, get sizes, get components, get largest
 - Articulation points, bridges
- Bi-connected, 1-connected

```
MxWcc = snap.GetMxWcc(G) Get largest WCC  
print "max wcc nodes %d, edges %d" %  
      (MxWcc.GetNodes(), MxWcc.GetEdges())
```

```
WccV = snap.TIntPrV()  
snap.GetWccSzCnt(G, WccV) Get WCC sizes
```

```
print "# of connected component sizes", WccV.Len()  
for comp in WccV:  
    print "size %d, number of components %d" %  
          (comp.GetVal1(), comp.GetVal2())
```

Node Degrees

■ Analyze node connectivity

- Find node degrees, maximum degree, degree distribution
- In-degree, out-degree, combined degree

```
NId = snap.GetMxDegNId(GPA)
print "max degree node", NId
```

Get node with max degree

```
DegToCntV = snap.TIntPrV()
snap.GetDegCnt(GPA, DegToCntV)
for item in DegToCntV:
    print "%d nodes with degree %d" % (
        item.GetVal2(), item.GetVal1())
```

Get degree distribution

```
max degree node 1
13 nodes with degree 3
4 nodes with degree 4
3 nodes with degree 5
2 nodes with degree 6
1 nodes with degree 7
1 nodes with degree 9
2 nodes with degree 10
2 nodes with degree 11
1 nodes with degree 13
1 nodes with degree 15
```


Node Centrality

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

```
PRankH = snap.TIntFltH()  
snap.GetPageRank(G, PRankH)
```

Calculate node
PageRank scores

```
for item in PRankH:  
    print item, PRankH[item]
```

Print them out

Triads and 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(GPA)  
print "triads", Triads
```

Count triads

```
CC = snap.GetClustCf(GPA)  
print "clustering coefficient", CC
```

Calculate clustering
coefficient

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
```

Calculate diameter

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

Calculate effective
diameter

Community Detection

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

```
CmtyV = snap.TCnComV()                                Clauset-Newman-Moore
modularity = snap.CommunityCNM(UGraph, CmtyV)

for Cmty in CmtyV:
    print "Community: "
    for NI in Cmty:
        print NI
print "The modularity of the network is %f" % modularity
```

Spectral properties of a graph

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

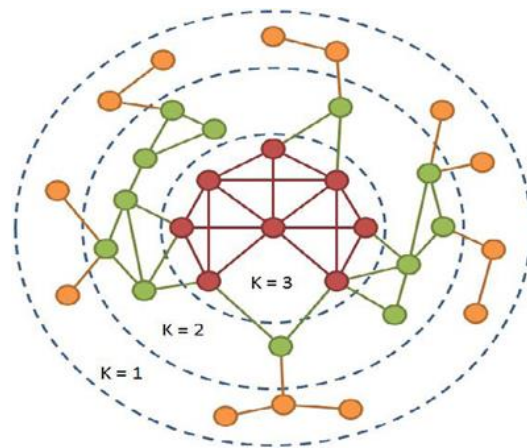
```
EigV = snap.TF1tV()  
snap.GetEigVec(G, EigV)
```

Get leading
eigenvector

```
nr = 0  
for f in EigV:  
    nr += 1  
    print "%d: %.6f" % (nr, f)
```

K-core decomposition

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



Core3 = snap.GetKCore(G, 3)

Calculate 3-core