Tutorial: Large Scale Network Analytics with SNAP

http://snap.stanford.edu/proj/snap-www

Rok Sosič, Jure Leskovec
Stanford University

WWW-15, Florence, Italy **May, 2015**

Network Analytics with SNAP

Rok Sosič, Jure Leskovec
Stanford University

WWW-15, Florence, Italy **May, 2015**

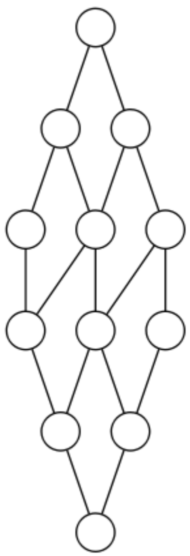
Overview of Network Analytics

- **How to get a network**
  - From a **real-world dataset**
  - Generate a **synthetic network**
  - From an **existing 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

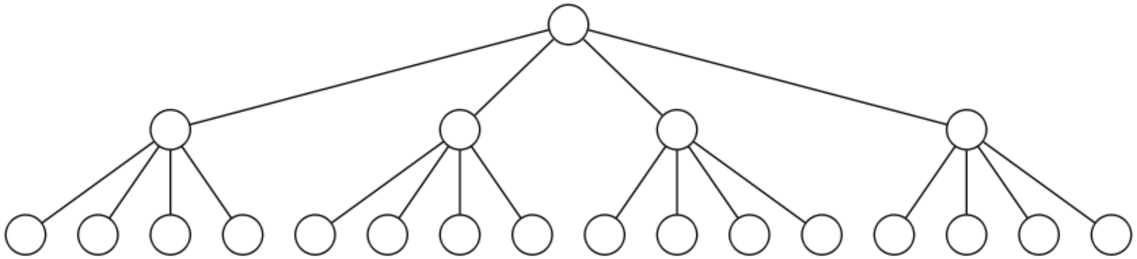
Basic Graph Generators

- **Complete, circle, grid, star, tree graphs**

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



G-4-3

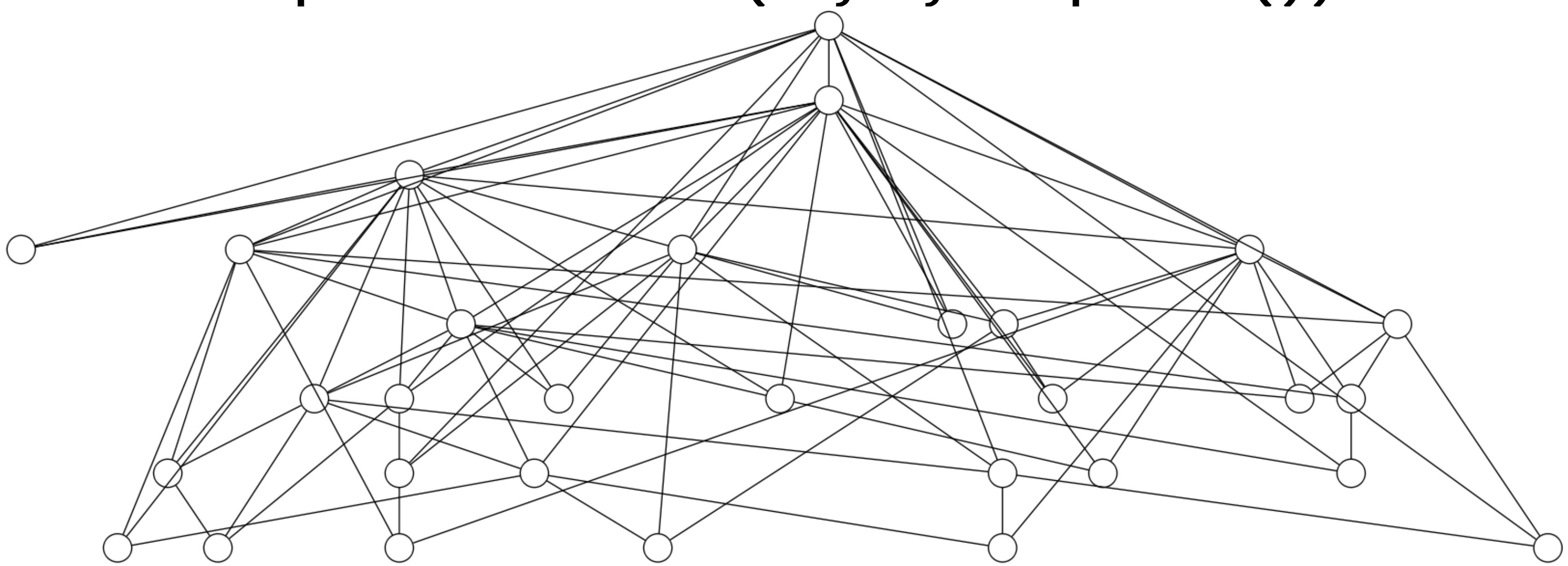


T-4-2

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

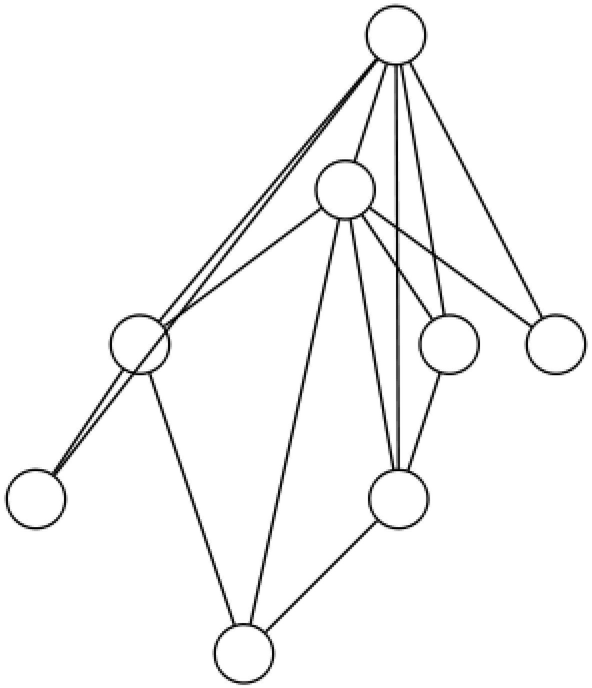
Subgraphs and Conversions

- 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

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:                       188406
  Edges:                       415174
  Zero Deg Nodes:              0
  Zero InDeg Nodes:            108618
  Zero OutDeg Nodes:           38319
  NonZero In-Out Deg Nodes:    41469
  Unique directed edges:       415174
  Unique undirected edges:     415027
  Self Edges:                  26924
  BiDir Edges:                 27218
  Closed triangles:            46992
  Open triangles:              69426319
  Frac. of closed triads:      0.000676
  Connected component size:    0.886745
  Strong conn. comp. size:     0.025758
  Approx. full diameter:       13
  90% effective diameter:      5.751723
```

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)
print "max wcc nodes %d, edges %d" %
      (MxWcc.GetNodes(), MxWcc.GetEdges())
```

Get largest WCC

```
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

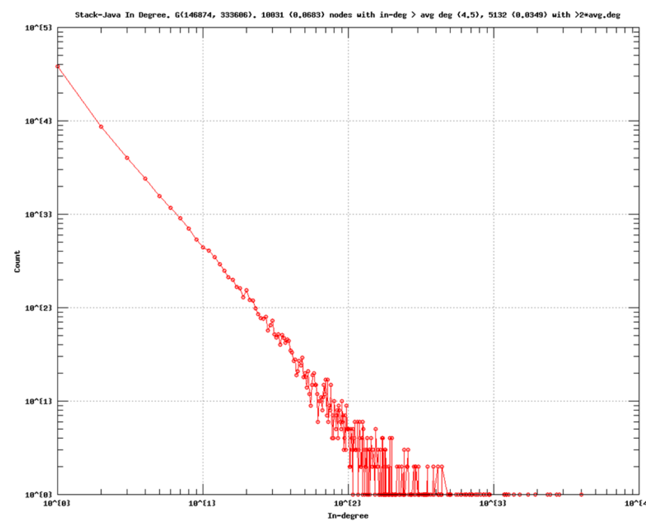
```
NId = snap.GetMxDegNId(GPA)
print "max degree node", NId

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

Get node with max degree

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 (HITS)
  - 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, Cesna (C++ only)

Clauset-Newman-Moore

```
CmtyV = snap.TCnComV()
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

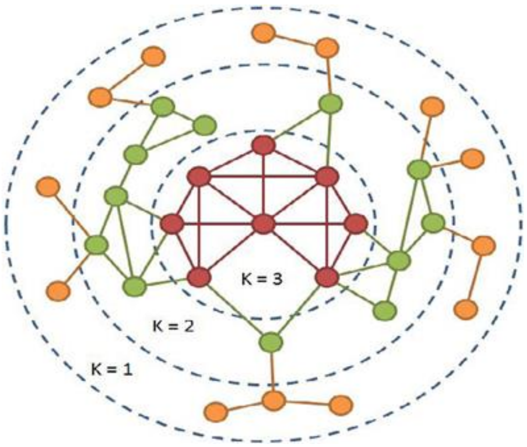
```
EigV = snap.TFltV()
snap.GetEigVec(G, EigV)

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

Get leading eigenvector

K-core Decomposition

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




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

Calculate 3-core