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
SNAP Tutorial: Content

- Motivation
- Introduction to SNAP
- Snap.py for Python
- Network analytics
- SNAP network datasets
- SNAP for C++
- Hands-on exercise

Contact information:
Rok Sosič, rok@cs.stanford.edu
Slides available at:
http://snap.stanford.edu/proj/snap-icwsm
Why Networks?

Networks are a general language for describing complex systems.
Friends & Family
Media & Information
World economy
Roads
Brain
What is SNAP?

- **Stanford Network Analysis Platform (SNAP)** is a general purpose, high-performance system for analysis and manipulation of large networks
  - [http://snap.stanford.edu](http://snap.stanford.edu)
  - Scales to massive networks with hundreds of millions of nodes and billions of edges

- **SNAP software**
  - Snap.py for Python, SNAP C++

- **SNAP datasets**
  - Over 70 network datasets
In the context of SNAP software

- **Graphs** consists of nodes and edges
  - An edge connects two points (or is a loop)
- **Networks** are graphs where nodes and edges can have attributes (features, values)

In presentation and documentation, terms “graph” and “network” are often used interchangeably to mean graph and/or network

- Specific meaning is usually evident from the context
Snap.py Resources

- **Prebuilt packages** available for Mac OS X, Windows, Linux

- **Snap.py documentation:**

- **SNAP user mailing list**
  [http://groups.google.com/group/snap-discuss](http://groups.google.com/group/snap-discuss)

- **Developer resources**
  - Software available as open source under BSD license
  - GitHub repository
    [https://github.com/snap-stanford/snap-python](https://github.com/snap-stanford/snap-python)
SNAP C++ Resources

- **Source code** available for Mac OS X, Windows, Linux
  

- **SNAP documentation**
  

  - Quick Introduction, User Reference Manual
  - Source code, see tutorials

- **SNAP user mailing list**
  
  [http://groups.google.com/group/snap-discuss](http://groups.google.com/group/snap-discuss)

- **Developer resources**
  
  - Software available as open source under BSD license
  - GitHub repository
    
    [https://github.com/snap-stanford/snap](https://github.com/snap-stanford/snap)
  
  - SNAP C++ Programming Guide
Collection of over 70 social network datasets: http://snap.stanford.edu/data

Mailing list: http://groups.google.com/group/snap-datasets

- **Social networks**: online social networks, edges represent interactions between people
- **Twitter and Memetracker**: Memetracker phrases, links and 467 million Tweets
- **Citation networks**: nodes represent papers, edges represent citations
- **Collaboration networks**: nodes represent scientists, edges represent collaborations (co-authoring a paper)
- **Amazon networks**: nodes represent products and edges link commonly co-purchased products
Snap.py: SNAP for Python

Rok Sosič, Jure Leskovec
Stanford University

ICWSM-14, Ann Arbor, MI  June, 2014
**What is Snap.py?**

- **Snap.py** (pronounced “snappy”): SNAP for Python

  [http://snap.stanford.edu/snappy](http://snap.stanford.edu/snappy)

<table>
<thead>
<tr>
<th>Solution</th>
<th>Fast Execution</th>
<th>Easy to use, interactive</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Python</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Snap.py (C++, Python)</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Installing Snap.py

- Download the Snap.py for your platform:
  http://snap.stanford.edu/snappy/
  - Packages for Mac OS X, Windows, Linux (CentOS)
    - OS must be 64-bit
    - Mac OS X, 10.7.5 or later
    - Windows, install Visual C++ Redistributable Runtime

- Download Python

- Installation:
  - Follow instructions on the Snap.py webpage
    python setup.py install

If you encounter problems, please report them to us or post to the mailing list
The most important step: 

**Import the snap module!**

```
$ python
>>> import snap
```
Snap.py Tutorial

- **On the Web:**

- **We will cover:**
  - Basic Snap.py data types
  - Vectors, hash tables and pairs
  - Graphs and networks
  - Graph creation
  - Adding and traversing nodes and edges
  - Saving and loading graphs
  - Plotting and visualization
Variable types/names:
- ...Int: an integer operation, variable: GetValInt()
- ...Flt: a floating point operation, variable; GetValFlt()
- ...Str: a string operation, variable; GetDateStr()

Classes vs. Graph Objects:
- T...: a class type; TUNGraph
- P...: type of a graph object; PUNGraph

Data Structures:
- ...V: a vector, variable TIntV  InNIdV
- ...VV: a vector of vectors (i.e., a matrix), variable FltVV
  - TFltVV ... a matrix of floating point elements
- ...H: a hash table, variable NodeH
  - TIntStrH ... a hash table with TInt keys, TStr values
- ...HH: a hash of hashes, variable NodeHH
  - TIntIntHH ... a hash table with TInt key 1 and TInt key 2
- ...Pr: a pair; type TIntPr
Snap.py Naming Conventions (2)

- **Get...**: an access method, `GetDeg()`
- **Set...**: a set method, `SetXYLabel()`
- **...I**: an iterator, `NodeI`
- **Id**: an identifier, `GetUID()`
- **NId**: a node identifier, `GetNId()`
- **EId**: an edge identifier, `GetEId()`
- **Nbr**: a neighbor, `GetNbrNId()`
- **Deg**: a node degree, `GetOutDeg()`
- **Src**: a source node, `GetSrcNId()`
- **Dst**: a destination node, `GetDstNId()`
Basic Types in Snap.py (and SNAP)

- **TInt**: Integer
- **TFlt**: Float
- **TStr**: String

- Used primarily for constructing composite types
- In general no need to deal with the basic types explicitly
  - Data types are automatically converted between C++ and Python
  - An illustration of explicit manipulation:
    >>> i = snap.TInt(10)
    >>> print i.Val
    10

- **Note**: do not use an empty string "" in TStr parameters
For more information check out Snap.py Reference Manual

Class TInt

- **TInt(val)**
  - Returns a new TInt initialized with the value specified by optional parameter val. If no value is given, the TInt object is initialized with the default value 0. If SInt is provided, the value is loaded from the binary stream SIn. In Snap.py, TInt is automatically converted to Python type int.

Below is a list of functions supported by the TInt class:

- **Load(SIn)**
  - Loads the int from a binary stream SIn.

- **Save(SOut)**
  - Saves the int to a binary stream SOut.

- **Abs(val)**
  - A static method that returns the absolute value of val, an int.

- **GetHexStr(val)**
  - A static method that returns a string with the hexadecimal representation of int val.

- **GetInRange(val, min, max)**
  - A static method that returns int val if it is between min and max. If val is smaller than min, it returns min. If val is larger than max, it returns max.

- **GetKiloStr(val)**
  - A static method that returns the int val as a kilo-formatted string. If val is less than 1000, it returns val as a string. If val is greater than or equal to 1000, it returns a string in form of 'x.yK', where x is some digit from 1-9 and y from 0-9.

- **GetMegaStr(val)**
  - A static method that returns the int val as a mega-formatted string. If val is less than 10000000, it returns the equivalent of GetKiloStr(val). If val is greater than or equal to 10000000, it returns a string in the form of 'x.yM', where x is some digit from 1-9 and y from 0-9.
SNAP User Reference Manual
http://snap.stanford.edu/snap/doc.html

SNAP Library 2.1, User Reference

Public Member Functions

- Tint()
- Tint(const int &Val)
- operator int() const
- Tint(TSIN &SN)
- void Load(TSIN &SN)
- void Save(TSOOut &SOOut) const
- void LoadXml(const PXmItok &XmItok, const TStr &Nm)
- void SaveXml(TSOOut &SOOut, const TStr &Nm) const
- Tint & operator=(const Tint &int)
- Tint & operator=(const int &int)
- bool operator==(const Tint &int) const
- bool operator==(const int &int) const
Vector Types

- **Sequences of values of the same type**
  - New values can be added to the end
  - Existing values can be accessed or changed

- **Naming convention:** `<type_name>V`
  - Examples: TIntV, TFltV, TStrV

- **Common operations:**
  - `Add(<value>):` add a value
  - `Len():` vector size
  - ` [<index>]:` get or set a value of an existing element
  - `for i in V:` iteration over the elements
v = snap.TIntV()

v.Add(1)
v.Add(2)
v.Add(3)
v.Add(4)
v.Add(5)

print v.Len()

print v[3]
print v[3]

for item in v:
    print item

for i in range(0, v.Len()):
    print i, v[i]
Hash Table Types

- **A set of (key, value) pairs**
  - Keys must be of the same types, values must be of the same type (could be different from the key type)
  - New (key, value) pairs can be added
  - Existing values can be accessed or changed via a key

- **Naming:** `<type1><type2>H`
  - **Examples:** TIntStrH, TIntFltH, TStrIntH

- **Common operations:**
  - `[<key>]`: add a new or get or set an existing value
  - `Len()`: hash table size
  - `for k in H:` iteration over keys
  - `BegI(), IsEnd(), Next()`: element iterators
  - `GetKey(<i>):` get i-th key
  - `GetDat(<key>):` get value associated with a key
Hash Table Example

```python
h = snap.TIntStrH()

h[5] = "five"
h[3] = "tree"
h[9] = "nine"
h[6] = "six"
h[1] = "one"

print h.Len()


h[3] = "three"

for key in h:
    print key, h[key]
```

Create an empty table
Add elements
Print table size
Get element value
Set element value
Print table elements
**Hash Tables: KeyId**

- \texttt{THash\<key type, value type\>}
  - **Key**: item key, provided by the caller
  - **Value**: item value, provided by the caller
  - **KeyId**: integer, unique slot in the table, calculated by SNAP

<table>
<thead>
<tr>
<th>KeyId</th>
<th>0</th>
<th>2</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>100</td>
<td>89</td>
<td>95</td>
</tr>
<tr>
<td>Value</td>
<td>“David”</td>
<td>“Ann”</td>
<td>“Jason”</td>
</tr>
</tbody>
</table>
Pair Types

- A pair of (value1, value2)
  - Two values, type of value1 could be different from the value2 type
  - Existing values can be accessed

- **Naming**: <type1,type2>Pr
  - **Examples**: TIntStrPr, TIntFltPr, TStrIntPr

- **Common operations**:
  - **GetVal1**: get value1
  - **GetVal2**: get value2
Pair Example

```python
>>> p = snap.TIntStrPr(1,"one")
>>> print p.GetVal1()
1
>>> print p.GetVal2()
one
```

- **TIntPrV**: a vector of (integer, integer) pairs
- **TIntPrFltH**: a hash table with (integer, integer) pair keys and float values
Graphs vs. Networks Classes:

- **TUNGraph**: undirected graph
- **TNGraph**: directed graph
- **TNEANet**: multigraph with attributes on nodes and edges

Object types start with **P...**, since they use wrapper classes for garbage collection

- **PUNGraph**, **PNGraph**, **PNEANet**

Guideline

- For class methods (functions) use **T**
- For object instances (variables) use **P**
Graph Creation

G1 = snap.TNGraph.New()

G1.AddNode(1)
G1.AddNode(5)
G1.AddNode(12)

G1.AddEdge(1,5)
G1.AddEdge(5,1)
G1.AddEdge(5,12)

G2 = snap.TUNGraph.New()
N1 = snap.TNEANet.New()
### Graph Traversal

#### Node traversal

```python
for NI in G1.Nodes():
    print "node id %d, out-degree %d, in-degree %d"
    % (NI.GetId(), NI.GetOutDeg(), NI.GetInDeg())
```

#### Edge traversal

```python
for EI in G1.Edges():
    print "(%d, %d)" % (EI.GetSrcNId(), EI.GetDstNId())
```

#### Edge traversal by nodes

```python
for NI in G1.Nodes():
    for DstNId in NI.GetOutEdges():
        print "edge (%d %d)" % (NI.GetId(), DstNId)
```
Graph Saving and Loading

FOut = snap.TFOut("test.graph")
G2.Save(FOut)
FOut.Flush()

FIn = snap.TFIn("test.graph")
G4 = snap.TNGraph.Load(FIn)

snap.SaveEdgeList(G4, "test.txt", "List of edges")

G5 = snap.LoadEdgeList(snap.PNGraph,"test.txt",0,1)
Example file: wiki-Vote.txt
  - Download from http://snap.stanford.edu/data

# Directed graph: wiki-Vote.txt
# Nodes: 7115 Edges: 103689
# FromNodeId   ToNodeId
0   1
0   2
0   3
0   4
0   5
2   6
...

G5 = snap.LoadEdgeList(snap.PNGraph,"test.txt",0,1)
Plotting in Snap.py

- Plotting graph properties
  - Gnuplot: http://www.gnuplot.info

- Visualizing graphs
  - Graphviz: http://www.graphviz.org

- Other options
  - Matplotlib: http://www.matplotlib.org
Install Gnuplot:
http://www.gnuplot.info/

Make sure that the directory containing wgnuplot.exe (for Windows) or gnuplot (for Linux, Mac OS X) is in your environmental variable $PATH
import snap
G = snap.LoadEdgeList(snap.PNGraph, "qa.txt", 1, 5)
snap.PlotInDegDistr(G, "Stack-Java", "Stack-Java In Degree")

Graph of Java QA on StackOverflow: in-degree distribution
Snap.py + Gnuplot

- **Snap.py** generates three files:

  - `.png` or `.eps` is the plot
  - `.tab` file contains the data (tab separated file)
  - `.plt` file contains the plotting commands
Drawing Graphs

- **InstallGraphViz:**
  http://www.graphviz.org/

- Make sure that the directory containing GraphViz is in your environmental variable $PATH
G1 = snap.TNNGraph.New()  # Create graph
G1.AddNode(1)
G1.AddNode(5)
G1.AddNode(12)
G1.AddEdge(1,5)
G1.AddEdge(5,1)
G1.AddEdge(5,12)

NIdName = snap.TIntStrH()  # Set node labels
NIdName[1] = "1"
NIdName[5] = "5"
NIdName[12] = "12"

snap.DrawGViz(G1, snap.gvlDot, "G1.png", "G1", NIdName)  # Draw
Network Analytics with SNAP

Rok Sosič, Jure Leskovec
Stanford University

ICWSM-14, Ann Arbor, MI                June, 2014
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

\[
\begin{align*}
GG &= \text{snap.GenGrid}(\text{snap.PUNGraph}, 4, 3) \\
GT &= \text{snap.GenTree}(\text{snap.PUNGraph}, 4, 2)
\end{align*}
\]
Advanced Graph Generators

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

\[ \text{GPA} = \text{snap.GenPrefAttach}(30, 3, \text{snap.TRnd}()) \]
Extract subgraphs, convert from one graph type to another

Get an induced subgraph on a set of nodes \( \text{NIdV} \):

\[
\text{NIdV} = \text{snap.TIntV}() \\
\text{for } i \text{ in range}(1,9): \text{NIdV}.\text{Add}(i)
\]

\[
\text{SubGPA} = \text{snap.GetSubGraph}(@\text{GPA}, \text{NIdV})
\]
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

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

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

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

```python
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
  - Degree-, betweenness-, closeness-, farness-, and eigen-centrality

```python
PRankH = snap.TIntFltH()
snap.GetPageRank(G, PRankH)  # Calculate node PageRank scores

for item in PRankH:
    print item, PRankH[item]  # Print them out
```
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

CC = snap.GetClustCf(GPA)
print "clustering coefficient", CC
Distances between nodes

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

\[
D = \text{snap.GetBfsFullDiam}(G, 100) \\
\text{print "diameter", D}
\]

\[
ED = \text{snap.GetBfsEffDiam}(G, 100) \\
\text{print "effective diameter", ED}
\]
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(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

```python
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)
SNAP Network Datasets

Rok Sosič, Jure Leskovec
Stanford University

ICWSM-14, Ann Arbor, MI  June, 2014
SNAP Network Datasets

- [http://snap.stanford.edu/data/](http://snap.stanford.edu/data/)
- Public collection of large network datasets
  - Over 15 network types
  - Over 70 datasets
  - Varying sizes from 20K up to 1.8B edges
- Popular resource for network scientists
  - Method development, study, benchmarking
- Contribute your dataset
  - We welcome new additions
- SNAP Dataset Users mailing list
  - [http://groups.google.com/group/snap-datasets](http://groups.google.com/group/snap-datasets)
Datasets in SNAP (1)

- **Social networks**
  - Online social networks, edges represent interactions between users
- **Location-based online social networks**
  - Social networks with geographic check-ins
- **Online communities**
  - Data from online communities such as Reddit and Flickr
- **Networks with ground-truth communities**
  - Ground-truth network communities in social/information networks
- **Online reviews**
  - Data from online review systems such as Amazon
- **Amazon networks**
  - Nodes represent products, edges link co-purchased products

http://snap.stanford.edu/data/
Datasets in SNAP (2)

- **Twitter and Memetracker**
  - Memetracker phrases, links and 467 million Tweets

- **Signed networks**
  - Networks with positive and negative edges (friend/foe, trust/distrust)

- **Communication networks**
  - Email communication networks with edges representing emails

- **Wikipedia networks and metadata**
  - Talk, editing and voting data from Wikipedia

- **Citation networks**
  - Nodes represent papers, edges represent citations

- **Collaboration networks**
  - Nodes represent scientists, edges represent collaboration (paper co-authoring)

- [http://snap.stanford.edu/data/](http://snap.stanford.edu/data/)
Datasets in SNAP (3)

- **Web graphs**
  - Nodes represent webpages and edges are hyperlinks

- **Internet networks**
  - Nodes represent computers and edges communication

- **Autonomous systems**
  - Graphs of the internet

- **Road networks**
  - Nodes represent intersections and edges roads connecting the intersections

- **http://snap.stanford.edu/data/**
Friends lists from Facebook
- Includes user profiles, circles, ego networks
- Collected via Social Circles App on Facebook
- Social circle detection Kaggle competition:

http://snap.stanford.edu/data/egonets-Facebook.html

<table>
<thead>
<tr>
<th>Dataset statistics</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>4039</td>
</tr>
<tr>
<td>Edges</td>
<td>88234</td>
</tr>
<tr>
<td>Nodes in largest WCC</td>
<td>4039 (1.000)</td>
</tr>
<tr>
<td>Edges in largest WCC</td>
<td>88234 (1.000)</td>
</tr>
<tr>
<td>Nodes in largest SCC</td>
<td>4039 (1.000)</td>
</tr>
<tr>
<td>Edges in largest SCC</td>
<td>88234 (1.000)</td>
</tr>
<tr>
<td>Average clustering coefficient</td>
<td>0.6055</td>
</tr>
<tr>
<td>Number of triangles</td>
<td>1612010</td>
</tr>
<tr>
<td>Fraction of closed triangles</td>
<td>0.2647</td>
</tr>
<tr>
<td>Diameter (longest shortest path)</td>
<td>8</td>
</tr>
<tr>
<td>90-percentile effective diameter</td>
<td>4.7</td>
</tr>
</tbody>
</table>
Location Based Social Networks

Friendship network and check-ins in Gowalla location-based social network

<table>
<thead>
<tr>
<th>Dataset statistics</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>196591</td>
</tr>
<tr>
<td>Edges</td>
<td>950327</td>
</tr>
<tr>
<td>Nodes in largest WCC</td>
<td>196591 (1.000)</td>
</tr>
<tr>
<td>Edges in largest WCC</td>
<td>950327 (1.000)</td>
</tr>
<tr>
<td>Nodes in largest SCC</td>
<td>196591 (1.000)</td>
</tr>
<tr>
<td>Edges in largest SCC</td>
<td>950327 (1.000)</td>
</tr>
<tr>
<td>Average clustering coefficient</td>
<td>0.2367</td>
</tr>
<tr>
<td>Number of triangles</td>
<td>2273138</td>
</tr>
<tr>
<td>Fraction of closed triangles</td>
<td>0.007952</td>
</tr>
<tr>
<td>Diameter (longest shortest path)</td>
<td>14</td>
</tr>
<tr>
<td>90-percentile effective diameter</td>
<td>5.7</td>
</tr>
<tr>
<td>Check-ins</td>
<td>6,442,890</td>
</tr>
</tbody>
</table>

http://snap.stanford.edu/data/loc-gowalla.html
Online Communities: Reddit

- **Post submissions to Reddit**
  - Includes an image with multiple submissions
  - **Features per posts**: number of ratings, the title, number of comments

<table>
<thead>
<tr>
<th>Dataset statistics</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of submissions</td>
<td>132,308</td>
</tr>
<tr>
<td>Number of unique images</td>
<td>16,736</td>
</tr>
<tr>
<td>Average number of times an image is resubmitted</td>
<td>7.9</td>
</tr>
<tr>
<td>Timespan</td>
<td>July 2008 - Jan 2013</td>
</tr>
</tbody>
</table>

http://snap.stanford.edu/data/web-Reddit.html
18 years of Amazon reviews up to March 2013

- Product and user information, ratings, review text

<table>
<thead>
<tr>
<th>Dataset statistics</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of reviews</td>
<td>34,686,770</td>
</tr>
<tr>
<td>Number of users</td>
<td>6,643,669</td>
</tr>
<tr>
<td>Number of products</td>
<td>2,441,053</td>
</tr>
<tr>
<td>Users with &gt; 50 reviews</td>
<td>56,772</td>
</tr>
<tr>
<td>Median no. of words per review</td>
<td>82</td>
</tr>
<tr>
<td>Timespan</td>
<td>Jun 1995 - Mar 2013</td>
</tr>
</tbody>
</table>

http://snap.stanford.edu/data/web-Amazon.html
SNAP C++ Installation

- Download the latest version of SNAP C++
  
  http://snap.stanford.edu/snap/download.html

Download SNAP

Current SNAP Release

Download the current SNAP distribution package:

SNAP 2.2 (Mar 12, 2014)

A public development SNAP repository is available at GitHub:

snap-stanford/snap
SNAP C++ Repository

- **Graph and network library**: directory `snap-core`
  - Graph and network generation, manipulation, algorithms

- **Data structures**: directory `glib-core`
  - STL-like library
  - Contains basic data structures, like vectors, hash tables and strings
  - Provides serialization for loading and saving

- **Tutorials**: directory `tutorials`
  - Short programs that demonstrate basic functionality

- **Example applications**: directory `examples`
  - Complete sample applications

- **Advanced capabilities**: directories `snap-adv, snap-exp`
SNAP Quick Start Guide

- **Download and unzip** Snap package

- **Compile programs** in subfolder **examples**
  - Windows Visual Studio
    - Project file `SnapExamples*.sln`
  - Mac OS X with Xcode
    - Project file `snap-examples*.xcodeproj`
  - Command line on Linux, Mac OS X, Cygwin
    - Makefile

- For **your own project**, copy examples/testgraph and modify it
Installation on Windows

- Install Visual Studio or Visual Studio Express
  - http://www.visualstudio.com/

- Download and Unzip Snap package

- Go to subfolder examples
- Open project SnapExamples*.sln
  - Visual Studio 2008 and 2010 projects are available
1) Open Visual Studio and create a project
   - Or start with examples/testgraph and modify it

2) Include `Snap.h` in your main program
   ```
   #include "Snap.h"
   ```

3) Include the path to directories “`snap-core`”, “`glib-core`” and “`snap-adv`” in your project
   - Properties ➔ `Configuration Properties` ➔ `VC++ Directories` ➔ `Include Directories`

4) Character set must be configured to Multi-Byte:
   - Properties ➔ `Configuration Properties` ➔ `General` ➔ `Projects Defaults` ➔ `Character Set` ➔ `Select “Use Multi-Byte Character Set”`
Installation on Mac OS X with Xcode

- Install Xcode

- Download and Unzip Snap package

- Go to subfolder examples

- Open project snap-examples*.xcodeproj

- Build the project and execute examples
Open Xcode and create a project
  - Or start with examples/testgraph and modify it

Include “Snap.h” in your main program
#include ”Snap.h“
For command line-based systems (e.g., Linux, OsX, Cygwin), use the `Makefile` in the example folder.

Makefiles are available in all folders in "examples", e.g., `examples/kronfit/Makefile"`
Basic Graph Types

- **TUNGraph**: undirected graph
- **TNGraph**: directed graph
- **TNEANet**: directed multi-graph with attributes
Graph Creation

- **Create a graph:**
  
  ```cpp
  PNGraph Graph = TNGraph::New();
  Graph->AddNode(1);
  Graph->AddNode(5);
  Graph->AddEdge(1,5);
  ```

- **Use smart-pointers**
  - typedef TPtr<TNGraph> PNGraph
  - Memory management
    - Objects are automatically released when not needed

- Add nodes (G->AddNode(i)) before adding edges (G->AddEdge(i,j))
Graph Traversal

- Traverse the nodes

```cpp
for (TNGraph::TNodeI NI = Graph->BegNI(); NI < Graph->EndNI(); NI++)
    printf("%d %d %d\n", NI.GetId(), NI.GetOutDeg(), NI.GetInDeg());
```

- Traverse the edges, globally

```cpp
for (TNGraph::TEdgeI EI = Graph->BegEI(); EI < Graph->EndEI(); EI++)
    printf("edge (%d, %d)\n", EI.GetSrcNId(), EI.GetDstNId());
```

- Traverse the edges, per node

```cpp
for (TNGraph::TNodeI NI = Graph->BegNI(); NI < Graph->EndNI(); NI++)
    for (int e = 0; e < NI.GetOutDeg(); e++)
        printf("edge (%d %d)\n", NI.GetId(), NI.GetOutNId(e));
```
Node and Edge Iterators

- Get a node iterator from node id:
  
  \[
  \text{TNGraph}::\text{TNodeI} \ NI = \text{Graph}->\text{GetNI}(\text{NIId});
  \]

- Get an edge iterator from node ids:
  
  \[
  \text{TNGraph}::\text{TEdgeI} \ EI = \text{Graph}->\text{GetEI}(\text{SrcNIId}, \text{DstNIId});
  \]
## Loading/Saving of Graphs

- **Loading a graph in the edge list, text format**
  ```cpp
  PUNGraph G2 =
  TSnap::LoadEdgeList<PUNGraph>("as20graph.txt", 0, 1);
  - 0 (1) is the column id for source (target) node
  ```

- **Saving a graph in the edge list, text format**
  ```cpp
  TSnap::SaveEdgeList<PUNGraph>(G2, "as20graph.txt", "");
  ```

- **Loading/Saving in a binary format – faster**
  ```cpp
  { TFin FIn("test.graph"); PNGraph G2 =
    TNGraph::Load(FIn); }
  { TFOut FOut("test.graph"); G2->Save(FOut); }
  - Note the parenthesis {}!
  ```
Edge List, Text File Format

- **Example file:**
  ```
  as20graph.txt in subfolder examples
  ```

  # Directed Node Graph
  # Autonomous systems ...
  # Nodes: 6474  Edges: 26467
  # SrcNIId  DstNIId
  1   3
  1   6
  1   32
  1   48
  1   63
  1   70
  ```
Get degree distribution (degree, count)

```cpp
TSnap::GetOutDegCnt(G, CntV);
```

Get distribution of connected components (component size, count)

```cpp
TSnap::GetWccSzCnt(G, CntV);
```

- CntV is a vector of pairs of integers:
  ```cpp
  TVec<TPair<TInt, TInt>> CntV;
  ```
Generating Graphs

- Generate graphs with specific properties
- Use functions `TSnap::Gen...`
  
  `TSnap::GenRndGnm()`: $G_{nm}$ Erdős–Rényi graph
  
  `TSnap::GenForestFire`, Forest Fire Model
  
  `TSnap::GenPrefAttach`, Preferential Attachment

- **Example:**
  
  - Create a directed random graph on 100 nodes and 1k edges
    
    ```cpp
    PNGraph Graph = TSnap::GenRndGnm<PNGraph>(100, 1000);
    ```
Graph Operations (Examples 2)

- **Generate a network using Forest Fire model**
  
  PNGraph G = TSnap::GenForestFire(1000, 0.35, 0.35);

- **Convert to undirected graph TUNGraph**
  
  PUNGraph UG = TSnap::ConvertGraph<PUNGraph, PNGraph> (G);

- **Get largest weakly connected component of G**
  
  PNGraph WccG = TSnap::GetMxWcc(G);

- **Get a subgraph induced on nodes {0,1,2,3,4}**
  
  PNGraph SubG = TSnap::GetSubGraph(G, TIntV::GetV(0,1,2,3,4));
SNAP Network Types

- **TNodeNet<TNodeData>:** directed graph with TNodeData object for each node

- **TNodeEDatNet<TNodeData, TEdgeData>:** directed graph with TNodeData on each node and TEdgeData on each edge

- **TNodeEdgeNet<TNodeData, TEdgeData>:** directed multi-edge graph with TNodeData on each node and TEdgeData on each edge
Example Applications

- In SNAP directory “examples”
  - **TestGraph**: Demonstrates basic functionality of the library, modify this example for your own project
  - **ForestFire**: ForestFire graph generative model
  - **Cliques**: Clique Percolation Method for detecting overlapping communities
  - **Cascades**: Simulate susceptible-infected model on a network
  - **AGMFit, BigClam, CODA, Cesna**: Community detection methods
SNAP Data Structures and Types

- In directory `glib-core`

- **Key files:**
  - `dt.h`: Data Types (`TInt`, `TFlt`)
  - `ds.h`: Data Structures (`TVec`)

- **Numbers:**
  - Integers: `TInt`
  - Real numbers: `TFlt`
  - Example:
    ```c
    TInt A = 5;
    printf("%d\n", A.Val);
    ```
Basic SNAP Types

- **String: TStr**
  
  **Examples:**
  
  ```
  TStr A = "abc";
  TStr B = "ccc";
  printf("string %s\n", A.CStr()); // -- abc
  printf("length %d\n", A.Len()); // -- 3
  printf("A[0] %c\n", A[0]); // -- a
  printf("A==B %d\n", A == B); // -- 0
  ```
SNAP Data Structures

- **Pair**
  - `TPair <Type1, Type2>`
    (Types can also be complex types like TVec, TPair...)
    ```
    TPair<TInt, TFlt> A;
    A.Val1 = 3;
    A.Val2 = 3.14;
    ```
  - **Predefined types in ds.h**
    ```
    typedef TPair<TInt, TInt> TIntPr;
    typedef TPair<TInt, TIntPr> TIntIntPrPr;
    ```

- **Triple**
  - `TTriple <Type1, Type2, Type3>```
SNAP Vectors

- **TVec<Type>**
  - Example:
    ```
    TVec<TInt> A;
    A.Add(10);
    A.Add(20);
    A.Add(30);
    printf("length %d\n", A.Len()); // -- 3
    printf("A[0] %d\n", A[0].Val); // -- 10
    ```
  - “Type” can be a complex type
    ```
    TVec< TVec< TVec<TFlt> > >
    ```
  - Predefined types in ds.h
    ```
    typedef TVec<TInt> TIntV;
    Typedef TVec<TFlt> TFltV;
    ```
SNAP Hash Tables

- **THash <key, value>**
  - **Key**: item key, provided by the caller
  - **Value**: item value, provided by the caller
  - **KeyId**: integer, unique slot in the table, calculated by SNAP

<table>
<thead>
<tr>
<th>KeyId</th>
<th>0</th>
<th>2</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>100</td>
<td>89</td>
<td>95</td>
</tr>
<tr>
<td>Value</td>
<td>“David”</td>
<td>“Ann”</td>
<td>“Jason”</td>
</tr>
</tbody>
</table>
SNAP Hash Tables

- Example:

  ```c
  THash<TInt, TStr> A;
  A.AddDat(100) = "David";
  A.AddDat(89) = "Ann";
  A.AddDat(95) = "Jason";
  printf("%s\n", A.GetDat(89).CStr()); // -- Ann, Key to Value
  printf("%d\n", A.GetKeyId(95)); // -- 5, Key to KeyId
  printf("%d\n", A.GetKey(5).Val); // -- 95, KeyId to Key
  printf("%s\n", A[5].CStr()); // -- Jason, KeyId to Value
  ```

- Predefined types in hash.h

  ```c
  typedef THash<TInt, TInt> TIntIntH;
  typedef THash<TInt, TFlt> TIntFltH;
  ```
Saving and Loading Objects

- **Binary files**
  - Fast save/load
  - Memory efficient

- **Save()**: 
  ```
  TIntStrH A;
  { TFOut fout("a.bin");
    A.Save(fout); }
  ```

- **Load()**: 
  ```
  { TFIn fin("a.bin"); A.Load(fin); }
  ```
Generating Distributions

- **TRnd class**
  - Generate random numbers according to various probability distributions

- **Example:**
  ```c
  TRnd A;
  // sample from an exponential distribution
  for (int i=0; i<10; ++i){
      printf("%f\n", A.GetExpDev(1));
  }
  ```
Calculating Statistics

- **File glib-core/xmath.h**
  - Useful for calculating moments, correlation coefficients, t-test, ...

- **Example of computing moments (TMom):**

```c
TMom Mom;
Mom.Add(5);  Mom.Add(6);  Mom.Add(8);
Mom.Def();
printf("Avg: %f\n", Mom.GetMean());
printf("Min: %f\n", Mom.GetMn());
printf("Max: %f\n", Mom.GetMx());
```
Making Plots

- Making a plot in SNAP
1) Install Gnuplot [http://www.gnuplot.info/]
   - Make sure that the directory containing wgnuplot.exe (for Windows) or gnuplot (for Linux, Mac OS X) is in your environmental variable $PATH.

2) Use TGnuPlot (glib-core/gnuplot.h):
   
   ```
   TVec<TPair<TFlt, TFlt>> XY1, XY2; ...
   TGnuPlot Gp("file name", "title name");
   Gp.AddPlot(XY1, gpwLinesPoints, "curve1");
   Gp.AddPlot(XY2, gpwPoints, "curve2");
   Gp.SetXYLabel("x-axis name", "y-axis name");
   Gp.SavePng(); //or Gp.SaveEps();
   ```
After executing, three files are generated:

- **.plt** file includes plotting commands for gnuplot
- **.tab** file contains the tab separated data
- **.png** or **.eps** is the plot
Drawing SNAP Graphs

- Use TGraphViz
  - Need to install GraphViz software first
    http://www.graphviz.org/
  - Add GraphViz path to environment variable
Drawing SNAP Graphs

```cpp
PNGraph G = TNGraph::New();
G->AddNode(1); G->AddNode(2);
G->AddNode(3); G->AddNode(4);
G->AddEdge(1,2); G->AddEdge(2,3);
G->AddEdge(1,3); G->AddEdge(2,4);
G->AddEdge(3,4);
TIntStrH Name;
Name.AddDat(1)="David";
Name.AddDat(2)="Emma";
Name.AddDat(3)="Jim";
Name.AddDat(4)="Sam";
TGraphViz::Plot<PNGraph>(G, gvlDot,
    "gviz_plot.png", ",", Name);
```
SNAP Hands-on Exercise

Rok Sosič, Jure Leskovec
Stanford University

ICWSM-14, Ann Arbor, MI       June, 2014
Stack Overflow Dataset

- Publicly available by Stack Overflow
  
  https://archive.org/download/stackexchange/stackoverflow.com-Posts.7z

- 5.2GB compressed, 26GB uncompressed

- 19,881,020 posts from Jul 2008 to May 2014
Task:
- Find top Java experts on Stack Overflow

Possible approaches for finding experts:
- Use Stack Overflow reputation score:
  - Not Java specific
  - No control
- Count the number of answers:
  - No measure of answer importance or usefulness
- Create a social network and compute user centrality:
  - Pagerank
Plan:
- Use node centrality measure, Pagerank
- Need a graph

Constructing a graph:
- Nodes, each user a node
- Edges, a question owner points to the owner of the accepted answer
Questions XML format in Posts.xml:

- Total 7,214,697 questions, Java 632,493

```xml
<row Id="4" PostTypeId="1"
    OwnerUserId="8" AcceptedAnswerId="7"
    Tags="&lt;c#&gt;&lt;winforms&gt;&lt;forms&gt;&lt;opacity&gt;"
    .. />
```

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question Id</td>
<td>4</td>
</tr>
<tr>
<td>Post Type</td>
<td>1 (question)</td>
</tr>
<tr>
<td>Question Owner</td>
<td>8</td>
</tr>
<tr>
<td>Accepted Answer</td>
<td>7</td>
</tr>
<tr>
<td>Tags</td>
<td>c#, winforms, forms, opacity</td>
</tr>
</tbody>
</table>
Stack Overflow: Answers

- **Answers XML format in Posts.xml:**
  - total 12,609,623

  ```xml
  <row Id="12" PostTypeId="2" OwnerUserId="1" ... />
  ```

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer Id</td>
<td>12</td>
</tr>
<tr>
<td>Post Type</td>
<td>2 (answer)</td>
</tr>
<tr>
<td>Answer Owner</td>
<td>1</td>
</tr>
</tbody>
</table>
Workflow to Find Java Experts

- **Step 1**, process input file, extract relevant fields
  - Get lists of questions and answers, identify Java posts
  - Convert XML format to TSV (tab separated values)

```plaintext
getQuestions.py Posts.xml
getTag.py Posts.xml java
getAnswers.py Posts.xml
```

ICWSM-14, Ann Arbor, MI
Rok Sosič and Jure Leskovec, Stanford University
Step 2, Select only questions about Java

doJoin.py java.txt questions.txt 1 1

- **java.txt**
- **questions.txt**
- **java-posts.txt**
Step 3, Find owners of accepted answers

- `answers.txt`
- `java-posts.txt`

`doJoin.py answers.txt java-posts.txt 1 3`

```
Question
QuestionOwner
Answer
Question
Answer
AnswerOwner
```
Step 4, analyze the graph

- Find top Java experts

Program calculations
- # of nodes, edges
- Distribution of weakly connected components
- In and out-degree distributions
- Top 10 experts by Pagerank
- Top 10 experts by Hits
- Top 10 learners by HIts

```
Run analysis
getStats.py qa.txt 2 6
```

```
top 10 experts by PageRank
id 992484, pagerank 0.013981
id 135152, pagerank 0.010006
id 22656, pagerank 0.007104
id 139985, pagerank 0.005521
id 157882, pagerank 0.004597
...```
Find Java Experts: Hands-on Exercise

- Download and install Snap.py

- Download programs and data for the exercise: icwsm14-T4-code.zip and icwsm14-T4-data.zip, for finding experts on Stack Overflow

- Unpack zip files icwsm14-T4-code.zip and icwsm14-T4-data.zip

- Find experts by executing the programs from command line
  - stackoverflow.sh on Mac OS X and Linux
  - stack.bat on Windows

- Explore getStats.py
  - Extend it with different graph analysis methods

- Extra exercise
  - Find Javascript experts

- Stack Overflow original data
  [https://archive.org/download/stackexchange/stackoverflow.com-Posts.7z](https://archive.org/download/stackexchange/stackoverflow.com-Posts.7z)

Contact information: Rok Sosic, rok@cs.stanford.edu