An Introduction to Snap.py: SNAP for Python

http://snap.stanford.edu/snappy

Rok Sosič, Jure Leskovec
Stanford University

CS224W, Autumn 2015 September, 2015
Snap.py Tutorial: Content

- Introduction to SNAP
- Snap.py for Python
- Network analytics

Contact information:
Rok Sosič, rok@cs.stanford.edu
Office: Gates 428
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
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](http://snap.stanford.edu/snap/doc.html)

- **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
SNAP Network Datasets

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

CS224W, Autumn 2015  September, 2015
What is Snap.py?

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

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

- Requires Python 2.x
  - Download and install Python 2.x:
    http://www.python.org
- 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
- Installation:
  - Follow instructions on the Snap.py webpage
    (sudo) 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
```
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
Snap.py Naming Conventions (1)

**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:
    ```python
    >>> 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
SNAP User Reference Manual

SNAP Library 2.4, User Reference

SNAP, a general purpose, high performance system for analysis and manipulation of large networks

TNGraph Class Reference

Directed graph. More...

#include <graph.h>

Classes

class TEdge
Edge iterator. Only forward iteration (operator++) is supported. More...

class TNode

class TNode
Node iterator. Only forward iteration (operator++) is supported. More...

Public Types

typedef TNGraph TNet

typedef TPtr< TNGraph > PNet

Public Member Functions

TNGraph ()

TNGraph (const int &Nodes, const int &Edges)
Constructor that reserves enough memory for a graph of Nodes nodes and Edges edges. More...
Vector Types

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

- **Naming convention:** $T<$value_type$>V$
  - Examples: TIntV, TFltV, TStrV

- **Common operations:**
  - Add(<value>): append a value at the end
  - Len(): vector size
  - [<index>]: get or set a value of an existing element
  - for i in V: iteration over the elements
Vector Example

```python
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]
```

Create an empty vector
Add elements
Print vector size
Get and set element value
Print vector elements
Hash Table Types

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

- **Naming:** $T<\text{key\_type}><\text{value\_type}>H$
  - **Examples:** $T\text{IntStrH}$, $T\text{IntFltH}$, $T\text{StrIntH}$

- **Common operations:**
  - $[<\text{key}>]$: add a new value or get or set an existing value
  - $\text{Len}()$: hash table size
  - $\text{for } k \text{ in } H$: iteration over keys
Hash Table Example

```python
h = snap.TIntStrH()

h[5] = "apple"
h[3] = "tomato"
h[9] = "orange"
h[6] = "banana"
h[1] = "apricot"

print h.Len()


h[3] = "peach"

for key in h:
    print key, h[key]
```
Hash Tables: KeyID

- $T<$key_type$><$value_type$>H
  - **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: T<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
```

- **TIntStrPrV**: a vector of (integer, string) pairs
- **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

\[
G_1 = \text{snap.TNGraph.New}()
\]

\[
G_1.\text{AddNode}(1)
G_1.\text{AddNode}(5)
G_1.\text{AddNode}(12)
\]

\[
G_1.\text{AddEdge}(1,5)
G_1.\text{AddEdge}(5,1)
G_1.\text{AddEdge}(5,12)
\]

\[
G_2 = \text{snap.TUNGraph.New}()
N_1 = \text{snap.TNEANet.New}()
\]

Directed graph
Add nodes before adding edges
Undirected graph, directed network
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 "(%d %d)" % (NI.GetId(), DstNId)
```
Graph Saving and Loading

```c
snap.SaveEdgeList(G4, "test.txt", "List of edges")  // Save text
G5 = snap.LoadEdgeList(snap.PNGraph,"test.txt",0,1)  // Load text

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

FIn = snap.TFIn("test.graph")  // Load binary
G4 = snap.TNGraph.Load(FIn)
```
Text File Format

- **Example file:** *wiki-Vote.txt*
  - Download from [http://snap.stanford.edu/data](http://snap.stanford.edu/data)

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

Load text:

```python
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")
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.TNGraph.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
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

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

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

\[
\text{GPA} = \text{snap.GenPrefAttach}(30, 3, \text{snap.TRnd}())
\]
Subgraphs and Conversions

- Extract subgraphs
- Convert from one graph type to another

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

\[
NIdV = \text{snap.TIntV}()
\]

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

\[
\text{SubGPA} = \text{snap.GetSubGraph}(\text{GPA}, \ NIdV)
\]
G = snap.LoadEdgeList(snap.PNGraph, "qa.txt", 1, 5)
snap.PrintInfo(G, "QA Stats", "qa-info.txt", False)

**Output:**

QA Stats: Directed

<table>
<thead>
<tr>
<th>Stat</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>188406</td>
</tr>
<tr>
<td>Edges</td>
<td>415174</td>
</tr>
<tr>
<td>Zero Deg Nodes</td>
<td>0</td>
</tr>
<tr>
<td>Zero InDeg Nodes</td>
<td>108618</td>
</tr>
<tr>
<td>Zero OutDeg Nodes</td>
<td>38319</td>
</tr>
<tr>
<td>NonZero In-Out Deg Nodes</td>
<td>41469</td>
</tr>
<tr>
<td>Unique directed edges</td>
<td>415174</td>
</tr>
<tr>
<td>Unique undirected edges</td>
<td>415027</td>
</tr>
<tr>
<td>Self Edges</td>
<td>26924</td>
</tr>
<tr>
<td>BiDir Edges</td>
<td>27218</td>
</tr>
<tr>
<td>Closed triangles</td>
<td>46992</td>
</tr>
<tr>
<td>Open triangles</td>
<td>69426319</td>
</tr>
<tr>
<td>Frac. of closed triads</td>
<td>0.000676</td>
</tr>
<tr>
<td>Connected component size</td>
<td>0.886745</td>
</tr>
<tr>
<td>Strong conn. comp. size</td>
<td>0.025758</td>
</tr>
<tr>
<td>Approx. full diameter</td>
<td>13</td>
</tr>
<tr>
<td>90% effective diameter</td>
<td>5.751723</td>
</tr>
</tbody>
</table>
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())

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
```

Get node with max degree

Get degree distribution
Node Centrality

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

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

Calculate node PageRank scores
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

```python
Triads = snap.GetTriads(GPA)
print "triads", Triads
```

### Calculate clustering coefficient

```python
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)

Calculate diameter

```python
D = snap.GetBfsFullDiam(G, 100)
print "diameter", D
```

Calculate effective diameter

```python
ED = snap.GetBfsEffDiam(G, 100)
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
```
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)

Calculate 3-core
Q&A

Thank you!

Contact information:
Rok Sosič, rok@cs.stanford.edu
Office: Gates 428