An Introduction to Snap.py: SNAP for Python

http://snap.stanford.edu/snappy

Rok Sosič, Jure Leskovec
Stanford University

CS224W, Autumn 2014
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
  http://snap.stanford.edu/snap/download.html

- **SNAP documentation**
  http://snap.stanford.edu/snap/doc.html
  - Quick Introduction, User Reference Manual
  - Source code, see tutorials

- **SNAP user mailing list**
  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
  - 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 2014  September, 2014
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>

User Code

- C++
- Python
- Python

SNAP

- C++
- Python

CS224W, Autumn 2014
Rok Sosič and Jure Leskovec, Stanford University
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

- 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 for using Snap.py:

**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
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
http://snap.stanford.edu/snap/doc.html
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<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]
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: T<key_type><value_type>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()  # Create an empty table

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

print h.Len()  # Print table size


h[3] = "peach"  # Set element value

for key in h:  # Print table elements
    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 \((\text{value1}, \text{value2})\)
  - Two values, type of \text{value1} could be different from the \text{value2} type
  - Existing values can be accessed

- Naming: \text{T<type1><type2>Pr}
  - Examples: \text{TIntStrPr}, \text{TIntFltPr}, \text{TStrIntPr}

- Common operations:
  - \text{GetVal1}: get value1
  - \text{GetVal2}: get value2
Pair Example

```python
>>> p = snap.TIntStrPr(1, "one")
Create a pair

>>> print p.GetVal1()
1
Print pair values

>>> 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

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

**Traverse nodes**

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

**Traverse edges**

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

**Traverse edges by nodes**

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

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

G5 = snap.LoadEdgeList(snap.PNGraph, "test.txt", 0, 1)

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

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

Save text

Load text

Save binary

Load binary
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
G = snap.LoadEdgeList(snap.PNGraph, "stackoverflow-Java.txt", 0, 1)
snap.PlotInDegDistr(G, "Stack-Java", "Stack-Java In Degree")

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

- **Snap.py** generates three files:
  - `.png` is the plot
  - `.tab` file contains the data (tab separated file)
  - `.plt` file contains the plotting commands
Drawing Graphs

- Install GraphViz:
  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
Network Analytics with Snap.py

Rok Sosič, Jure Leskovec
Stanford University

CS224W, Autumn 2014

September, 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

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

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

GPA = snap.GenPrefAttach(30, 3, snap.TRnd())
Extract subgraphs, convert from one graph type to another

Get an induced subgraph on a set of nodes $\mathcal{N}_{idV}$:

$\mathcal{N}_{idV} = \text{snap.TIntV}()$

for $i$ in range(1,9): $\mathcal{N}_{idV}.\text{Add}(i)$

SubGPA = snap.GetSubGraph(GPA, $\mathcal{N}_{idV}$)
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())
```

Analyze graph connectedness

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

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

PRankH = snap.TIntFltH()
print PRankH

for item in PRankH:
    print item, PRankH[item]
Triads and Clustering Coefficient

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

\[
\text{Triads} = \text{snap.GetTriads(GPA)} \\
\text{print} \ "\text{triads}, \ Triads"
\]

\[
\text{CC} = \text{snap.GetClustCf(GPA)} \\
\text{print} \ "\text{clustering coefficient}, \ CC"
\]
Distances between nodes

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

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

```
print "diameter", D
```

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

Calculate diameter

Calculate effective diameter
Identify communities of nodes

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

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

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

\[
\text{Core}_3 = \text{snap.GetKCore}(G, 3)
\]
Snap.py Tutorial: Conclusion

- Q&A

- Thank you!

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