An Introduction to SNAP

Liu Jie

2020-02-19

What is SNAP?

- Stanford Network Analysis Platform (SNAP) is a general purpose, high performance system for analysis and manipulation of large networks.
  - https://snap.stanford.edu/
  - written in C++ and easily scales to massive networks with hundreds of millions of nodes, and billions of edges
- SNAP Software
  - SNAP for C++
  - Snap.py: SNAP for Python
    - Snap.py provides performance benefits of SNAP, combined with flexibility of Python. Most of the SNAP functionality is available via Snap.py in Python.

SNAP C++ Installation

- Download the latest version of SNAP C++ and unzip
    - https://snap.stanford.edu/releases/Snap-5.0.zip
- A public development SNAP repository is available at GitHub:
    - https://github.com/snap-stanford/snap
    - you can find the useful information of SNAP in ReadMe

Compilation of SNAP Programs - option 1

- SNAP on MAC OS X, Linux and Windows with Cygwin (Command Line Installation)
  - Make sure that the GCC package with a C++ compiler is installed on your system.
  - Run "make all" in the main SNAP directory. This compiles the core SNAP library and all the application examples.
  - Note that it may report “ error: unsupported option '-fopenmp'” on Mac

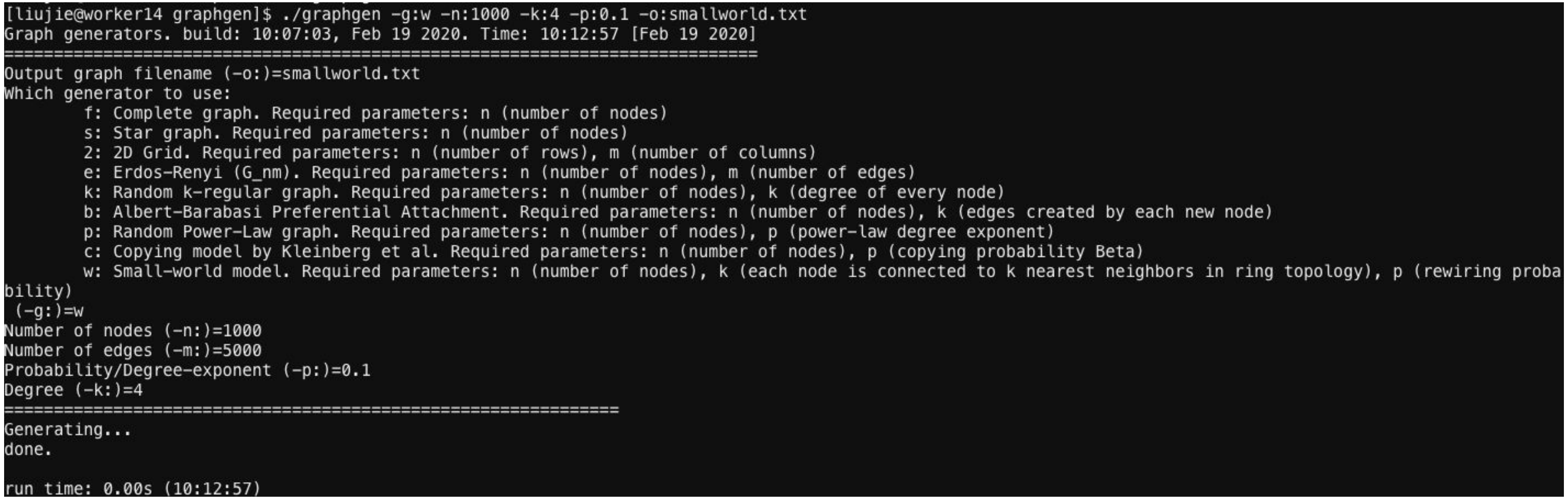
Compilation of SNAP Programs - option 2

- SNAP on Windows with Visual Studio
  - Start Visual Studio and open the solutions file SnapExamples in the examples directory.
  - Configure Visual Studio for use with SNAP, which requires that the character set is set to Multi-byte and that the locations of SNAP include directories are specified.
    - To set the character set to Multi-byte, right-click on your project, go to Properties --> Configuration Properties --> General --> Projects Defaults --> Character Set --> Select "Use Multi-Byte Character Set".
    - To specify the location of SNAP include directories, go to Options --> Preferences --> C++ Directories --> Include Directories and add the paths on your system to SNAP folders glib-core, snap-core and snap-adv.
  - Once Visual Studio is configured, build your solution, which compiles the core SNAP library all the examples.

Execution of SNAP Programs

- After the code is compiled, it can be executed. The graphgen application is used here as an example.
    - cd examples/graphgen
    - ./graphgen -g:w -n:1000 -k:4 -p:0.1 -o:smallworld.txt
- The above command generates a Watts-Strogatz small-world graph where each node is connected to k=4 closest nodes and with the rewiring probability of p=0.1.
    - You can find the detailed information, e.g. -g, -n, -k, options, in ReadMe of github repository (https://github.com/snap-stanford/snap)

graphgen example



```
[liujie@worker14 graphgen]$ ./graphgen -g:w -n:1000 -k:4 -p:0.1 -o:smallworld.txt
Graph generators. build: 10:07:03, Feb 19 2020. Time: 10:12:57 [Feb 19 2020]
============================================================================
Output graph filename (-o:)=smallworld.txt
Which generator to use:
        f: Complete graph. Required parameters: n (number of nodes)
        s: Star graph. Required parameters: n (number of nodes)
        2: 2D Grid. Required parameters: n (number of rows), m (number of columns)
        e: Erdos-Renyi (G_nm). Required parameters: n (number of nodes), m (number of edges)
        k: Random k-regular graph. Required parameters: n (number of nodes), k (degree of every node)
        b: Albert-Barabasi Preferential Attachment. Required parameters: n (number of nodes), k (edges created by each new node)
        p: Random Power-Law graph. Required parameters: n (number of nodes), p (power-law degree exponent)
        c: Copying model by Kleinberg et al. Required parameters: n (number of nodes), p (copying probability Beta)
        w: Small-world model. Required parameters: n (number of nodes), k (each node is connected to k nearest neighbors in ring topology), p (rewiring proba
bility)
 (-g:)=w
Number of nodes (-n:)=1000
Number of edges (-m:)=5000
Probability/Degree-exponent (-p:)=0.1
Degree (-k:)=4
==============================================================
Generating...
done.

run time: 0.00s (10:12:57)
```

Snap.py - SNAP for Python

- Snap.py is a Python interface for SNAP.
  - more info: https://snap.stanford.edu/snappy/index.html
  - Snap.py tutorial: https://snap.stanford.edu/snappy/doc/tutorial/tutorial.html
- System Requirements
  - macOS: macOS 10.14 with standard system Python 2.7 and Python 3.7 from Homebrew;
  - Windows 64-bit: Windows 10 with Python 2.7 and Python 3.7 from python.org;
  - Linux: Ubuntu 18.04 with standard Python 2.7 and Python 3.6, CentOS 6.5 with standard Python 2.6.

Installation of Snap.py

- Installation with pip
  - Execute pip from the command line as follows:
    - pip install snap-stanford
  - You might need to replace pip with pip3 for Python 3.x. If pip is not available on your system, try using python -m pip instead.
- Installation with setup.py
  - download and unpack the package (https://snap.stanford.edu/snappy/release) for your platform and run setup.py
  - Please refer to https://snap.stanford.edu/snappy/index.html for the detailed steps on macOS, Linux (Ubuntu and CentOS) and Windows 64-bit

Quick Introduction to Snap.py

- I will introduce some basic functions of SNAP using Snap.py
  - https://snap.stanford.edu/snappy/file/intro.py
  - for more details of Snap.py: https://snap.stanford.edu/snappy/doc/index.html
  - you may want to know more about C++ version from here: https://snap.stanford.edu/snap/index.html
- Make sure that you execute this line in Python before running any of the code below:
  - import snap

Snap.py - Graph and Network Types

- Snap.py supports graphs and networks.
    - Graphs describe topologies.
        - TUNGraph: undirected graph (single edge between an unordered pair of nodes)
        - TNGraph: directed graph (single directed edge between an ordered pair of nodes)
    - Networks are graphs with data on nodes and/or edges of the network.
        - TNEANet: directed multigraph with attributes for nodes and edges

Snap.py - Graph Creation

Example of how to create and use a directed graph:

```
# create a graph PNGraph
G1 = snap.TNGraph.New()
G1.AddNode(1)
G1.AddNode(5)
G1.AddNode(32)
G1.AddEdge(1,5)
G1.AddEdge(5,1)
G1.AddEdge(5,32)
```

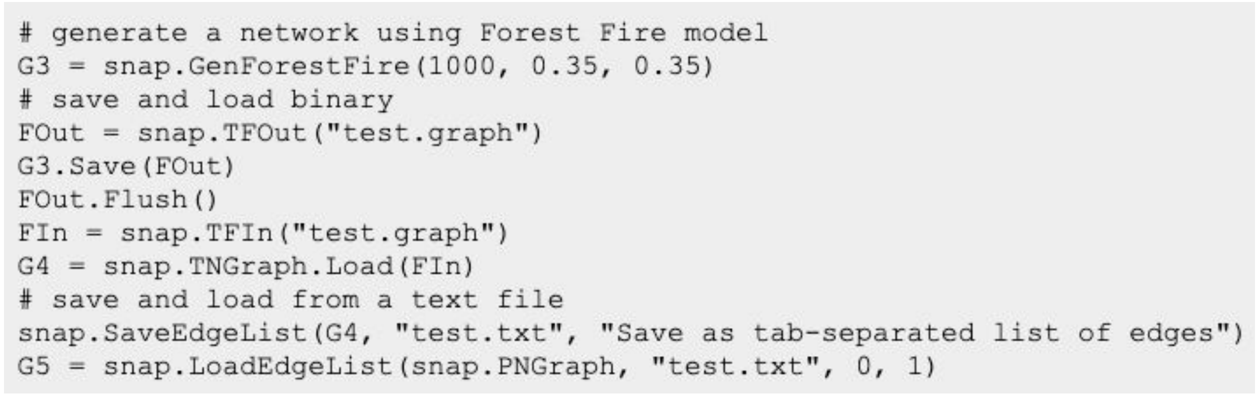
Snap.py - Iterators

- Many SNAP operations are based on node and edge iterators which allow for efficient implementation of algorithms that work on networks regardless of their type
  - GetId(): return node id
  - GetOutDeg(): return out-degree of a node
  - GetInDeg(): return in-degree of a node
  - GetOutNId(e): return node id of the endpoint of e-th out-edge
  - GetInNId(e): return node id of the endpoint of e-th in-edge
  - IsOutNId(int NId): do we point to node id n
  - IsInNId(n): does node id n point to us
  - IsNbrNId(n): is node n our neighbor

You can find related sample code here: https://snap.stanford.edu/snappy/file/intro.py

Snap.py - Input/Output

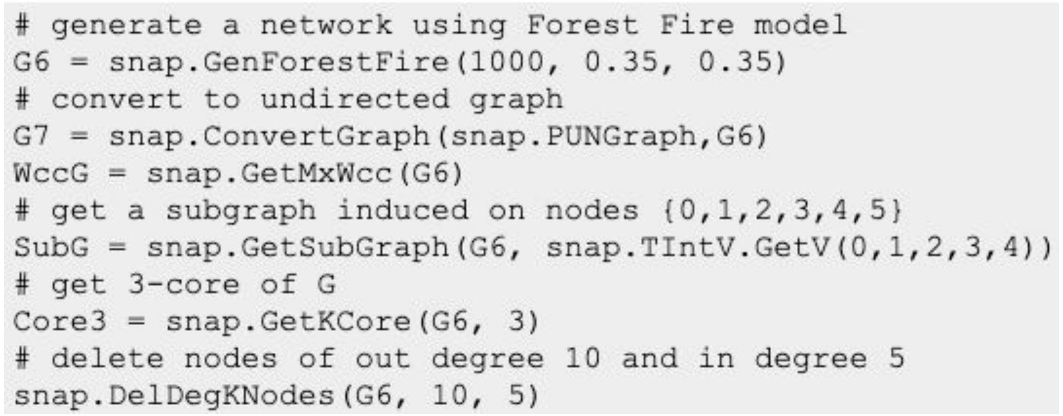
- With SNAP it is easy to save and load networks in various formats. Please check the example below



```
# generate a network using Forest Fire model
G3 = snap.GenForestFire(1000, 0.35, 0.35)
# save and load binary
FOut = snap.TFOut("test.graph")
G3.Save(FOut)
FOut.Flush()
FIn = snap.TFIn("test.graph")
G4 = snap.TNGraph.Load(FIn)
# save and load from a text file
snap.SaveEdgeList(G4, "test.txt", "Save as tab-separated list of edges")
G5 = snap.LoadEdgeList(snap.PNGraph, "test.txt", 0, 1)
```

Snap.py - Manipulating Graphs and Networks

- SNAP provides rich functionality to efficiently manipulate graphs and networks.
    - For more details on Snap.py functionality, check out the Snap.py Manuals https://snap.stanford.edu/snappy/doc/index.html.



```
# generate a network using Forest Fire model
G6 = snap.GenForestFire(1000, 0.35, 0.35)
# convert to undirected graph
G7 = snap.ConvertGraph(snap.PUNGraph,G6)
WccG = snap.GetMxWcc(G6)
# get a subgraph induced on nodes {0,1,2,3,4,5}
SubG = snap.GetSubGraph(G6, snap.TIntV.GetV(0,1,2,3,4))
# get 3-core of G
Core3 = snap.GetKCore(G6, 3)
# delete nodes of out degree 10 and in degree 5
snap.DelDegKNodes(G6, 10, 5)
```

Snap.py - Computing Structural Properties of Networks

- SNAP provides rich functionality to efficiently compute structural properties of networks.
  - e.g. get distribution of connected components, or first eigenvector of graph adjacency matrix, or diameter of the graph, or even the clustering coefficient of the graph

```
# generate a Preferential Attachment graph on 1000 nodes and node out degree of 3
G8 = snap.GenPrefAttach(1000, 3)
# vector of pairs of integers (size, count)
CntV = snap.TIntPrV()
# get distribution of connected components (component size, count)
snap.GetWccSzCnt(G8, CntV)
# get degree distribution pairs (degree, count)
snap.GetOutDegCnt(G8, CntV)
# vector of floats
EigV = snap.TFltV()
# get first eigenvector of graph adjacency matrix
snap.GetEigVec(G8, EigV)
# get diameter of G8
snap.GetBfsFullDiam(G8, 100)
# count the number of triads in G8, get the clustering coefficient of G8
snap.GetTriads(G8)
snap.GetClustCf(G8)
```

Thank you