SNAP

Justin DeBenedetto

The College of Engineering at the University of Notre Dame

Stanford Network Analysis Project

- Originally Stanford Network Analysis Platform
- Actively developed since 2004
- Largest dataset analyzed was Microsoft Instant Messenger network of 240 million nodes and 1.3 billion edges
- Built to:
  - Handle large graphs efficiently
  - Implement many common algorithms
  - Allow dynamic network changes

SNAP Overview

- 8 graph/network types
- 20 graph generation methods
- >100 graph algorithms
- Available in C++ and as Python module
- Open source

Stanford Large Network Dataset Collection

- Maintained alongside SNAP
- ~80 network datasets
  - Online social networks
  - Communication networks
  - Scientific citation networks
  - Collaboration networks
  - etc.

Comparison to NetworkX

- They consider NetworkX to be similar
- They find SNAP runs 1 to 2 orders of magnitude faster
- SNAP uses 50 times less memory
- Both can be used for Python
- NetworkX has more flexibility
- Run on single machine

Input, Output, Save, and Load

- Can read in various formats
  - One edge per line (source target)
  - One node per line (source target1 target2 …)
  - Other established systems like DyNet and Pajek
- Can also build
  - Generators
  - One node/edge at a time
- Save and load as binary
  - Internal representation for faster save/load

Containers

- Each graph/network is implemented in a container

Table I. SNAP Graph and Network Containers.

Graph Containers	
TUNGraph	Undirected graphs
TNGraph	Directed graphs
TNEGraph	Directed multigraphs
TBPGraph	Bipartite graphs
Network Containers	
TNodeNet	Directed graphs with node attributes
TNodeEDatNet	Directed graphs with node and edge attributes
TNodeEdgeNet	Directed multigraphs with node and edge attributes
TNEANet	Directed multigraphs with dynamic node and edge attributes

Interchangeable

- All functionality available on all containers
- To change network type, only change container
- Implementing algorithm on one container works on all
- Each node/edge has unique integer id

Graph Storage

Balance vector and hash table benefits

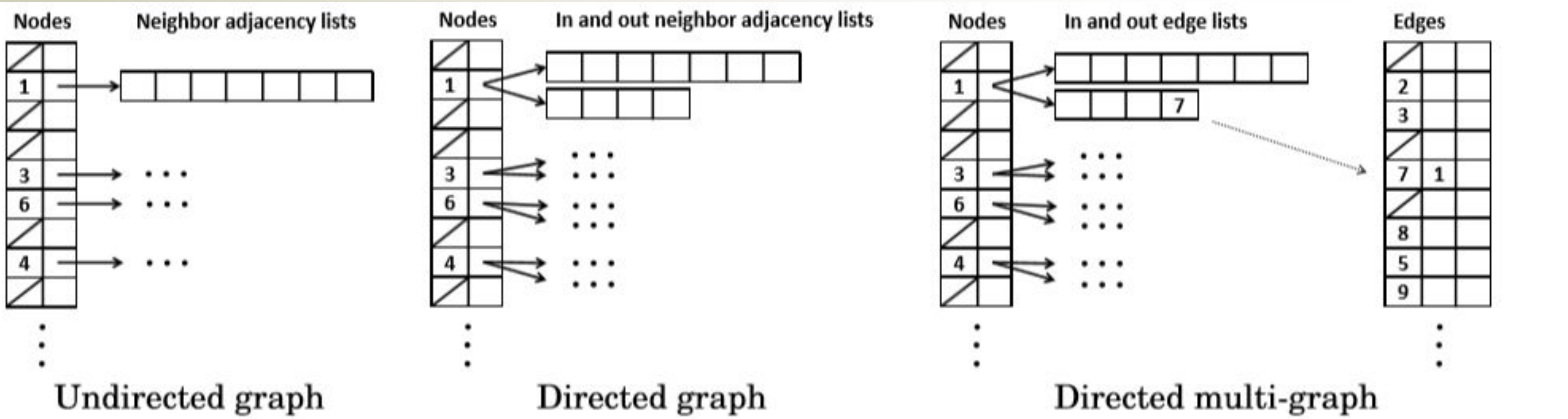

Fig. 2. A diagram of graph data structures in SNAP. Node ids are stored in a hash table, and each node has one or two associated vectors of neighboring node or edge ids.

Common Methods

Table II. Common Graph and Network Methods.

Nodes	
`AddNode`	Adds a node
`DelNode`	Deletes a node
`IsNode`	Tests, if a node exists
`GetNodes`	Returns the number of nodes
Edges	
`AddEdge`	Adds an edge
`DelEdge`	Deletes an edge
`IsEdge`	Tests, if an edge exists
`GetEdges`	Returns the number of edges
Graph Methods	
`Clr`	Removes all nodes and edges
`Empty`	Tests, if the graph is empty
`Dump`	Prints the graph in a human readable form
`Save`	Saves a graph in a binary format to disk
`Load`	Loads a graph in a binary format from disk
Node and Edge Iterators	
`BegNI`	Returns the start of a node iterator
`EndNI`	Returns the end of a node iterator
`GetNI`	Returns a node (iterator)
`NI++`	Moves the iterator to the next node
`BegEI`	Returns the start of an edge iterator
`EndEI`	Returns the end of an edge iterator
`GetEI`	Returns an edge (iterator)
`EI++`	Moves the iterator to the next edge

Sample Iterating

```
// traverse all the nodes, print out-degree for each node
for (TNGraph::TNodeI NI=Graph->BegNI(); NI<Graph->EndNI(); NI++) {
  printf("node %d, outdegree %d\n", NI.GetId(), NI.GetOutDeg());
}

// traverse all the edges, print source and destination nodes
for (TNGraph::TEdgeI EI=Graph->BegEI(); EI<Graph->EndEI(); EI++) {
  printf("edge (%d, %d)\n", EI.GetSrcNId(), EI.GetDstNId());
}
```

Listing 1. Iterating over Nodes and Edges. Top example prints out the ids and out-degrees of all the nodes. Bottom example prints out all the edges as pairs of edge source node id and edge destination node id. These traversals can be executed on any type of a graph/network container.

Benchmarks

- Uses 50x less memory than NetworkX
- Uses slightly more memory than other vector-based systems (iGraph)
- 15x faster than iGraph for save/load
- 200x faster than NetworkX for save/load
- Comparable speed to iGraph otherwise
  - Much faster than NetworkX
  - Allows dynamic networks (unlike iGraph)

Graph Generators

Table III. Graph generators in SNAP.

Category	Graph Generators
Regular graphs	Complete graphs, circles, grids, stars, and trees;
Basic random graphs	Erdős-Rényi graphs, Bipartite graphs, Graphs where each node has a constant degree, Graphs with exact degree sequence;
Advanced graph models	Configuration model [Bollobás 1980], Ravasz-Barabasi model [Ravasz and Barabási 2003], Copying model [Kumar et al. 2000], Forest Fire model [Leskovec et al. 2005], Geometric preferential model [Flaxman et al. 2006], Barabasi-Albert model [Barabási and Albert 1999], Rewiring model [Milo et al. 2003], R-MAT [Chakrabarti et al. 2004], Graphs with power-law degree distribution, Watts-Strogatz model [Watts and Strogatz 1998], Kronecker graphs [Leskovec et al. 2010], Multiplicative Attribute Graphs [Kim and Leskovec 2012b].

Some Included Algorithms

Table IV. Graph manipulation and analytics methods in SNAP.

Category	Graph Manipulation and Analytics
Graph manipulation	Graph rewiring, decomposition to connected components, subgraph extraction, graph type conversions;
Connected components	Analyze weakly, strongly, bi- and 1-connected components;
Node connectivity	Node degrees, degree distribution, in-degree, out-degree, combined degree, Hop plot, Scree plot;
Node centrality algorithms	PageRank, Hits, degree-, betweenness-, closeness-, farness-, and eigen-centrality, personalized PageRank;
Triadic closure algorithms	Node clustering coefficient, triangle counting, clique detection;
Graph traversal	Breadth first search, depth first search, shortest paths, graph diameter;
Community detection	Fast modularity, clique percolation, link clustering, Community-Affiliation Graph Model, BigClam, CoDA, CESNA, Circles;
Spectral graph properties	Eigenvectors and eigenvalues of the adjacency matrix, spectral clustering;
K-core analysis	Identification and decomposition of a given graph to k-cores;
Graph motif detection	Counting of small subgraphs;
Information diffusion	Infopath, Netinf;
Network link and node prediction	Predicting missing nodes, edges and attributes.

Citations

http://snap.stanford.edu/

SNAP: A General-Purpose Network Analysis and Graph-Mining Library (ACM 2016)

Questions?

More Examples

Get degree distribution pairs (out-degree, count):

```
>>> snap.GetOutDegCnt(G9, CntV)
>>> for p in CntV:
>>>     print "degree %d: count %d" % (p.GetVal1(), p.GetVal2())
```

Generate a Preferential Attachment graph on 100 nodes and out-degree of 3:

```
>>> G10 = snap.GenPrefAttach(100, 3)
```

Define a vector of floats and get first eigenvector of graph adjacency matrix:

```
>>> EigV = snap.TFltV()
>>> snap.GetEigVec(G10, EigV)
>>> nr = 0
>>> for f in EigV:
>>>     nr += 1
>>>     print "%d: %.6f" % (nr, f)
```

Get an approximation of graph diameter:

```
>>> diam = snap.GetBfsFullDiam(G10, 10)
```

Count the number of triads:

```
>>> triads = snap.GetTriads(G10)
```

Get the clustering coefficient:

```
>>> cf = snap.GetClustCf(G10)
```