



Tutorial: Large Scale Network Analytics with SNAP

<http://snap.stanford.edu/proj/snap-www>

Rok Sosič, Jure Leskovec
Stanford University





SNAP C++

Rok Sosič, Jure Leskovec
Stanford University

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 2.4 (May 11, 2015)

A public development SNAP repository is available at GitHub:

[snap-stanford/snap](https://github.com/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
 - <http://snap.stanford.edu/snap/download.html>
- **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**
 - <http://snap.stanford.edu/snap/download.html>
- Go to subfolder **examples**
- Open project **SnapExamples*.sln**
 - Visual Studio 2008 and 2010 projects are available

Visual Studio: Creating New Project

- **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
 - <https://developer.apple.com/xcode/>
- **Download and Unzip Snap package**
 - <http://snap.stanford.edu/snap/download.html>
- Go to subfolder **examples**
- Open project **snap-examples*.xcodeproj**
- Build the project and execute examples

Xcode – Creating New Project

- **Open Xcode and create a project**
 - Or start with **examples/testgraph** and modify it
- Include “**Snap.h**” in your main program
#include “Snap.h”

Command Line Installation on Linux, Mac OS X, Windows with Cygwin

- 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:**

```
PNGraph Graph = TNGraph::New();  
Graph->AddNode(1);  
Graph->AddNode(5);  
Graph->AddEdge(1,5);
```

- **Use smart-pointers**

- typedef TPt<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

```
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

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

■ Traverse the edges, per node

```
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:**

```
TNGraph::TNodeI NI = Graph->GetNI(NId);
```

- **Get an edge iterator from node ids:**

```
TNGraph::TEdgeI EI = Graph->GetEI(SrcNId,DstNId);
```

Loading/Saving of Graphs

- **Loading a graph in the edge list, text format**

```
PUNGraph G2 =
```

```
    TSnap::LoadEdgeList<PUNGraph>("as20graph.txt", 0, 1);
```

- 0, 1 are the columns of source, target nodes

- **Saving a graph in the edge list, text format**

```
TSnap::SaveEdgeList<PUNGraph>(G2, "as20graph.txt", "");
```

- **Loading/Saving in a binary format – faster**

```
{ 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
# SrcNId  DstNId
1    3
1    6
1    32
1    48
1    63
1    70
...
```


Graph Operations (Examples 1)

- **Get degree distribution (degree, count)**

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

- **Get distribution of connected components (component size, count)**

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

- `CntV` is a vector of pairs of integers:

`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

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
- **TNodeEdgeNet<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:

```
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** $\langle \text{key}, \text{value} \rangle$
 - **Key**: item key, provided by the caller
 - **Value**: item value, provided by the caller
 - **Keyld**: integer, unique slot in the table, calculated by SNAP

Keyld	0	2	5
Key	100	89	95
Value	"David"	"Ann"	"Jason"

SNAP Hash Tables

■ Example:

```
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

```
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:**

```
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`**

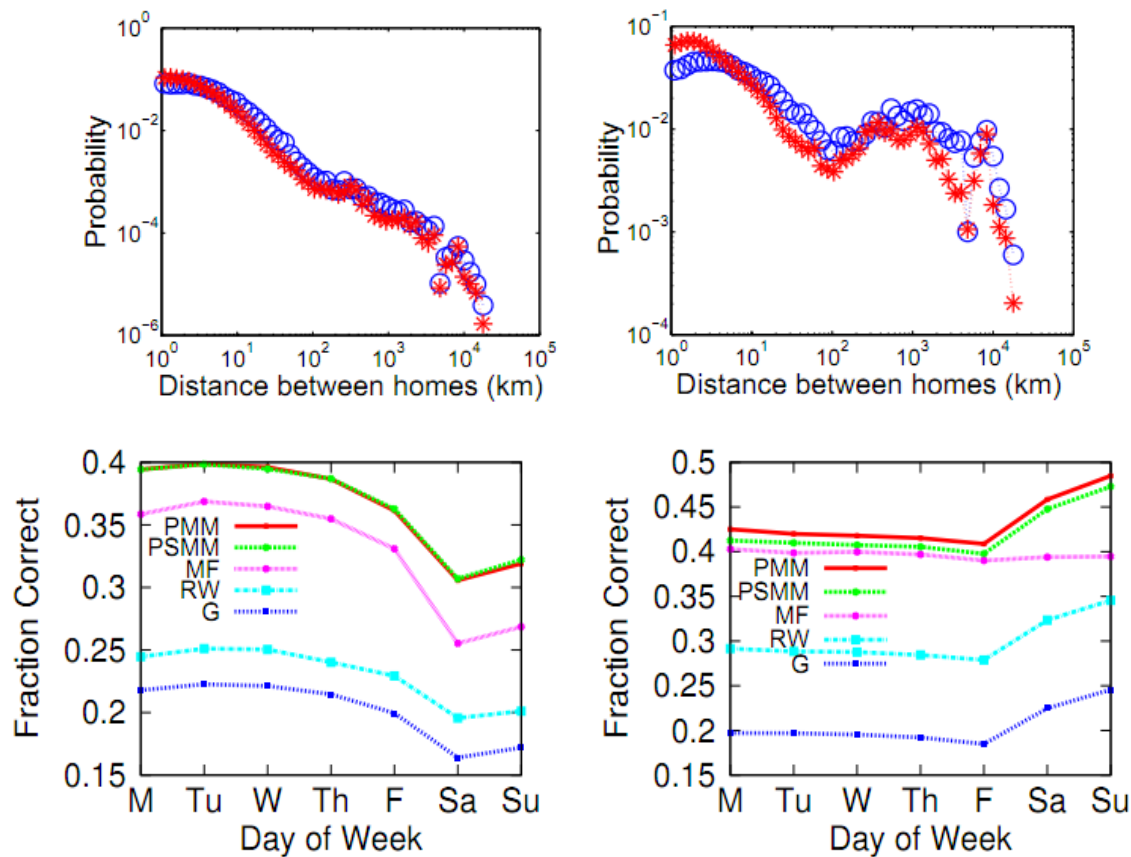
- Useful for calculating moments, correlation coefficients, t-test, ...

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

```
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



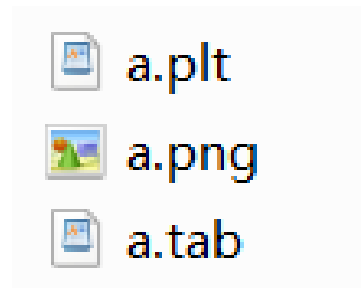
Making Plots in SNAP

- **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();
```


Gnuplot in SNAP

- After executing, three files are generated



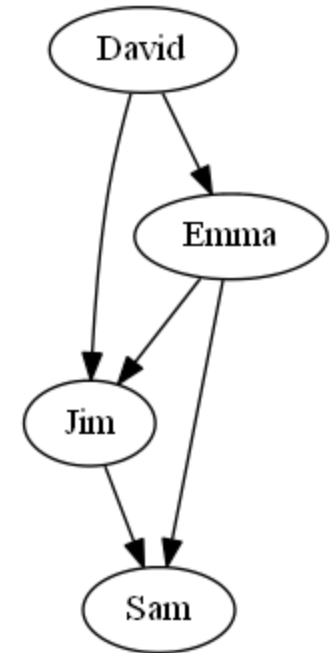
- **.plt** file includes plotting commands for gnuplot
- **.tab** file contains the tab separated data
- **.png** or **.eps** is the plot

Drawing SNAP Graphs

- Use **TGraphViz**
 - Need to install GraphViz software first
<http://www.graphviz.org/>
- Add GraphViz path to environment variable

Drawing SNAP Graphs

```
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, gv1Dot,
    "gviz_plot.png", "", Name);
```



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