The World is Big and Linked: Whole Spectrum Industry Solutions towards Big Graphs — Graph Computing and Tutorial of IBM System G

IBM System G Team
Presenters: Ching-Yung Lin (lead), Toyotaro Suzumura, Yinglong Xia
IBM T. J. Watson Research Center

October 31st, 2015
Agenda:

• 4:00 – 4:30 Introduction of IBM System G
• 4:30 – 4:40 IBM System G Visualizer & Demo
• 4:40 – 5:10 Quick Exploration of IBM System G
  • gShell, py-gShell, gremlin-gShell (groovy), REST API
  • gShell Analytics
  • Programming/User-Defined Plugins
• 5:10 – 5:50 Glance at IBM System G Eco-system
  • GraphBIG
  • ScaleGraph
• 5:50 – 6:00 Q&A
Introduction to IBM System G

http://systemg.research.ibm.com
Network / Graph is the way we remember, we associate, and we understand.

Demo
IBM System G — Graph Computing as an Intelligence Machine

Graph Database

- Property Graph

Memory

- Charles Flint
  - born: 1850
  - died: 1934

- IBM
  - HQ: "Armonk"
  - employees: 433,362
  - industry: Software, Hardware, Services

Graph Analytics

- Relation Graph

Perception

Graphical Models

- Reasoning Graph

Intelligence

Contextual Analysis

- Personal Event
  - Unusual Activities
  - Planning
  - Stress
  - Workplace Conflict

Related Information

- Machine Reasoning & Deep Learning

http://systemg.research.ibm.com
What is IBM System G?

A Complete Graph Computing Suite — Toolkits, Solutions, & Cloud

http://systemG.research.ibm.com (Internet) or http://systemG.ibm.com (IBM internal site)

Rich Graph Algorithm/Functions Primitives

- Centralities
- Communities
- Graph Sampling
- Network Info Flow
- Shortest Paths
- Ego Net Features
- Graph Matching
- Graph Query
- Graph Search
- Bayesian Networks
- Latent Net Inference
- Markov Networks
- Spatio-Temporal Ana.

Multi Graph Type Support

- Few, very large graphs (e.g. social, Internet of things)
- Many, many small graphs (e.g. protein, healthcare)
- Large semantic graph (Semantic web, RDF, Graph search, Graph recommendation)
- Large Probabilistic graphical models: Bayesian networks, Markovian networks, HMMs, etc.

And More:

- Graph Visualizations
- Graph Databases
- Graph Middleware for Hardware Platform Optimization
- Cognitive Networks and Cognitive Analytics
- Graph-Enabled Industry Solutions

Based on 100+ innovations including 8 best paper awards; $22M+ R&D investment

“IBM System G” is an IBM corporate approved external naming (April 2014). First production solution (SmallBlue): Oct 2008; Based on 30+ graph-related projects in IBM Research since 2003
Big Data: “While enterprises struggle to consolidate systems and collapse redundant databases to enable greater operational, analytical, and collaborative consistencies, changing economic conditions have made this job more difficult. E-commerce, in particular, has exploded data management challenges along three dimensions: volumes, velocity and variety. In 2001/02, IT organizations much compile a variety of approaches to have at their disposal for dealing each.” – Doug Laney, Gartner, 2001
Graph is a missing pillar in the existing Big Data foundation.

Graph Computing is difficult because data cannot be easily partitioned.
Graph Database key differentiator — native store

Native Graph DB stores nodes and relationships directly, it makes retrieval efficient.

In Relational DB, relationships are distributed and stored as tables.

Retrieving multi-step relationships is a 'graph traversal' problem.

Technology ==> Top Layer: Graph, Bottom Layer: Graph

Cited “Graph Database” O’liey 2013
Forrester: Over 25% of enterprise will use Graph DB by 2017

Graph DB is in the significant success trajectory, and has the highest business value among the upcoming DBs.
GraphDB has the largest Popularity Change among DBMS lately.
Comparison of graph size

No. of edges

\[ \log_2(m) \]

\[ \log_2(n) \]

1 trillion edges

1 billion edges

USA-Road Network

USA-road-d.LKS.gr

USA-road-d.NY.gr

USA-road-d.USA.gr

Symbolic Network

Human Brain Project

Graph500 (Toy)

Graph500 (Mini)

Graph500 (Small)

Graph500 (Medium)

Graph500 (Large)

Graph500 (Huge)

Twitter (tweets/day)

1 billion nodes

1 trillion nodes

© 2015 IBM Corporation
July 2015: IBM Research’s Software powered all Top 3 winners of Graph 500 benchmark and 9 out of the Top 10 winners (supercomputers in US, Japan, France, UK, and Germany; except in China).

The July 2015 winner, K-computer supercomputer of 83K nodes and 663K cores, achieved graph search of up to 38,621,400,000 vertices per second.
IBM System G Application Use Cases

1. System G for Expertise Location
2. System G for Recommendation
3. System G for Commerce
4. System G for Financial Analysis
5. System G for Social Media Monitoring
6. System G for Telco Customer Analysis
7. System G for Watson
8. System G for Data Exploration and Visualization
9. System G for Personalized Search
10. System G for Anomaly Detection (Espionage, Sabotage, etc.)
11. System G for Fraud Detection
12. System G for Cybersecurity
13. System G for Sensor Monitoring (Smarter another Planet)
14. System G for Celluar Network Monitoring
15. System G for Cloud Monitoring
17. System G for Traffic Navigation
18. System G for Image and Video Semantic Understanding
19. System G for Genomic Medicine
20. System G for Brain Network Analysis
21. System G for Data Curation
22. System G for Near Earth Object Analysis
Value of Social Connections

Insider Newsletter
A weekly summary of the best in BusinessWeek and BusinessWeek.com

THIS WEEK'S TOP STORY

Putting a Price on Social Connections

Researchers at IBM and Massachusetts Institute of Technology say that indulging in certain types of electronic communications makes for higher productivity at work. Our Insider Top Story this week, "Putting a Price on Social Connections," is part of our Special Report, The Value of Virtual Friends. Find out what sort of networking increases your value as an employee.

You've seen the advertisements on TV and in the paper—apparently there has never been a better time to sell Grandma's tiara. But hold on: Click through our slide show, "Selling Your Jewelry," and read the accompanying story first to see what you can get for what you've got. Have you ever had the thing appraised?

IBM

According to happy employees and thousands of IBM employees who have been encouraged to "connect" were the better they were in $948 in revenue. IBM's Dr. Ching-Yung Lin has shown, prove the value of your connections.
Value of Social Network

15,000 contributors in 76 countries; 92,000 unique IBM users
25,000,000 emails & SameTime messages (incl. Content features)
1,000,000 Learning clicks; 14M KnowledgeView, SalesOne, …, access data
1,000,000 Lotus Connections (blogs, file sharing, bookmark) data
200,000 people's consulting financial databases
400,000 organization/demographic data
Decades of Social Science studies demonstrates that (social) network structure is the key indicator determining a person's influence, organizational operation efficiency, social capital to get help, potential to be successful, etc.

Who are the key bridges? Who have the most connections? How do these experts cluster?

Analogy – Google founders utilized the concept of network analysis on webpages to create ranking.

SmallBlue analyzes underlying dynamic network structure in enterprise
Network Value Analysis - First Large-Scale Economical Social Network Study

Productivity effect from network variables

- An additional person in network size ~ $948 revenue per year
- Each person that can be reached in 3 steps ~ $0.163 in revenue per month
- A link to manager ~ $1074 in revenue per month
- 1 standard deviation of network diversity (1 - constraint) ~ $758
- 1 standard deviation of btw ~ -$300K
- 1 strong link ~ $-7.9 per month

- Structural Diverse networks with abundance of structural holes are associated with higher performance.
  - Having diverse friends helps.
  - Betweenness is negatively correlated to people but highly positive correlated to projects.
    - Being a bridge between a lot of people is bottleneck.
    - Being a bridge of a lot of projects is good.
  - Network reach are highly corrected.
  - The number of people reachable in 3 steps is positively correlated with higher performance.
  - Having too many strong links — the same set of people one communicates frequently is negatively correlated with performance.
  - Perhaps frequent communication to the same person may imply redundant information exchange.
Project Team Composition—Managers

The number of managers in a project exhibit an inverted-U shaped curve.

1. Having managers in a project is correlated with team performance initially.
2. Too many managers in a project is negatively associated with team performance.

\[ \text{revenue} = \alpha + \beta_1 \cdot \text{mgr} + \beta_2 \cdot \text{mgr}^2 + \gamma_1 \cdot \text{otherfactor}_1 + \ldots + \gamma_k \cdot \text{otherfactor}_k + \varepsilon \]

<table>
<thead>
<tr>
<th># Managers in project</th>
<th>( \beta_1 )</th>
<th>2733.9***</th>
</tr>
</thead>
<tbody>
<tr>
<td>(# Managers in project) (^2)</td>
<td>( \beta_2 )</td>
<td>-682.02***</td>
</tr>
</tbody>
</table>

\( S = .027 \)  \( S = -.056 \)
Behavior depends on Culture and Roles

Culture difference of normal behavior (ICIS 2011)

Role difference of normal behavior

Organization difference of normal behavior

© 2015 IBM Corporation
System G Graph Analytical Tools

- **Network topological analysis** tools
  - Centralities (degree, closeness, betweenness)
  - PageRank
  - Communities (connected component, K-core, triangle count, clustering coefficient)
  - Neighborhood (egonet, K-neighborhood)

- **Graph matching and search** tools
  - Graph search/filter by label, vertex/edge properties (including geo locations)
  - Graph matching
  - Collaborative filtering

- **Graph path and flow** tools
  - Shortest paths
  - Top K-shortest paths

- **Probabilistic graphical model** tools
  - Bayesian network inference
  - Deep learning
Characteristics of IBM System G Graph Analytics

• Cover a wide range of graph analytics to support many application use cases in different domains, e.g.:
  • Enterprise social network analysis, expertise search, knowledge recommendation
  • Financial/security anomaly/fraud detection
  • Social media monitoring and analysis
  • Cellular network analytics in Telco operation
  • Patient and disease analytics for healthcare
  • Live neural brain network analysis

• Provide efficient in-memory computation as well as on-disk persistence
• Optimal performance enabled by IBM System G graph database technologies that focus on efficient use of available computing resources with architecture-aware design to leverage system/architecture advantages
• Single-threaded, concurrent (shared memory), and distributed versions
• Multiple deployment options to suit different customer preferences and needs
  • C++ executables in Linux environments (Redhat CentOS, Ubuntu, Mac OS X, Power)
  • TinkerPop (Blueprints) API
  • gShell (a shell-like environment with interactive, batch, and server/client modes to operate multiple graph stores simultaneously)
  • Gremlin console
  • REST API Web service
  • Python wrapper
Existing foundation of 16 types of graph visualization assets in these 4 categories:

- **Multivariate Graphs**: nodes and edges have multivariate attributes. E.g., healthcare graphs, workflow graphs, behavior reasoning graphs, etc.

- **Heterogeneous Graphs**: graphs in which nodes and edges are in different categories and types. E.g.: bipartite/tripartite/multi-partite graphs, geospatial graphs, etc.

- **Dynamic Graphs**: graphs whose topology and attributes change over time. E.g., relationship graphs, information propagation graphs, etc.

- **Big Graphs**: graphs with millions or even billions of nodes and edges. Hierarchical-based visualization or infinite-plane based visualization. E.g., social graphs, knowledge graphs, etc.

Web-based:
- HTML5
- WebGL

http://systemg.ibm.com
And Romance belongs to the old days

Demo — Exploration
Social Media Monitoring

Modeling, Tracking and Affecting Information Dissemination in Context

==> 26 Fundamental Research Tasks organized in 3 Thrusts — Modeling, Tracking and Affecting
Demo — Live Example
A novel **Cognitive Security System** to detect and predict abnormal behaviors in organizations from large-scale multimodality data of people through graph computing, cognitive analytics, data mining, and machine learning.

**Detection, Prediction, and Exploration Interface**

- **Analytics Infrastructure**
- **Graph analysis**
- **Behavior analysis**
- **Content analysis**
- **Emotion analysis**

**Available existing data**
- Emails
- Instant Messaging
- Web Access
- Executed Processes
- Printing
- Copying
- Log On/Off

**Cognition Layer**

**Semantics Layer**

**Concept Layer**

**Feature Layer**

**Sensor Layer**

~500 original detector scores / per day / per person

Espionage Detection
Sabotage Detection
Fraud Detection

3 potential ‘Attack’ scores per person

© 2015 IBM Corporation
Demo — Reasoning
Download IBM System G Standard Edition (on-premise)

http://systemg.research.ibm.com/download.html

or

Resources

- IBM System G on Bluemix (need registration)
  - http://systemg.mybluemix.net

- IBM System G Graph Analytics Overview

- IBM System G Graph Tools Trial Download

- IBM System G Graph Tools Installation Guide and Documentation
IBM System Visualizer (SystemG-Lite)
IBM System G Visualizer – Graph Data Explorer

Visual Query Panel

Visualization Panel

Visual Mapping Panel

Console Panel
Panel Introduction

• Visual Query Panel
  • Providing users a friendly UI to create, delete, and query graphs from the System G native store.

• Console Panel
  • Display all the interaction information with System G native store.
  • Execute user defined query.

• Visualization Panel
  • Rendering graph structure on screen for users to visually explore graphs.

• Visual Mapping Panel
  • Customizing rendering effects to show desired graph information.
Visual Query Panel – Creating a graph

1: Click “Create Graph”; 2: Prepare the graph data; 3: Set the graph name; 4: Upload node files; 5: Upload edge files and finalize creating the graph.
“analytics_degree <= 10 and (group == “center” or group == “guard”)"
Visual Mapping Panel

<table>
<thead>
<tr>
<th>Name</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Background Color</td>
<td>Change the background color of the canvas.</td>
</tr>
<tr>
<td>Node Default Color</td>
<td>Set a unified color for all nodes.</td>
</tr>
<tr>
<td>Edge Default Color</td>
<td>Set a unified color for all edges.</td>
</tr>
<tr>
<td>Show Nodes</td>
<td>Set the visibility of all nodes.</td>
</tr>
<tr>
<td>Node Color Mapping</td>
<td>Assign color to nodes according to selected property of nodes.</td>
</tr>
<tr>
<td>Node Size Mapping</td>
<td>Assign the radius of nodes according to selected property of nodes.</td>
</tr>
<tr>
<td>Filter Node Label by Node Size</td>
<td>Selectively show the node label according to the threshold. Labels will be shown for the nodes of which the size is larger than the threshold.</td>
</tr>
<tr>
<td>Node Label Mapping</td>
<td>Set the label value according to selected property of nodes.</td>
</tr>
<tr>
<td>Node Label Size</td>
<td>Adjust the font size of node labels</td>
</tr>
<tr>
<td>Show Edges</td>
<td>Set the visibility of all edges.</td>
</tr>
<tr>
<td>Edge Color Mapping</td>
<td>Assign color to edges according to selected property of edges.</td>
</tr>
<tr>
<td>Edge Label Mapping</td>
<td>Set the label value according to selected property of edges.</td>
</tr>
<tr>
<td>Edge Label Size</td>
<td>Adjust the font size of edge labels</td>
</tr>
<tr>
<td>Edge Style</td>
<td>Select the rendering style of edges. For directed graphs, users also can choose if showing the arrows or not.</td>
</tr>
<tr>
<td>Edge Thickness Mapping</td>
<td>Assign thickness to edges according to selected property of edges.</td>
</tr>
</tbody>
</table>
Visualization Panel – After Customization

Node Color Mapping: 
- Blue: Actor
- Orange: Movie
- Green: undefined
- Red: Male Actor

Edge Color Mapping: 
- Gray: ACTS_IN

© 2015 IBM Corporation
Users can further specify colors by clicking the color blocks shown in the legend area.
http://systemg.ibm.com/tool/visualizer/
Quick Exploration of IBM System G

- gShell
- py-gShell
- gremlin-gShell
- REST API
- Programming/User-Defined Plugins
gShell → a Straightforward Way to Feel Native Store

- gShell is a simple implementation based on the native graph API
- Accepts a string of characters locally or remotely as the input
- Assumes properties are of identical format (can be k/v pairs)
- Outputs results with some format (plain text, json, etc) and interfaces with System G Visualization component
- Wrapped into C/S mode
gShell and IBM System G Native Store

- Native store organizes graph data for representing a graph with both structure and the vertex properties and edge properties using multiple files in Linux file system
  - Creating a list called ID → Offset where each element translates a vertex (edge) ID into two offsets, pointing to the earliest and latest data of the vertex/edge, respectively
  - Creating a list called Time_stamp → Offset where each element has a time stamp, an offset to the previous time stamp of the vertex/edge, and a set of indices to the adjacent edge list and properties
  - Create a list of chained block list to store adjacent list and properties
Download & Use — So Simple!

Download:  http://systemg.research.ibm.com/download.html

Download and Support
The System G Graph Tools Trial Download version is free, intended for experimentation, research and application development. You can use it to support your commercial or non-commercial applications. But, please note that, this software cannot be redistributed or sold. It is the users’ own risk using the software.

You can download the IBM System G Graph Tools Trial Download from here.

There is no online support for this version and IBM may choose to update the version at our discretion. Feedback & enhancement suggestions may be sent to systemg@us.ibm.com (remove white space).
Use gShell

- **README**: a text file that describes the content of the package and provides references to documentation files
- **systemg.sh**: a script to set up environment variables required to run IBM System G Graph Tools
- **doc/**: documentation files
- **data/**: sample data files for tests
- **gshell/**: gShell executable files, sample data, and test scripts
- **lib/**: library files for gShell
- **python/**: Python interface to gShell
- **blueprints-gremlin/**: Blueprints API and Gremlin
- **resapi/**: REST API executable files and scripts
- **systemg-lite/**: IBM System G Lite visualization
Use gShell - 2

```
Yinglons-MacBook-Pro:gshell yxiawen@yxiawen:~$ gShell
add_edge
analytic_auction
analytic_closeness_centrality
analytic_connected_component
analytic_k_core
analytic_reset_engine
analytic_stop_engine
delete_eprop
eport_csv
find_edge
find_random_edges
find_vertex_max_degree
get_num_vertices
indexer_clucene
load_csv_vertices
update_edge
close
delete
help
```

```bash
.gShell interactive
```

```
add_vertex
add_vertex_json
analytic_betweenness
analytic_clustering
analytic_degree_centrality
analytic_k_shortest_path
analytic_shortest_path
analytic_triangle_closure
```

```
gShell>> list_all
["list_all"]
{
"warning": ["MESSAGE" : "store is empty!"]
}
```

```
gShell>> list_all --help
["list_all"] [--help]
{
"info": {
"MESSAGE" : "list_all - list all graphs"},
"MESSAGE" : "--format: [optional] output format"},
"MESSAGE" : "--help: [optional] help information"
}
```

Write Python Code based on System G

```python
#!/usr/bin/python
from py_gShell import _py_gshell as gShell
import json

g = gShell()
g.delete_graph("testu")
g.create_graph("testu", "undirected")
g.load_csv_vertices(csvfile="data/test.vertices.dat", keypos=0, labelpos=1)
g.load_csv_edges(csvfile="data/test.edges.dat", srcpos=0, targpos=1, labelpos=3)
g.add_vertex(vertex_id="7", label="C", prop={"tag": "T2", "value": 0.1})
g.add_vertex(vertex_id="8", prop={"value": 0.4})
g.add_vertex(vertex_id="9", label="C", prop={"value": 0.5})
g.update_vertex(vertex_id="9", prop={"value": 0.55, "other": "1"})
g.add_edge(src="7", targ="8", edgelabel="c")
g.add_edge(src="7", targ="1", edgelabel="c", prop={"weight": 8.0})
g.update_edge(src="1", targ="7", prop={"weight": 8.6, "other": "2"})
g.add_edge(src="8", targ="9")
g.update_edge(src="1", targ="2", prop={"weight": 6.5})
g.analytic_start_engine(edgeweightpropname="weight")
print json.dumps(json.loads(g.analytic_find_path(src="1",sink="2")), indent = 4)

print json.dumps(json.loads(g.analytic_find_path(src="1",sink="2",label="b")), indent = 4)
g.analytic_stop_engine()
```

Output of the above Python script:  
```json
{
  "paths": [
    {
      "src": "1",
      "path": "1->3->5->2",
      "sink": "2",
      "distance": 3.0
    },
    {
      "src": "1",
      "path": "1->2",
      "sink": "2",
      "distance": 1.0
    }
  ],
  "time": [
    {
      "TIME": "3.31402e-05"
    },
    {
      "TIME": "2.09808e-05"
    }
  ]
}
```
Open Source TinkerPop Stack (Apache Incubator)

http://tinkerpop.incubator.apache.org

SQL2 Gremlin

http://sql2gremlin.com
Use Gremlin-gShell

gremlin> g = CreateGraph.openGraph("nativemem_authors","awesome")
==>nsgraph[vertices:7 edges:8]
gremlin> g.class
==>class com.ibm.research.systemg.nativestore.tinkerpop.NSGraph
gremlin> // lets look at all the vertices
==>true
gremlin> g.V

```java
gremlin> gs = new GShell()
==>com.ibm.research.systemg.nativestore.gshell.GShell@5e88a3de
gremlin> gs.exec("create --graph test --type directed")
140711320353584
[create] [--graph] [test] [--type] [directed]
===>
"info":{"MESSAGE":"store [test] is created!"}"
}
gremlin> gs.exec("add_vertex --graph test --id "test node" --prop tag:"test tag""")
139868232521952
[add_vertex] [--graph] [test] [--id] [test node] [--prop] [tag] [test tag]
===>
"info":{"MESSAGE":"vertex is added"},
"time":{"TIME":"0.000422001"}"
```
User-Defined Analytics

- a header file template
- a cpp file template
- add .o file to link
THAT’S IT!

```c
#include "plugin_helloWorld.h"

REGISTER_QUERY_NAME(example_helloWorld, "example_helloWorld");

void example_helloWorld::options(command_options &opts)
{
    opts.add_command_info("this is an example of gShell plugin");
    opts.add_option("arg1", true, HAS_ARGUMENT, "arg1 is a mandatory argument with value ");
    opts.add_option("arg2", false, HAS_ARGUMENT, "arg2 is an optional argument with value ");
    opts.add_option("arg3", false, NO_ARGUMENT, "arg3 is an optional flag");
    opts.add_option("arg4", false, MULTIPLE_ARGUMENT, "arg4 is an optional flag with multiple values");
}

int example_helloWorld::run(struct query_param_type param)
{
    if (param.directness == TYPE_UNDIRECT)
        param.internal_output->info("this is a undirected graph");
    else
        param.internal_output->warning("this is a directed graph");

    string arg1, arg2;
    param.opts->get_value("arg1", arg1);
    param.opts->get_value("arg2", arg2);

    bool arg3 = param.opts->get_flag("arg3");
    if (arg3)
        param.internal_output->info("arg3 is true");
    else
        param.internal_output->info("arg3 is false");
}
```
IBM System G Eco-System (GraphBIG)
GraphBIG

A group of graph analytics for benchmarking underlying platforms

A simplified IBM System G in-memory graph layer, with similar APIs

Come with performance profiler by taking hardware performance counters, breaking down the execution time into multiple stages to reveal the performance bottleneck

Fetch Code

Code: https://github.com/graphbig/graphBIG
Doc: https://github.com/graphbig/GraphBIG-Doc

```
-bash:~$ git clone https://github.com/graphbig/graphBIG.git GraphBIG
Cloning into 'GraphBIG'...
remote: Counting objects: 497, done.
remote: Compressing objects: 100% (110/110), done.
remote: Total 497 (delta 57), reused 0 (delta 0), pack-reused 386
Receiving objects: 100% (497/497), 2.07 MiB | 0 bytes/s, done.
Resolving deltas: 100% (229/229), done.
Checking connectivity... done.
-bash:~$
```
Understand Graph Computational Challenges

76%

76% of the total execution is spent inside the framework by invoking primitive graph operations framework.

framework actually plays a critical role
Graph Data Representations

(a) Graph G

(b) CSR Representation of G

(c) Vertex-centric Representation of G

CSR format is compact, and maybe good for cache performance. But it is static, and cannot support structure changes. However, in practices, graphs are usually dynamic. This is why vertex-centric representation is popular across multiple graph frameworks.
Graph Computing Types

• Computation on graph structure (CompStruct)
  • Example: Breadth-first search
  • Irregular access pattern, heavy read access

• Computation on dynamic graph (CompDyn)
  • Example: Streaming Graph
  • Dynamic graph structure, dynamic memory usage

• Computation on graph property (CompProp)
  • Example: Belief propagation
  • Heavy numeric operations on graph property
We start from the use cases of IBM System G. By analyzing the use cases, we are able to summarize the computation and data types. Meanwhile, we select workloads and data from them. After that, we then have a reselection stage. In the reselection stage, we reselect workloads and data to ensure that they cover all computation and data types.
In total, we analyzed 21 use cases from 6 different categories, from science exploration to security. Different categories contain different use cases and different selected workloads also have different popularities across the use cases. But in general, all workloads are widely used in multiple real-world use cases.
# Workload Summary and Experiments to Show

<table>
<thead>
<tr>
<th>Category</th>
<th>Workload</th>
<th>Computation Type</th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph traversal</td>
<td>BFS</td>
<td>CompStruct</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>DFS</td>
<td>CompStruct</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Graph update</td>
<td>Graph construction (GCons)</td>
<td>CompDyn</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Graph update (GUp)</td>
<td>CompDyn</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Topology morphing (TMorph)</td>
<td>CompDyn</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Graph analytics</td>
<td>Shortest path (SPath)</td>
<td>CompStruct</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>kCore</td>
<td>CompStruct</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Connected component (CComp)</td>
<td>CompStruct</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Graph coloring (GColor)</td>
<td>CompStruct</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Triangle counting (TC)</td>
<td>CompProp</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Gibbs Inference (GI)</td>
<td>CompProp</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Social analytics</td>
<td>Betweenness Centrality (BCentr)</td>
<td>CompStruct</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Degree Centrality (DCentr)</td>
<td>CompStruct</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data set</th>
<th>Type</th>
<th>Vertex #</th>
<th>Edge #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Twitter Graph</td>
<td>Type 1</td>
<td>120M</td>
<td>1.9B</td>
</tr>
<tr>
<td>IBM Knowledge Repo</td>
<td>Type 2</td>
<td>154K</td>
<td>1.72M</td>
</tr>
<tr>
<td>IBM Watson Gene Graph</td>
<td>Type 3</td>
<td>2M</td>
<td>12.2M</td>
</tr>
<tr>
<td>CA Road Network</td>
<td>Type 4</td>
<td>1.9M</td>
<td>2.8M</td>
</tr>
<tr>
<td>LDBC Graph</td>
<td>Synthetic</td>
<td>Any</td>
<td>Any</td>
</tr>
</tbody>
</table>
GraphBIG Hands-on

Fetch Code

Code: https://github.com/graphbig/graphBIG
Doc: https://github.com/graphbig/GraphBIG-Doc

```
-bash:~$ git clone https://github.com/graphbig/graphBIG.git GraphBIG
Cloning into 'GraphBIG'...
remote: Counting objects: 497, done.
remote: Compressing objects: 100% (110/110), done.
remote: Total 497 (delta 57), reused 0 (delta 0), pack-reused 386
Receiving objects: 100% (497/497), 2.07 MiB | 0 bytes/s, done.
Resolving deltas: 100% (229/229), done.
Checking connectivity... done.
```

GraphBIG is open sourced under BSD license. We have an organization in github named as graphbig. To obtain the GraphBIG code is pretty simple. Just do use git to perform a “git clone” More detailed documents can also be found in a separate repository in the same organization in github.
GraphBIG is a standalone package. It doesn’t require any external libraries. But of course, you need a gcc and for gpu workloads, you need cuda sdk.

To compile it, just “make all”.

To compile the full suite, you can “make all” at the top level. If you just want to compile CPU benchmarks, get into “benchmark/” directory and “make all”. Similarly for GPU workloads, get into “gpu_bench/” and “make all”.

```bash
-bash:~$ cd GraphBIG/
bash:GraphBIG$ ls
benchmark CHANGELOG.md common csr_bench dataset gpu_bench LICENSE openG README.md tools
-bash:GraphBIG$ cd benchmark/
bash:benchmark$ ls
bench_betweennessCentr bench_degreeCentr bench_graphConstruct bench_shortestPath
bench_BFS bench_DFS bench_graphUpdate bench_TopoMorph
bench_connectedComp bench_gibbsInference bench_kCore
bench_triangleCount
-bash:benchmark$ make all
make -C ./tools all
make[1]: Entering directory `/home/lifeng/GraphBIG/tools'
rm -rf librpm-4.5.0.tgz
librpm-4.5.0.
librpm-4.5.0./COPYING
librpm-4.5.0./lib/
librpm-4.5.0./lib/pfmlib_ppc970.c
librpm-4.5.0./lib/pfmlib_powerpc.c
librpm-4.5.0./lib/pfmlib_sparc_priv.h
librpm-4.5.0./lib/pfmlib_powerpc_perf_event.c
```
It is also pretty simple to make a test run of GraphBIG workloads. We include the simple test run already in the makefile. You can get into the directory of any benchmark and use “make run”. Then, a test run will be performed and the output will be stored in a log file named “output.log”

To get more info about the arguments of a specific benchmark, just run it with “--help”
Characterization

Methodology
Real machine + hardware performance counters

CPU: linux perf event kernel calls (integrated with benchmarks)
GPU: CUDA nvprof

<table>
<thead>
<tr>
<th>Processor</th>
<th>Type</th>
<th>Xeon E5-2670</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>2.6 GHz</td>
<td></td>
</tr>
<tr>
<td>Core #</td>
<td>2 sockets x 8 cores x 2 threads</td>
<td></td>
</tr>
<tr>
<td>Cache</td>
<td>32KB L1, 256KB L2, 20MB L3</td>
<td></td>
</tr>
<tr>
<td>Memory BW</td>
<td>51.2 GB/s (DDR3)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GPU</th>
<th>Type</th>
<th>Nvidia Tesla K40</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA Core</td>
<td>2880</td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td>12 GB</td>
<td></td>
</tr>
<tr>
<td>Memory BW</td>
<td>288 GB/s</td>
<td></td>
</tr>
<tr>
<td>Frequency</td>
<td>Core-745 MHz, mem-3 GHz</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System</th>
<th>Memory</th>
<th>192 GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk</td>
<td>2 TB HDD</td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td>RHEL 6</td>
<td></td>
</tr>
</tbody>
</table>
We breakdown the total execution time into four categories. Both frontend and backend represent the CPU stall cycles. One is stall cycles caused by frontend issues, the other is stall cycles caused by backend issues. Badspeculation represents the wasted cycles because of wrong branch predictions. The retiring is the actual running and useful cycles.

We can see that for most workloads, backend is the dominant, it is the bottleneck here. Backend may include instruction execution, retiring, memory sub-systems. But outliers also exist, for TC (triangle counting) and Gibbs (gibbs inference), they are not suffering from backend issues. It shows an interesting diversity across benchmarks.
IBM System G Eco-System (ScaleGraph)
ScaleGraph Library

Build an open source **Highly Scalable Large Scale Graph Analytics Library** beyond the scale of billions of vertices and edges on Distributed Systems
Graph Algorithms

Currently supported algorithms

- PageRank
- Degree Distribution
- Betweenness Centrality
- Shortest path
- Breadth First Search
- Minimum spanning tree (forest)
- Strongly connected component
- Spectral clustering
- Separation of Degree (HyperANF)
- Cluster Coefficient

The algorithms that will be supported in the future.

- Blondel clustering
- Eigen solver for sparse matrix
- Connected component
- Random walk with restart
- etc.
Weak Scaling and Strong Scaling Performance up to 128 nodes (1536 cores)

Weak Scaling Performance of Each Algorithm (seconds): RMAT Graph of Scale 22 per node

<table>
<thead>
<tr>
<th></th>
<th>PageRank</th>
<th>BFS</th>
<th>SSSP</th>
<th>WCC</th>
<th>SC</th>
<th>HyperANF</th>
<th>Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMAT, Scale 22, 1 nodes</td>
<td>13.7</td>
<td>1.9</td>
<td>8.9</td>
<td>5.6</td>
<td>351.1</td>
<td>50.3</td>
<td>33.1</td>
</tr>
<tr>
<td>RMAT, Scale 26, 16 nodes</td>
<td>28.3</td>
<td>4.0</td>
<td>13.5</td>
<td>12.0</td>
<td>701.4</td>
<td>88.9</td>
<td>36.3</td>
</tr>
<tr>
<td>RMAT, Scale 28, 64 nodes</td>
<td>37.9</td>
<td>7.5</td>
<td>18.8</td>
<td>17.0</td>
<td>1166.0</td>
<td>103.5</td>
<td>39.4</td>
</tr>
<tr>
<td>RMAT, Scale 29, 128 nodes</td>
<td>45.3</td>
<td>11.2</td>
<td>24.5</td>
<td>22.1</td>
<td>1438.8</td>
<td>142.3</td>
<td>41.1</td>
</tr>
<tr>
<td>Random, Scale 29, 128 nodes</td>
<td>46.5</td>
<td>8.8</td>
<td>20.6</td>
<td>21.4</td>
<td>1106.6</td>
<td>162.3</td>
<td>42.7</td>
</tr>
</tbody>
</table>

Strong Scaling Performance of Each Algorithm (seconds): RMAT Graph of Scale 28

<table>
<thead>
<tr>
<th></th>
<th>PageRank</th>
<th>BFS</th>
<th>SSSP</th>
<th>WCC</th>
<th>SC</th>
<th>HyperANF</th>
<th>Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 nodes</td>
<td>124.1</td>
<td>21.9</td>
<td>65.8</td>
<td>55.9</td>
<td>2969.9</td>
<td>38.0</td>
<td>16.1</td>
</tr>
<tr>
<td>32 nodes</td>
<td>91.7</td>
<td>18.7</td>
<td>36.9</td>
<td>30.2</td>
<td>1639.0</td>
<td>27.0</td>
<td>11.6</td>
</tr>
<tr>
<td>64 nodes</td>
<td>38.1</td>
<td>7.5</td>
<td>20.1</td>
<td>17.2</td>
<td>1169.9</td>
<td>10.6</td>
<td>4.9</td>
</tr>
<tr>
<td>128 nodes</td>
<td>26.5</td>
<td>5.8</td>
<td>14.7</td>
<td>10.5</td>
<td>706.4</td>
<td>6.8</td>
<td>3.1</td>
</tr>
</tbody>
</table>

Evaluation Environment: TSUBAME 2.5 (Each node is equipped with two Intel® Xeon® X5760 2.93 GHz CPUs by each CPU having 6 cores and 12 hardware threads, 54GB of memory. All compute nodes are connected with InifinitBand QDR)
## Performance of XPregel

The execution time of PageRank 30 iteration for the Scale 20 (1 million vertices, 16 million edges) RMAT graph with 4 TSUBAME nodes.

![Graph showing execution times](image)

<table>
<thead>
<tr>
<th>Framework</th>
<th>Execution Time (second)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Giraph</td>
<td>153</td>
</tr>
<tr>
<td>GPS</td>
<td>100</td>
</tr>
<tr>
<td>Optimized X-Pregel</td>
<td>2.4</td>
</tr>
</tbody>
</table>

Giraph and GPS data is from [Bao and Suzumura, LSNA 2013 WWW Workshop].
Web Site and Source Code Repository

Official web site – http://scalegraph.org
  Project information
  Source code distribution
  Documentation
Source code repository - http://github.com/scalegraph/
  License: Eclipse Public License v1.0
  Project information and Documentation
  Source code distribution / VM Image
ScaleGraph Software Stack

User Program

Graph Algorithm

X10

XPregel
Graph Processing System

BLAS for Sparse Matrix

File IO

X10 & C++

Third-Party Library Interface

Third-Party Libraries
(ARPACK, METIS)

Optimized Team

X10 Core Lib

ScaleGraph Core Lib

X10 Native Runtime

MPI

X10 & C++
Developing Graph Algorithms (e.g. PageRank)

```scala
xpgraph.iterate[Double,Double](
  // Compute closure
  (ctx :VertexContext[Double, Double, Double, Double], messages :MemoryChunk[Double]) => {
    val value :Double;
    if(ctx.superstep() == 0) {
      // calculate initial page rank score of each vertex
      value = 1.0 / ctx.numberOfVertices();
    } else {
      // for step onward,
      value = (1.0-damping) / ctx.numberOfVertices() +
        damping * MathAppend.sum(messages);
    }
    // sum score
    ctx.aggregate(Math.abs(value - ctx.value()));
    // set new rank score
    ctx.setValue(value);
    // broadcast its score to its neighbors
    ctx.sendMessageToAllNeighbors(value / ctx.outEdgesId().size());
  },

  // Aggregate closure: calculate aggregate value
  (values :MemoryChunk[Double]) => MathAppend.sum(values),

  // End closure: should continue?
  (superstep :Int, aggVal :Double) => {
    return (superstep >= maxIter || aggVal < eps);
  });

public def iterate[M,A](
  compute :(ctx:VertexContext[V,E,M,A],
           messages:MemoryChunk[M])
    => void,
  aggregator : (MemoryChunk[A])=>A,
  end :(Int,A)=>Boolean)
```

Developing Graph Algorithms (e.g. PageRank)
Developing Graph Algorithms (e.g. PageRank)

The core algorithm of a graph kernel can be implemented by calling `iterate` method of XPregelGraph as shown in the example.

Users are also required to specify the type of messages (M) as well as the type of aggregated value (V).

The method accepts three closures: `compute` closure, `aggregator` closure, and `end` closure.

In each superstep (iteration step), a vertex contributes its value, which depends on the number of links, to its neighbors.

Each vertex summarizes the score from its neighbors and then set the score as its value.

The computation continues until the aggregated value of change in vertex’s value less than a given criteria or the number of iterations less than a given value.
Installation and Developing Graph Algorithms

Installation and Execution Guide

http://www.scalegraph.org/web/index.php/documentation/getting-started-guides

PageRank Example:

https://github.com/scalegraph/scalegraph/blob/develop/src/example/PageRankSimple.x10
Understanding time-series nature of large-scale social networks (e.g. separation of degree, diameter, clustering, ..)

Crawled the entire Twitter follower/followee network of 826.10 million vertices and 29.23 billion edges. How could we analyze this gigantic graph?
Crawling Billion-Scale Twitter Follower-Followee Network

with Twitter API (v1.0*) from Jul. 2012 to Oct. 2012 (around 3 months).
begin with top 1,000 users*¹ with the largest number of followers according to breadth-first search along the direction of follower

*¹: Twitaholic. http://twitaholic.com/top100/followers/
Crawled Data Set

We stopped our crawling at depth 29
Because the user after depth 26 was less than 100.
Finally, we collected **469.9 million user data**.

Collect two kind of user data by crawling for 3 months
1. User profile
   Include user id, screen_name, description, account creation time, time zone, etc.
   The serialized data size is **91GB**

2. Follower-friend
   Adjacency list of followers and friends
   The compressed(gzip) data size is **231GB**

To perform the Twitter network analysis
- **Apache Hadoop** for large-scale data processing
- **HyperANF** for approximate calculation of degree of separation and diameter
  Lars Backstrom\(^1\) also use HyperANF for Facebook network analysis

\(^1\) : “Four degrees of separation” ACM Web Science 2012
Explore Twitter Evolution (1/2)
- Transition of the number of users

Total user count (left fig.)
Twitter started at June 2006 and rapidly expanded from beginning of 2009.
Haewoon Kwak *1 studied Twitter network on July 2009

Monthly increase of users (right fig.)
Twitter users increase, but it seems a little unstable...

*1 : “What is Twitter, a social network or a news media?”
Explore Twitter Evolution (2/2)
- Transition of the number of users by regions -

Classify 131 million users by “Time zone” property under 6 regions
Africa, Asia, Europe, Latin America and Caribbean (Latin), Northern America (NA), Oceania
Only 131 million user correctly set one’s own “Time zone”

Massive change of ratio of users by region
Asia users : 8.30% => 20.8% (12.5% up)
NA users : 54.4% => 40.4% (14.0% down)

Characteristic of Twitter network also change?

<table>
<thead>
<tr>
<th></th>
<th>July 2009</th>
<th>October 2012</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># users</td>
<td>ratio (%)</td>
</tr>
<tr>
<td>Africa</td>
<td>0.13M</td>
<td>0.66</td>
</tr>
<tr>
<td>Asia</td>
<td>1.65M</td>
<td>8.30</td>
</tr>
<tr>
<td>Europe</td>
<td>3.01M</td>
<td>15.1</td>
</tr>
<tr>
<td>Latin</td>
<td>3.80M</td>
<td>19.0</td>
</tr>
<tr>
<td>NA</td>
<td>10.9M</td>
<td>54.6</td>
</tr>
<tr>
<td>Oceania</td>
<td>0.45M</td>
<td>2.29</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>19.9M</strong></td>
<td><strong>100</strong></td>
</tr>
</tbody>
</table>
Monthly Increase of Users by Regions

- Africa
- Asia
- Europe
- Latin America and the Caribbean
- Northern America
- Oceania
**Degree Distribution: Unexpected value in in-degree distribution**

“Scale-free” is one of the features of a social graph. Unexpected value in in-degree distribution at $x=20$ due to Twitter recommendation system at $x=2000$ due to upper bound of friends before 2009.

Out-degree distribution (follower)  In-degree distribution (friend)
Reciprocity: decline from 22.1% to 19.5%

Reciprocity is a quantity to specifically characterize directed networks. Traditional Definition:

\[ r = \frac{L_{\leftrightarrow}}{L} \]

- \( L_{\leftrightarrow} \): # of edges pointing in both directions
- \( L \): # of total edges

As a result, **Twitter network reciprocity decline from 22.1% to 19.5%**

<table>
<thead>
<tr>
<th></th>
<th>July 2009</th>
<th>October 2012</th>
</tr>
</thead>
<tbody>
<tr>
<td># of users</td>
<td>41.6 M</td>
<td>465.7 M</td>
</tr>
<tr>
<td># of edges</td>
<td>1.47 B</td>
<td>28.7 B</td>
</tr>
<tr>
<td><strong>Reciprocity</strong></td>
<td>22.1% *1</td>
<td>19.5%</td>
</tr>
</tbody>
</table>

*1: “What is Twitter, a social network or a news media?”
93% users have less than or equal to 100 followers

99.94% users have less than or equal to 10,000 followers

However, their followers count are only 11% of total followers count

But still... 57.6% of total followers count

Only 0.06% celebrities control most of edges in Twitter network

How many edges do celebrities have in Twitter network? → Only 0.06% celebrities control most of edges
Degree of Separation and Network Diameter (1/3)

Both degree of separation and diameter are measures to characterize networks in terms of scale of graph.

Definition

Degree of Separation:
Average value of the shortest-path length of all pairs of users.

Diameter:
Maximum value of the shortest-path length of all pairs of users

* Note: unreachable pairs are excluded from calculation

(A, B) = 1
(A, C) = 1
(B, A) = \infty
(B, C) = 1
(C, A) = \infty
(C, B) = 1

Degree of Separation : 1
Diameter : 1
Degree of Separation and Network Diameter (2/3)

Experimental environment

Using an approximate algorithm named HyperANF [Paolo, WWW’12] on TSUBAME 2.0 (Supercomputer at TITECH)

TSUBAME 2.0 Fat node

64 cores, 512 GB memory, SUSE Linux Enterprise Server 11 SP1

HyperANF Parameters

We set the logarithm of the number of registers per counter to 6 in order to reduce an error.

Four times executions

Degree of Separation

take a average of 4 calculation

Diameter

take a minimum value of 4 calculation

because HyperANF guarantee lower bound of diameter

Each execution on 2012 took more than 42,000 sec.
Degree of Separation and Network Diameter (3/3)

Degree of Separation

Only a little difference between ‘09 and ’12 in spite of the lapse of three years.

Diameter

Diameter of 2012 is much larger than the one of 2009.

Cumulative Distribution

In 2009
- 89.2% of node pairs whose path length is 5 or shorter
- 99.1% pairs whose it is 6 or shorter.

In 2012
- 85.2% pairs whose it is 5 or shorter
- 94.6% pairs whose it is 6 or shorter.

<table>
<thead>
<tr>
<th>Degree of Separation</th>
<th>Diameter</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2009</td>
</tr>
<tr>
<td>1st</td>
<td>4.39</td>
</tr>
<tr>
<td>2nd</td>
<td>4.46</td>
</tr>
<tr>
<td>3rd</td>
<td>4.53</td>
</tr>
<tr>
<td>4th</td>
<td>4.62</td>
</tr>
<tr>
<td>Result</td>
<td><strong>4.50</strong></td>
</tr>
</tbody>
</table>

Cumulative Distribution
The scale-28 graphs we used have $2^{28} \approx 268$ million of vertices and $16 \times 2^{28} \approx 4.29$ billion of edges.

Strong-scaling result of HyperANF (scale 28)
Degree of Separation and Diameter for Time-Evolving Twitter Network
# Classifying Degree of Separation by Spoken Language

<table>
<thead>
<tr>
<th></th>
<th>Spanish</th>
<th>Portuguese</th>
<th>Japanese</th>
<th>Turkish</th>
<th>French</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Users</td>
<td>64,927,267</td>
<td>22,456,938</td>
<td>20,279,402</td>
<td>10,402,846</td>
<td>10,743,511</td>
</tr>
<tr>
<td>Follow ratio to its own language</td>
<td>64%</td>
<td>58%</td>
<td>89%</td>
<td>57%</td>
<td>51%</td>
</tr>
<tr>
<td>Follow ratio to English</td>
<td>31%</td>
<td>36%</td>
<td>9%</td>
<td>39%</td>
<td>44%</td>
</tr>
<tr>
<td># of Nodes for DOS</td>
<td>60,708,434</td>
<td>21,152,308</td>
<td>19,682,116</td>
<td>9,638,906</td>
<td>8,964,888</td>
</tr>
<tr>
<td># of Edges for DOE</td>
<td>2,266,838,184</td>
<td>1,098,723,999</td>
<td>1,394,986,423</td>
<td>271,513,323</td>
<td>177,419,512</td>
</tr>
<tr>
<td>Average Degree</td>
<td>37.33</td>
<td>51.94</td>
<td>70.87</td>
<td>28.16</td>
<td>19.79</td>
</tr>
<tr>
<td>Degree of Separation (Average path length between two users)</td>
<td>4.625</td>
<td>4.253</td>
<td>4.014</td>
<td>4.340</td>
<td>4.699</td>
</tr>
<tr>
<td>Diameter (Lower bound value)</td>
<td>42</td>
<td>23</td>
<td>27</td>
<td>39</td>
<td>22</td>
</tr>
</tbody>
</table>
Q & A
Backup
A graph:

\[ G = (V, E) \]

- \( V = \) Vertices or Nodes
- \( E = \) Edges or Links

- The number of vertices: “Order” \( N_v = |V| \)
- The number of edges: “Size” \( N_e = |E| \)
In-degrees and out-degrees

- For Directed graphs:

  ![Diagram](image)
  
  In-degree = 8
  Out-degree = 8
Degree Distribution Example: Power-Law Network


\[
p_k = e^{-m} \cdot \frac{m^k}{k!}
\]

Newman, Strogatz and Watts, 2001

\[
p_k = C \cdot k^{-\tau} e^{-k/k}
\]
Six Degree Separation:
adding long range link, a regular graph can be transformed into a small-world network, in which the average number of degrees between two nodes become small.

from Watts and Strogatz, 1998

C: Clustering Coefficient, L: path length, 
(C(0), L(0)): (C, L) as in a regular graph 
(C(p), L(p)): (C,L) in a Small-world graph with randomness p.
Some examples of Degree Distribution

(a) scientist collaboration: biologists (circle) physicists (square), (b) collaboration of movie actors, (d) network of directors of Fortune 1000 companies
Network size is positively correlated with performance.

- Each person in your email address book at work is associated with $948 dollars in annual revenue.

1 direct contact in a person’s network

$74.07 increase in monthly revenues or $948 annual revenues

Std error = (26.38)***
Significant at p < 0.01
“There is certainly no unanimity on exactly what centrality is or its conceptual foundations, and there is little agreement on the procedure of its measurement.” – Freeman 1979.

Degree (centrality)
Closeness (centrality)
Betweenness (centrality)
Eigenvector (centrality)
Closeness: A vertex is ‘close’ to the other vertices

\[ c_{CI}(v) = \frac{1}{\sum_{u \in V} dist(v,u)} \]

where \( dist(v,u) \) is the geodesic distance between vertices \( v \) and \( u \).
Betweenness ==> Bridges

Example: Healthcare experts in the world

Connections between different divisions

Example: Healthcare experts in the U.S.

Key social bridges
Betweenness measures are aimed at summarizing the extent to which a vertex is located ‘between’ other pairs of vertices.

Freeman’s definition:

\[
C_B(v) = \sum_{s \neq t \neq v \in V} \frac{\sigma(s, t \mid v)}{\sigma(s, t)}
\]

Calculation of all betweenness centralities requires calculating the lengths of shortest paths among all pairs of vertices. Computing the summation in the above definition for each vertex.
Eigenvector Centrality

Try to capture the ‘status’, ‘prestige’, or ‘rank’.
More central the neighbors of a vertex are, the more central the vertex itself is.

\[ c_{Ei}(v) = \alpha \sum_{\{u,v\} \in E} c_{Ei}(u) \]

The vector \( c_{Ei} = (c_{Ei}(1), \ldots, c_{Ei}(N_v))^T \) is the solution of the eigenvalue problem:

\[ \mathbf{A} \cdot c_{Ei} = \alpha^{-1} c_{Ei} \]
PageRank Algorithm (Simplified)
Connectivity of Graph

- A measure related to the flow of information in the graph
- Connected ➔ every vertex is reachable from every other
- A connected component of a graph is a maximally connected subgraph.
- A graph usually has one dominating the others in magnitude ➔ giant component.
- **Reachable**: A vertex $v$ in a graph $G$ is said to be reachable from another vertex $u$ if there exists a walk from $u$ to $v$.

- **Connected**: A graph is said to be connected if every vertex is reachable from every other.

- **Component**: A component of a graph is a maximally connected subgraph.
- **Complete Graph**: every vertex is linked to every other vertex.

- **Clique**: a complete subgraph.
A k-core of a graph $G$ is a subgraph $H$ in which all vertices have degree at least $k$.

Batagelj et. al., 1999. A maximal k-core subgraph may be computed in as little as $O(N_{v} + N_{e})$ time.

Computes the shell indices for every vertex in the graph

Shell index of $v = \text{the largest value, say } c, \text{ such that } v \text{ belongs to the } c\text{-core of } G \text{ but not its } (c+1)\text{-core.}$

For a given vertex, those neighbors with lesser degree lead to a decrease in the potential shell index of that vertex.
The density of a subgraph $H = (V_H, E_H)$ is:

$$den(H) = \frac{|E_H|}{|V_H|(|V_H| - 1) / 2}$$

Range of density

$$0 \leq den(H) \leq 1$$

and

$$den(H) = (|V_H| - 1)\bar{d}(H)$$

average degree of H
Clustering coefficient

A triangle is a complete subgraph of order three.
A connected triple is a subgraph of three vertices connected by two edges (regardless how the other two nodes connect).
The local clustering coefficient can be expressed as:

\[ \text{den}(H_v) = cl(v) = \frac{\tau_\Delta(v)}{\tau_3(v)} \]

The clustering coefficient of \( G \) is then:

\[ cl(G) = \frac{1}{V'} \sum_{v \in V'} cl(v) \]

Where \( V' \subseteq V \) is the set of vertices \( v \) with \( d_v \geq 2 \).
Which one is better -- Structural Diverse Networks or Cohesive Networks?

- Structural diverse networks with abundance of structural holes are associated with higher performance.
  
  - When friends of your friends are not friends of each other or belong to the same social group.

1 standard deviation increase in network diversity
Diversity = (1-constraint)

276.64 \% increase in monthly revenues
Std error = (113.88)
Significant at p < 0.01
Studied 2,038 global practitioners for 2 years by Prof. Wu, Wharton School, U Penn.

Controlled factors: temporal shocks, individual characteristics such as job roles and hierarchies, and the characteristics of each project including line of business and the region.

Compare individual performance == billable revenue since adopting SmallBlue.

We saw a revenue of $584.15 per month == $7,010 in a year
ROI – Sales opportunities & signings as a result of using SmallBlue SNA tool

Based on 324 random surveys in 2011 (0.35% sampling rate):

• $40M in unqualified sales opportunities identified
• $9M in identified sales based on unqualified opportunities
• 5% of the sample pool of survey respondents realized revenue
• All achieved through using SmallBlue

S&T Sr. Managing Consultant, US
… I was able to turn a 2 minute hallway conversation with my client into a real OpenPages Compliance Solution opportunity!

S&D Practitioner, India
… multiple number of times I tracked down assets across SWG to develop value propositions and connect with the right expert

Studied by IBM
Global Business Services