Columbia University

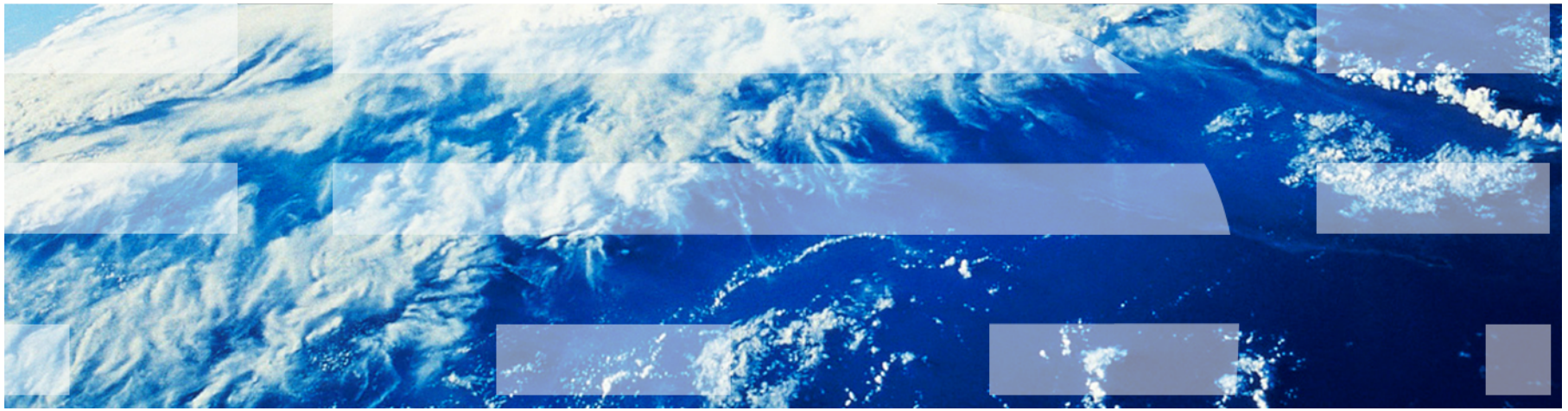
E6893 Big Data Analytics Lecture 9:

Linked Big Data — Graph Computing (II)

Ching-Yung Lin, Ph.D.

Adjunct Professor, Dept. of Electrical Engineering and Computer Science

IBM Chief Scientist, Graph Computing



November 6th, 2015

Final Project Proposal (First) Presentation

Columbia University

Date/Time: November 19, 7pm - 9:30pm

Each Team — about 3 mins:

1. Team members and expected contributions of each member;
2. Motivation of your project (The problem you would like to solve);
3. Dataset, algorithm, and tools for your project;
4. Current Status of your project.

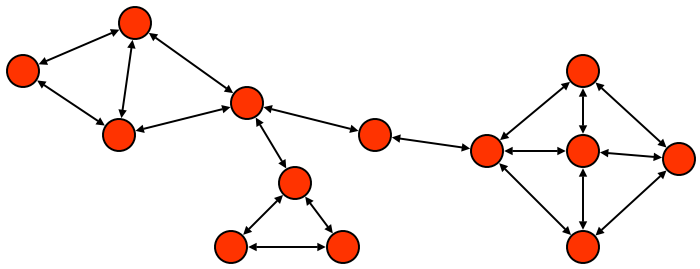
Please update your team info in the Project webpage by November 13. The presentation schedule will be announced on November 17.

The website will be opened to allow you upload your slides by November 19.

If a project is purely by CVN students, please submit your slides without oral presentation.

- A graph:

$$G = (V, E)$$



- V = Vertices or Nodes
- E = Edges or Links

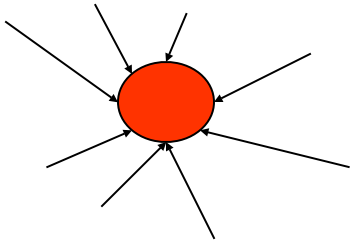
- The number of vertices: "Order"

$$N_v = |V|$$

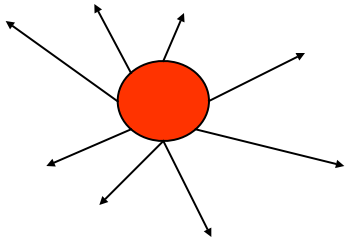
$$N_e = |E|$$

In-degrees and out-degrees

- For Directed graphs:



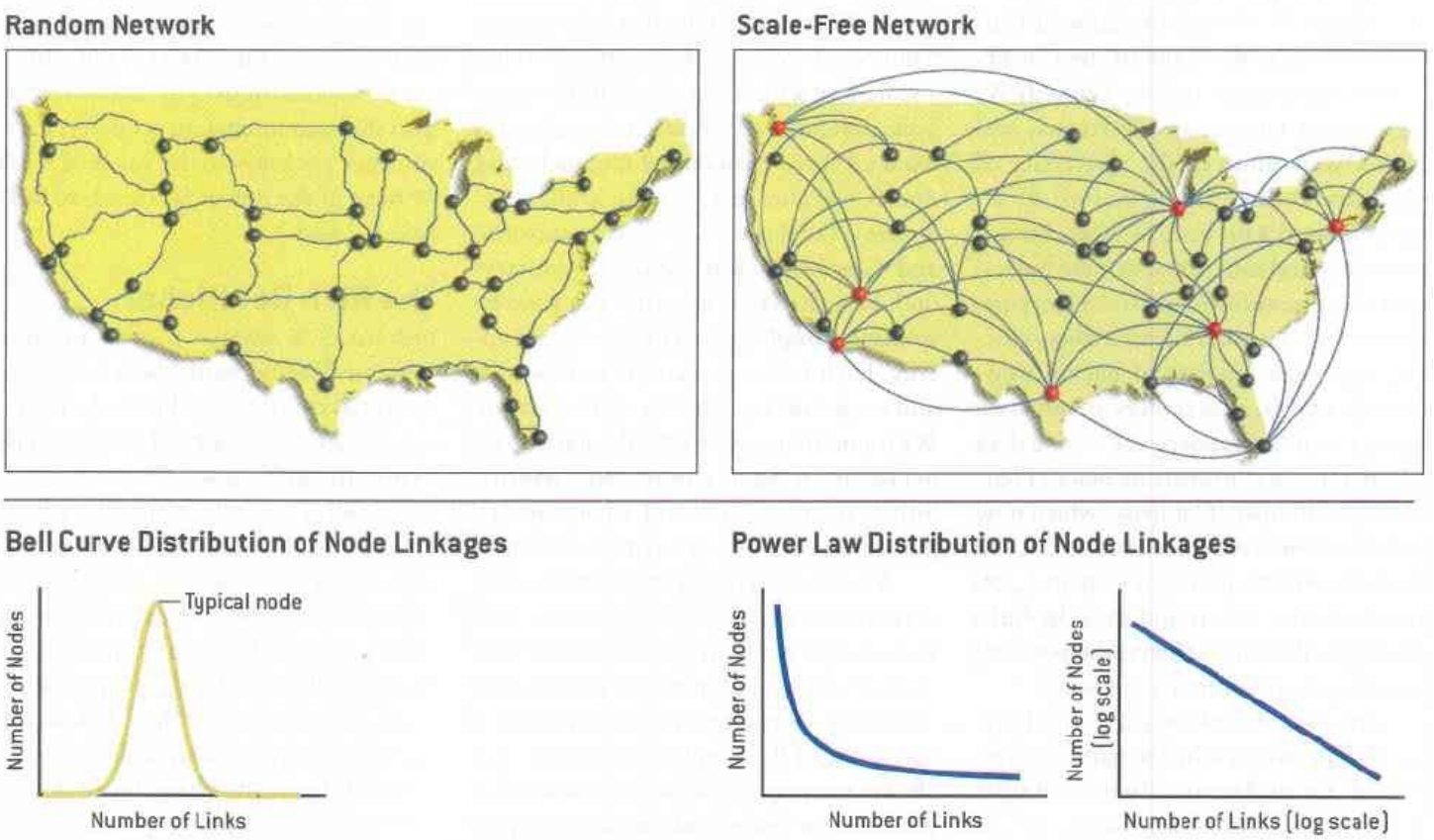
In-degree = 8



Out-degree = 8

Degree Distribution Example: Power-Law Network

A. Barbasi and E. Bonabeau, "Scale-free Networks", Scientific American 288: p.50-59, 2003.



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

$$p_k = C \cdot k^{-\tau} e^{-k/\kappa}$$

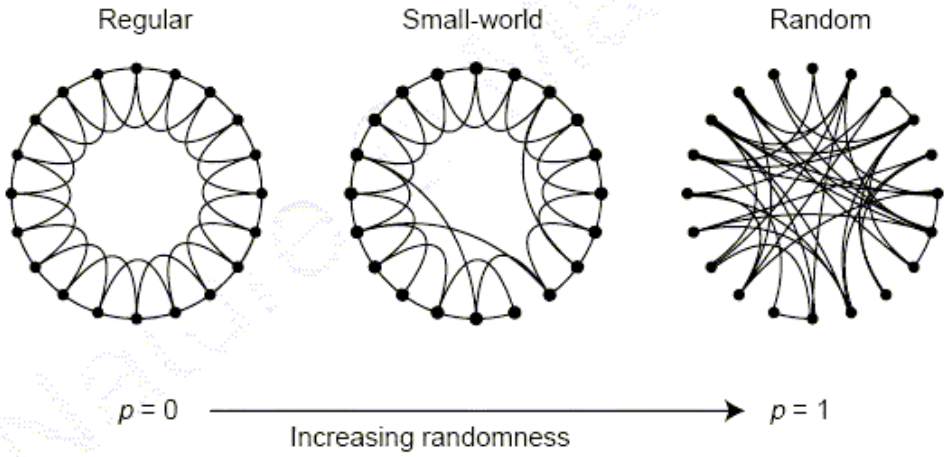
Newman, Strogatz and Watts, 2001

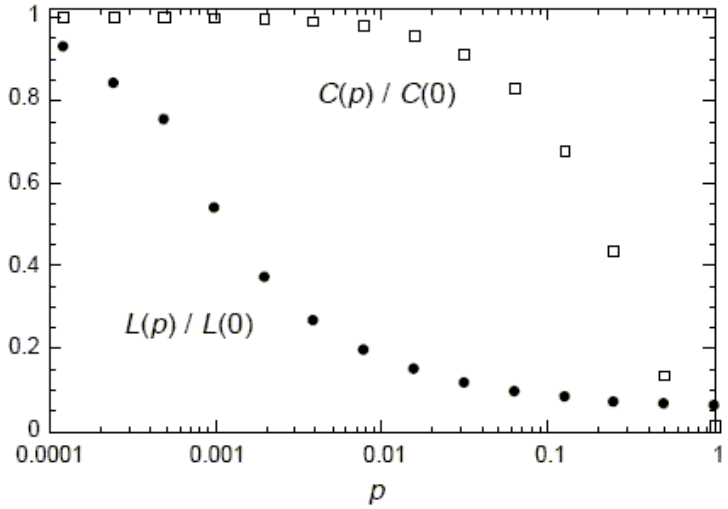
Another example of complex network: Small-World Network

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

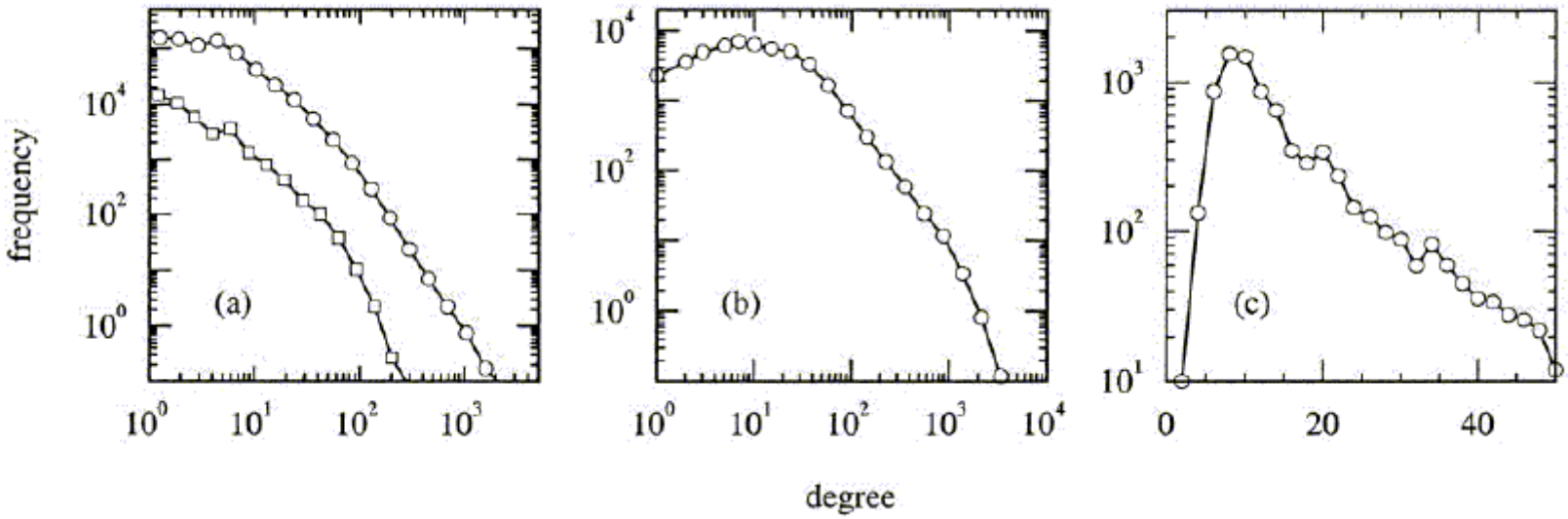




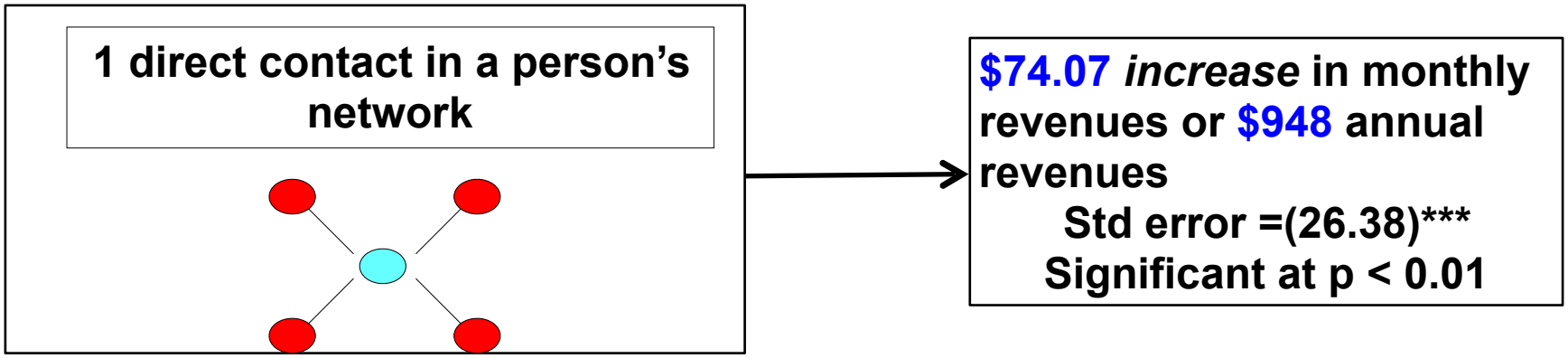
C: Clustering Coefficient, L: path length,
(C(0), L(0)): (C, L) as in a regular graph
(C(p), L(p)): (C,L) in a Small-world graph with randomness p.

Some examples of Degree Distribution

(a) scientist collaboration: biologists (circle) physicists (square), (b) collaboration of move 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.



Centrality

"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)

Betweeness (centrality)

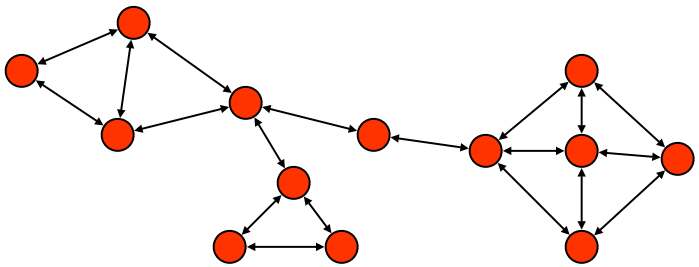
Eigenvector (centrality)

Closeness

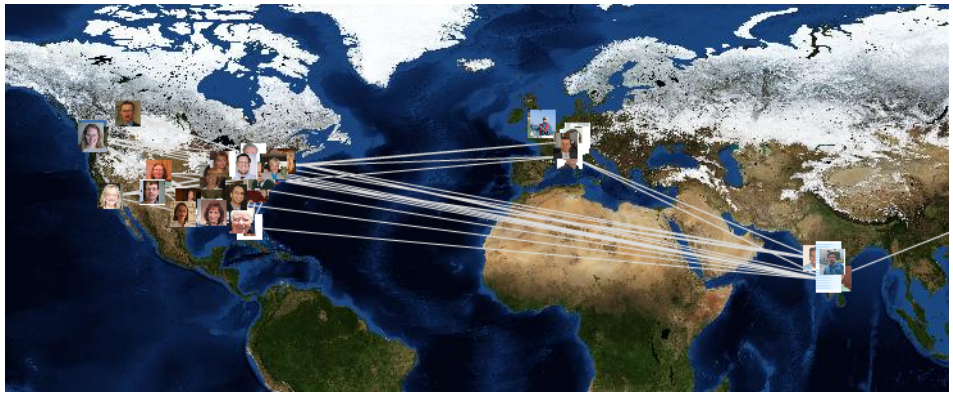
Closeness: A vertex is ‘close’ to the other vertices

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

where dist(v,u) is the geodesic distance between vertices v and u.

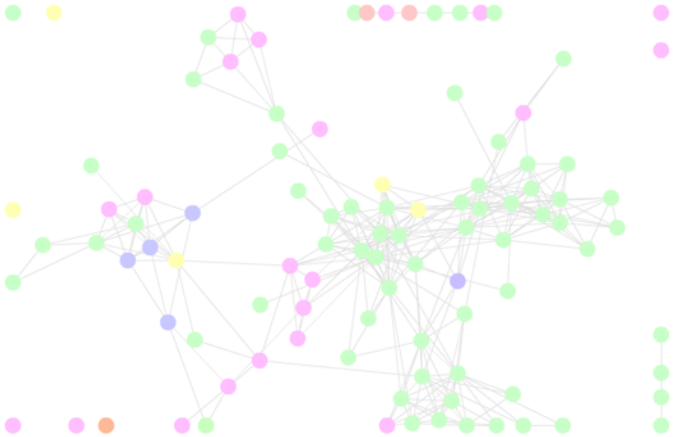


Betweenness ==> Bridges

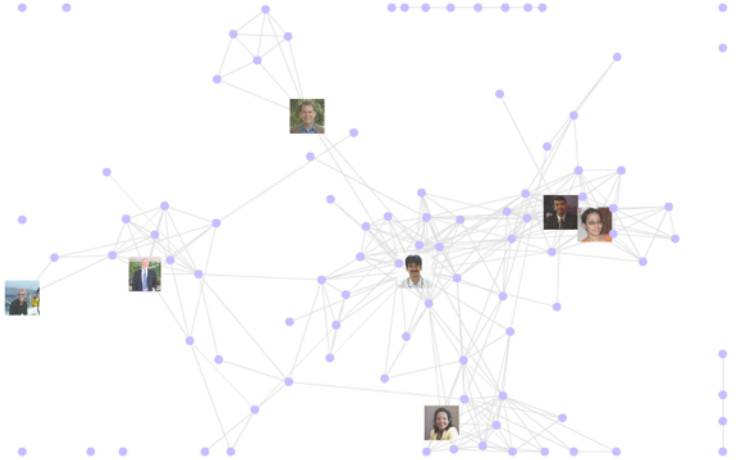


Example: Healthcare experts in the world

Example: Healthcare experts in the U.S.



Connections between different divisions



Key social bridges

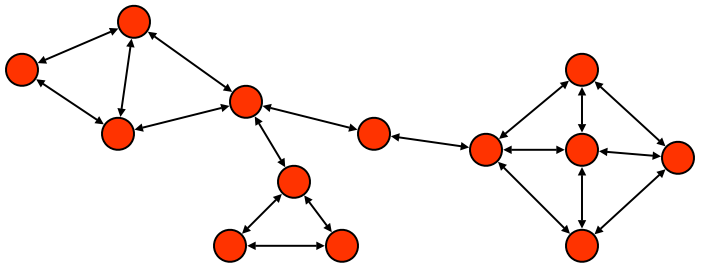
Betweenness

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

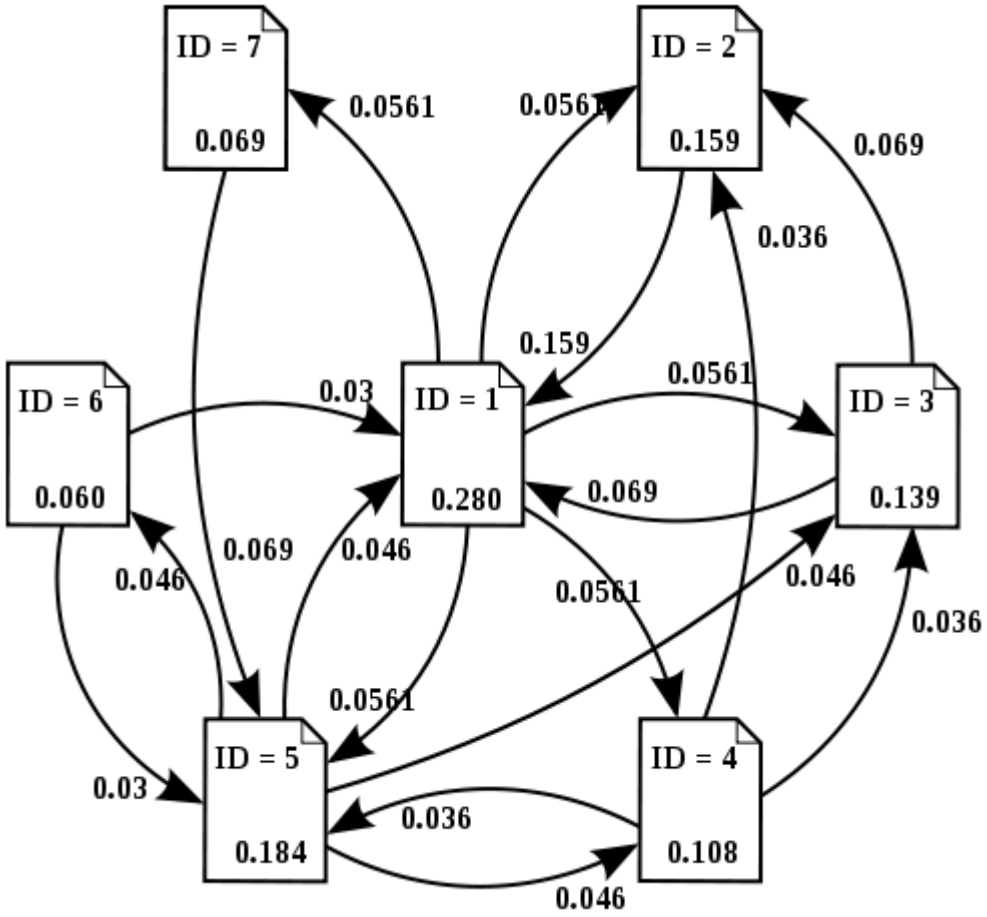
Try to capture the 'status', 'prestige', or 'rank'.
More central the neighbors of a vertex are, the more central the vertex itself is.

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

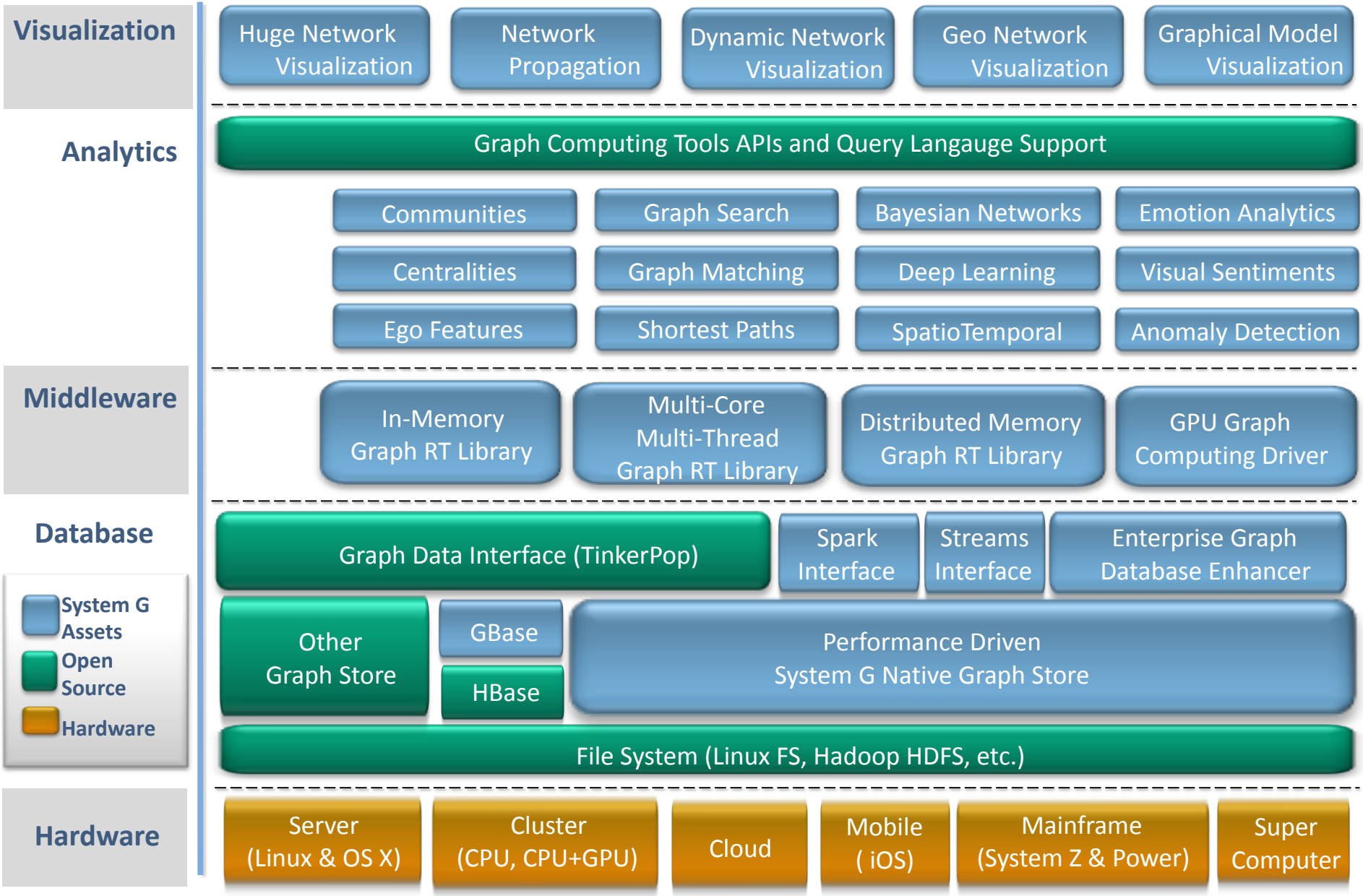
The vector $\mathbf{c}_{Ei} = (c_{Ei}(1), \ldots, c_{Ei}(N_v))^T$ is the solution of the eigenvalue problem:

$$\mathbf{A} \cdot \mathbf{c}_{Ei} = \alpha^{-1} \mathbf{c}_{Ei}$$

PageRank Algorithm (Simplified)



IBM System G Graph Computing Tools



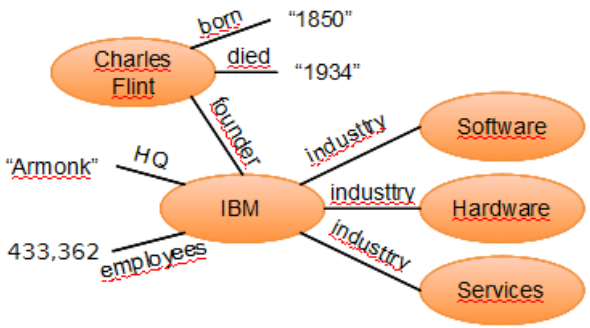
System G — Graph Computing as an Intelligence Machine Machine

COLUMBIA UNIVERSITY

Graph Database

Property Graph

Memory



subject	predicate	object
Charles Flint	born	"1850"
Charles Flint	died	"1934"
Charles Flint	founder	IBM
IBM	HQ	"Armonk"
IBM	employees	433,362
IBM	industry	Software
IBM	industry	Hardware
IBM	industry	Services

Related Information

Graph Analytics

Relation Graph

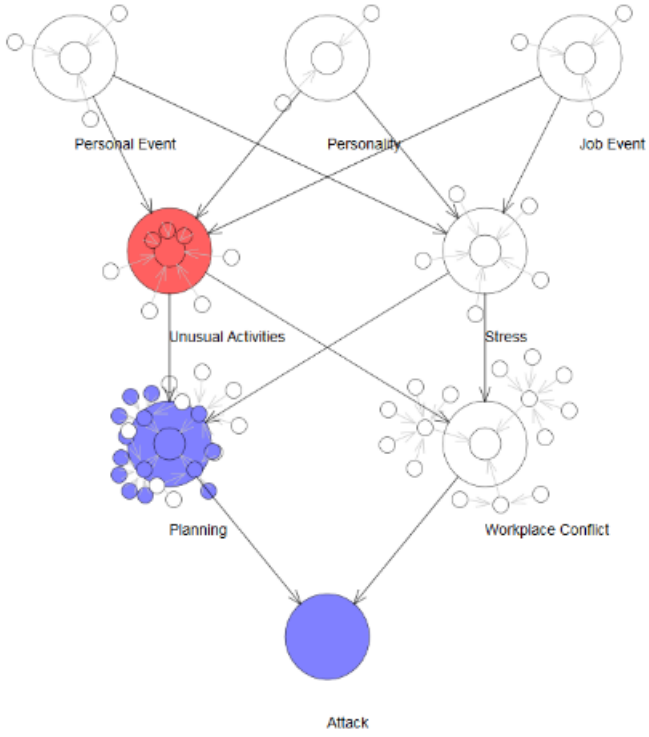
Perception

Contextual Analysis

Graphical Models

Reasoning Graph

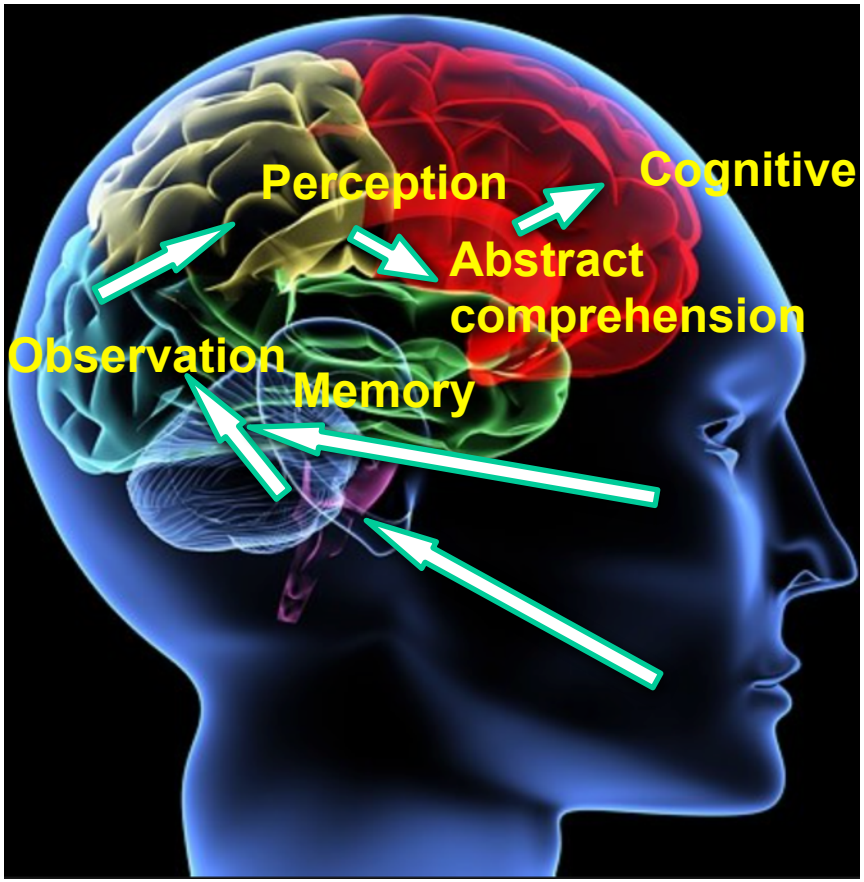
Intelligence

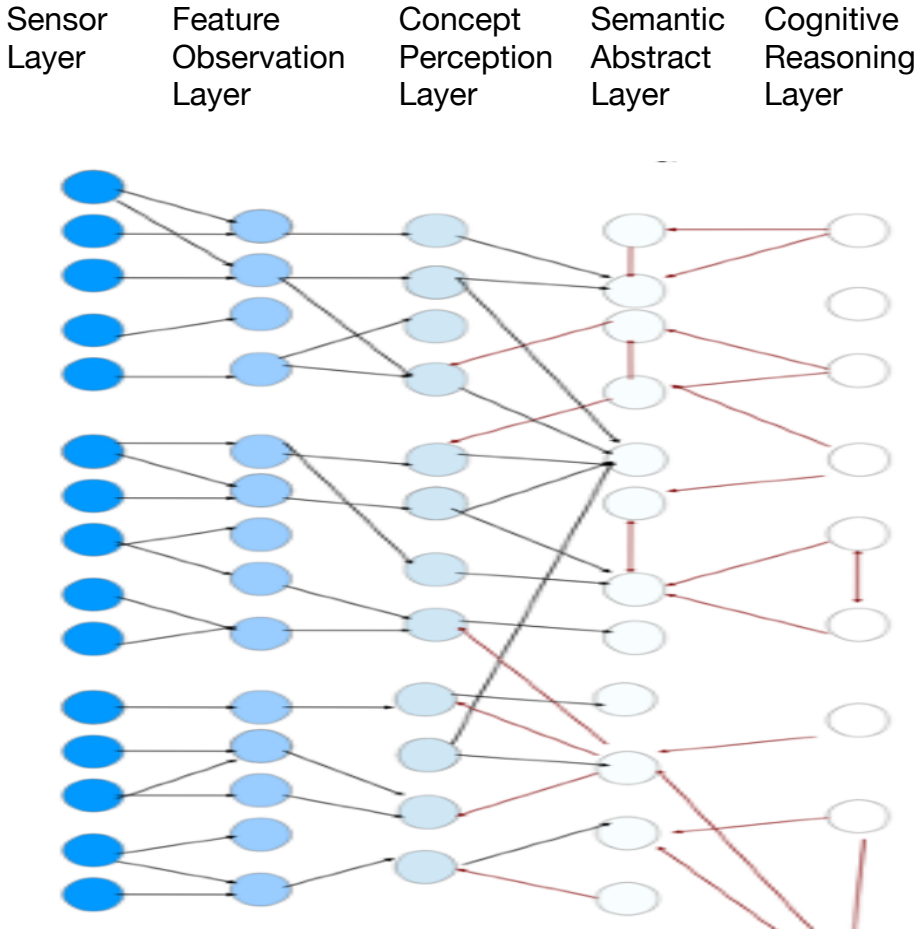


Machine Reasoning & Deep Learning

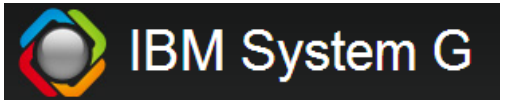
http://systemg.research.ibm.com

Machine Reasoning





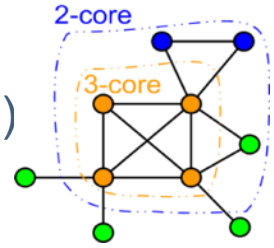
IBM System G Application Use Cases



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
16. System G for Code Life Cycle Management
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

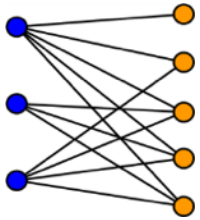
System G Graph Analytical Tools

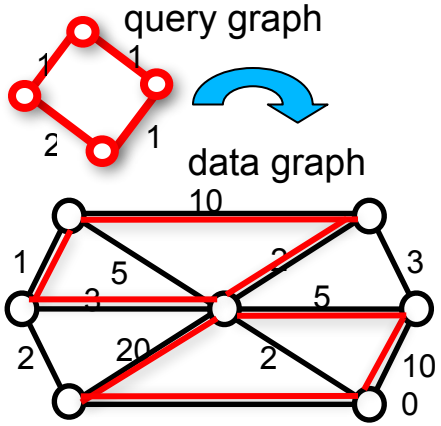
- ***Network topological analysis*** tools
  - •Centralities (degree, closeness, betweenness)
  - •PageRank
  - •Communities (connected component, K-core, triangle count, clustering coefficient)
  - •Neighborhood (egonet, K-neighborhood)
- ***Graph matching and search*** tools
  - •Graph search/filter 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



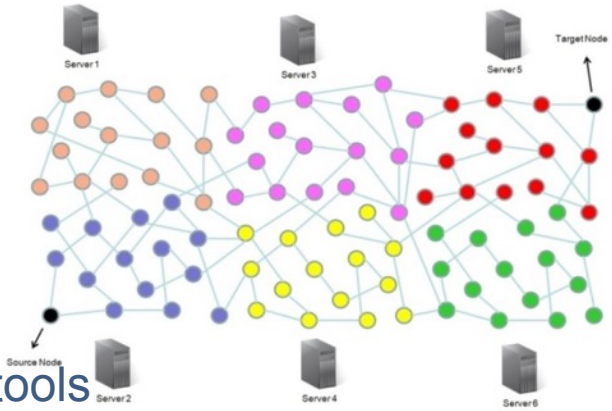
K-core

K-neighborhood


Collaborative filtering
Bipartite weighted graph matching



Graph matching



Top k-shortest paths

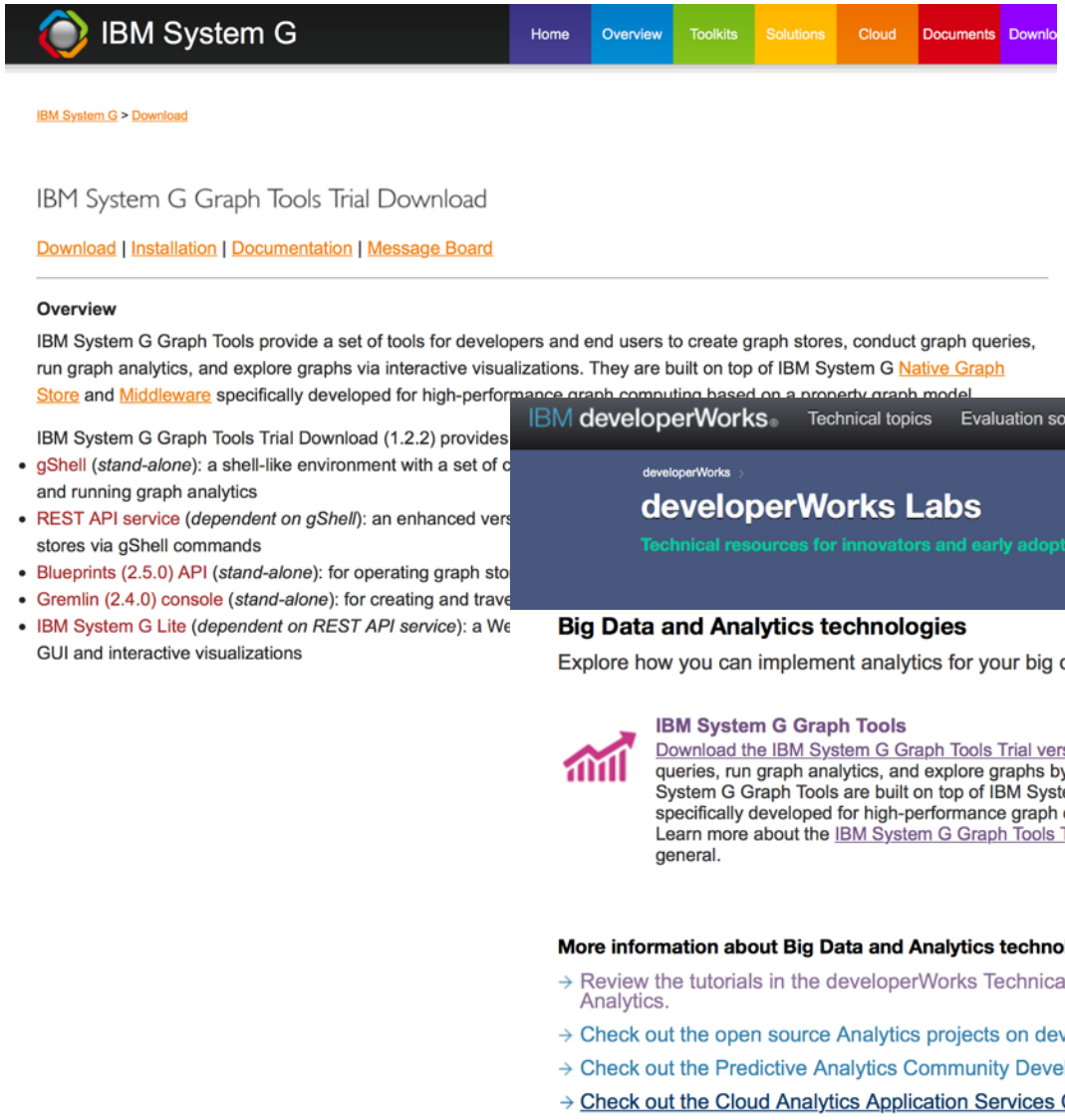

Bayesian network inference

Characteristics of IBM System G Graph Analytics

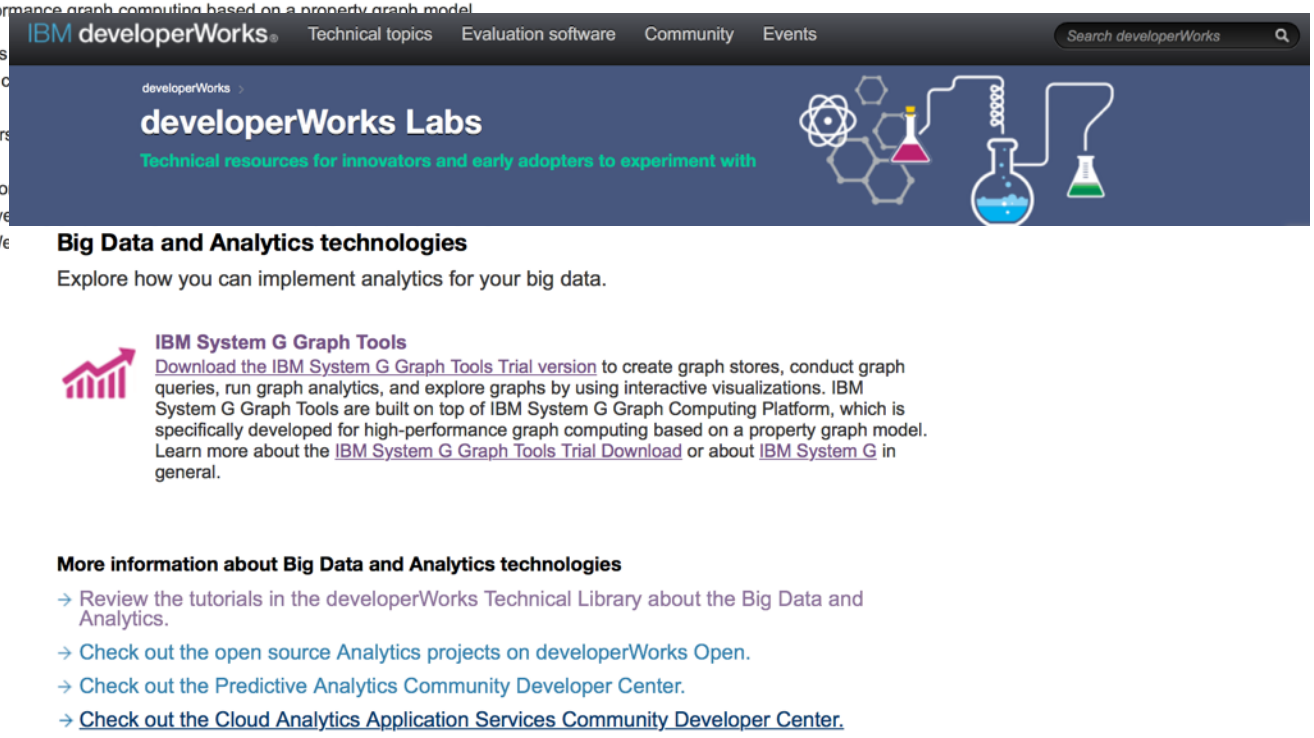
- Cover a wide range of graph analytics to support many application use cases in different domains, e.g.:
  - Enterprise social network analysis, expertise search, knowledge recommendation
  - Financial/security anomaly/fraud detection
  - Social media monitoring and analysis
  - Cellular network analytics 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

Download IBM System G Standard Edition (on-premise)

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



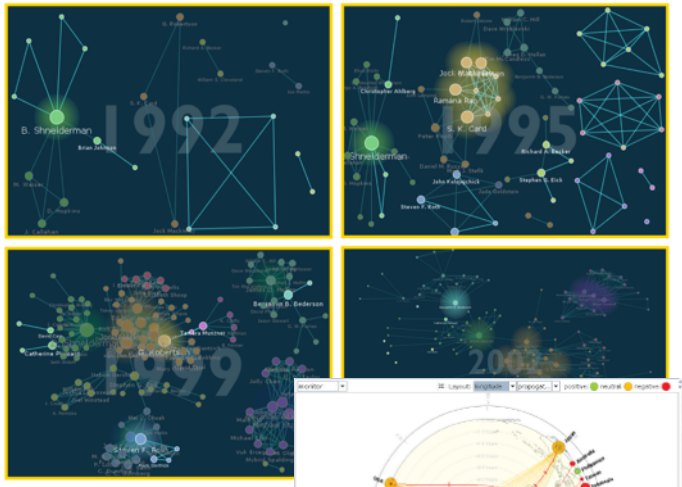
or
http://www.ibm.com/developerworks/labs/

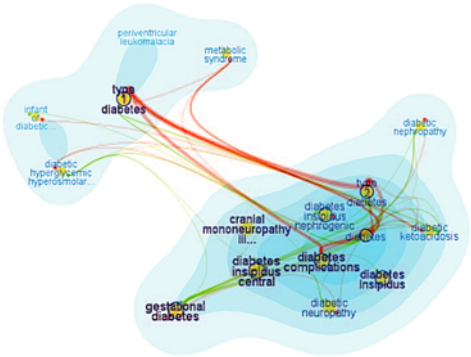


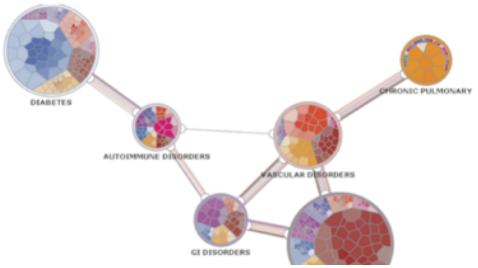
Existing foundation of 16 types of graph visualization assets in these 4 categories:

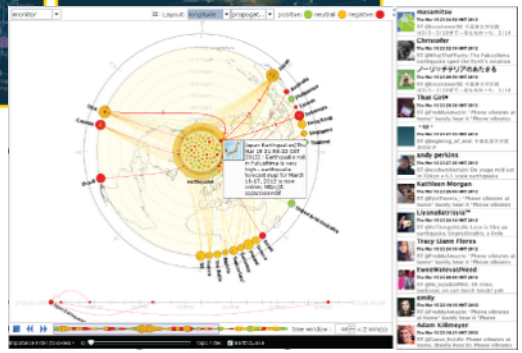
- **Multivariate Graphs**: nodes and edges have multivariate attributes. E.g., healthcare graphs, workflow graphs, behavior reasoning graphs, etc.
- **Heterogeneous Graphs**: graphs in which nodes and edges are in different categories and types. E.g.: bipartite/tripartite/multi-partite graphs, geospatial 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.

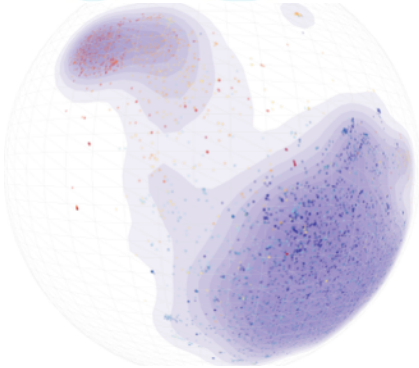
http://systemg.ibm.com

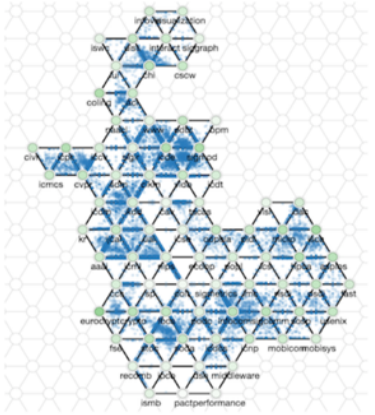












Web-based:
HTML5
WebGL

Resources

- IBM System G on Bluemix (need registration)
  - http://systemg.mybluemix.net
- IBM System G Graph Analytics Overview
  - http://systemg.research.ibm.com/analytics.html
- IBM System G Graph Tools Trial Download
  - http://systemg.research.ibm.com/download.html
- IBM System G Graph Tools Installation Guide and Documentation
  - http://systemg.research.ibm.com/setup.html

IBM System G Graph Tools - Overview

Columbia University

Overview

IBM System G Graph Tools provide a set of tools for developers and end users to create graph stores, conduct graph queries, run graph analytics, and explore graphs via interactive visualizations. They are built on top of IBM System G Native Graph Store and Middleware specifically developed for high-performance graph computing based on a property graph model.

IBM System G Graph Tools Trial Download (1.2.2) provides the following tools *in a single-node environment*:

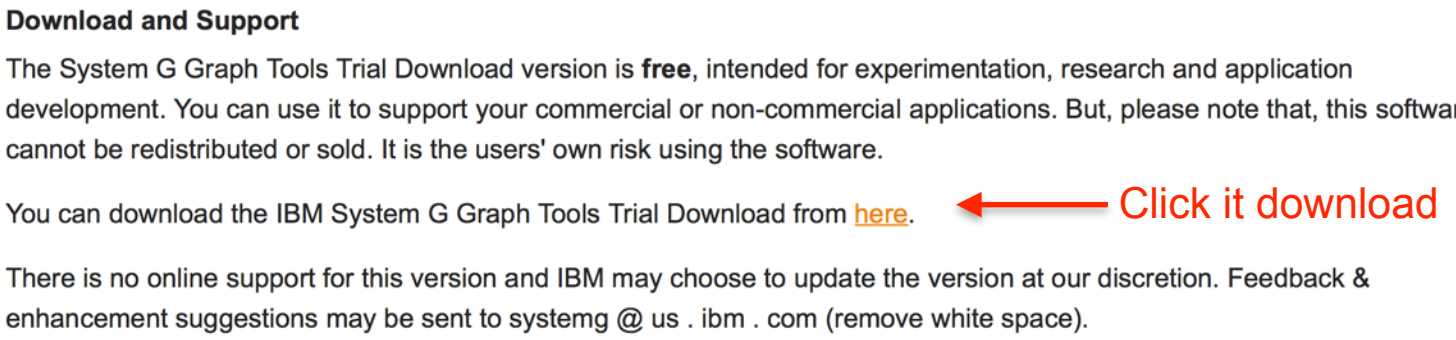
- gShell (*stand-alone*): a shell-like environment with a set of commands for creating, updating, querying multiple graph stores and running graph analytics
- REST API service (*dependent on gShell*): an enhanced version of TinkerPop 2 Rexster Basic REST API to operate graph stores via gShell commands
- Python interface to gShell: Python wrappers of all gShell commands
- Blueprints (2.5.0) API (*stand-alone*): for operating graph stores in Java applications
- Gremlin (2.4.0) console (*stand-alone*): for creating and traversing graphs using the Gremlin-Groovy graph query language
- IBM System G Lite (*dependent on REST API service*): a Web application for creating, querying, and exploring graphs through GUI and interactive visualizations

Packages for CentOS (Redhat) 6.5, Ubuntu 14.04, Power 8 and Mac OS X are currently available.

Note: A new version (1.3.0) will be available on Friday 11/6 or Monday 11/9. HW#3 will be delayed to 11/19.

IBM System G Graph Tools - Download

1. Download IBM System G Graph Tools, from http://systemg.research.ibm.com/download.html
2. In the middle of webpage, find download and support section



Download and Support

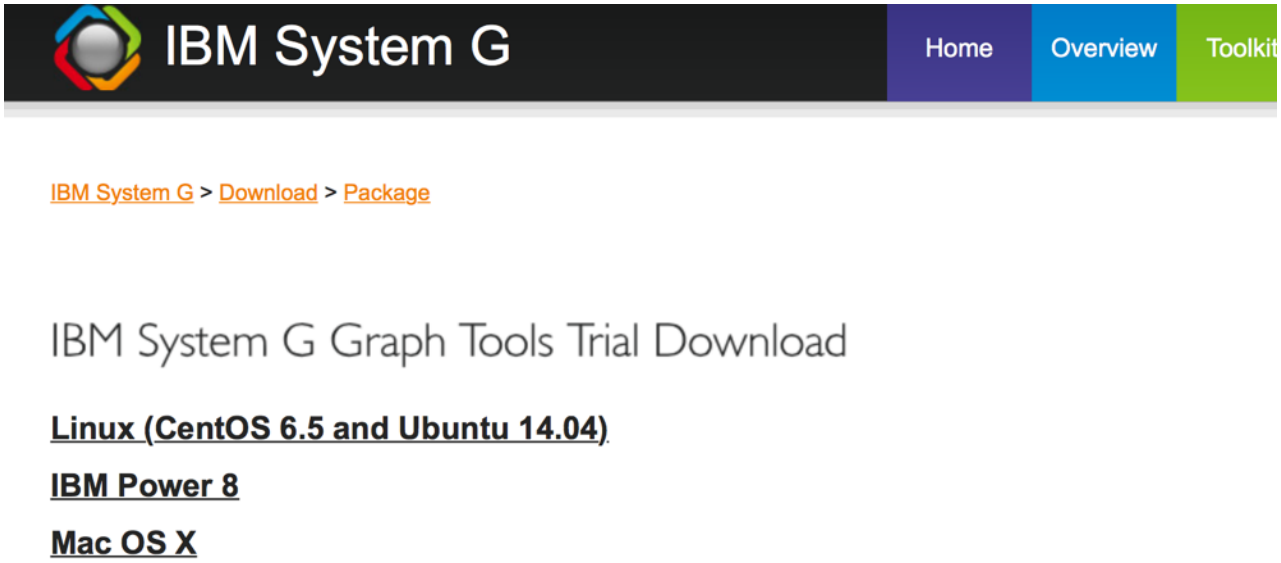
The System G Graph Tools Trial Download version is **free**, intended for experimentation, research and application development. You can use it to support your commercial or non-commercial applications. But, please note that, this software cannot be redistributed or sold. It is the users' own risk using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

IBM System G Graph Tools - Download

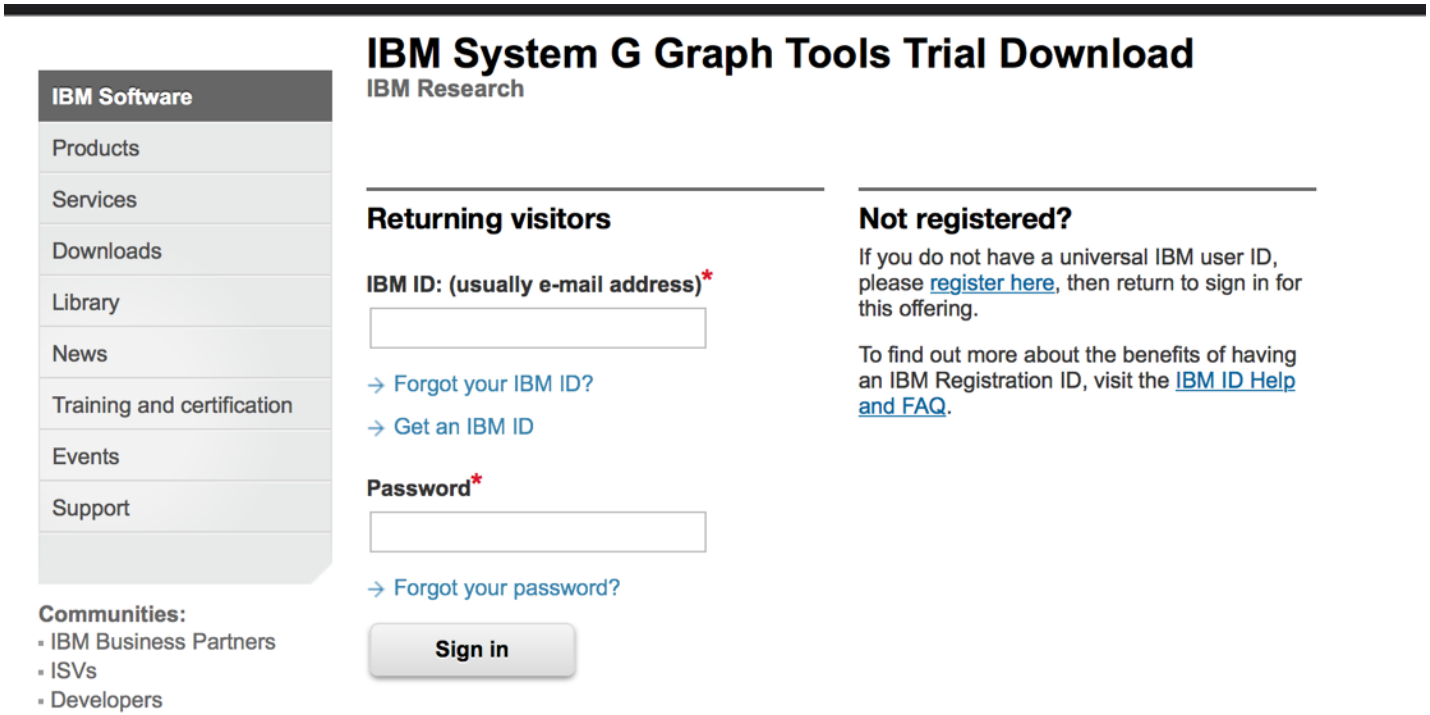
This is temporary page, you can download directly.



IBM System G Graph Tools - Download

Download from IBM developerWorks:

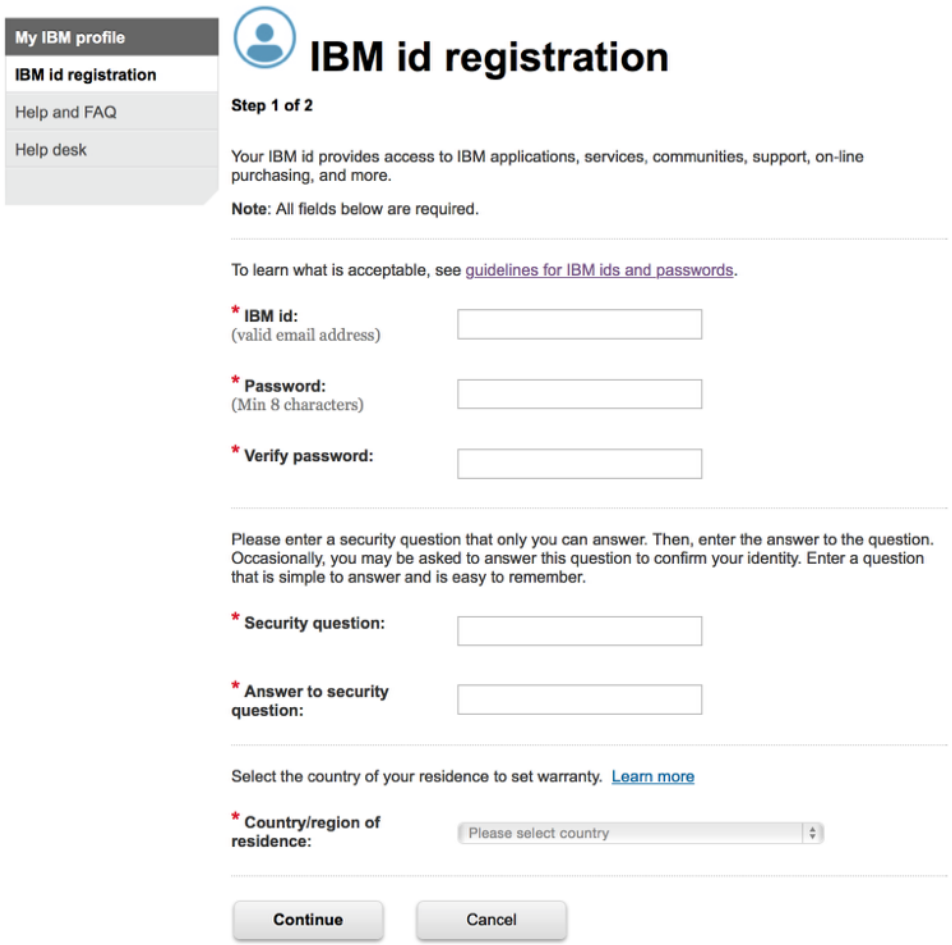
1. You need a **IBM id** to download the package
2. If you have, just log in, or register one

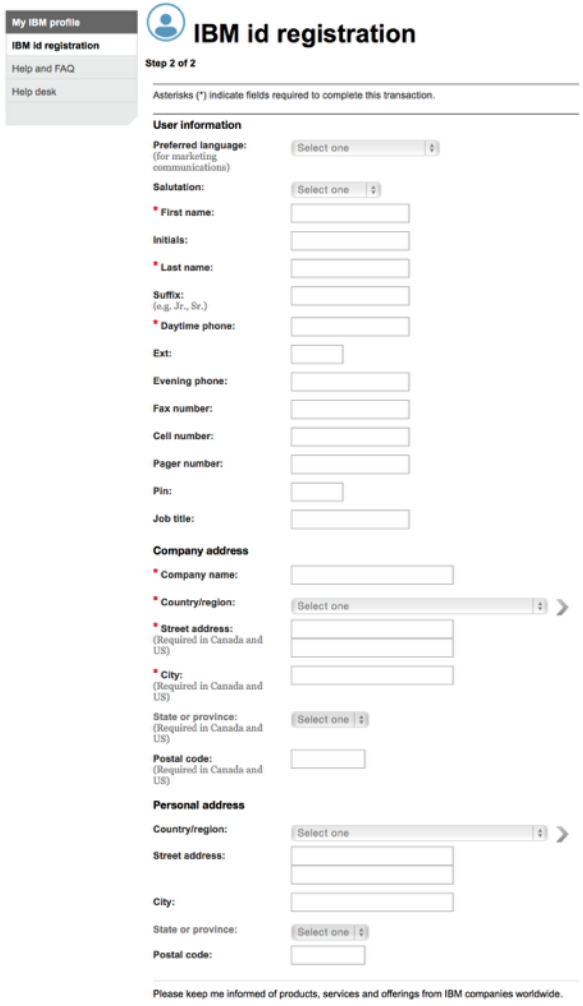


IBM System G Graph Tools - Download

IBM id registration page:

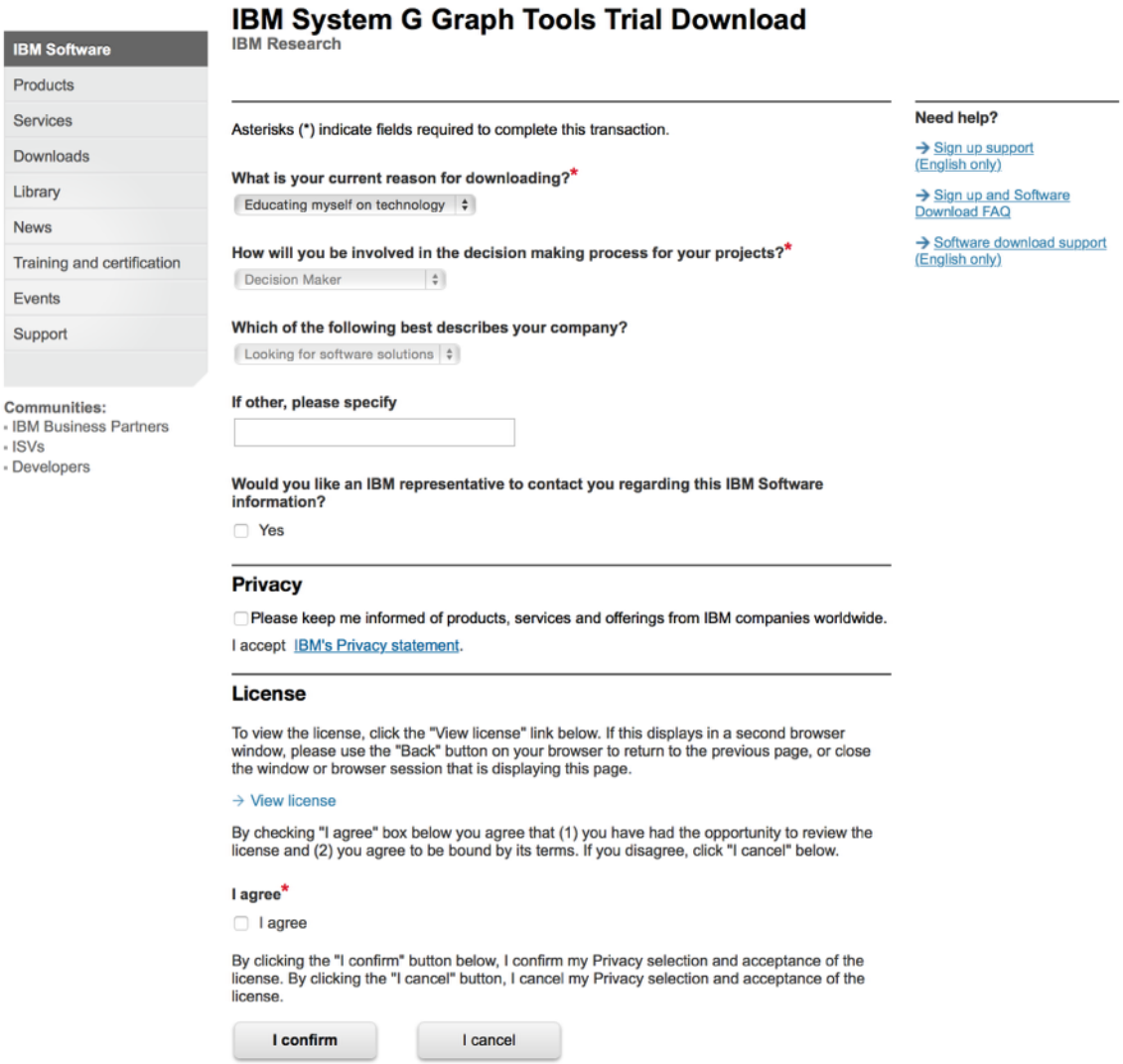
1. Fill out the form to get your IBM id





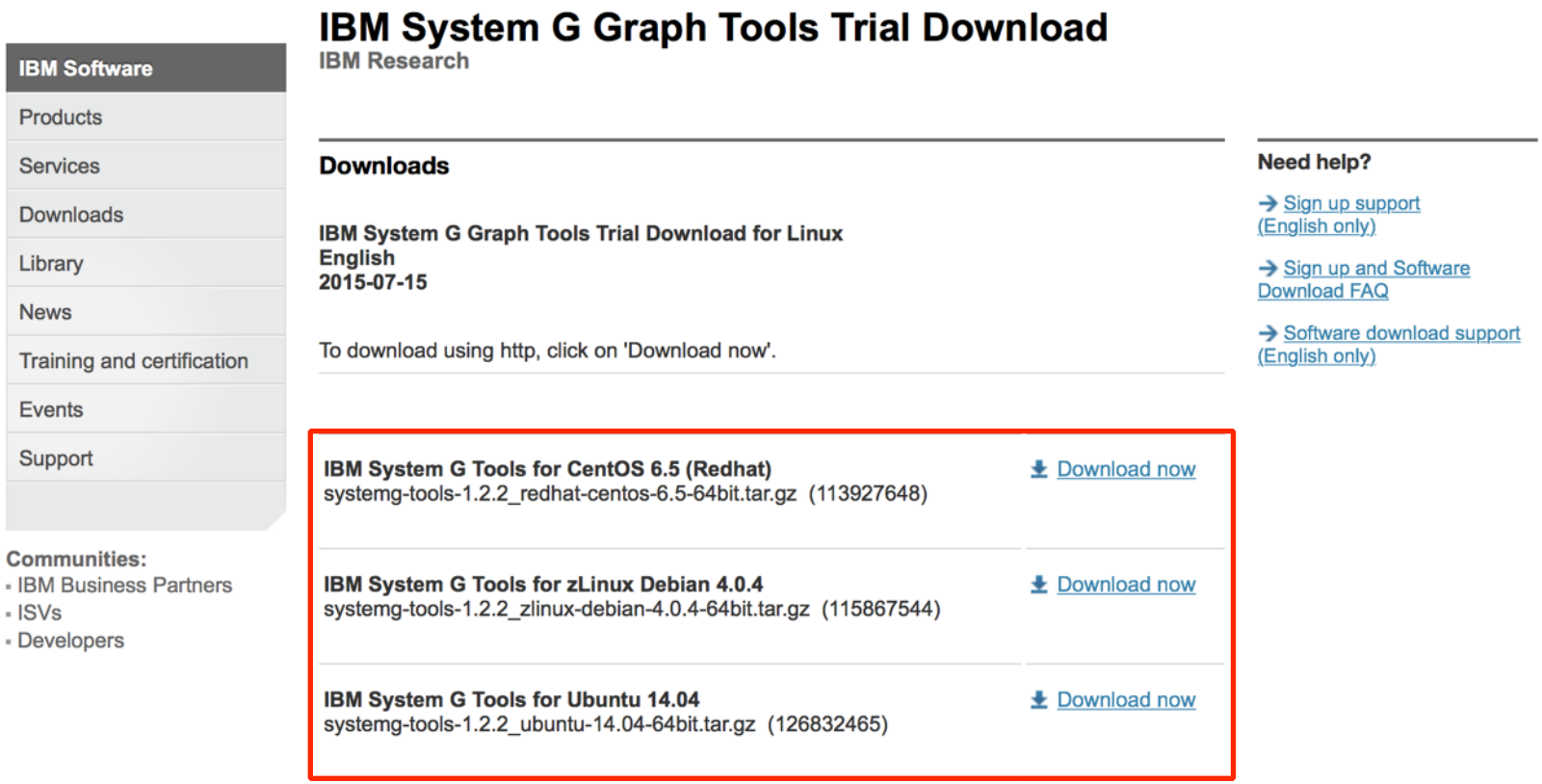
IBM System G Graph Tools - Download

Accept usage terms to download IBM System G Graph Tools



IBM System G Graph Tools - Download

This is old page, now, we support Linux (CentOS and Ubuntu), Mac OS (Yosemite), and IBM Power.



This is old version, new release is **1.3.0**
(will be available on developer work soon)

IBM System G Graph Tools - Installation

Columbia University

All following instructions are used for Ubuntu 14.04, installation of some packages might be varied based on your OS.

Installation

After downloading the package, simply untar the .tar.gz file (`tar -xvzf`), then follow a few steps here to set up the tools in your environment.

IBM System G Graph Tools Setup

After downloading the .tar.gz package file, use `tar -xvzf` to untar it. The unpacked directory contains the following files/sub-directories:

- `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

Setup instructions: http://systemg.research.ibm.com/setup.html

IBM System G Graph Tools - Installation

Columbia University

Package step

Set path

Most components of IBM System G require LD_LIBRARY_PATH to locate the library files needed for execution. To get started, first open `systemg.sh` to edit the path of `<systemg_home_path>` (the absolute path to the unpacked directory) according to the setup in your environment, then copy the file to `/etc/profile.d/`. You need to log out and back in again for the script to be executed to set the path.

If there isn't `/etc/profile.d/` in your environment, do the following before copying `systemg.sh`:

1. Create `/etc/profile.d/`, e.g. `sudo mkdir /etc/profile.d`.
2. Edit `/etc/profile` to add the following to the bottom:

```
for sh in /etc/profile.d/*.sh ; do
  [ -r "$sh" ] && . "$sh"
done
unset sh
```

3. Log out and back in.

If you don't have permission to put `systemg.sh` under `/etc/profile.d/`, the command listed in `systemg.sh` needs to be executed manually to set `LD_LIBRARY_PATH` properly.

You are ready to go for gShell with command line and Python interface.

Columbia University

gShell

gShell is a shell-like environment on top of IBM System G Graph Store. It allows users to create, update and query multiple graph stores simultaneously and supports both directed and undirected graphs.

Interactive mode

```
./gShell interactive
```

gShell displays a prompt >> waiting for input commands. To exit, type `close_all` or `Ctrl-c`.

Batch mode

```
./gShell interactive < <batch_file> (e.g., ./gShell interactive < script.txt)
```

The batch mode is a variant of the interactive mode, where the commands stored in a text file `<batch_file>` are redirected to the interactive mode (using <) and executed sequentially. **Note that `close_all` needs to be included as the last command in the file in order to exit from gShell.**

All gShell commands share the same format:

```
command --arg1 value1 --arg2 value2 ...
```

Online documentation: http://systemg.research.ibm.com/doc/gshell.html

IBM System G Graph Tools - gShell

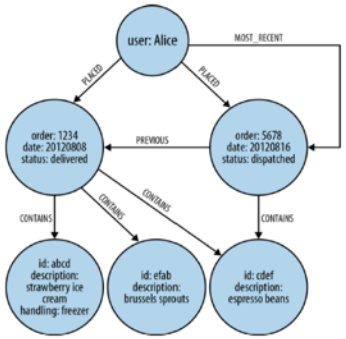
./gShell interactive

```
Yinglongs-MacBook-Pro:gshell yxia ./gShell  interactive
gShell>>
add_edge                        add_vertex                      add_vertex_json
analytic_auction                analytic_betweennes
analytic_closeness_centrality   analytic_clustering
analytic_connected_component    analytic_degree_cen
analytic_k_core                 analytic_k_shortest
analytic_reset_engine           analytic_shortest_p
analytic_stop_engine            analytic_triangle_c
delete_eprop                    delete_vertex
export_csv                      filter_edges
find_edge                       find_multiple_verti
find_random_edges               find_random_vertice
find_vertex_max_degree          get_egonet
get_num_vertices                get_subgraph
indexer_clucene                 indexer_leveldb
load_csv_vertices               print_all
update_edge                     update_vertex
close                           close_all
delete                          create
help                            version
gShell>>
```

```
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 infomation"}
]
}
```

double click "**tab**" to show possible auto-completion

IBM System G Graph Tools - gShell



Create a simple graph:

- ./gShell interactive
- delete --graph g
- create --graph g --type directed

- add_vertex --graph g --id Alice --label user
- add_vertex --graph g --id 1234 --label order --propwithtype date:string:20120808 status:string:delivered
- add_vertex --graph g --id 5678 --label order --propwithtype date:string:20120816 status:string:delivered
- add_vertex --graph g --id abcd --label item --prop description:"strawberry ice cream" handling:freezer
- add_vertex --graph g --id efab --label item --prop description:"brussels sprouts”
- add_vertex --graph g --id cdef --label item --prop description:"expresso beans”

- add_edge --graph g --src Alice --targ 1234 --edgelabel _ --prop type:PLACED
- add_edge --graph g --src Alice --targ 5678 --edgelabel _ --prop type:PLACED time:MOST_RECENT
- add_edge --graph g --src 5678 --targ 1234 --edgelabel _ --prop time:PREVIOUS
- add_edge --graph g --src 1234 --targ abcd --edgelabel _ --prop type:CONTAINS
- add_edge --graph g --src 1234 --targ efab --edgelabel _ --prop type:CONTAINS
- add_edge --graph g --src 1234 --targ cdef --edgelabel _ --prop type:CONTAINS
- add_edge --graph g --src 5678 --targ cdef --edgelabel _ --prop type:CONTAINS
- close_all

gShell with Python interface

1. Download and install the respective version of Python if it is not already available in your environment.
2. Copy the files from the sub-directory corresponding to the chosen Python version (i.e. `python2.6`, `python2.7`, `python3.4`) to the parent directory `python/`.
3. Make sure `LD_LIBRARY_PATH` is set properly by following the instruction in the "Set path" section above.
4. Test if the setup is successful by typing:

```
$ cd python
$ python -c "import py_gShell"
```

   If nothing is returned, the setup is successful. Otherwise, make sure that both `_py_gShell.so` and `py_gShell.py` are copied to the `python/` directory.
5. Run the test scripts provided.
6. Put "`from py_gShell import _py_gshell as gShell`" in your Python code, and you are good to go!

Online documentation: http://systemg.research.ibm.com/doc/python.html

IBM System G Graph Tools - gShell with Python interface

```
#!/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

g.analytic_find_path(src="1",sink="2")

```
{
    "paths": [
        {
            "src": "1",
            "path": "1-->2",
            "sink": "2",
            "distance": 1.0
        }
    ],
    "time": [
        {
            "TIME": "3.31402e-05"
        }
    ]
}
```

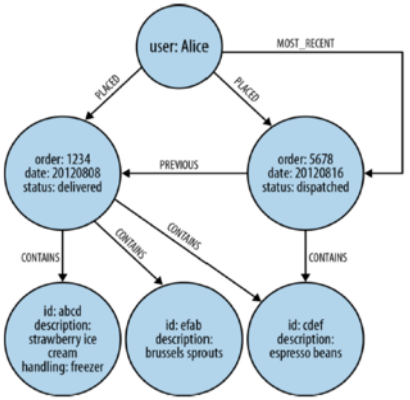
g.analytic_find_path(src="1",sink="2",label="b")

```
{
    "paths": [
        {
            "src": "1",
            "path": "1-->3-->5-->2",
            "sink": "2",
            "distance": 3.0
        }
    ],
    "time": [
        {
            "TIME": "2.09808e-05"
        }
    ]
}
```

IBM System G Graph Tools - gShell with Python interface

Columbia University

Create a simple graph:



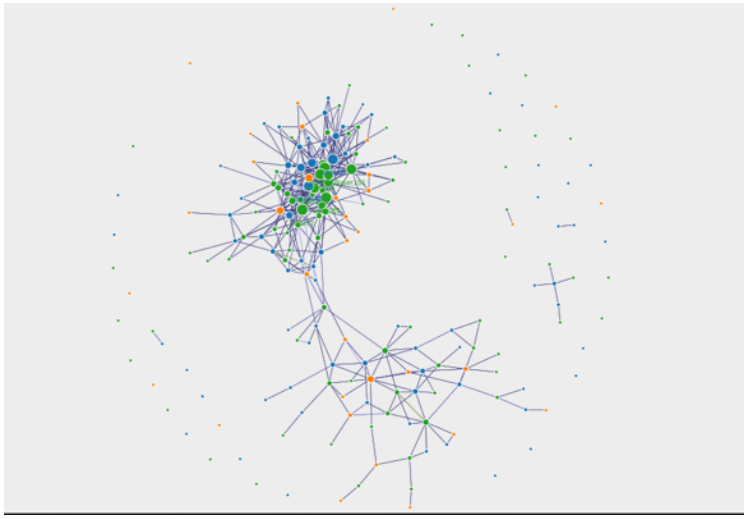

```
from py_gShell import _py_gshell as gShell
import json
g = gShell()
g.delete_graph("g")
g.create_graph("g", graph_type="directed")

g.add_vertex(vertex_id="Alice", label="user")
g.add_vertex(vertex_id="1234", label="order", prop={"date":"20120808", "status":"delivered"})
g.add_vertex(vertex_id="5678", label="order", prop={"date":"20120816", "status":"delivered"})
g.add_vertex(vertex_id="abcd", label="item", prop={"description":"strawberry ice cream", "handling":"freezer"})
g.add_vertex(vertex_id="efab", label="item", prop={"description":"brussels sprouts"})
g.add_vertex(vertex_id="cdef", label="item", prop={"description":"expresso beans"})

g.add_edge(src="Alice", targ="1234", edgelabel="_", prop={"type":"PLACED"})
g.add_edge(src="Alice", targ="5678", edgelabel="_", prop={"type":"PLACED","time":"MOST_RECENT"})
g.add_edge(src="5678", targ="1234", edgelabel="_", prop={"time":"PREVIOUS"})
g.add_edge(src="1234", targ="abcd", edgelabel="_", prop={"type":"CONTAINS"})
g.add_edge(src="1234", targ="efab", edgelabel="_", prop={"type":"CONTAINS"})
g.add_edge(src="1234", targ="cdef", edgelabel="_", prop={"type":"CONTAINS"})
g.add_edge(src="5678", targ="cdef", edgelabel="_", prop={"type":"CONTAINS"})
```

IBM System G Graph Tools - gShell with Python interface

Graph analytics:

For graph analytics in System G Graph Tools, it will return with the following fields, and all results will be written back to vertices/edges as a property. (it is easy to retrieve in the future.)

1. time: execution time
2. key field: major results (field name depends on analytics)
3. summary: if there are some statistic results, they will be stored in this field.
4. others: supplementary information



NBA graph

```
print json.dumps(json.loads(g.analytic_shortest_paths(src="user98", sink="user34")),indent=2)
```


```
{
  "paths": [
    {
      "src": "user98",
      "path": "user98-->user101-->user125-->user34 | user98-->user183-->user113-->user34 | user98-->user72-->user77-->use
r34",
      "num_paths": 3,
      "sink": "user34",
      "distance": 3.0
    }
  ],
  "time": [
    {
      "TIME": "0.000420809"
    }
  ]
}
```

IBM System G Graph Tools - gShell with Python interface

Columbia University

Another example: count number of triangular in the graph

```
json_print(g.analytic_triangle_count())
```


```
{
  "time": [
    {
      "TIME": "0.00218201"
    }
  ],
  "vertex": [
    {
      "analytic_triangle": 168,
      "id": "user1"
    },
    {
      "analytic_triangle": 24,
      "id": "user2"
    },
    {
      "analytic_triangle": 48,
      "id": "user3"
    },
    {
      "analytic_triangle": 192,
      "id": "user4"
    },
    {
      "analytic_triangle": 312,
```


⋮


```
"summary": [
  {
    "stat": "total",
    "metric": "analytic_triangle",
    "value": 10776
  },
  {
    "stat": "avg",
    "metric": "analytic_triangle",
    "value": 161.639999
  },
  {
    "stat": "max",
    "metric": "analytic_triangle",
    "value": 2136
  },
  {
    "stat": "min",
    "metric": "analytic_triangle",
    "value": 0
  }
]
```

IBM System G Graph Tools - gShell with Python interface

Interact with Python: (through JSON library)

since every result will be returned as “JSON string”, we can convert it as JSON object to interact with user-defined functions.

Example: with graph analytic tool to obtain connected components, and then get (I) number of connected components, (II) size of the maximal connected component, and (III) one of the vertex in the connected component.

```
ret = json.loads(g.analytic_connected_component())
print len(ret["summary"])
max_component = 0
max_component_label = ""
for summary in ret["summary"]:
    if int(summary["count"]) > max_component:
        max_component = int(summary["count"])
        max_component_label = summary["label"]
print max_component
print max_component_label
```

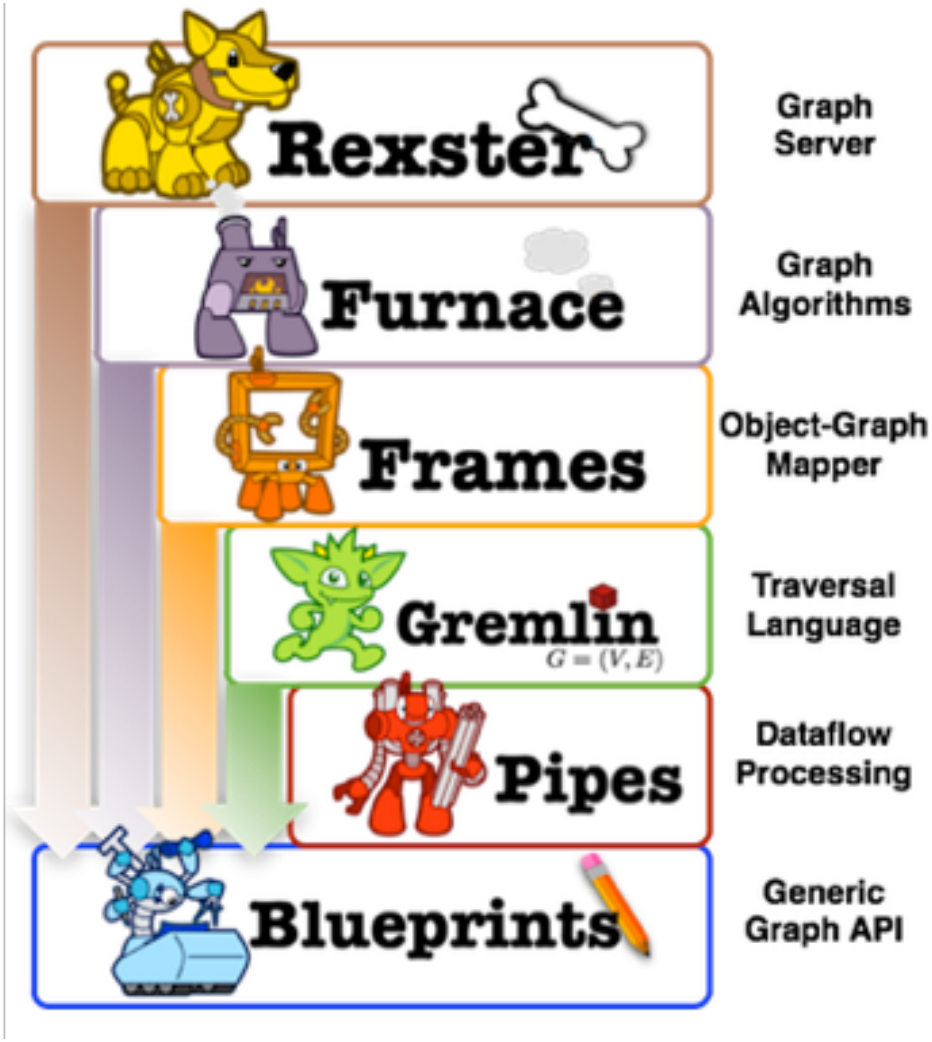
Results:

```
49
144
user1
```

Columbia University

Demo
gShell with Python Interface

Open Source TinkerPop Stack (Apache Incubator)



http://tinkerpop.incubator.apache.org

http://sql2gremlin.com

Graph Language

Gremlin

$G = (V, E)$

Gremlin is a graph traversal language. The documentation herein will provide all the information necessary to understand how to use Gremlin for graph query, analysis, and manipulation. Gremlin works over those graph databases/frameworks that implement the Blueprints property graph data model. Gremlin is a style of graph traversal that can be used in various JVM languages. This distribution of Gremlin provides support for Java and Groovy. Except where otherwise stated, the documentation herein is respective of the Groovy implementation (minor syntactic tweaks are required to map the ideas over to other JVM implementations).

Please join the Gremlin users group at http://groups.google.com/group/gremlin-users for all TinkerPop related discussions. [1]

```
// calculate basic collaborative filtering for vertex 1
m = [:]
g.v(1).out('likes').in('likes').out('likes').groupCount(m)
m.sort{-it.value}

// calculate the primary eigenvector (eigenvector centrality) of a graph
m = [:]; c = 0;
g.V.as('x').out.groupCount(m).loop('x'){c++ < 1000}
m.sort{-it.value}
```


Clone this wiki locally

Gremlin

Gremlin is a domain-specific language for traversing property graphs. JDK 1.7+ is required. The package supports Gremlin 2.4.0 in two modes: command-line and browser-based.

A few external library files need to be downloaded by running the following script before starting Gremlin:

```
$ cd blueprints-gremlin
$ ./bin/download_dependencies.sh
```

The command-line Gremlin console can be started by simply running `gremlin.sh` in the `blueprints-gremlin/bin/` directory.

```
$ cd blueprints-gremlin
$ ./bin/gremlin.sh
         \,,,/
         (o o)
-----oOOo-(_)-oOOo-----
gremlin>
```

To exit the Gremlin console, simply enter `quit`.

Install Java and unbuffer: *sudo apt-get install openjdk-7-jdk expect-dev*

Online documentation: http://systemg.research.ibm.com/doc/gremlin.html
Blueprint API: http://www.tinkerpop.com/docs/javadocs/blueprints/2.5.0/com/tinkerpop/blueprints/Graph.html

IBM System G Graph Tools - 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
```

```
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"}]
}
```

Columbia University

REST API

To enable REST API service, copy `gshell` script, `gShellClientMulti`, `invoke_with_load_library.sh`, and `lib/` in a `<directory>` under the `cgi-bin` directory of the Apache HTTP Server (e.g. `/var/www/cgi-bin/restapi/`). The URLs of all API calls will have a prefix of http://<host>/cgi-bin/<directory>/gshell. If the port number used by `sysGSuperMgr` is not 6688, and/or `sysGSuperMgr` is located on a different machine, the IP address and port number specified in line #169 of `gshell` script (search for "gShellClientMulti") need to be modified accordingly. Use `chmod 755 *` to set proper file permissions.

You need to install ***apache http server*** and ***perl parser***

After the above set-up, run `sysGSuperMgr` to start REST API service:

```
$ cd gshell
$ ./sysGSuperMgr <port_number> <max_num_active_users>
```

For example,

```
$ nohup ./sysGSuperMgr 6688 10 > sysg.out 2> sysg.err < /dev/null &
```

To stop REST API service safely, go to the directory where `gShellClientMulti` is located, run:

```
$ ./gShellClientMulti "bye" admin <host_ip> <port_number>
```

For example,

```
./gShellClientMulti "bye" admin 127.0.0.1 6688
```

Online documentation: http://systemg.research.ibm.com/doc/restapi.html

IBM System G Graph Tools - Installation

REST API - Apache server installation

1. Install apache server: *sudo apt-get install apache2*
2. Activate your cgi-bin: *sudo a2enmod cgi*
3. Add your ServerName: *sudo emacs /etc/apache2/apache2.conf*
    1. Add ServerName at the end of line, “SeverName 127.0.0.1”
4. Change your **cgi-bin** path to **/var/www/cgi-bin**: (in Ubuntu, default path is /usr/lib/cgi-bin
    (if /var/www/cgi-bin did not exist, mkdir before you change cgi-bin path)
    1. sudo emacs /etc/apache2/conf-available/serve-cgi-bin.conf
        - Find ScriptAlias: change to "/cgi-bin/ /var/www/cgi-bin/“
        - Find <Directory>, change to "/var/www/cgi-bin"
5. Change your Document root: *sudo emacs /etc/apache2/sites-available/000-default.conf*
        - Find DocumentRoot, change to “/var/www”
6. Restart Apache2 server: *sudo service apache2 restart*

IBM System G Graph Tools - Installation

REST API - Perl parser installation

1. Install perl: *sudo apt-get install perl*
2. install other perl modules via CPAN Shell: *sudo perl -MCPAN -e shell* (type **yes** twice)
    1. *install Bundle::LWP*
    2. *install JSON*
    3. *install CGI*

IBM System G Graph Tools - Installation

Columbia University

System G Lite

IBM System G Lite requires an HTTP server with PHP version 5.0+ and the REST API service described earlier.

To get started, perform the following steps:

1. Copy `systemg-lite/` to the Web hosting directory (e.g. `/var/www/`) so that IBM System G Lite can be accessed through http://<host>/systemg-lite
2. Open `systemg-lite/php/config.php` and change the URL in the line of `define("GRAPHDB_URL", ...` to be the URL of the REST API service described earlier
3. Use `chmod 755` for all directories and files under `systemg-lite/`
4. Use `chmod 777` for `systemg-lite/tmp/` and `systemg-lite/upload/files/` directories (make sure the directories exist)

You need to install ***php5:*** *sudo apt-get install php5*

Online documentation: http://systemg.research.ibm.com/doc/lite.html

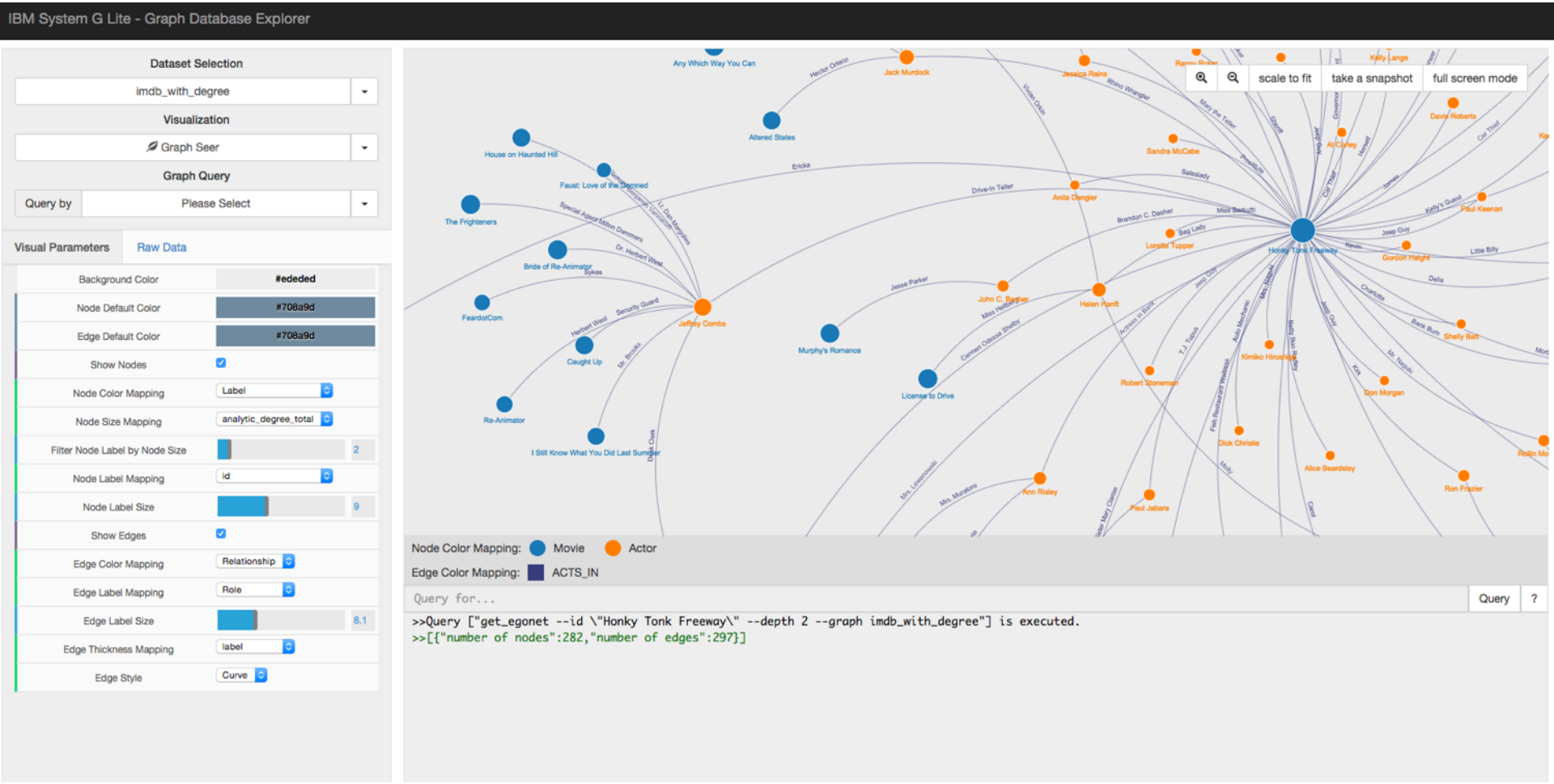
IBM System Visualizer (SystemG-Lite)

IBM System G Visualizer – Graph Data Explorer

Columbia University

Visual Query Panel

Visualization Panel



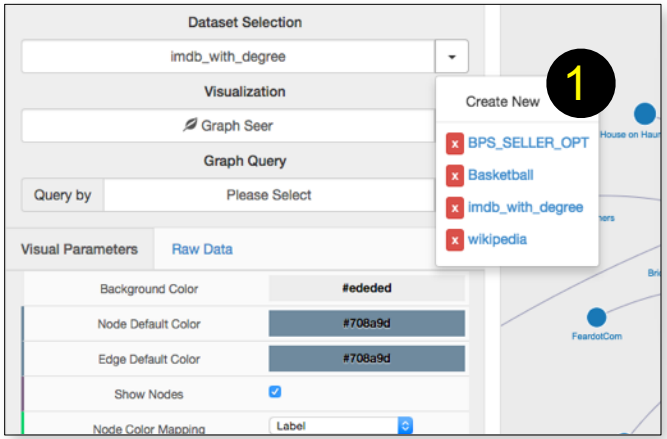
Visual Mapping Panel

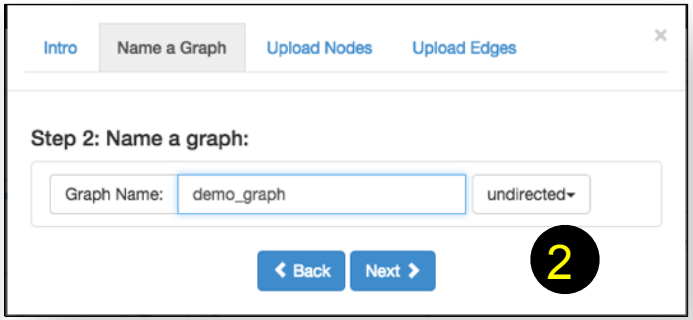
Console Panel

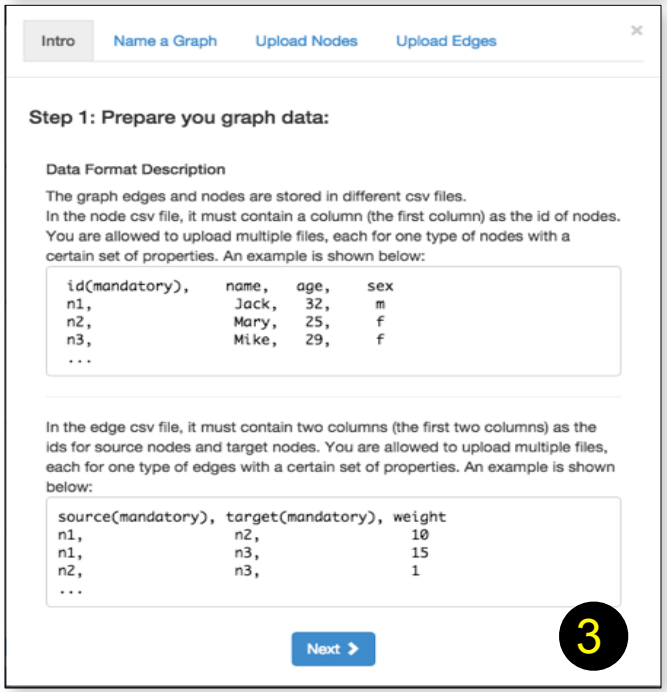
Panel Introduction

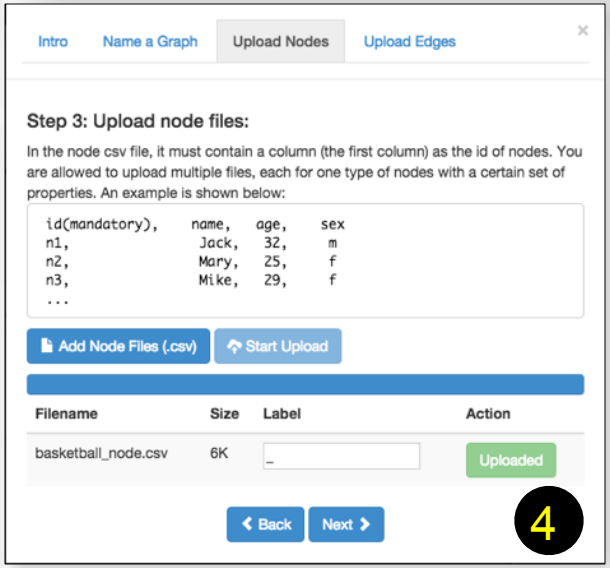
- Visual Query Panel
  - Providing users a friendly UI to create, delete, and query graphs from the System G native store.
- Console Panel
  - Display all the interaction information with System G native store.
  - Execute user defined query.
- Visualization Panel
  - Rendering graph structure on screen for users to visually explore graphs.
- Visual Mapping Panel
  - Customizing rendering effects to show desired graph information.

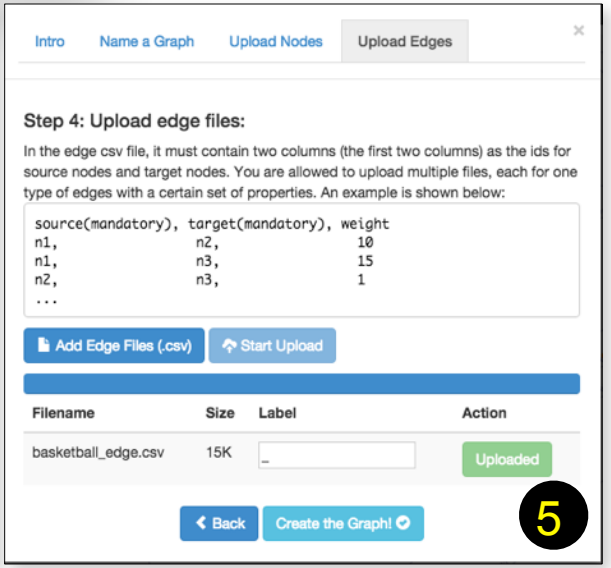
Visual Query Panel – Creating a graph





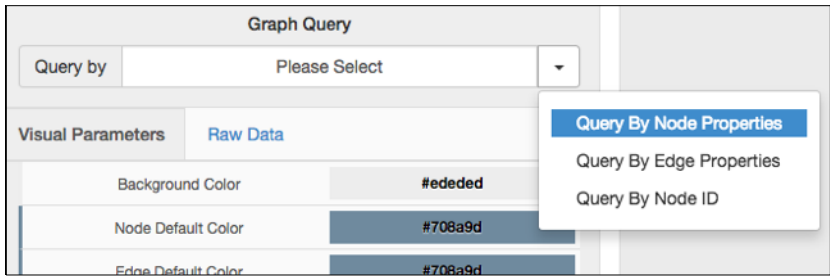


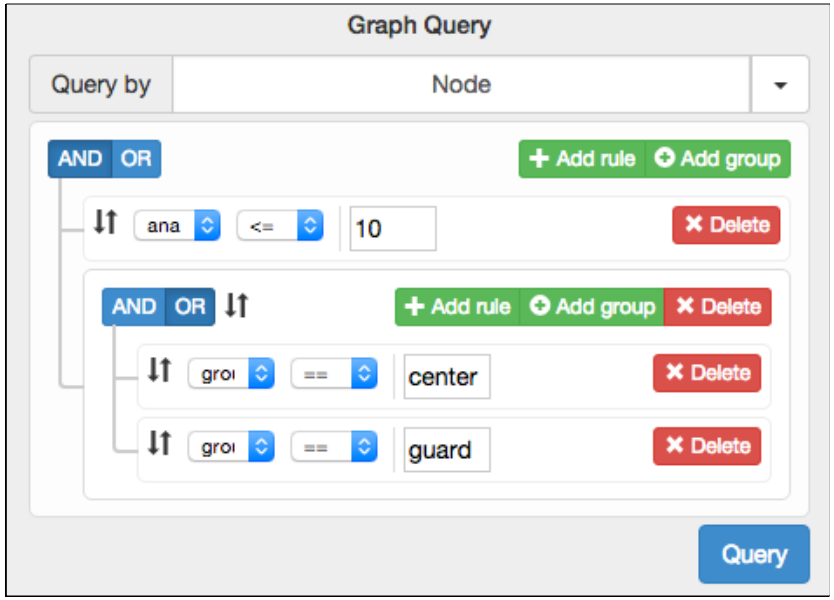


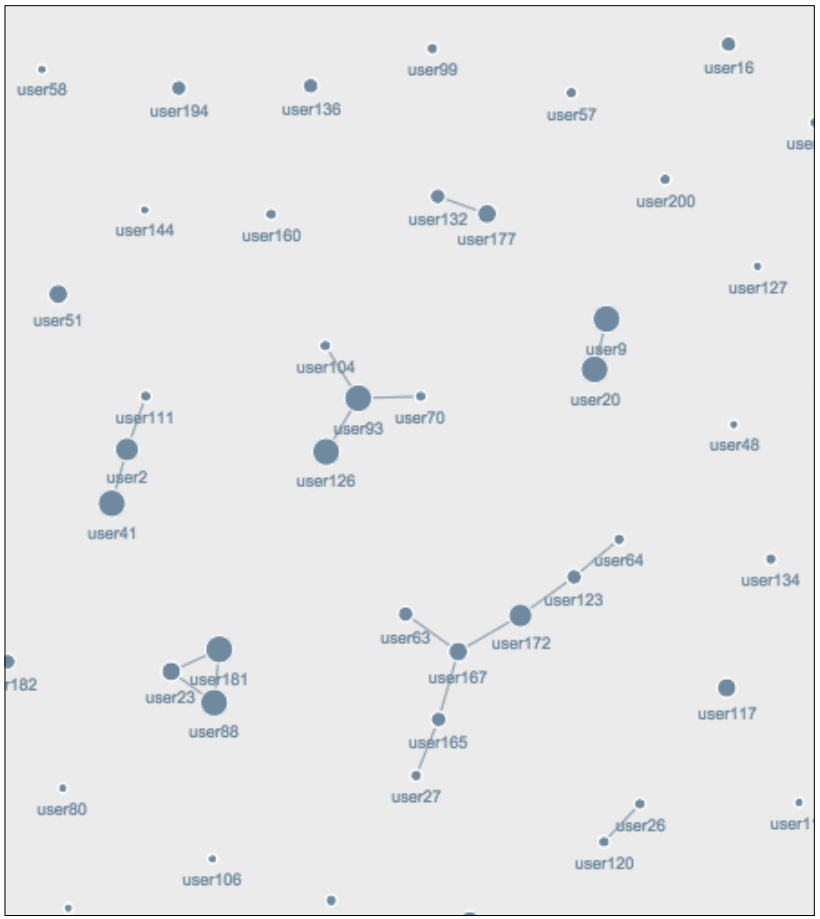


1: Click “Create Graph”; 2: Prepare the graph data
3: Set the graph name; 4: Upload node files;
5: Upload edge files and finalize creating the graph.

Visual Query Panel – Visual Query Builder








```
"analytics_degree <= 10 and (group == "center" or group == "guard")
```

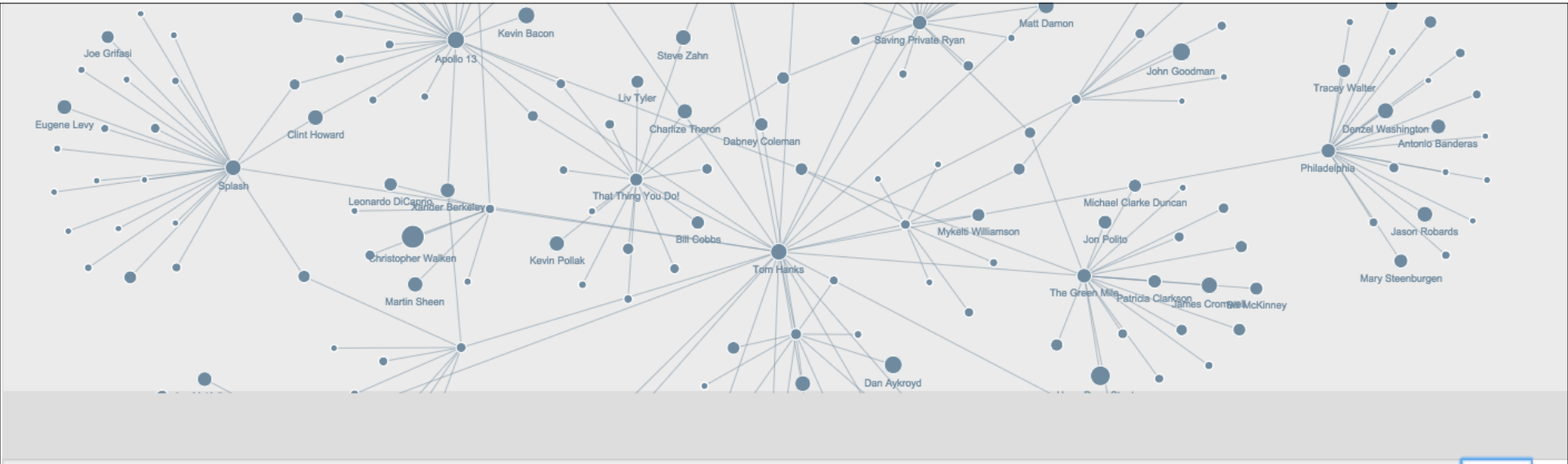
Console Panel – User typed query

```
find_vertex_max_degree --graph Basketball --edgetype all
```

Query ?

```
>>Query ["print_all --graph Basketball"] is executed.
>>[{"number of nodes":199,"number of edges":826}]
>>Query ["find_vertex_max_degree --edgetype all --graph Basketball"] is executed.
>>[{"vertex id":"user72"},{"all-degree":46}]
```

Query with no graph returned



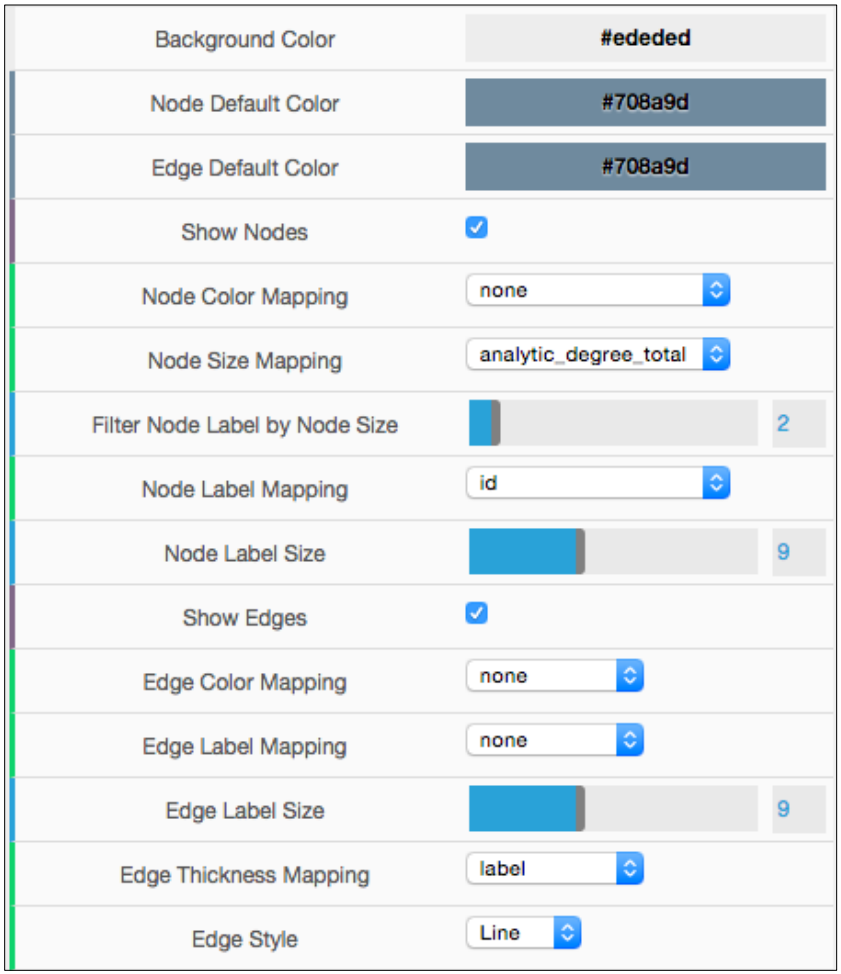

```
get_egonet --graph imdb_with_degree --id "Tom Hanks" --depth 2
```

Query ?

```
>>Query ["print_all --graph Basketball"] is executed.
>>[{"number of nodes":199,"number of edges":826}]
>>Query ["find_vertex_max_degree --edgetype all --graph Basketball"] is executed.
>>[{"vertex id":"user72"},{"all-degree":46}]
>>Query ["get_egonet --id \"Tom Hanks\" --depth 1 --graph imdb_with_degree"] is executed.
>>[{"number of nodes":26,"number of edges":25}]
>>Query ["get_egonet --id \"Tom Hanks\" --depth 2 --graph imdb_with_degree"] is executed.
>>[{"number of nodes":383,"number of edges":401}]
```

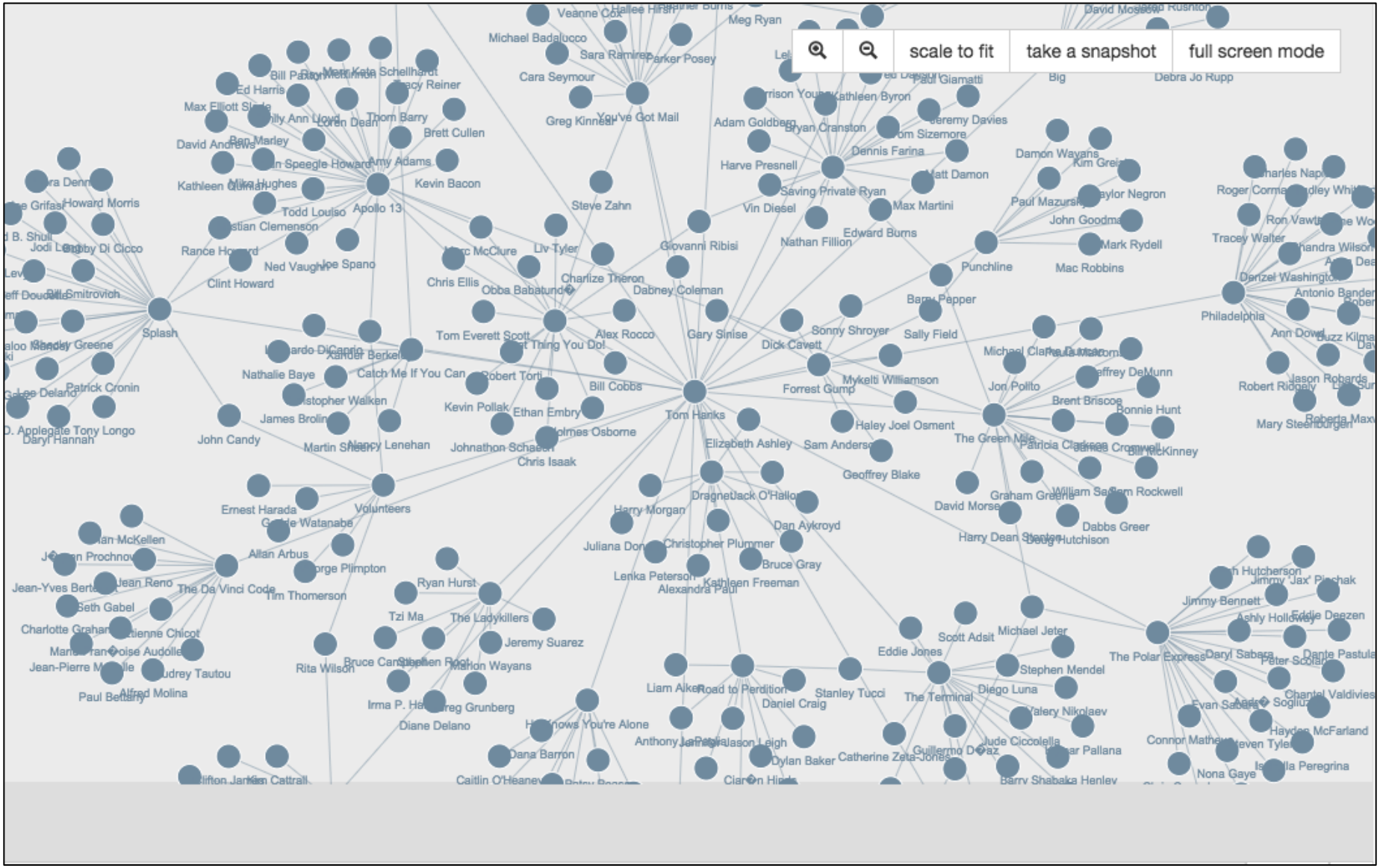
Query with graph returned

Visual Mapping Panel

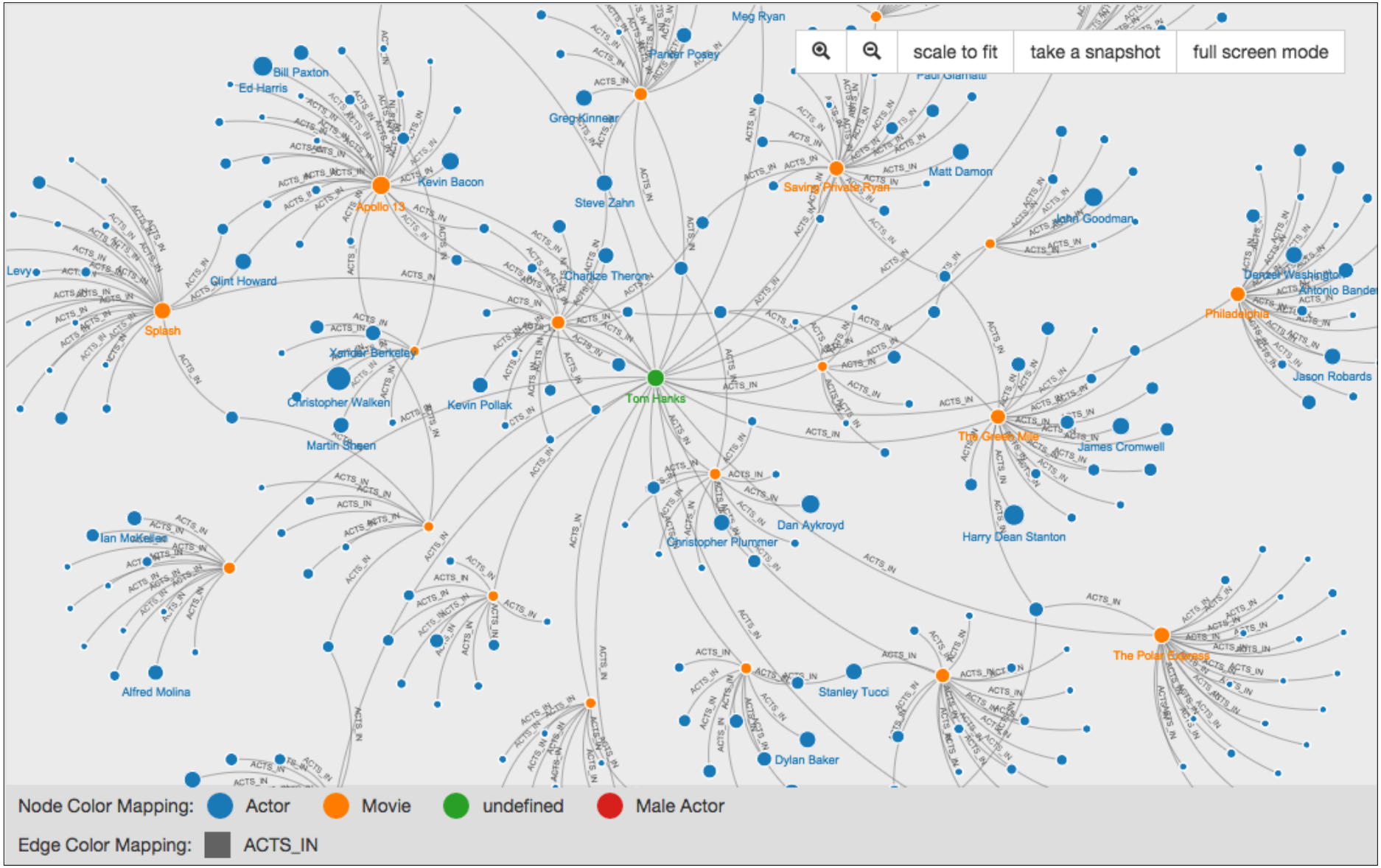


Name	Functionality
Background Color	Change the background color of the canvas.
Node Default Color	Set a unified color for all nodes.
Edge Default Color	Set a unified color for all edges.
Show Nodes	Set the visibility of all nodes.
Node Color Mapping	Assign color to nodes according to selected property of nodes.
Node Size Mapping	Assign the radius of nodes according to selected property of nodes.
Filter Node Label by Node Size	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.
Node Label Mapping	Set the label value according to selected property of nodes.
Node Label Size	Adjust the font size of node labels
Show Edges	Set the visibility of all edges
Edge Color Mapping	Assign color to edges according to selected property of edges.
Edge Label Mapping	Set the label value according to selected property of edges.
Edge Label Size	Adjust the font size of edge labels
Edge Thickness Mapping	Assign thickess to edges according to selected property of edges.
Edge Style	Select the rendering style of edges. For directed graphs, users also can choose if showing the arrows or not.

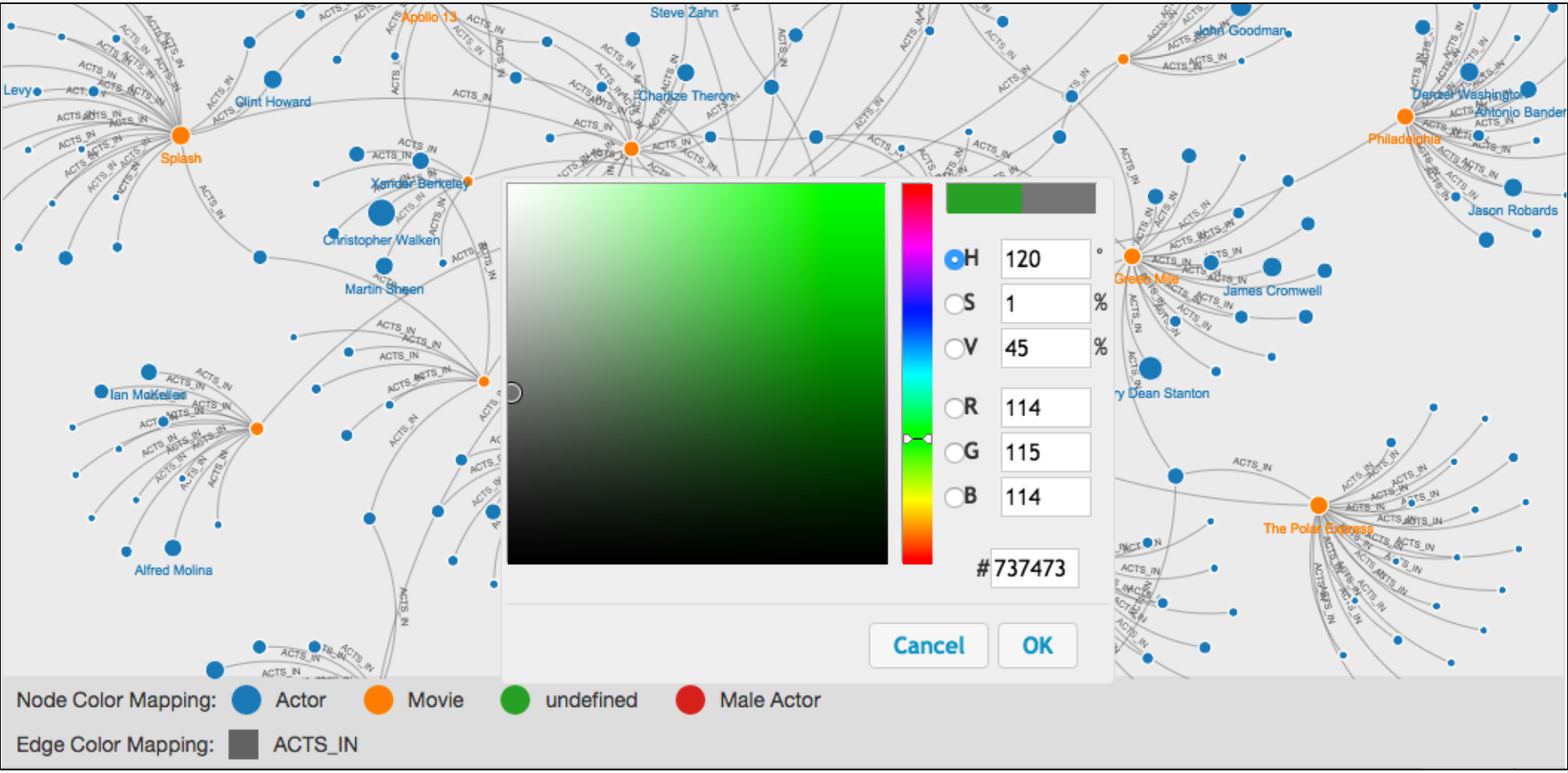
Visualization Panel – Before Customization



Visualization Panel – After Customization

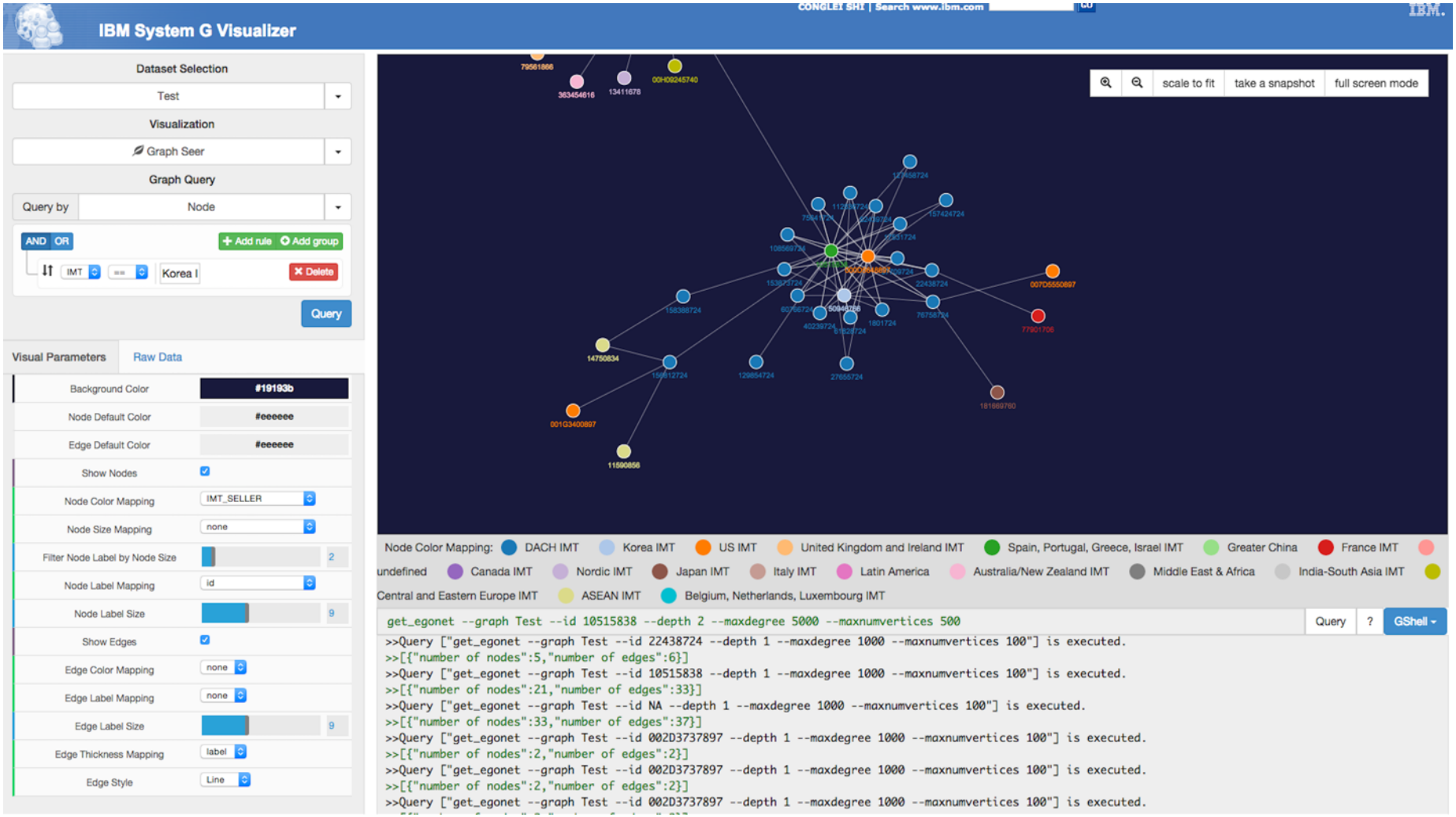


Visualization Panel – Further Customization



Users can further specify colors by clicking the color blocks shown in the legend area

http://systemg.ibm.com/tool/visualizer/



IBM System G Eco-System (GraphBIG)

A group of graph analytics for benchmarking underlying platforms

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

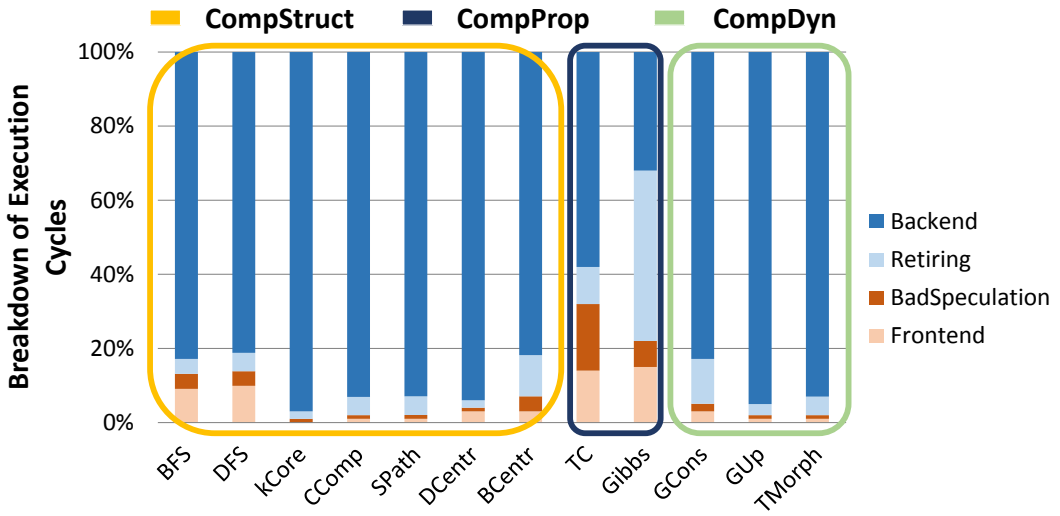
Come with performance profiler by taking hardware performance counters, breaking down the execution time into multiple stages to reveal the performance bottleneck

Fetch Code

Code: https://github.com/graphbig/graphBIG

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

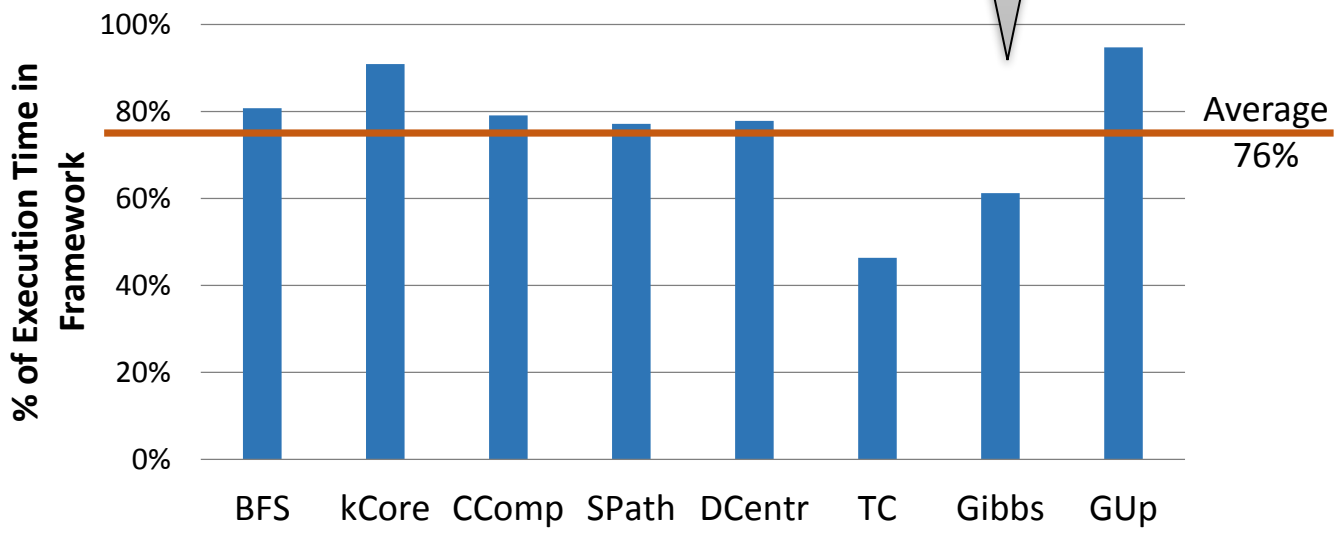
```
-bash:~$ git clone https://github.com/graphbig/graphBIG.git GraphBIG
Cloning into 'GraphBIG'...
remote: Counting objects: 497, done.
remote: Compressing objects: 100% (110/110), done.
remote: Total 497 (delta 57), reused 0 (delta 0), pack-reused 386
Receiving objects: 100% (497/497), 2.07 MiB | 0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



Understand Graph Computational Challenges

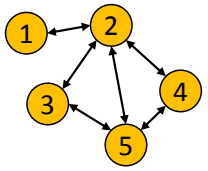
76%

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

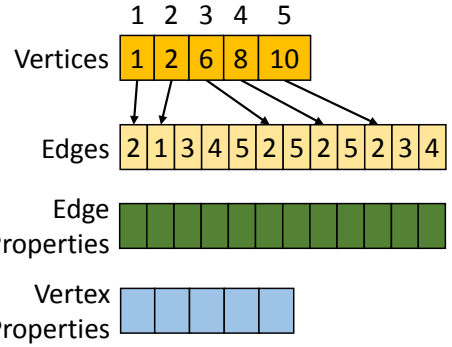


framework actually plays a critical role

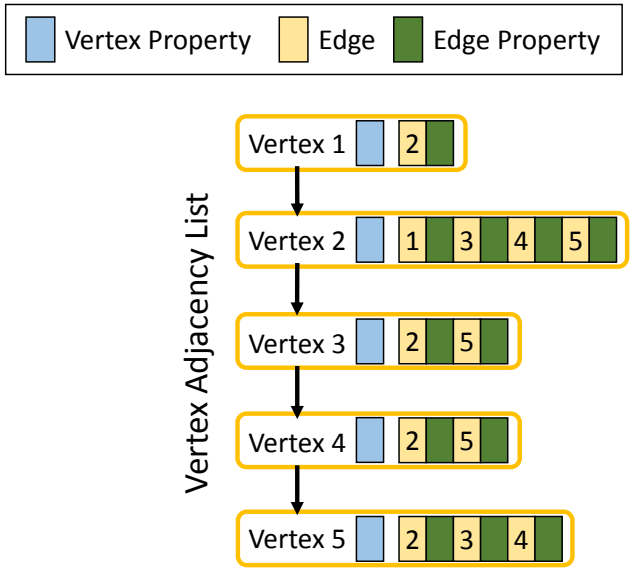
Graph Data Representations



(a) Graph G



(b) CSR Representation of G

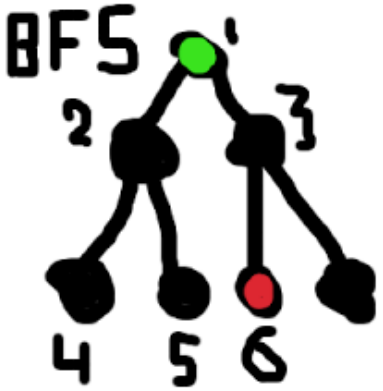


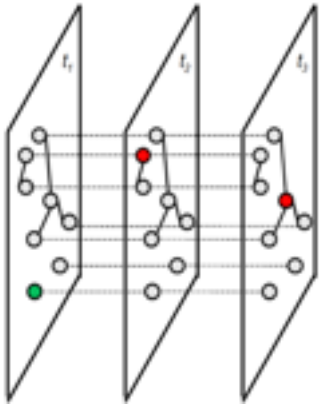
(c) Vertex-centric Representation of G

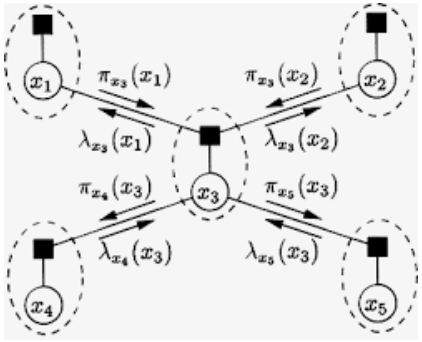
CSR format is compact, and maybe good for cache performance. But it is static, and cannot support structure changes. However, in practices, graphs are usually dynamic. This is why vertex-centric representation is popular across multiple graph frameworks.

Graph Computing Types

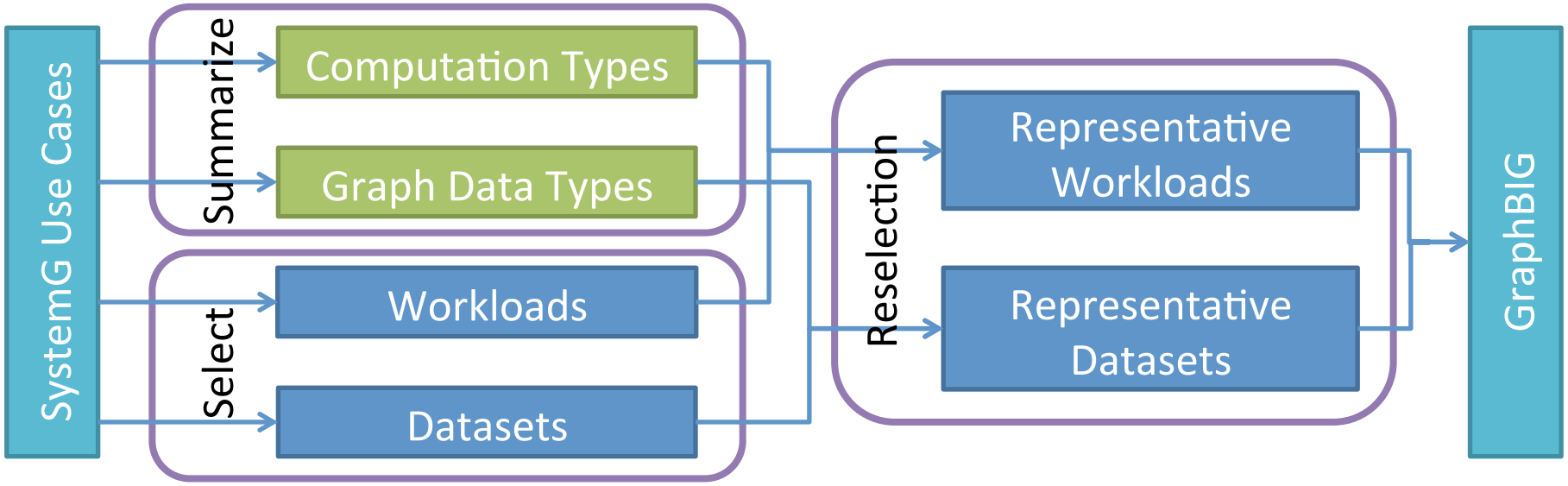
- Computation on graph structure (CompStruct)
  - Example: Breadth-first search
  - Irregular access pattern, heavy read access
- Computation on dynamic graph (CompDyn)
  - Example: Streaming Graph
  - Dynamic graph structure, dynamic memory usage
- Computation on graph property (CompProp)
  - Example: Belief propagation
  - Heavy numeric operations on graph property





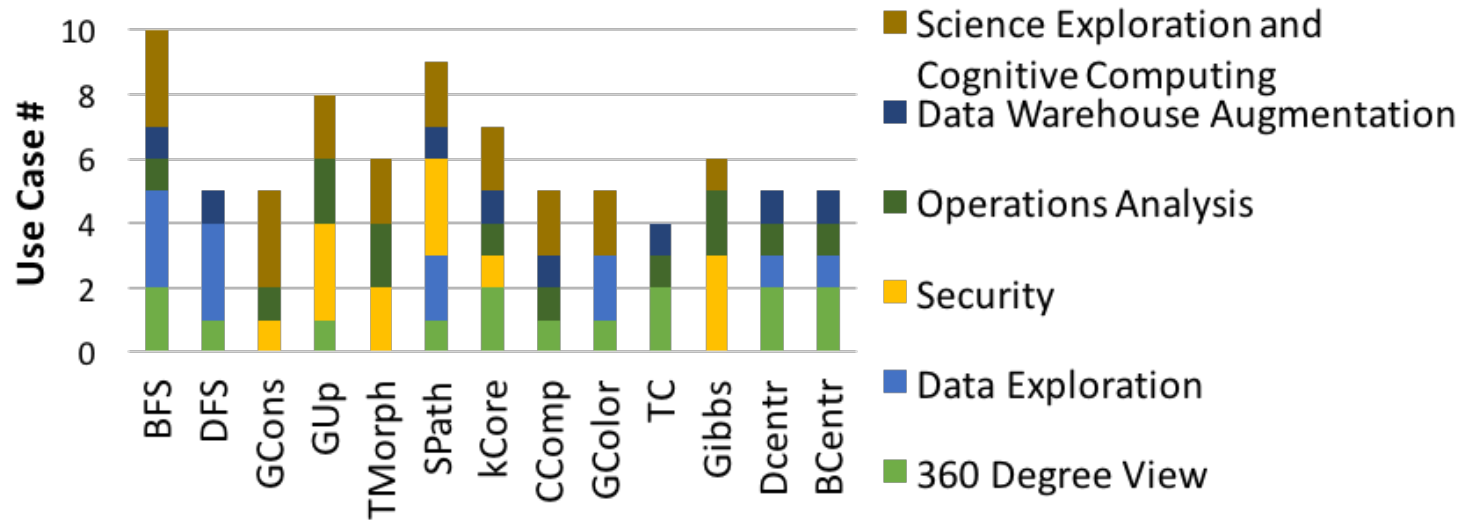


Graph Workload Selection to Form a Benchmark



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.

Workload Selection



(A) # of Use Cases with Each Workload

(B) Distribution of Selected Use Cases in 6 Categories

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

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

Data set	Type	Vertex #	Edge #
Twitter Graph	Type 1	120M	1.9B
IBM Knowledge Repo	Type 2	154K	1.72M
IBM Watson Gene Graph	Type 3	2M	12.2M
CA Road Network	Type 4	1.9M	2.8M
LDBC Graph	Synthetic	Any	Any

GraphBIG Hands-on

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.

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:~$
```

GraphBIG Hands-on - 2

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"

Compile

- Require: gcc/g++ (>4.3), gnu make
- Just *"make all"*

```
-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   common.mk  ubench_add     ubench_traverse
bench_BFS               bench_DFS             bench_graphUpdate     bench_TopoMorph      Makefile   ubench_delete
bench_connectedComp     bench_gibbsInference  bench_kCore           bench_triangleCount  README.txt ubench_find
-bash:benchmark$ make all
make -C ../tools all
make[1]: Entering directory `/home/lifeng/GraphBIG/tools'
rm -rf libpfm-4.5.0
tar xzvf libpfm-4.5.0.tar.gz
libpfm-4.5.0/./
libpfm-4.5.0/./COPYING
libpfm-4.5.0/./lib/
libpfm-4.5.0/./lib/pfmlib_ppc970.c
libpfm-4.5.0/./lib/pfmlib_powerpc.c
libpfm-4.5.0/./lib/pfmlib_sparc_priv.h
libpfm-4.5.0/./lib/pfmlib_powerpc_perf_event.c
```

GraphBIG Hands-on - 3

Test Run

- Just *"make run"*
- Using default "small" dataset

Help info: ./<exe> --help

```
-bash:benchmark$ cd bench_BFS/
-bash:bench_BFS$ make run
Running bfs, output in output.log
```

```
-bash:bench_BFS$ cat output.log
================================================================================

================================================================================
Benchmark: BFS
loading data...
== 1000 vertices  29790 edges
== time: 0.0530188 sec

BFS root: 31
BFS finish:
== time: 0.00118506 sec
PERF_COUNT_HW_CPU_CYCLES            ==>      2581036
PERF_COUNT_HW_INSTRUCTIONS          ==>      774222
PERF_COUNT_HW_BRANCH_INSTRUCTIONS   ==>      200529
PERF_COUNT_HW_BRANCH_MISSES         ==>      10486
PERF_COUNT_HW_CACHE_L1D_READ_ACCESS ==>      309284
PERF_COUNT_HW_CACHE_L1D_READ_MISS   ==>      99740
================================================================================
```

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"

Columbia University

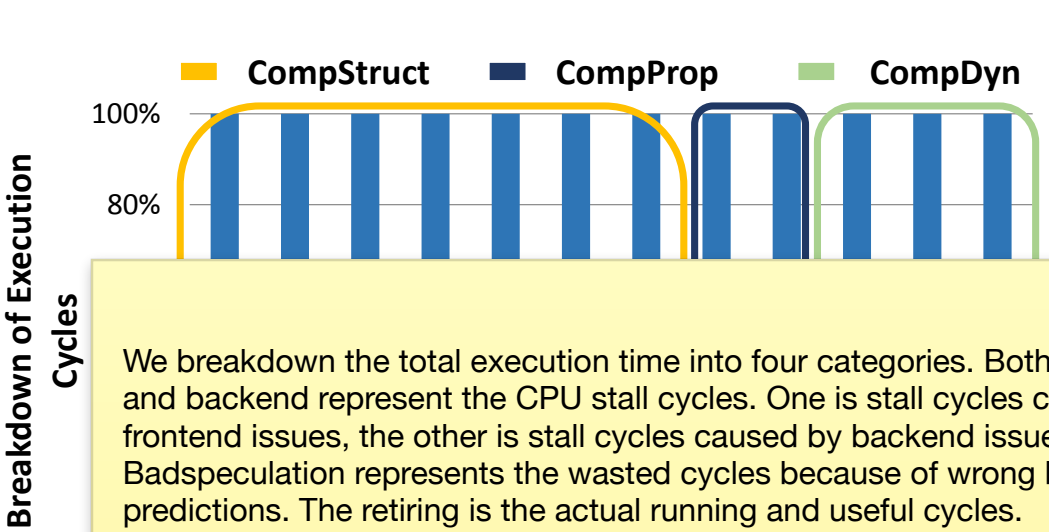
Methodology

Real machine + hardware performance counters

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

Processor	**Type**	**Xeon E5-2670**
	Frequency	2.6 GHz
	Core #	2 sockets x 8 cores x 2 threads
	Cache	32KB L1, 256KB L2, 20MB L3
	Memory BW	51.2 GB/s (DDR3)
GPU	Type	Nvidia Tesla K40
	CUDA Core	2880
	Memory	12 GB
	Memory BW	288 GB/s
	Frequency	Core-745 MHz, mem-3 GHz
System	Memory	192 GB
	Disk	2 TB HDD
	OS	RHEL 6

Execution Time Breakdown



Backend is the bottleneck

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 exsit, for TC (triangle counting) and Gibbs (gibbs inference), they are not suffering from backend issues.
It shows an interesting diversity across benchmarks

ation

he total cycles being spent by the CPU in 3 categories:

re instructions get retired (usefull work)

g spent in the Back-End (wasted)

nt in the Front-End (wasted).

Th rces
(u tc.).

The **cycles stalled in the front-end** are a waste because that means that the CPU does not feed the Back End with instructions. This can mean that you have misses in the Instruction cache, or complex instructions that are not already decoded in the micro-op cache.

IBM System G Eco-System (ScaleGraph)

ScaleGraph — an Open Source version of IBM System G

Columbia University

ScaleGraph

Home | About | Features and Roadmap | Documentation | Project Development | Workshops | Downloads

ScaleGraph

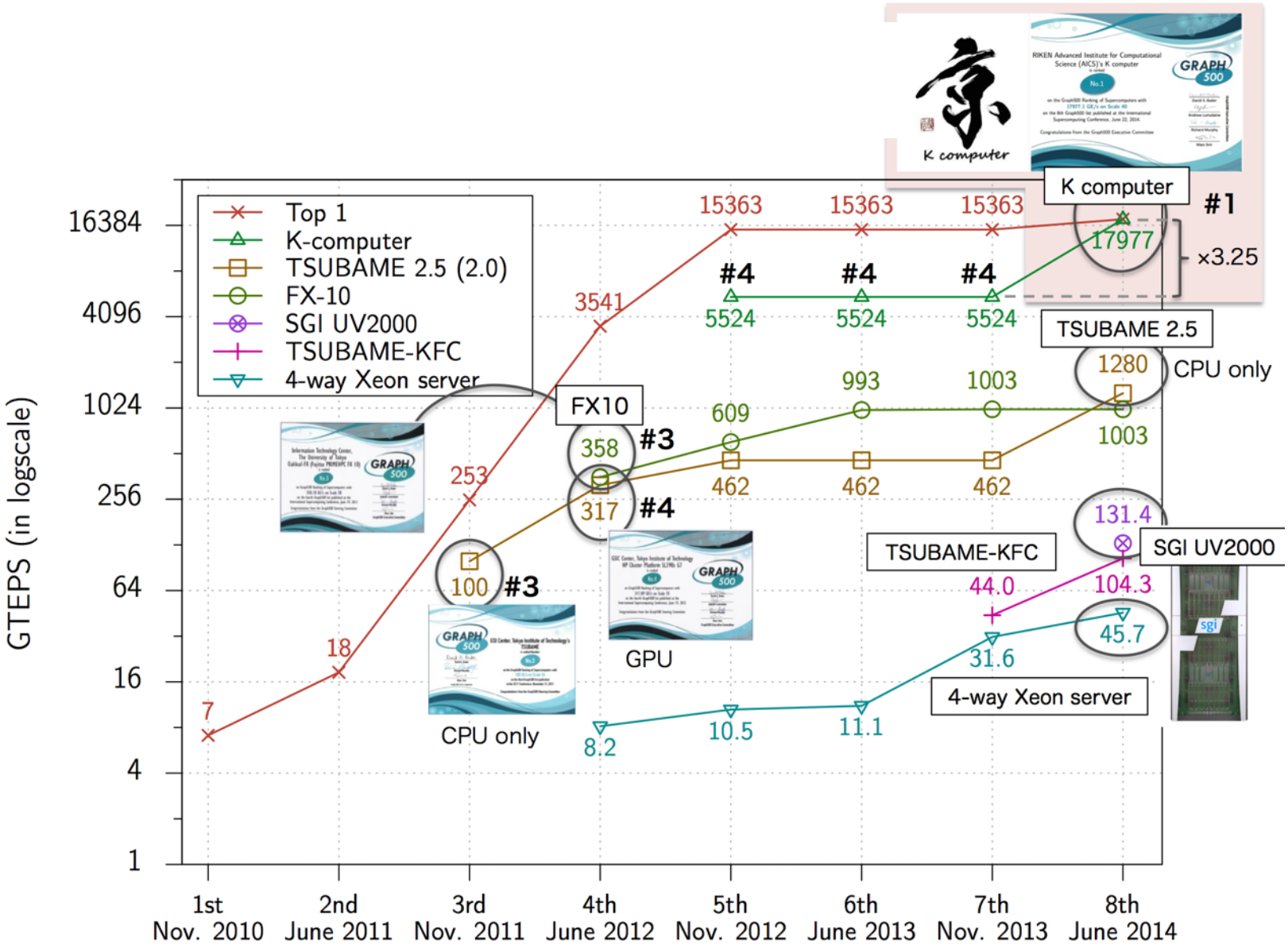
A graph library for **large-scale graph processing** on top of the state-of-the-art X10 programming language

Recently large-scale graphs with billions of vertices and edges have emerged in a variety of domains and disciplines especially in the forms of social networks, web link graphs, internet topology graphs, etc. Mining these graphs to discover hidden knowledge requires particular middleware and software libraries that can harness the full potential of large-scale computing infrastructures such as super computers.

ScaleGraph is a graph library based on the highly productive X10 programming language. The goal of ScaleGraph is to provide large-scale graph analysis algorithms and efficient distributed computing framework for graph analysts and for algorithm developers, respectively.

Search

ScaleGraph algorithms made Top #1 in Graph 500 benchmark



Source: Dr. Toyotaro Suzumura, ICPE2014 keynote

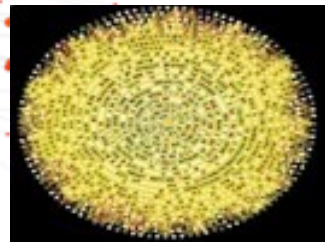
ScaleGraph Library

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

Internet Map

Symbolic Networks:

Social Networks



Protein Interactions

Cyber Security (15 billion log entries / day for large enterprise)

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

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

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

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

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

Web Site and Source Code Repository

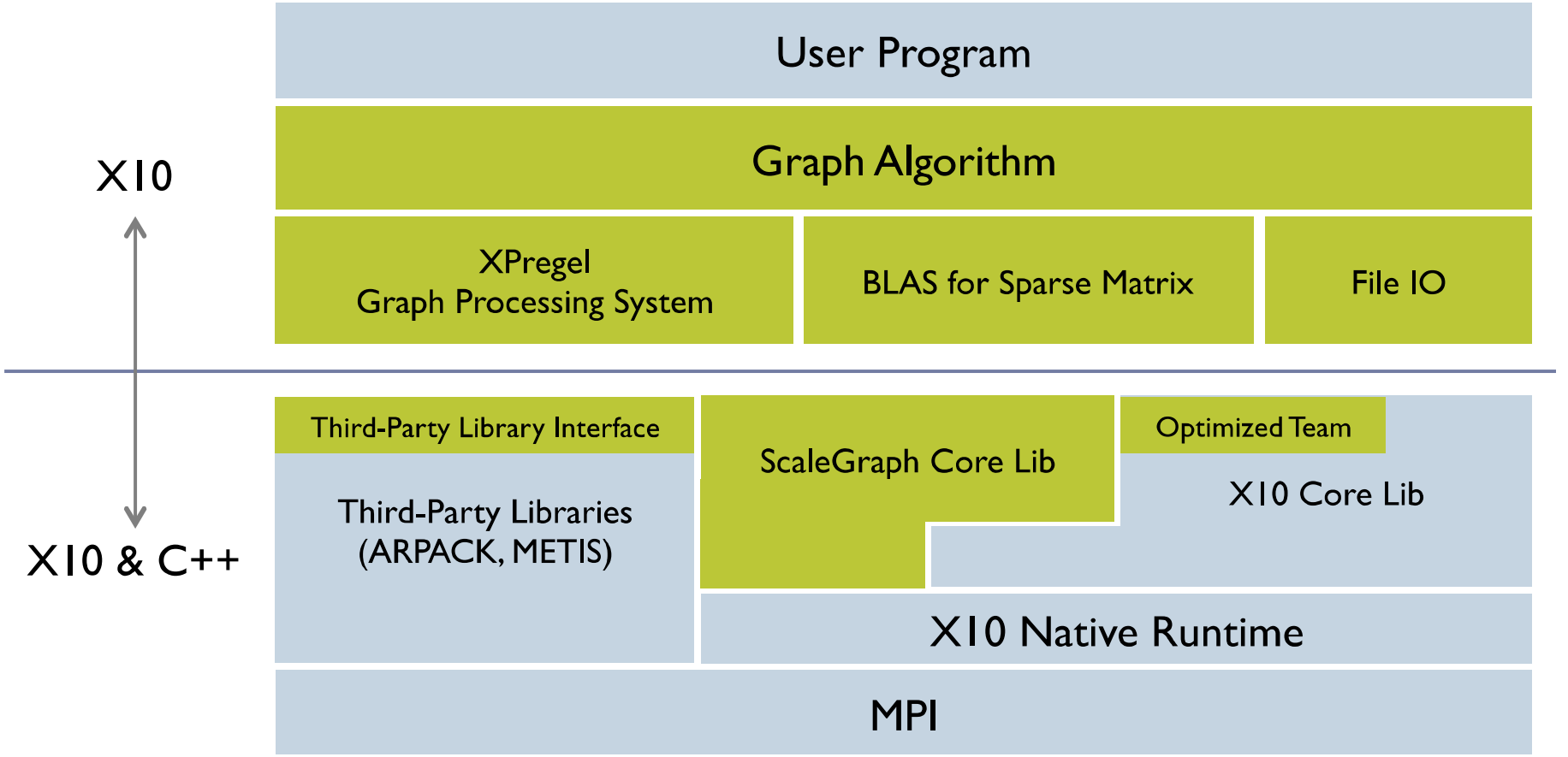
Official web site – http://scalegraph.org

- Project information
- Source code distribution
- Documentation

Source code repository - http://github.com/scalegraph/

- License: Eclipse Public License v1.0
- Project information and Documentation

ScaleGraph Software Stack



Developing Graph Algorithms (e.g. PageRank)

```
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)

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

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

Questions?