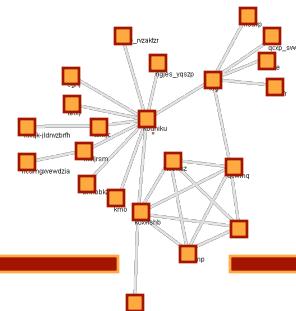


# The Java Universal Network/Graph Framework (JUNG): A Brief Tour

# Joshua O'Madadhain (UCI)

# Danyel Fisher (MSR), Tom Nelson (RABA Technologies) Scott White (UCI), Yan-Biao Boey (UCI)

# What's it for?



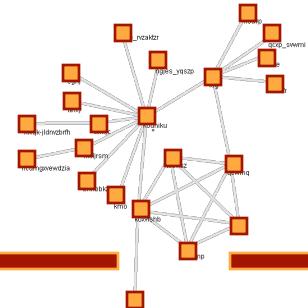
# phenomenae pertaining to relationships/links

- collaborations, citations/hyperlinks, email, biological systems, friendships, telecom networks, ecological and biological systems, Markov processes, Bayes nets...

# SUNBELT 2000

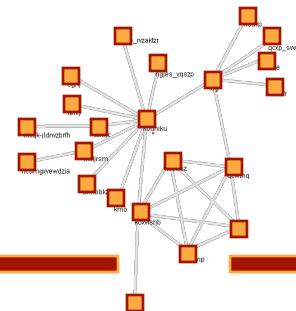
- 270 People & Structural Holes
  - Database *to* UCINET *to* Pajek *to* MAGE

# Programmatic Solution



- *Programmable*  
Make it easy to do the same thing again
- *Extensible*  
If you need a new routine, you can put it in yourself
- *Library*  
Can be used in a server, a client, or as part of a larger application
- *Open Source*  
Take it apart and put it together some other way  
Free!

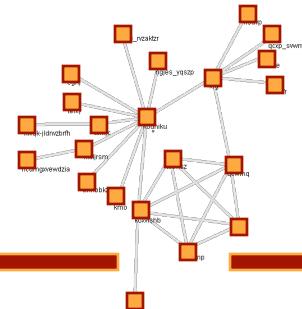
# What is JUNG?



# Framework for the modeling, analysis, and visualization of graphs in Java

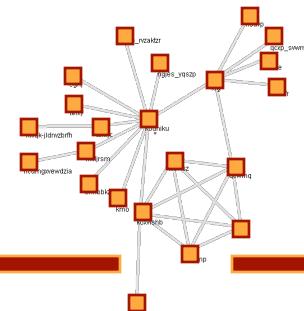
- supports most types of graphs/networks
  - separate, flexible visualization framework
  - rich attribute structure (metadata)
  - network event handling
  - predicates (subsets, constraints, filtering, ...)
  - extensive (and growing!) library of algorithms

# Dependencies



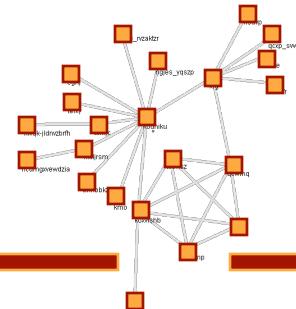
- **Java (1.4.2+)**
- **Commons-Collections API (3.1)**  
<http://jakarta.apache.org/commons/collections/>  
Predicate, set operations, wrappers, data types (BidiMap)
- **CERN Colt API (1.2.0)**  
<http://dsd.lbl.gov/~hoschek/colt/>  
matrix operations, statistics
- **Xerces (2.6.2)**  
<http://xml.apache.org/xerces2-j/index.html>  
XML parsing

# Overview



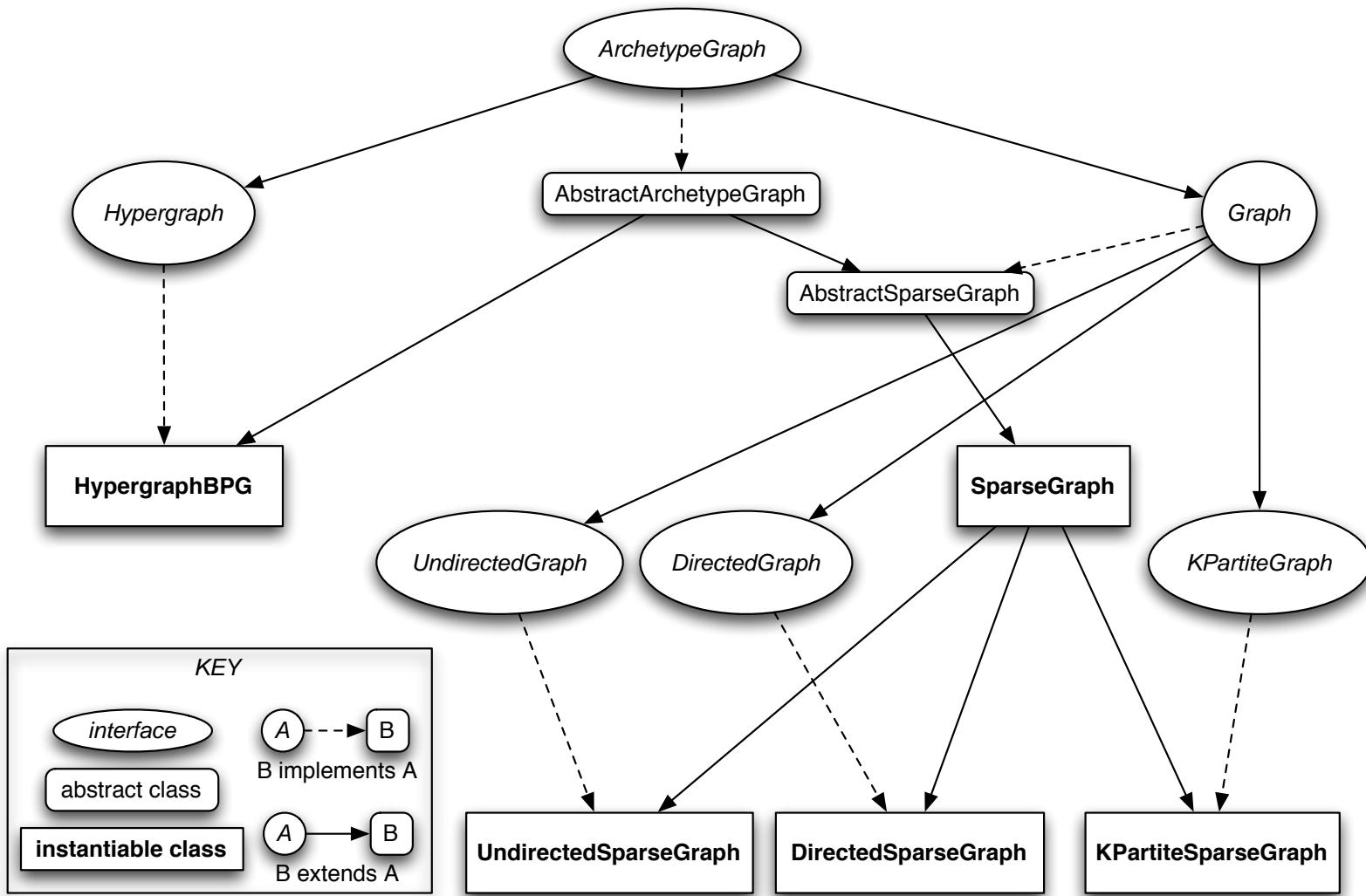
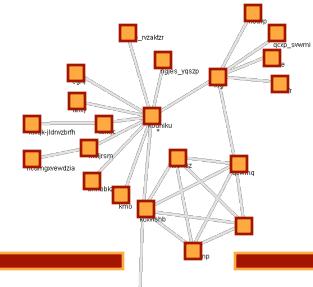
- Creating a graph
  - Annotating with data
  - Filters
  - Graph drawing
  - Algorithms
  - Demo

# Creating a graph

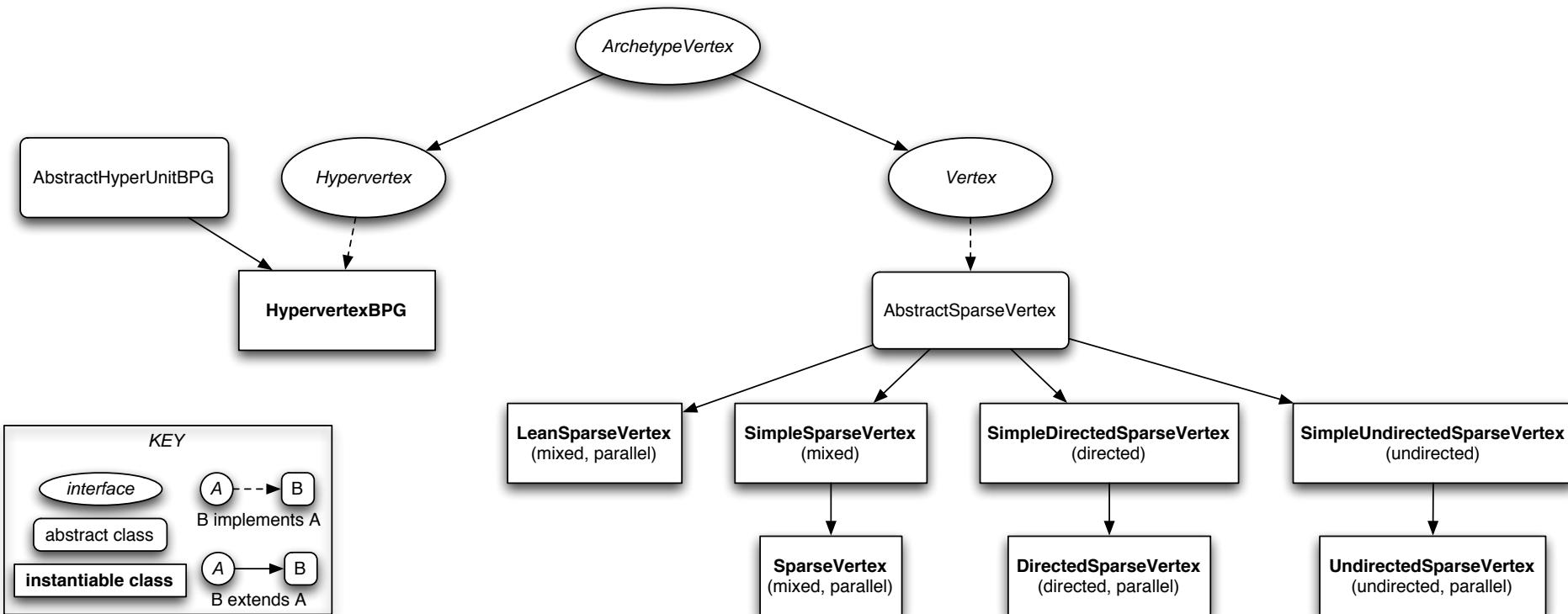
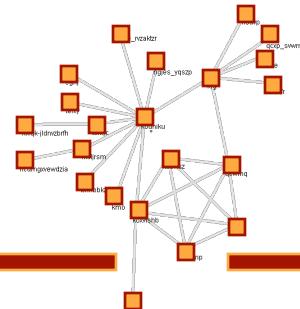


- Load from a file
    - Pajek
    - GraphML (XML)
  - Build from scratch
    - Create vertices individually
    - Specify edges
  - Generate from an algorithm/model
    - Small-world graphs
    - Power-law graphs

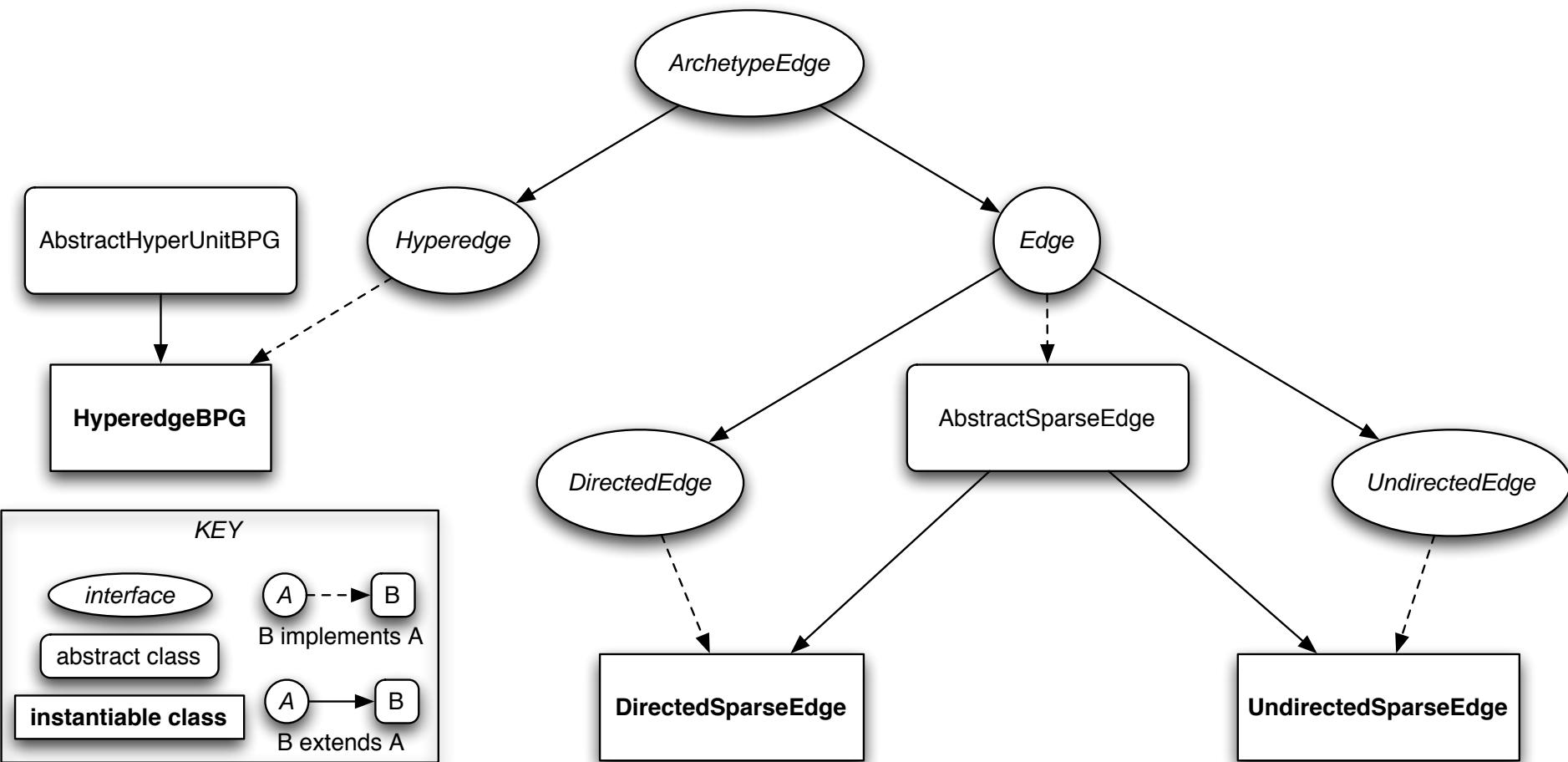
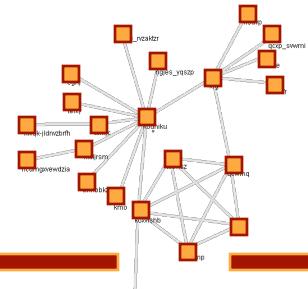
# Graph Types



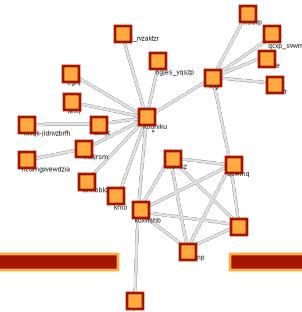
# Vertex Types



# Edge Types

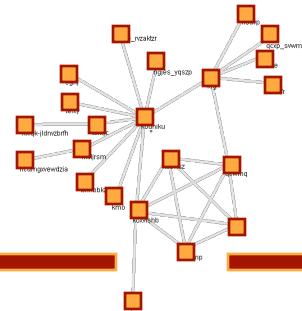


# Creating a Graph



```
Graph g = new DirectedSparseGraph();
Vertex v1 = new SparseVertex();
Vertex v2 = new SparseVertex();
g.addVertex( v1 );
g.addVertex( v2 );
Edge e = new Edge( v1, v2 );
g.addEdge( e );
```

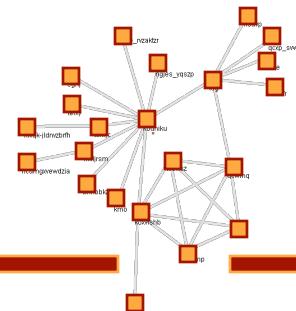
# Adding user-defined data



- key-value pairs associated with graphs, edges, vertices
- easy to write algorithms that operate on relational data

```
String name = vertex.getUserDatum(NAME);  
edge.setUserDatum(DATE, new Date());
```

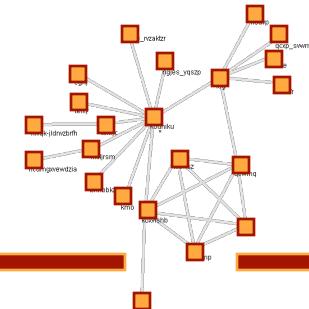
# Copying and Equality



- copies of vertices and edges are “equal to” the originals
    - `equals()`, `hashCode()`
    - subgraphs
    - allows experimentation with mutations to graph, without affecting original

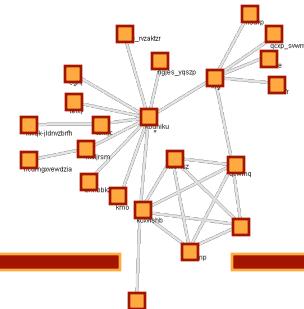
```
v2 = v1.getEqualVertex(g2)
```

# Predicates



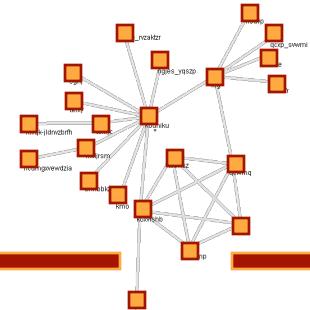
- logical expressions, e.g.  
`v.degree() > 4`  
`e instanceof DirectedEdge`
  - can be used to define subsets and constraints
    - k-partite graphs
    - multigraphs, simple graphs, trees, mixed-mode graphs, hypergraphs, ...

# Using graph filters



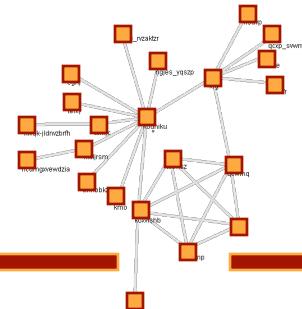
- Extract subgraphs of vertices and edges
    - All edges with user-data “weight” > 0.2
    - All nodes whose “location” is “California”
    - All nodes within distance 3.5 of v
  - Chain filters together
  - Automate complex graph transformations

# Visualizing graphs



- Basic layout algorithms:
  - Fruchterman-Rheingold
  - Kamada-Kawai
  - Eades
  - Self-Organizing Maps
- Plug and play architecture
  - Drop in your favorite renderer, layout
- Provided rendering engine is Swing-based
  - (but layouts are generic)

# Simple Graph Display

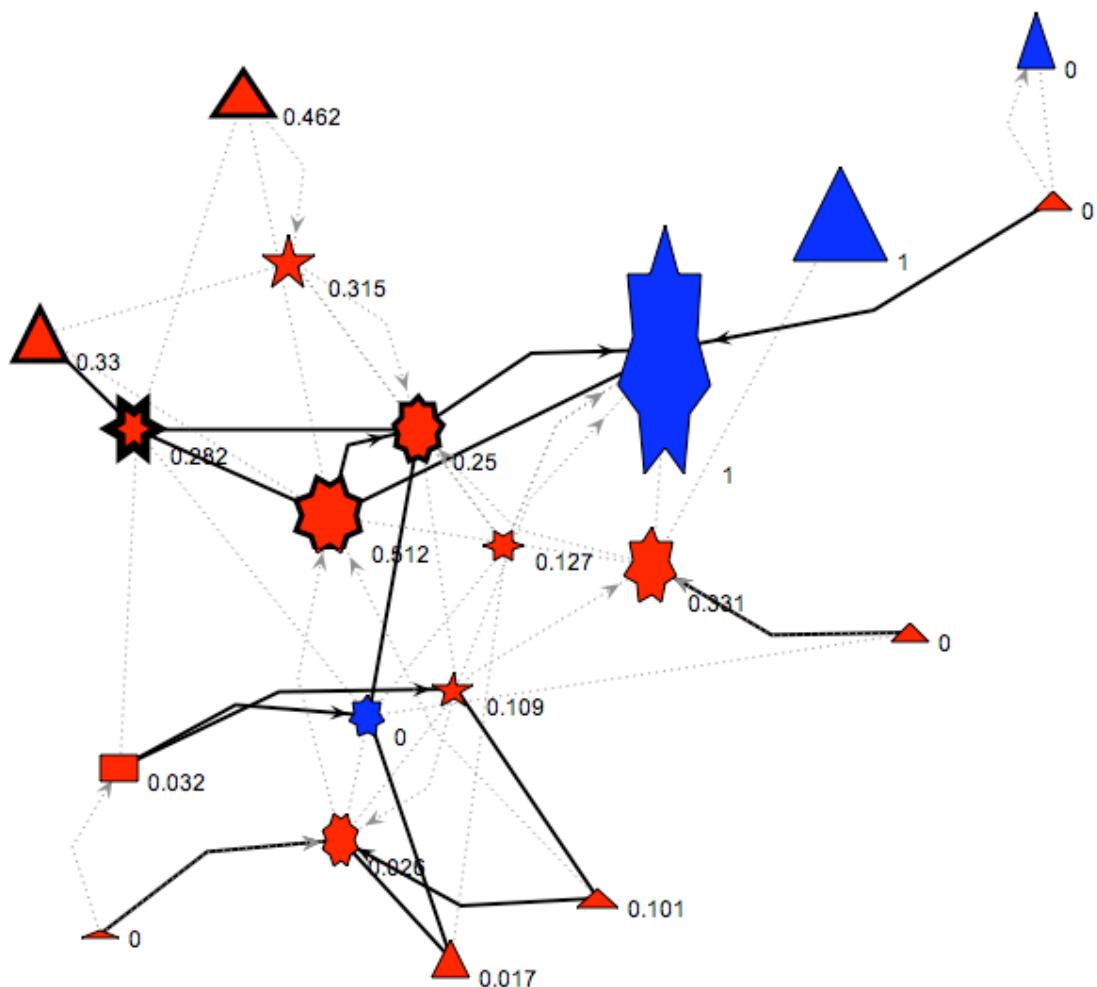


to display an existing graph:

```
l = new FRLayout(g);  
r = new PluggableRenderer();  
vv = new VisualizationViewer(l,r);  
jpanel.add(vv);
```

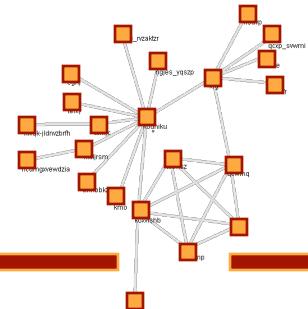
now, start customizing...

# Pluggable Renderer



- vertex seed coloring
- vertex selection stroke highlighting
- show vertex ranks (voltages)
- vertex degree shapes
- vertex voltage size
- vertex degree ratio stretch
- bold text
- show edge weights
- show undirected edge arrows

# Customizing

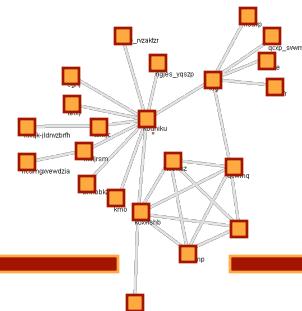


Layout algorithms specify vertex positions  
**advancePositions()**

Renderers draw vertices and edges.

```
paintEdge( g, Edge, x1, y1, x2, y2 )  
paintVertex( g, Vertex, x, y )
```

# Algorithms



# clustering

# k-neighborhood

# connected components

# k-means clustering

# connectivity

# maximum flow

# network distances

# structural equivalence

# centrality/importance

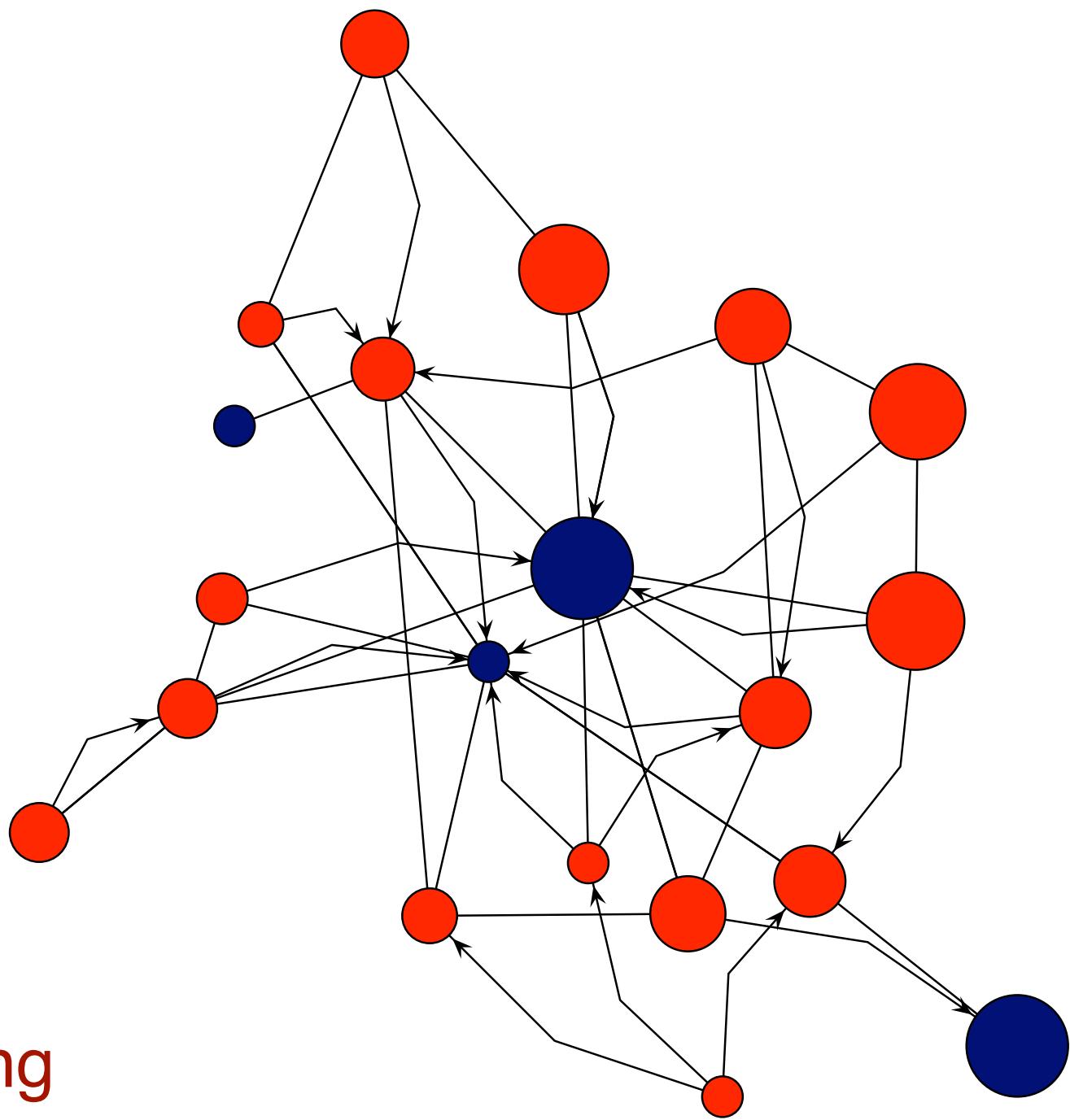
# betweenness,

# markov,

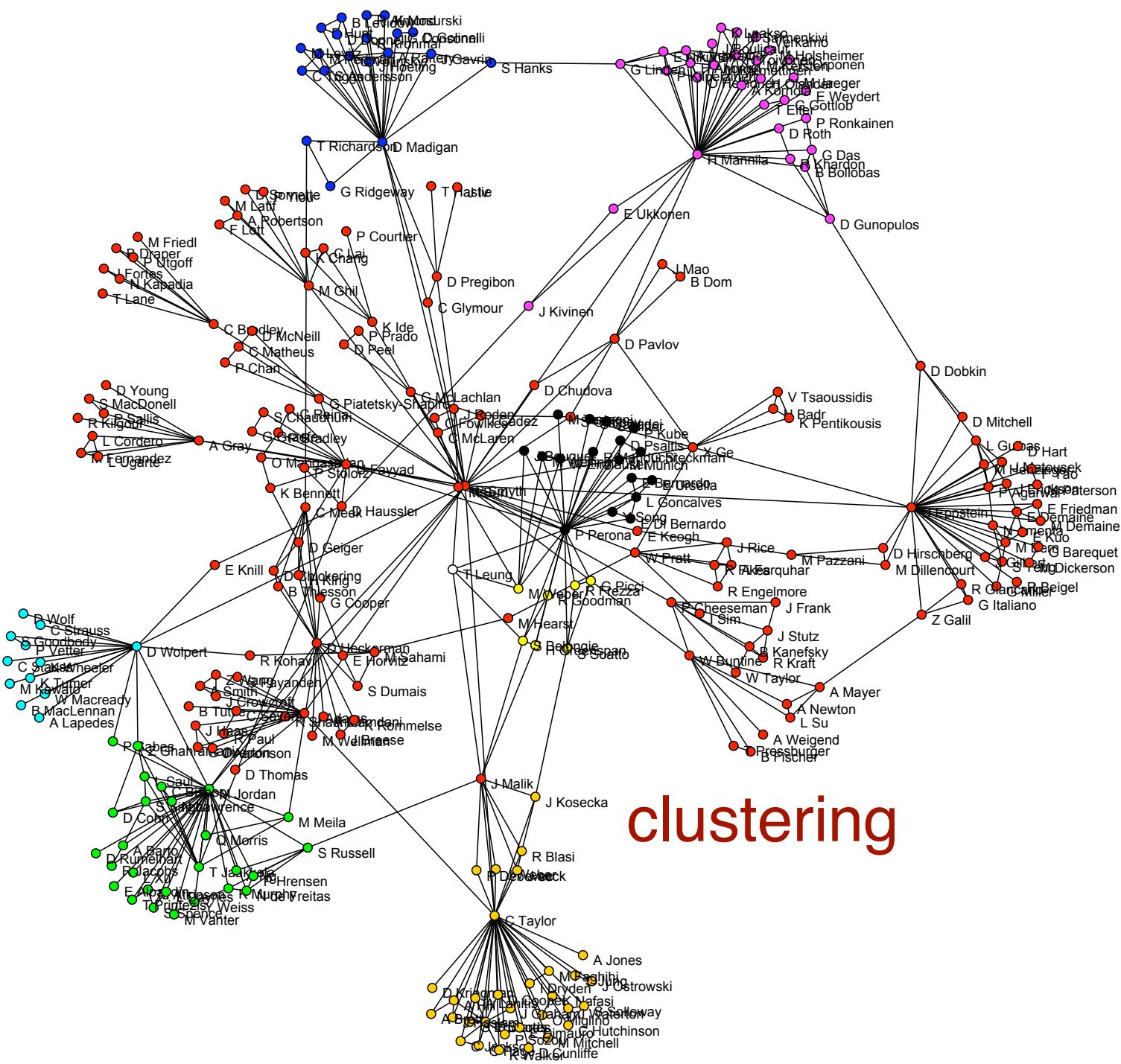
# PageRank,

hubs-and-authorities,  
voltage,

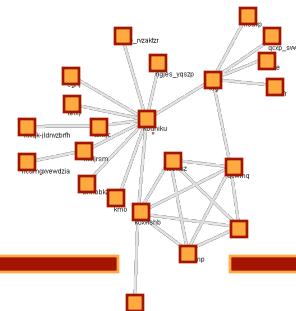
10



ranking



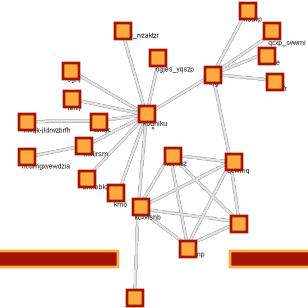
# Event Handling



- addition/removal of vertices/edges
    - graphs generate events
  - modification of user data repository
    - (JUNG) objects generate events
  - mouse events
    - can be parsed as selection, panning, zooming, pop-ups, property changes, functions, ...
  - objects can register as listeners

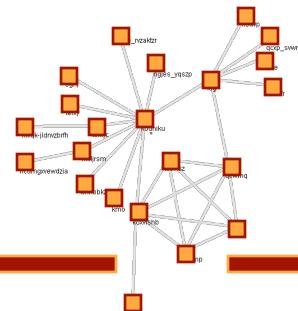
# Statistical Analysis

---



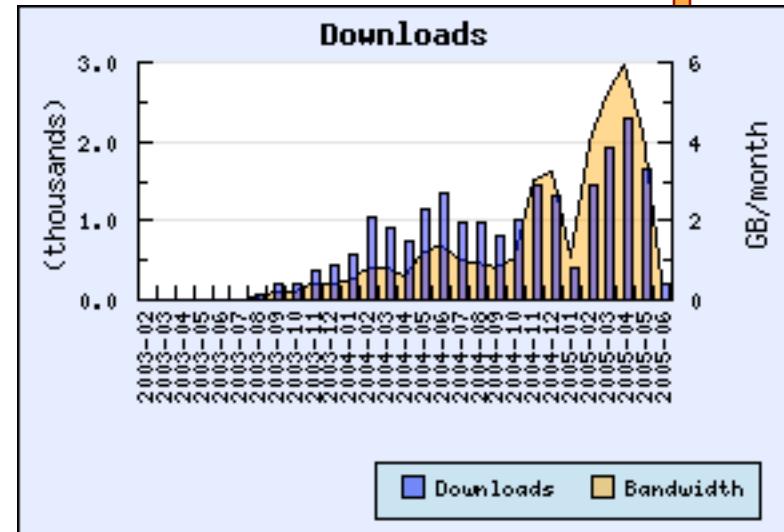
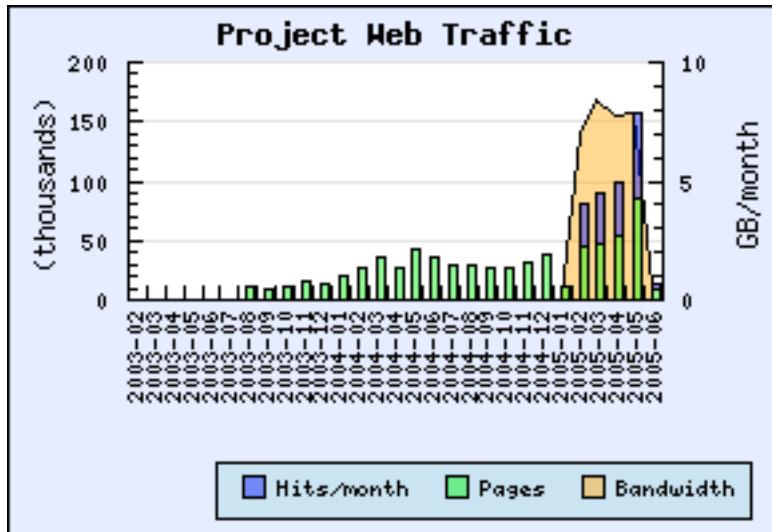
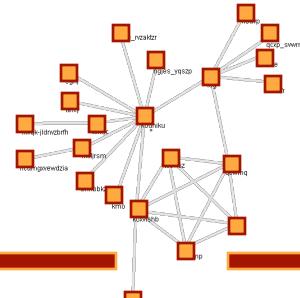
- Graph measures:
  - Degree distributions
  - Average shortest path length, diameter
  - Clustering coefficients
- can use existing Java statistical packages
  - CERN Colt Scientific Library
  - Visual Numeric's JMSL for Java

# Conclusion



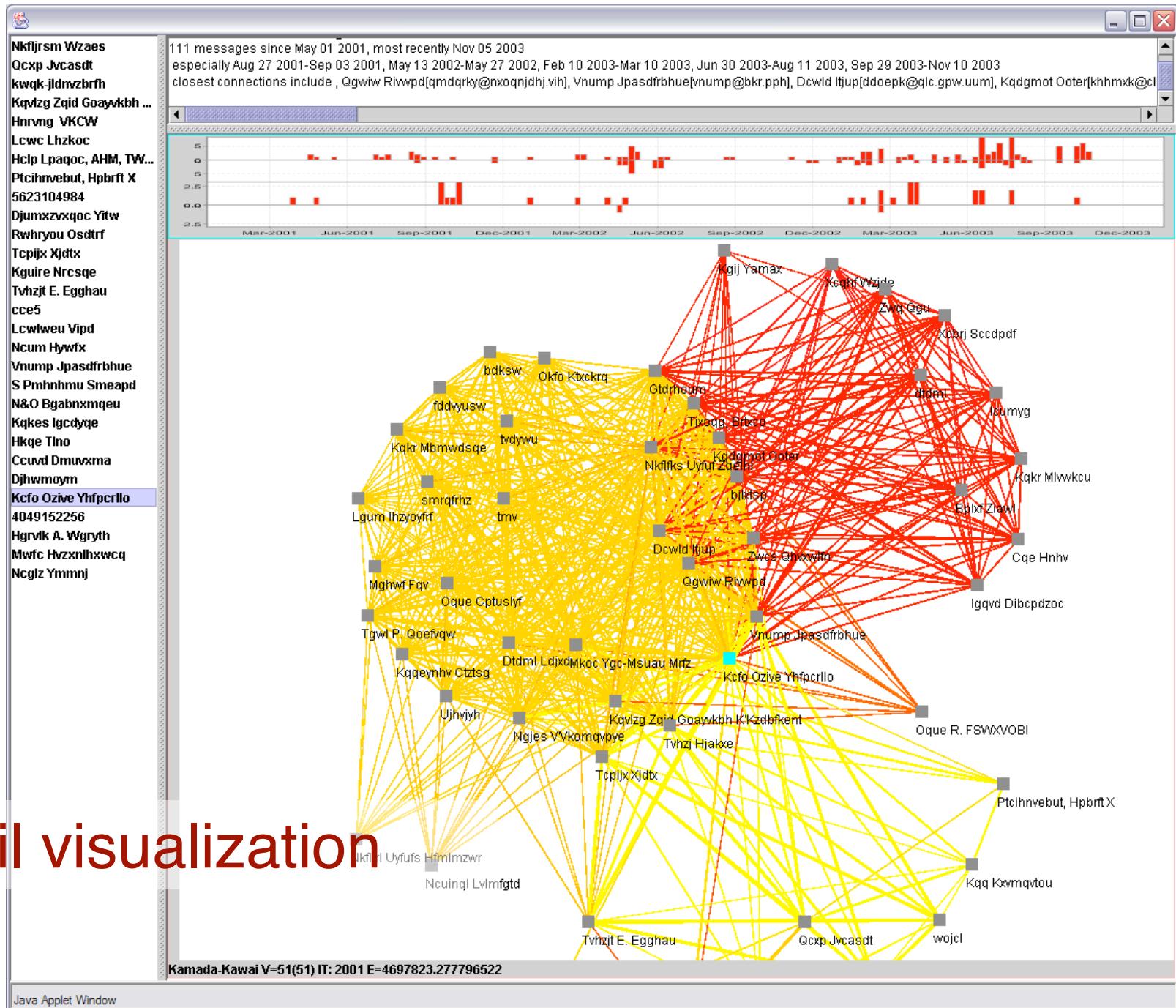
- Provides a common language for graphs
  - Complements rather than replaces other graph packages/network tools
  - A powerful framework for working with graphs with rich attribute structure
  - Ideally suited for building tools/applications related to network exploration and data mining

# Adoption



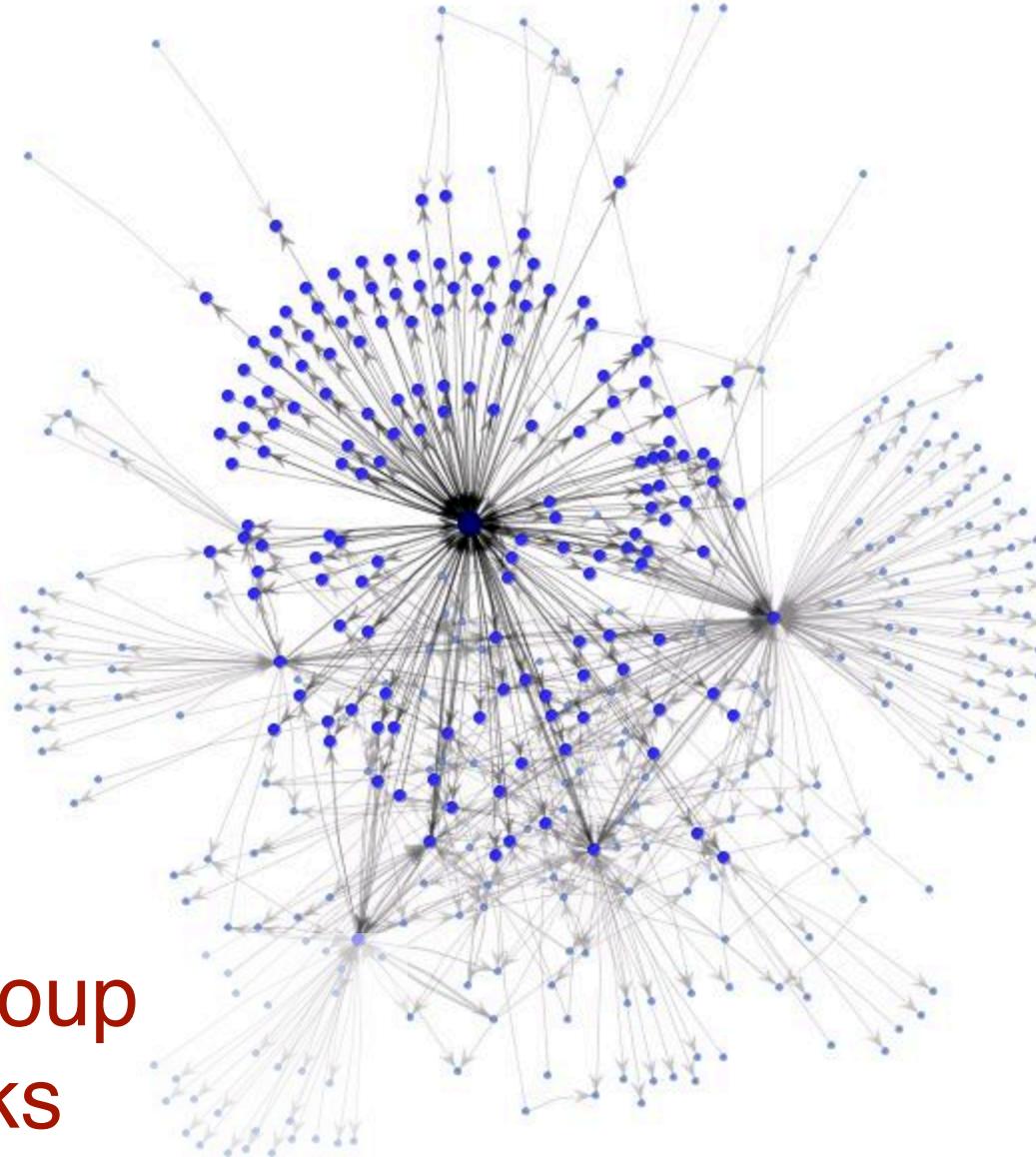
Users are writing

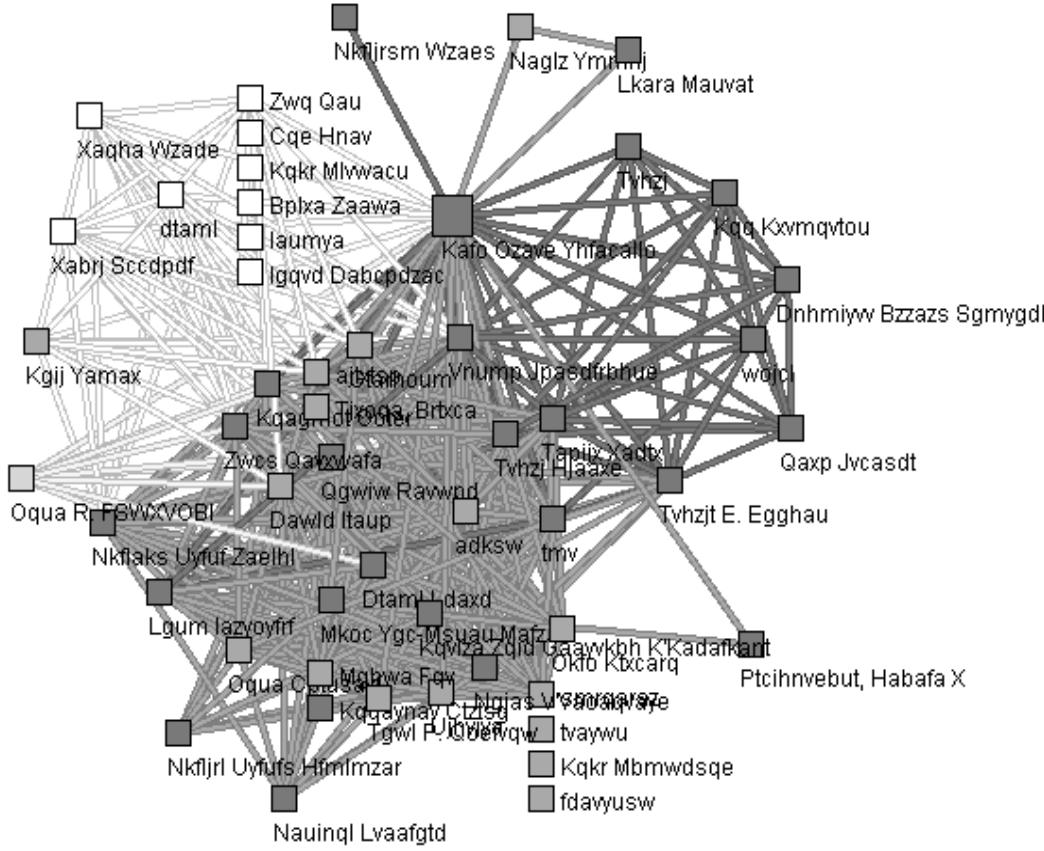
social network analyses, games, animated  
graph views, transportation network  
algorithms, Google Cartography, ...



# Email visualization

# Newsgroup Networks





# Temporal Egocentric Communication Networks



Display By: Age

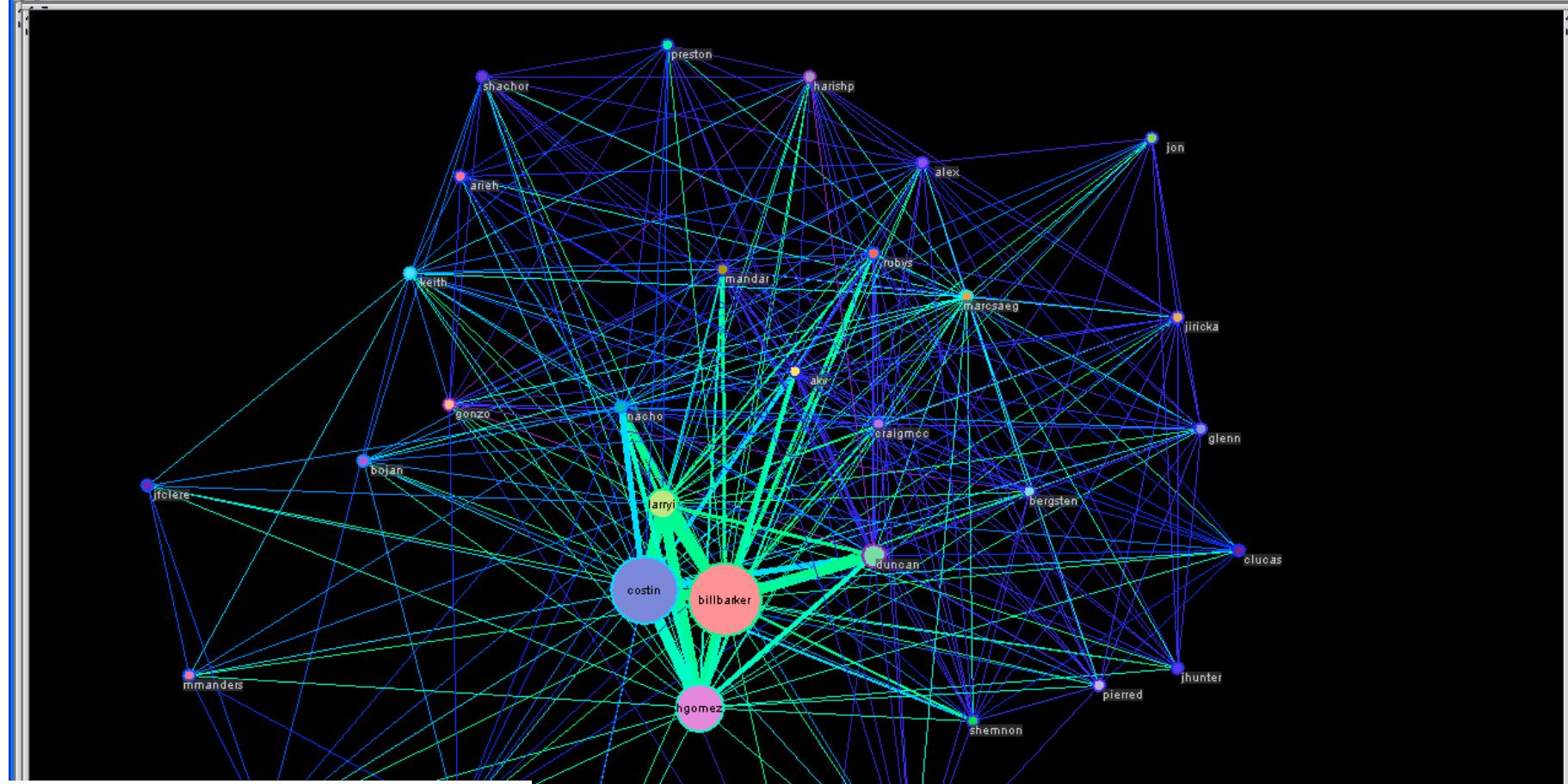
Indented: 

Size:

Time:

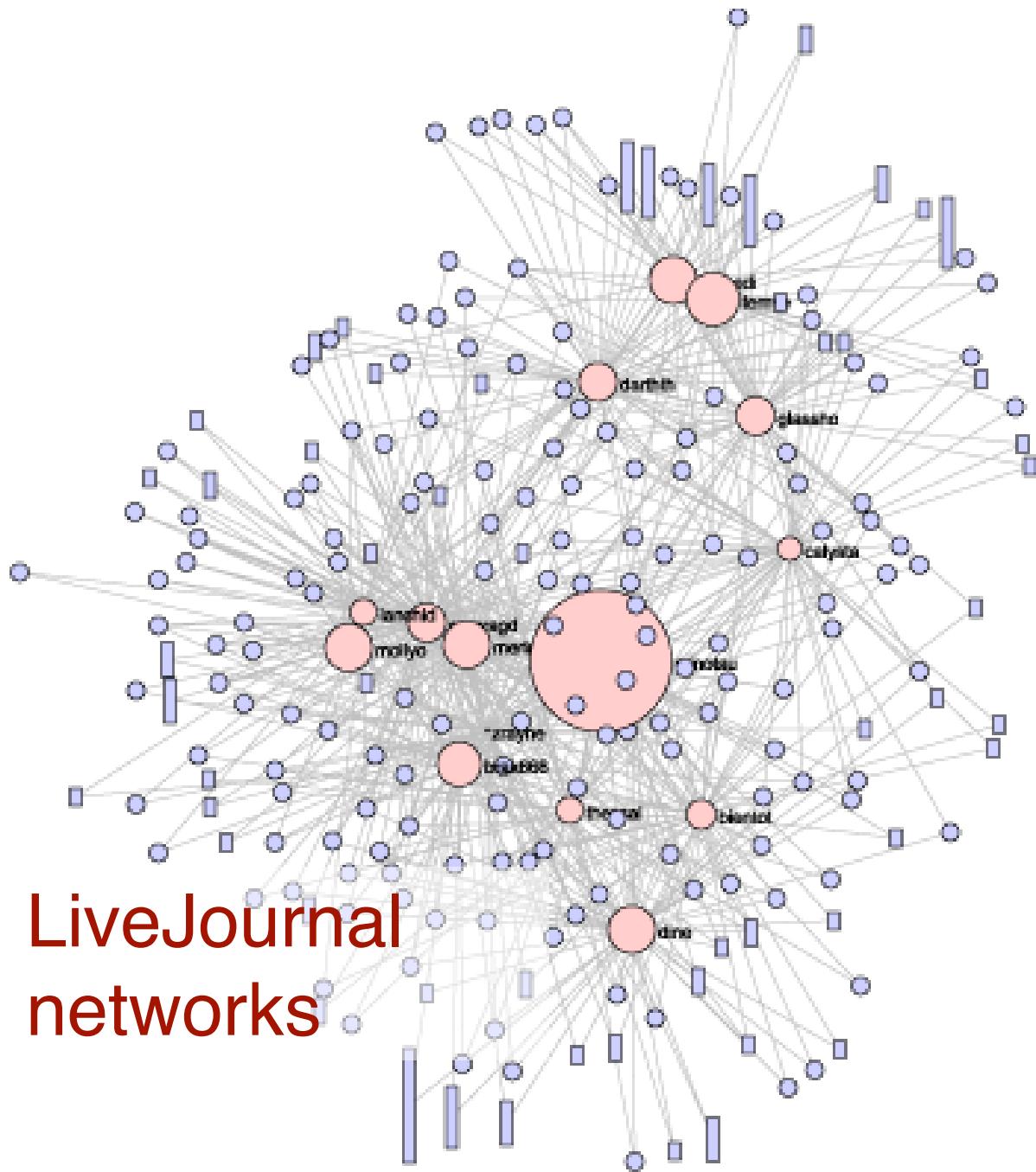
Past 4 years, 6 months

03/12/04 3.7 months 7.3 months 11 months 14.6 months 18.3 months 22 months 26.6 months 29.3 months 33 months 3 years 40.3 months 43.9 months 47.6 months 51.3 months 09/07/04

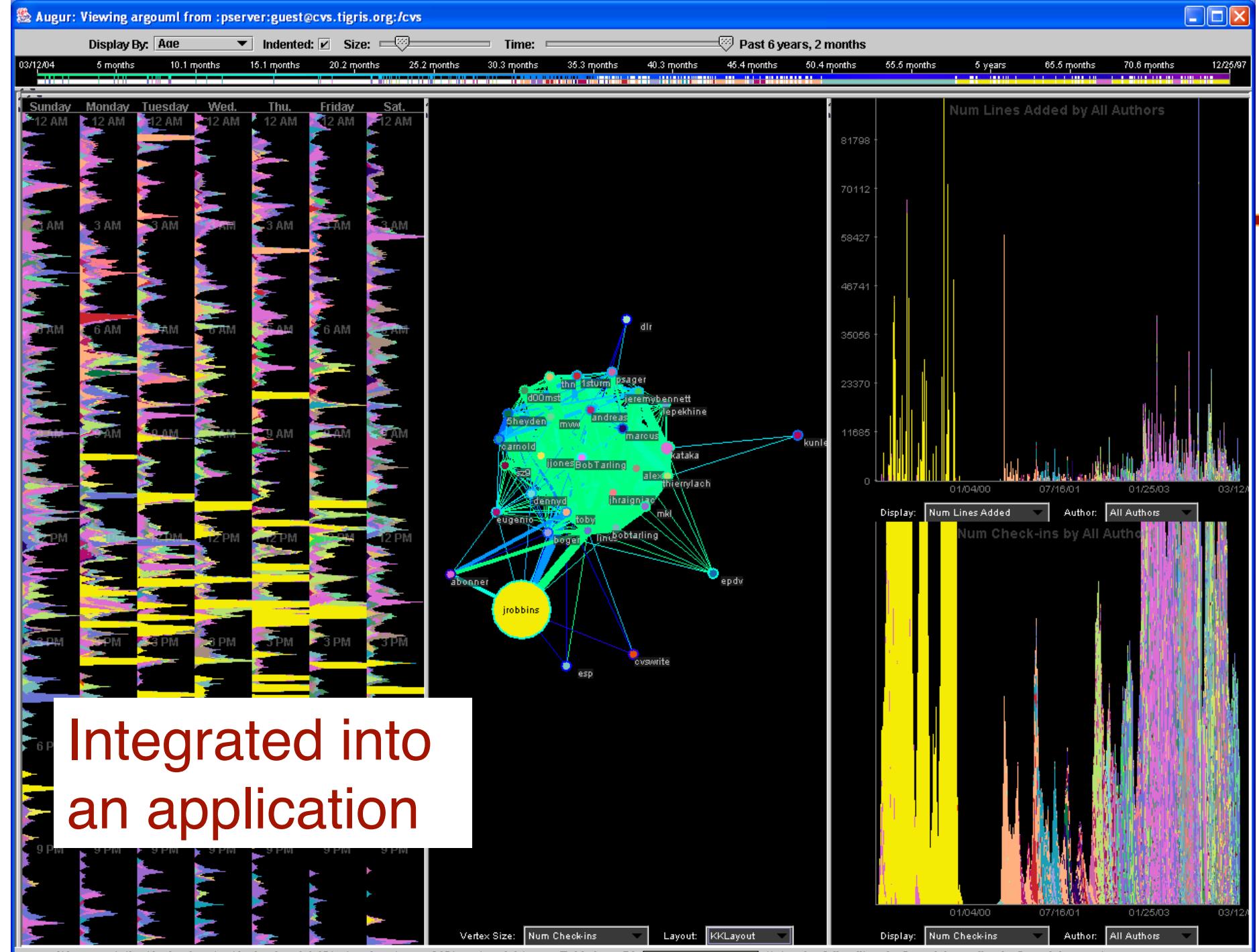


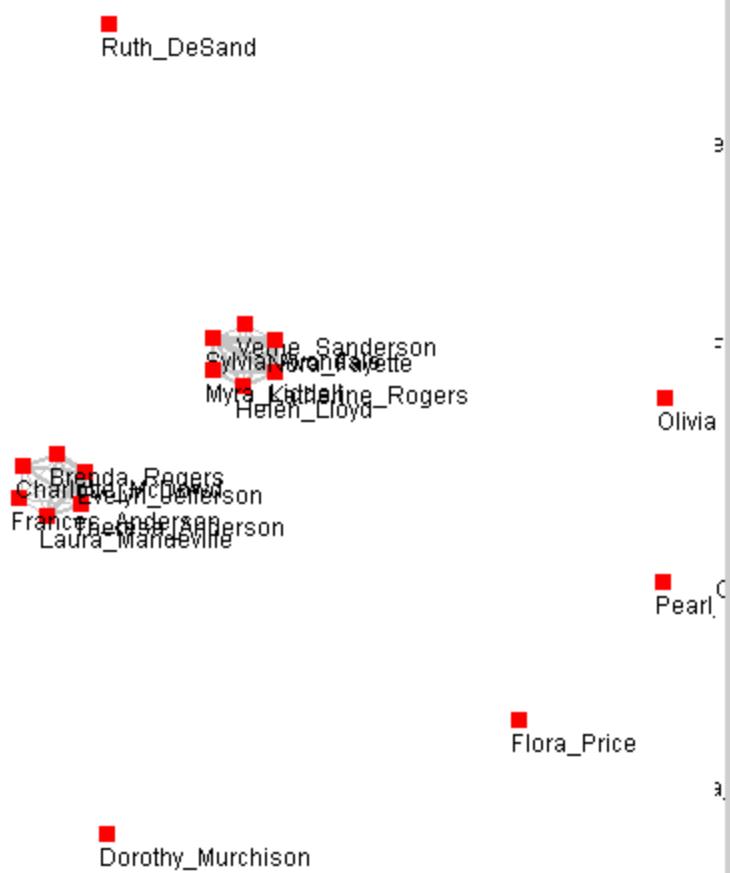
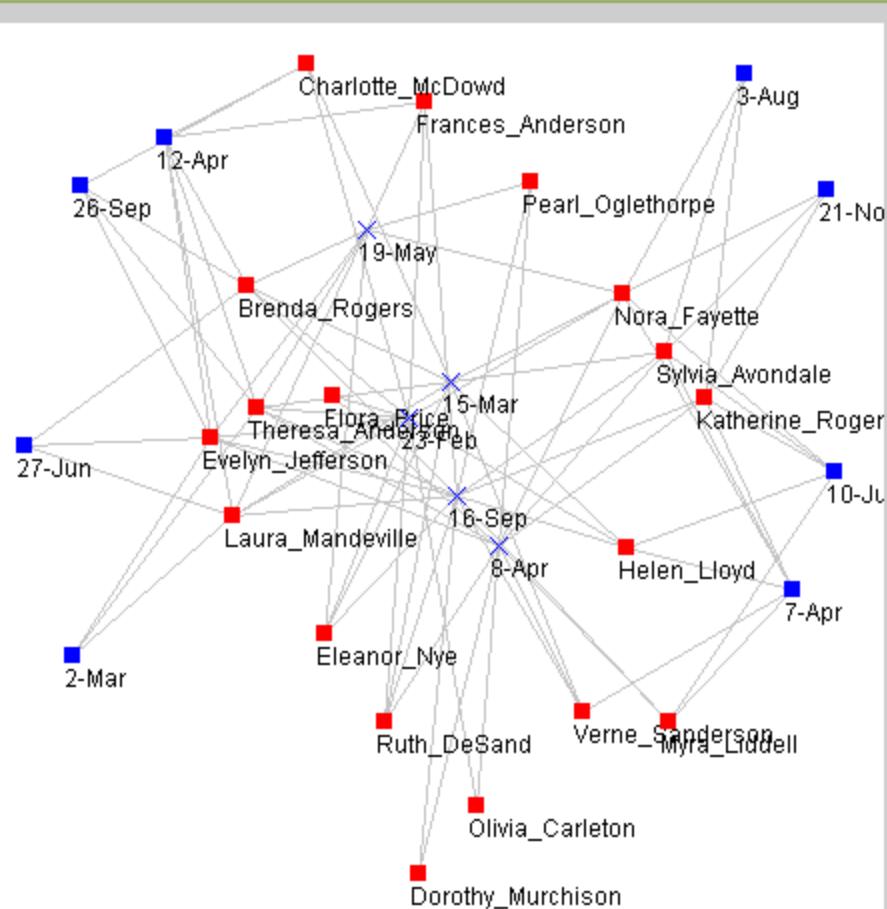
# Open-Source Collaboration

Vertex Size: Num Files Worked On Layout: KKLayout



# LiveJournal networks



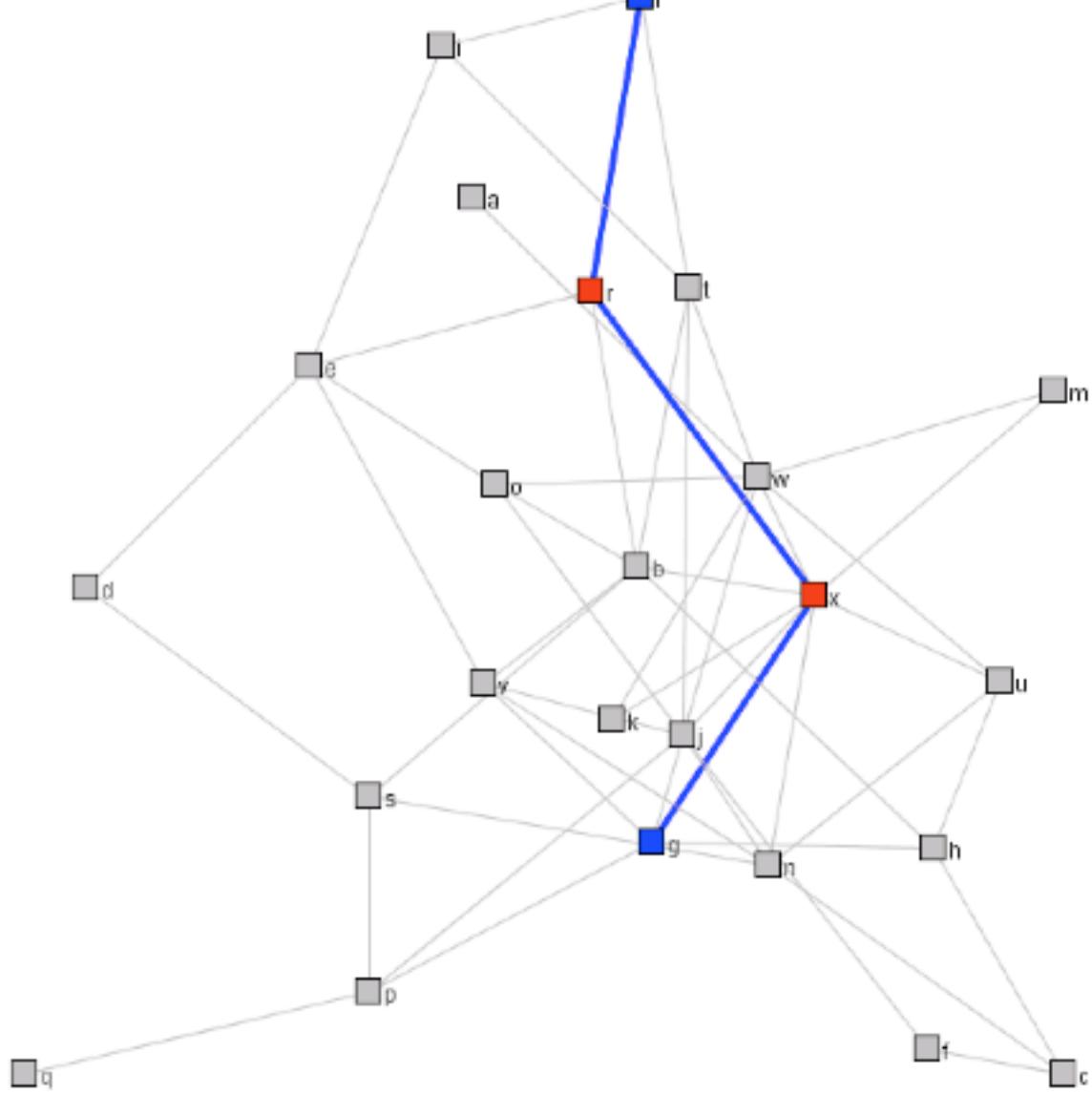


Java Applet Window

# Freeman's Southern Women

23-F  
2-Ma  
15-M  
7-Ap  
8-Ap  
12-A  
19-M  
10-J  
27-J  
3-Au  
16-S  
26-S  
21-N

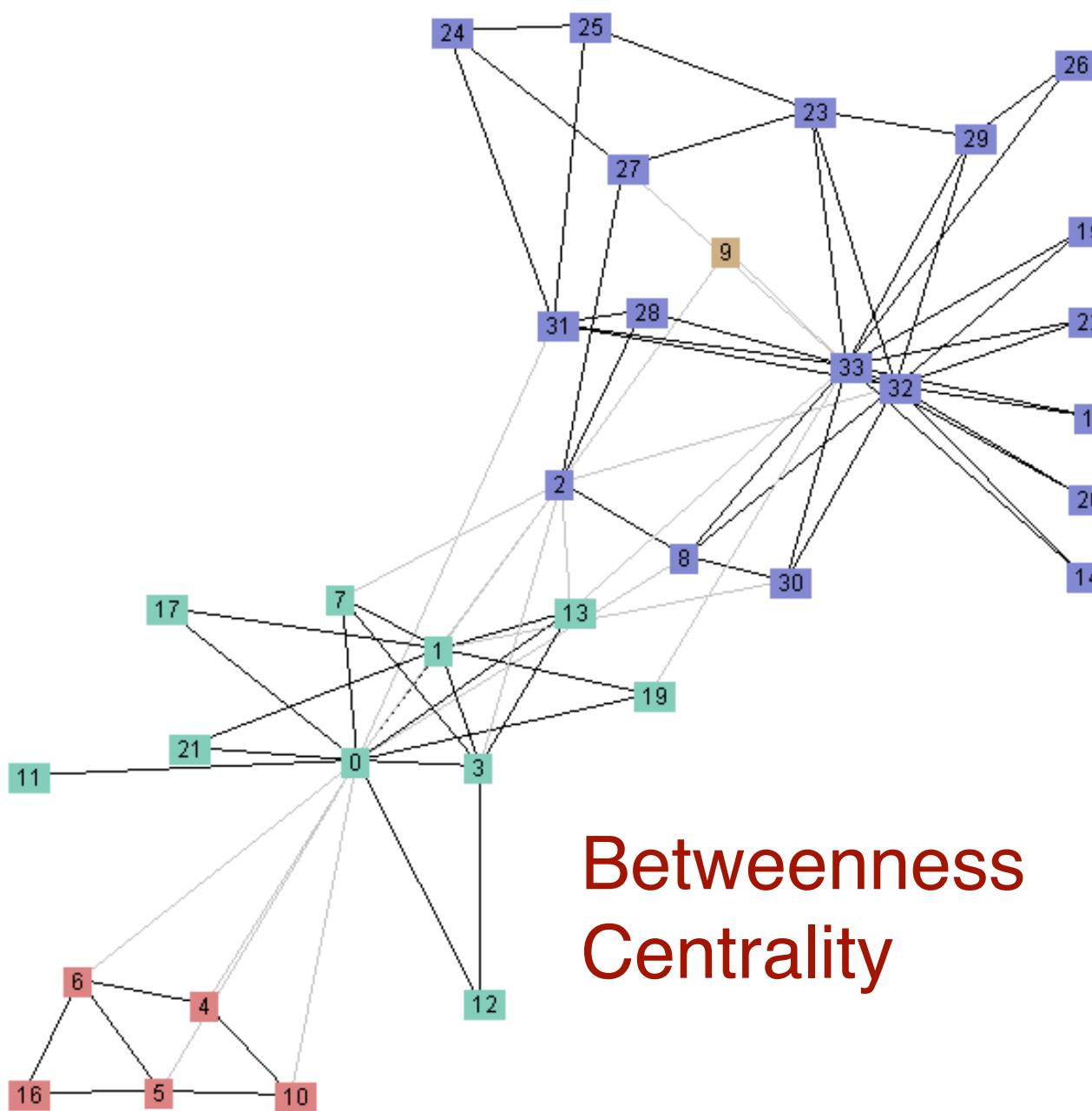
# Shortest Path



Select a pair of vertices to calculate a shortest path

vertex from

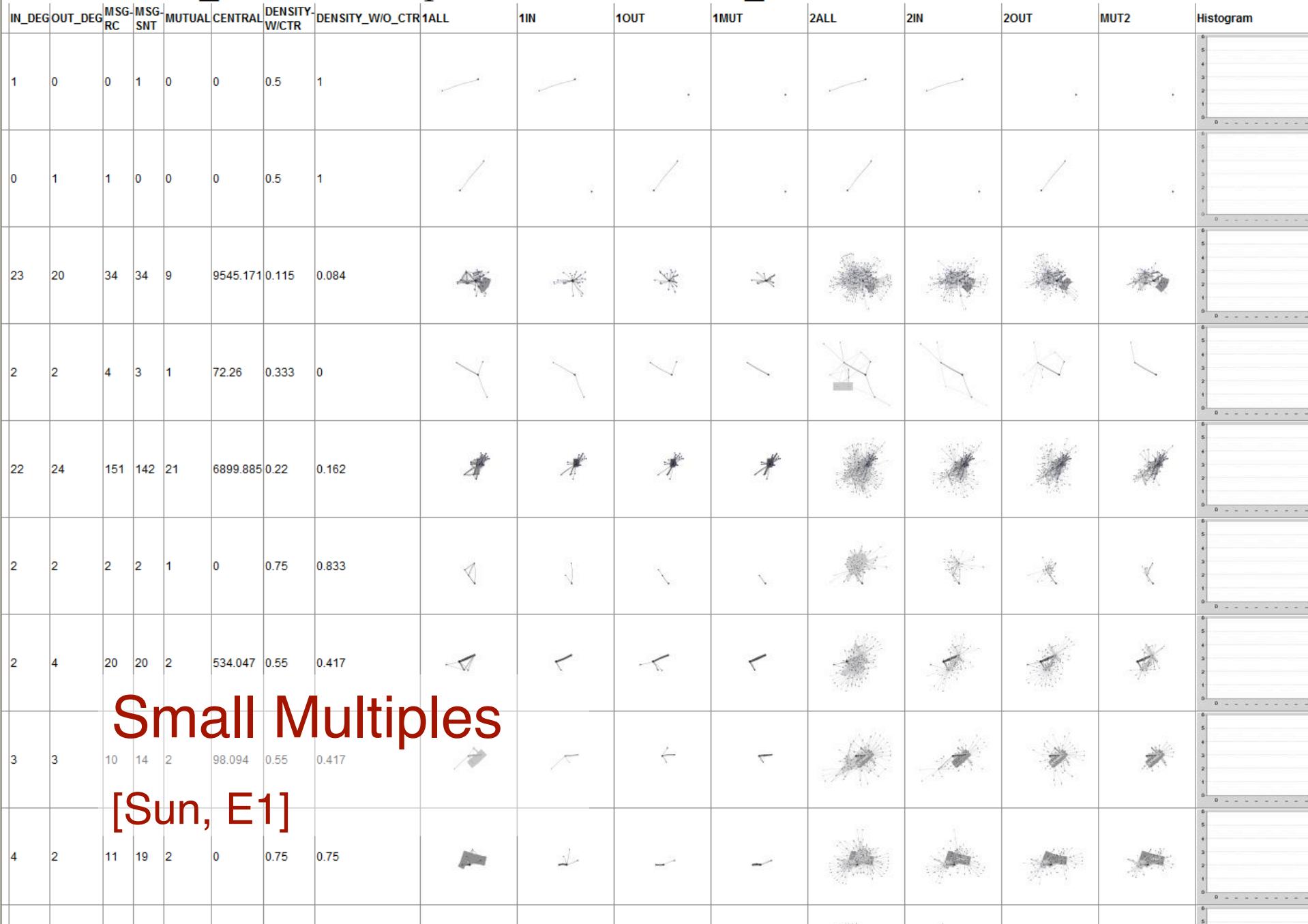
vertex to



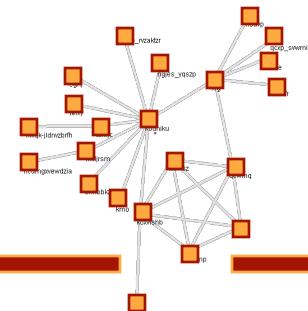
# Betweenness Centrality

[Restart](#)

# AuthToAuth\_AltAlienVampireFlonkFlonkFlonk\_Jan2001.txt



# Thanks



<http://jung.sourceforge.net>

**Feel free to just play with it...**

Or join us! Active collaboration is welcome.

Supported in part by NSF.