Subsampling Graphs

RECAP OF PAGERANK-NIBBLE

Why I'm talking about graphs

- Lots of large data *is* graphs
  - Facebook, Twitter, citation data, and other *social* networks
  - The web, the blogosphere, the semantic web, Freebase, Wikipedia, Twitter, and other *information* networks
  - Text corpora (like RCV1), large datasets with discrete feature values, and other *bipartite* networks
    - nodes = documents or words
    - links connect document → word or word → document
  - Computer networks, biological networks (proteins, ecosystems, brains, …), …
  - Heterogeneous networks with multiple types of nodes
    - people, groups, documents

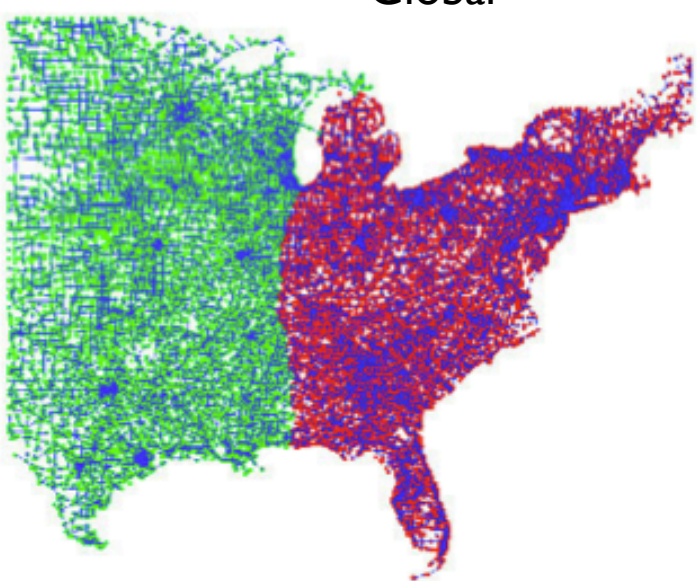
Our first operation on graphs

- Local graph partitioning
- Why?
  - it's about the best we can do in terms of subsampling/exploratory analysis of a graph

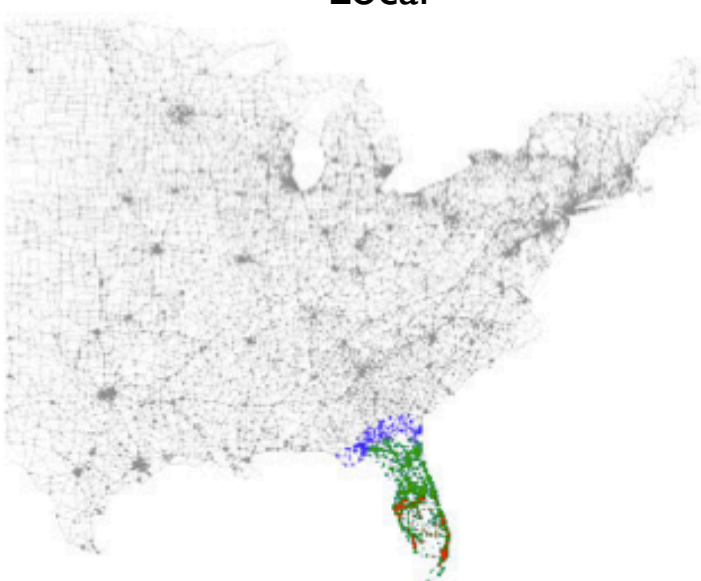
What is Local Graph Partitioning?

A local graph partitioning algorithm finds a small cut near the given seed(s) with running time depending only on the size of the output.

Global



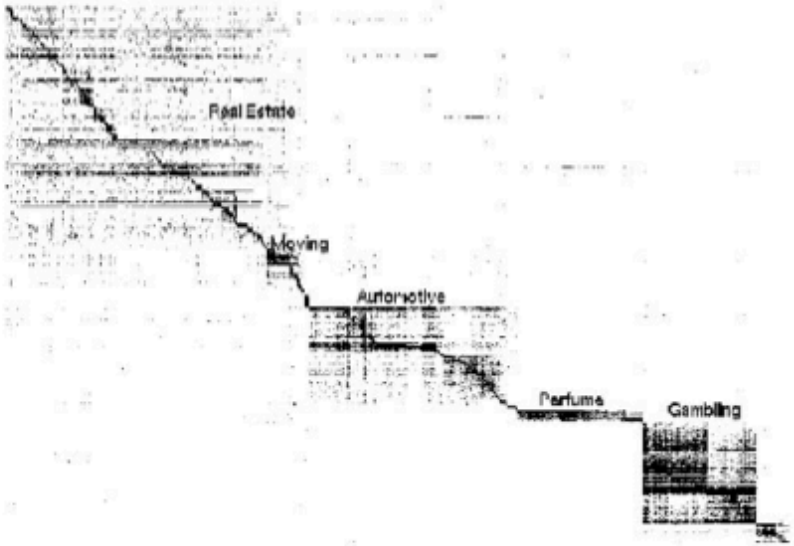
Local



What is Local Graph Partitioning?

Submarkets in the bigging graph

The bidding graph has numerous submarkets, related to real estate, flower delivery, hotels, gambling, ...



It is useful to identify these submarkets.

- Find groups of related phrases to suggest to advertisers.
- Find small submarkets for testing and experimentation.

Main results of the paper (Reid et al)

1. An *approximate* personalized PageRank computation that only touches nodes "near" the seed
   - but has small error relative to the true PageRank vector
2. A proof that a *sweep* over the approximate personalized PageRank vector finds a cut with conductance sqrt(α ln m)
   - unless no good cut exists
     - no subset S contains significantly more mass in the approximate PageRank than in a uniform distribution

Approximate PageRank: Key Idea

$$\mathrm{pr}(\alpha, s) = \alpha s + (1-\alpha)\mathrm{pr}(\alpha, sW). \quad W = \frac{I+P}{2}$$

$\mathtt{push}_u(p, r)$:

1. Let $p' = p$ and $r' = r$, except for the following changes:
   (a) $p'(u) = p(u) + \alpha r(u)$.
   (b) $r'(u) = (1-\alpha)r(u)/2$.
   (c) For each v such that $(u, v) \in E$: $\quad r'(v) = r(v) + (1-\alpha)r(u)/(2d(u))$.
2. Return (p', r').

- *p* is current approximation (start at **0**)
- *r* is set of "recursive calls to make"
  - residual error
  - start with all mass on *s*
- *u* is the node picked for the next call

ANALYSIS OF PAGERANK-NIBBLE

Analysis

Lemma 1. *Let p' and r' be the result of the operation* $\texttt{push}_u$ *on p and r. Then*

$$p' + \mathrm{pr}(\alpha, r') = p + \mathrm{pr}(\alpha, r).$$

Proof of Lemma 1. After the push operation, we have

$$\begin{aligned} p' &= p + \alpha r(u)\chi_u. \\ r' &= r - r(u)\chi_u + (1-\alpha)r(u)\chi_u W. \end{aligned}$$

Using equation (5),

$$\begin{aligned} p + \mathrm{pr}(\alpha, r) &= p + \mathrm{pr}(\alpha, r - r(u)\chi_u) + \underline{\mathrm{pr}(\alpha, r(u)\chi_u)} \\ &= p + \mathrm{pr}(\alpha, r - r(u)\chi_u) + \underline{[\alpha r(u)\chi_u + (1-\alpha)\mathrm{pr}(\alpha, r(u)\chi_u W)]} \\ &= [p + \alpha r(u)\chi_u] + \mathrm{pr}(\alpha, [r - r(u)\chi_u + (1-\alpha)r(u)\chi_u W]) \\ &= p' + \mathrm{pr}(\alpha, r'). \end{aligned}$$

linearity

re-group & linearity

$$\mathrm{pr}(\alpha, s) = \alpha s + (1-\alpha)\mathrm{pr}(\alpha, sW). \qquad (5)$$

Approximate PageRank: Key Idea

By definition PageRank is fixed point of:

$$\mathrm{pr}(\alpha, s) = \alpha s + (1-\alpha)\mathrm{pr}(\alpha, s)W,$$

Claim:

$$\mathrm{pr}(\alpha, s) = \alpha s + (1-\alpha)\mathrm{pr}(\alpha, sW).$$

Proof:

$$\begin{aligned} R_\alpha &= \alpha \sum_{t=0}^{\infty} (1-\alpha)^t W^t \\ &= \alpha I + (1-\alpha) W R_\alpha. \end{aligned}$$

$$\begin{aligned} \mathrm{pr}(\alpha, s) &= s R_\alpha \\ &= \alpha s + (1-\alpha) \underline{sW} R_\alpha \\ &= \alpha s + (1-\alpha)\mathrm{pr}(\alpha, sW). \end{aligned}$$

Approximate PageRank: Algorithm

`ApproximatePageRank` (v, α, ϵ):

1. Let $p = \vec{0}$, and $r = \chi_v$.
2. While $\max_{u \in V} \frac{r(u)}{d(u)} \geq \epsilon$:
   (a) Choose any vertex u where $\frac{r(u)}{d(u)} \geq \epsilon$.
   (b) Apply $\texttt{push}_u$ at vertex u, updating p and r.
3. Return p, which satisfies $p = \mathrm{apr}(\alpha, \chi_v, r)$ with $\max_{u \in V} \frac{r(u)}{d(u)} < \epsilon$.

$\texttt{push}_u(p, r)$:

1. Let $p' = p$ and $r' = r$, except for the following changes:
   (a) $p'(u) = p(u) + \alpha r(u)$.
   (b) $r'(u) = (1 - \alpha) r(u)/2$.
   (c) For each v such that $(u, v) \in E$: $\quad r'(v) = r(v) + (1 - \alpha) r(u)/(2d(u))$.
2. Return (p', r').

Analysis

Lemma 1. *Let p' and r' be the result of the operation* $\texttt{push}_u$ *on p and r. Then*

$$p' + \mathrm{pr}(\alpha, r') = p + \mathrm{pr}(\alpha, r).$$

So, at every point in the *apr* algorithm:

$$p + \mathrm{pr}(\alpha, r) = \mathrm{pr}(\alpha, \chi_v),$$

Also, at each point, $|r|_1$ **decreases** by $\alpha * \varepsilon$ *degree(u), so: after T push operations where degree(i-th u)=d_i, we know

$$\sum_i d_i \cdot \alpha\varepsilon \leq 1 \quad \Rightarrow \quad \sum_{i=1}^{T} d_i \leq \frac{1}{\epsilon\alpha}.$$

which bounds the size of r and p

Analysis

Theorem 1. `ApproximatePageRank`(v, α, ϵ) *runs in time* $O(\frac{1}{\epsilon\alpha})$, *and computes an approximate PageRank vector* $p = \mathrm{apr}(\alpha, \chi_v, r)$ *such that the residual vector* r *satisfies* $\max_{u \in V} \frac{r(u)}{d(u)} < \epsilon$, *and such that* $\mathrm{vol}(\mathrm{Supp}(p)) \leq \frac{1}{\epsilon\alpha}$.

With the invariant: $$p + \mathrm{pr}(\alpha, r) = \mathrm{pr}(\alpha, \chi_v),$$

This bounds the error of *p* relative to the PageRank vector.

Comments – API

`ApproximatePageRank` (v, α, ϵ):

1. Let $p = \vec{0}$, and $r = \chi_v$.
2. While $\max_{u \in V} \frac{r(u)}{d(u)} \geq \epsilon$:
   (a) Choose any vertex u where $\frac{r(u)}{d(u)} \geq \epsilon$.
   (b) Apply $\texttt{push}_u$ at vertex u, updating p and r.
3. Return p, which satisfies $p = \mathrm{apr}(\alpha, \chi_v, r)$ with $\max_{u \in V} \frac{r(u)}{d(u)} < \epsilon$.

p,r are hash tables – they are small (1/εα)

Could implement with API:

- List<Node> neighbor(Node *u*)
- int degree(Node *u*)

$\texttt{push}_u(p, r)$:

1. Let $p' = p$ and $r' = r$, except for the following changes:
   (a) $p'(u) = p(u) + \alpha r(u)$.
   (b) $r'(u) = (1 - \alpha) r(u)/2$.
   (c) For each v such that $(u, v) \in E$: $r'(v) = r(v) + (1 - \alpha) r(u)/(2d(u))$.
2. Return (p', r').

push just needs *p, r,* and neighbors of *u*

d(v) = api.degree(*v*)

Comments - Ordering

`ApproximatePageRank` (v, α, ϵ):

1. Let $p = \vec{0}$, and $r = \chi_v$.
2. While $\max_{u \in V} \frac{r(u)}{d(u)} \geq \epsilon$:
   (a) Choose any vertex u where $\frac{r(u)}{d(u)} \geq \epsilon$.
   (b) Apply $\texttt{push}_u$ at vertex u, updating p and r.
3. Return p, which satisfies $p = \mathrm{apr}(\alpha, \chi_v, r)$ with $\max_{u \in V} \frac{r(u)}{d(u)} < \epsilon$.

might pick the **largest** *r(u)/d(u)* ... ***or...***

$\texttt{push}_u(p, r)$:

1. Let $p' = p$ and $r' = r$, except for the following changes:
   (a) $p'(u) = p(u) + \alpha r(u)$.
   (b) $r'(u) = (1 - \alpha) r(u)/2$.
   (c) For each v such that $(u, v) \in E$: $r'(v) = r(v) + (1 - \alpha) r(u)/(2d(u))$.
2. Return (p', r').

Comments – Ordering for Scanning

`ApproximatePageRank` (v, α, ϵ):

1. Let $p = \vec{0}$, and $r = \chi_v$.
2. While $\max_{u \in V} \frac{r(u)}{d(u)} \geq \epsilon$:

Scan <u>repeatedly</u> through an adjacency-list encoding of the graph

For every line you read $u, v_1, \ldots, v_{d(u)}$ such that $r(u)/d(u) > \varepsilon$:

(b) Apply $\texttt{push}_u$ at vertex u, updating p and r.

3. Return p, which satisfies $p = \mathrm{apr}(\alpha, \chi_v, r)$ with $\max_{u \in V} \frac{r(u)}{d(u)} < \epsilon$.

benefit: storage is $O(1/\varepsilon\alpha)$ for the hash tables, avoids any *seeking*

Possible optimizations?

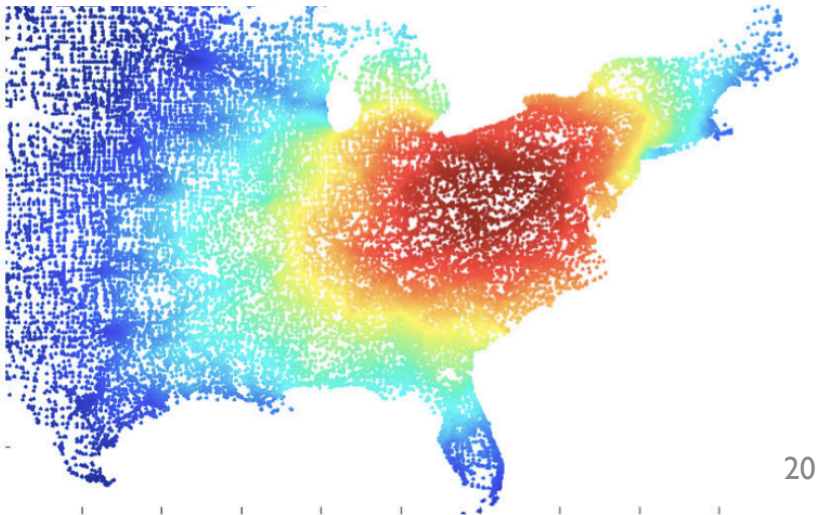
- Much faster than doing random access the first few scans, but then slower the last few
  - …there will be only a few 'pushes' per scan
- Optimizations you might imagine:
  - Parallelize?
  - Hybrid seek/scan:
    - Index the nodes in the graph on the first scan
    - Start seeking when you expect too few pushes to justify a scan
      - Say, less than one push/megabyte of scanning
  - Hotspots:
    - Save adjacency-list representation for nodes with a large $r(u)/d(u)$ in a separate file of "hot spots" as you scan
    - Then rescan that smaller list of "hot spots" until their score drops below threshold.

After computing apr…

- Given a graph
  - that's too big for memory, and/or
  - that's only accessible via API
- …we can extract a *sample* in an interesting area
  - Run the *apr*/rwr from a seed node
  - *Sweep* to find a low-conductance subset
- Then
  - compute statistics
  - test out some ideas
  - visualize it…

Key idea: a "sweep"

- Order vertices v_1, v_2, ... by highest apr(s)
- Pick a prefix S={ v_1, v_2, ... v_k }:
  - S={}; volS =0; B={}
  - For k=1,...n:
    - S += {v_k}
    - volS += degree(v_k)
    - B += {u: v_k→u and u not in S}
    - Φ[k] = |B|/volS
  - Pick k: Φ[k] is smallest



Scoring a graph prefix

$$\partial(S) = \{\{x, y\} \in E \mid x \in S, y \notin S\}$$

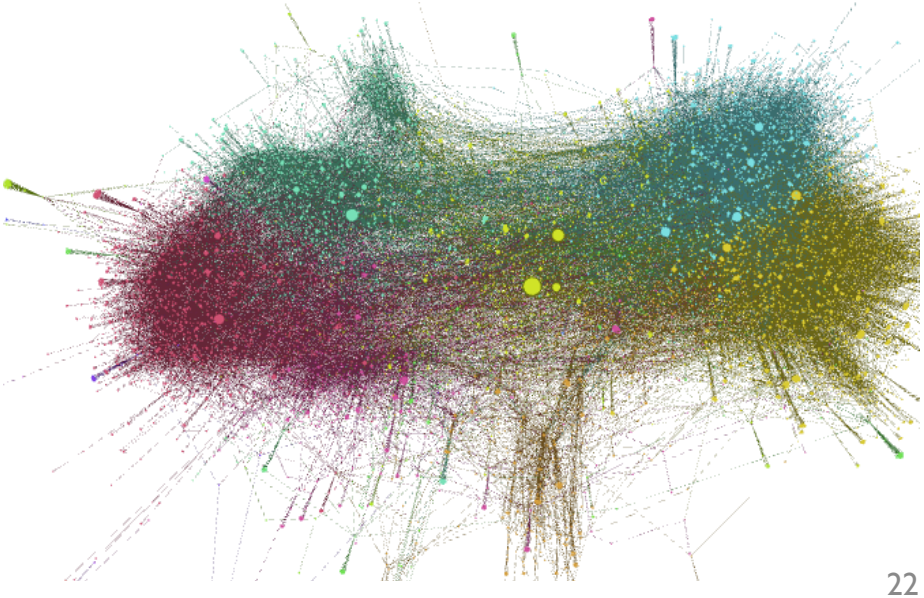
the edges leaving S

$$\Phi(S) = \frac{|\partial(S)|}{\min\left(\mathrm{vol}(S), 2m - \mathrm{vol}(S)\right)}.$$

- vol(S) is sum of deg(x) for x in S
- for small S: Prob(random *edge* leaves S)

Putting this together

- Given a graph
  - that's too big for memory, and/or
  - that's only accessible via API
- …we can extract a *sample* in an interesting area
  - Run the apr/rwr from a seed node
  - Sweep to find a low-conductance subset
- Then
  - compute statistics
  - test out some ideas
  - visualize it…



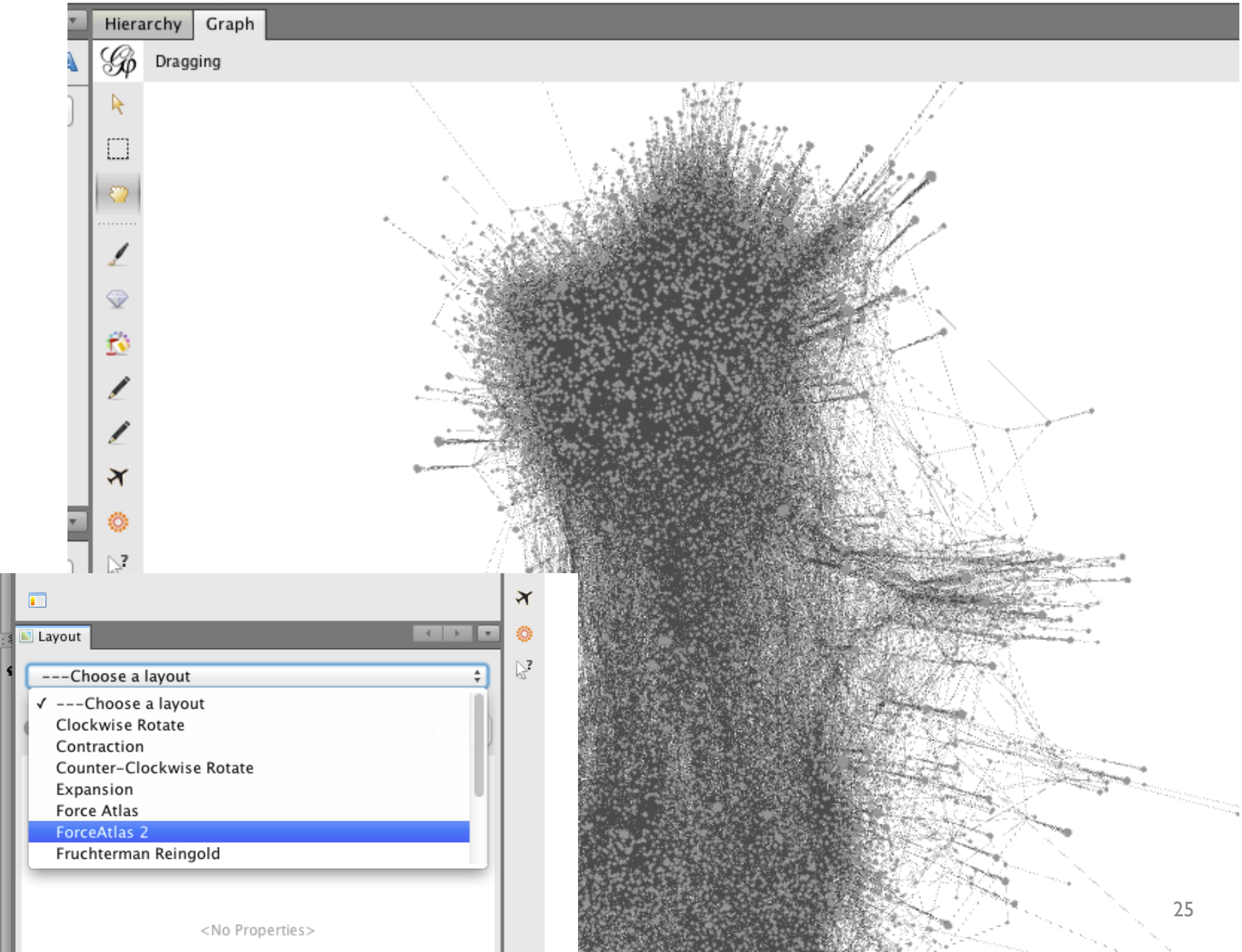
Visualizing a Graph with Gephi

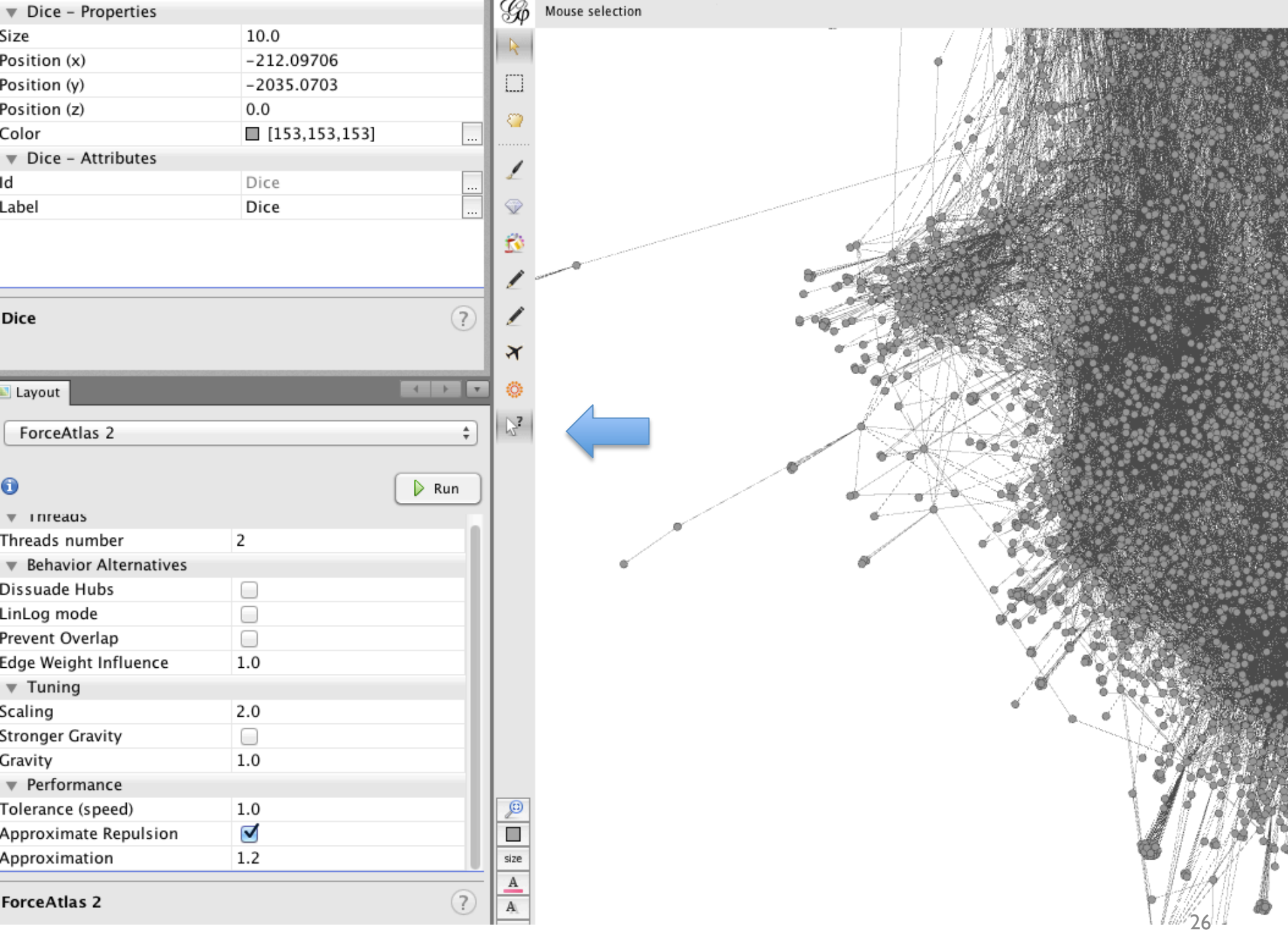
Screen shots/demo

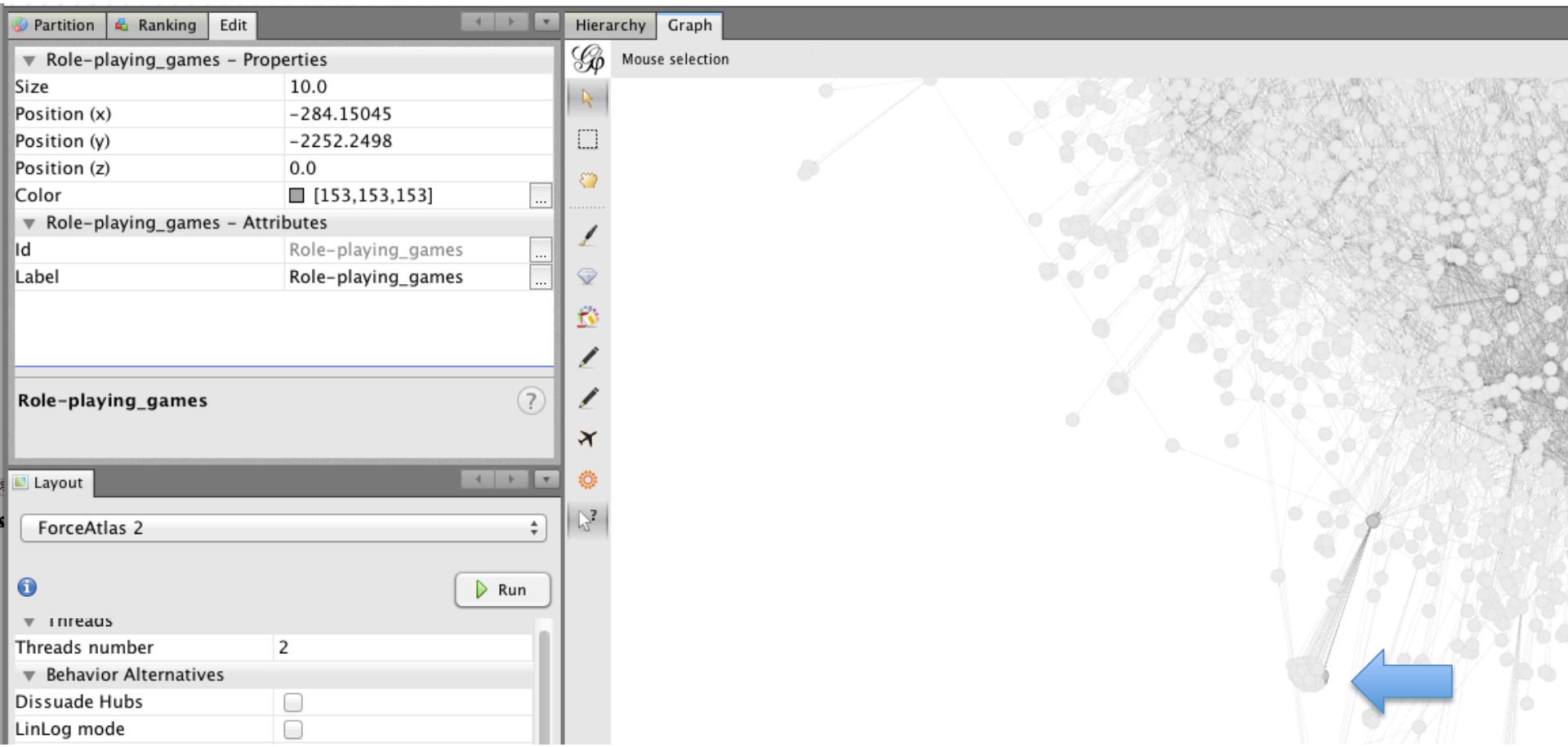
- Gelphi – java tool
- Reads several inputs
  - .csv file
  - [demo]

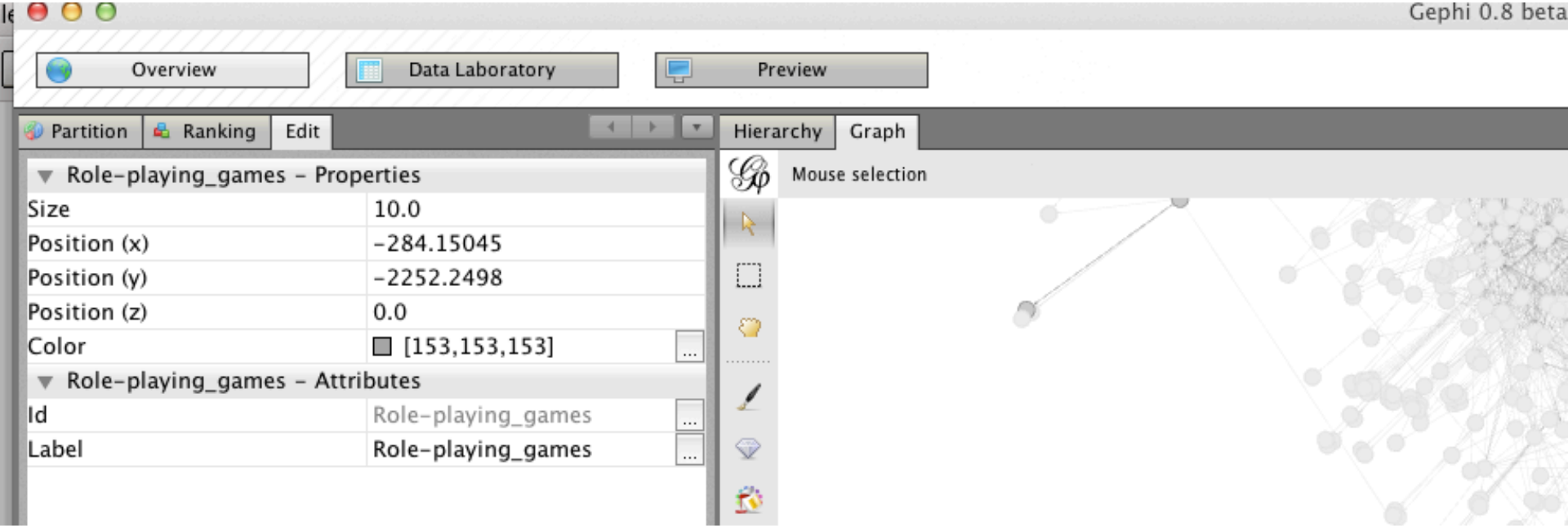
medium-crf-subgraph.csv

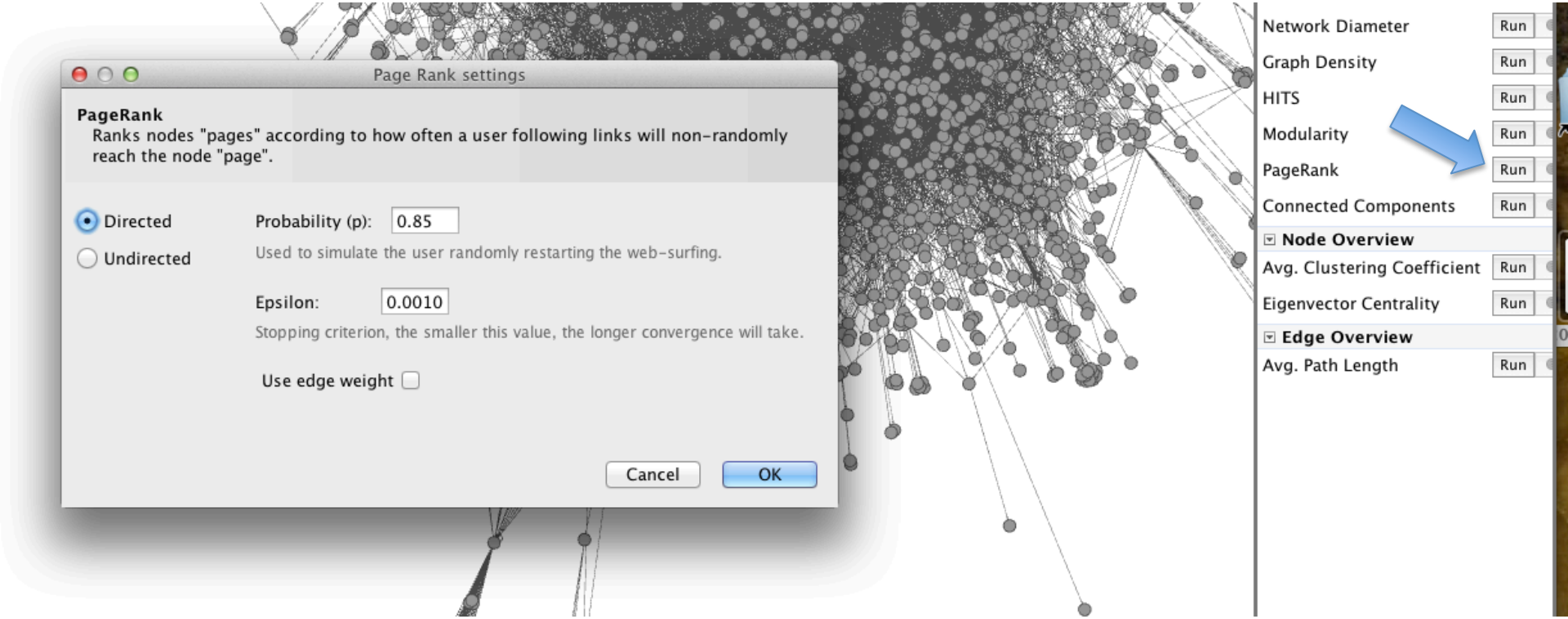
```
Arithmetic_mean;Means
Arithmetic_mean;Variance
Arithmetic_mean;Summary_statistics
Arithmetic_mean;Standard_deviation
Arithmetic_mean;Sample_size
Arithmetic_mean;Sample_mean_and_covariance
Arithmetic_mean;Median
Arithmetic_mean;Mean
Arithmetic_mean;Average
Arithmetic_mean;Average
Arithmetic_mean;Median
Arithmetic_mean;Law_of_large_numbers
Arithmetic_mean;Expected_value
Arithmetic_mean;Random_variable
Arithmetic_mean;Probability_distribution
Arithmetic_mean;Random_variable
Arithmetic_mean;Expected_value
Arithmetic_mean;Expected_value
Arithmetic_mean;Random_variable
Arithmetic_mean;Median
Arithmetic_mean;Average
Arithmetic_mean;Average
Arithmetic_mean;Statistic
Arithmetic_mean;Sampling_%28statistics%29
Arithmetic_mean;Statistical_population
Arithmetic_mean;Mean
Arithmetic_mean;Statistics
```


Hierarchy
Graph
Dragging
Layout
---Choose a layout
✓ ---Choose a layout
Clockwise Rotate
Contraction
Counter-Clockwise Rotate
Expansion
Force Atlas
ForceAtlas 2
Fruchterman Reingold
<No Properties>


Dice – Properties
Size 10.0
Position (x) -212.09706
Position (y) -2035.0703
Position (z) 0.0
Color [153,153,153]
Dice – Attributes
Id Dice
Label Dice
Dice
Layout
ForceAtlas 2
Run
Threads
Threads number 2
Behavior Alternatives
Dissuade Hubs
LinLog mode
Prevent Overlap
Edge Weight Influence 1.0
Tuning
Scaling 2.0
Stronger Gravity
Gravity 1.0
Performance
Tolerance (speed) 1.0
Approximate Repulsion
Approximation 1.2
ForceAtlas 2
Mouse selection
size


Partition
Ranking
Edit
Hierarchy
Graph
Role-playing_games - Properties
Size
10.0
Position (x)
-284.15045
Position (y)
-2252.2498
Position (z)
0.0
Color
[153,153,153]
Role-playing_games - Attributes
Id
Role-playing_games
Label
Role-playing_games
Role-playing_games
Mouse selection
Layout
ForceAtlas 2
Run
Threads number
2
Behavior Alternatives
Dissuade Hubs
LinLog mode


Gephi 0.8 beta
Overview
Data Laboratory
Preview
Partition
Ranking
Edit
Hierarchy
Graph
Role-playing_games - Properties
Size
10.0
Position (x)
-284.15045
Position (y)
-2252.2498
Position (z)
0.0
Color
[153,153,153]
Role-playing_games - Attributes
Id
Role-playing_games
Label
Role-playing_games
Mouse selection


Page Rank settings
PageRank
Ranks nodes "pages" according to how often a user following links will non-randomly reach the node "page".
Directed
Undirected
Probability (p):
0.85
Used to simulate the user randomly restarting the web-surfing.
Epsilon:
0.0010
Stopping criterion, the smaller this value, the longer convergence will take.
Use edge weight
Cancel
OK
Network Diameter
Graph Density
HITS
Modularity
PageRank
Connected Components
Node Overview
Avg. Clustering Coefficient
Eigenvector Centrality
Edge Overview
Avg. Path Length
Run

gephi File Workspace View Tools Window Plugins Help

Mon 1:50 PM

2012-03-20-rwr.pptx

Gephi 0.8 beta - Project 0

Overview | Data Laboratory | Preview

Edit

Role-playing_games - Properties

Size	10.0
Position (x)	-284.15045
Position (y)	-2252.2498
Position (z)	0.0
Color	[153,153,153]

Role-playing_games - Attributes

Id	Role-playing_games
Label	Role-playing_games
PageRank	4.769781822734278E-5

Role-playing_games

Data Table

Nodes | Edges | Configuration | Add node | Add edge | Clear graph | Clear edges | Search/Replace | Import Spreadsheet | Export table | Filte

Nodes	Id	Label	PageRank
Living_people	Living_people	Living_people	0.006
Mathematics	Mathematics	Mathematics	0.004
Computer_science	Computer_science	Computer_science	0.004
Statistics	Statistics	Statistics	0.003
Java_%28programming_language%29	Java_%28programming_language%29	Java_%28programming_language%29	0.003
C_%28programming_language%29	C_%28programming_language%29	C_%28programming_language%29	0.003
Programming_language	Programming_language	Programming_language	0.003
Compiler	Compiler	Compiler	0.003
Object-oriented_programming	Object-oriented_programming	Object-oriented_programming	0.003
Year_of_birth_missing_%28living_people%29	Year_of_birth_missing_%28living_people%29	Year_of_birth_missing_%28living_peop	0.002
C%2B%2B	C%2B%2B	C%2B%2B	0.002
Probability_theory	Probability_theory	Probability_theory	0.002
Real_number	Real_number	Real_number	0.002
Set_%28mathematics%29	Set_%28mathematics%29	Set_%28mathematics%29	0.002
Data_type	Data_type	Data_type	0.002
Computer	Computer	Computer	0.001
Computer_programming	Computer_programming	Computer_programming	0.001
Random_variable	Random_variable	Random_variable	0.001
Algorithm	Algorithm	Algorithm	0.001
Function_%28mathematics%29	Function_%28mathematics%29	Function_%28mathematics%29	0.001
Source_code	Source_code	Source_code	0.001
Operating_system	Operating_system	Operating_system	0.001
Machine_learning	Machine_learning	Machine_learning	0.001
Measure_%28mathematics%29	Measure_%28mathematics%29	Measure_%28mathematics%29	0.001
Python_%28programming_language%29	Python_%28programming_language%29	Python_%28programming_language%2	0.001
Probability	Probability	Probability	0.001
Subroutine	Subroutine	Subroutine	0.001
Probability_distribution	Probability_distribution	Probability_distribution	0.001
Fortran	Fortran	Fortran	0.001
Computer_program	Computer_program	Computer_program	0.001
Subset	Subset	Subset	0.001

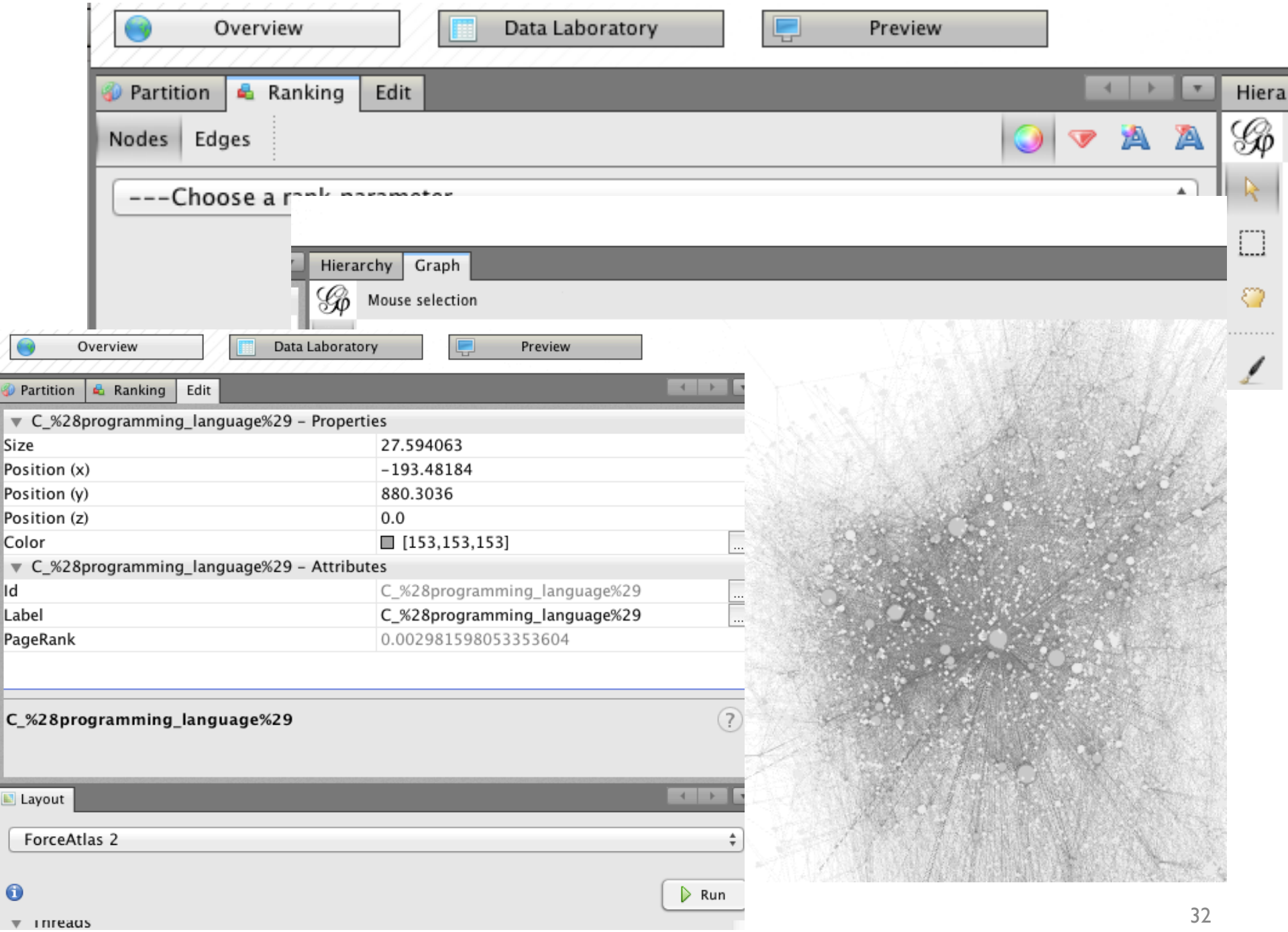
Add column | Merge columns | Delete column | Clear column | Copy data to other column | Fill column with a value | Duplicate column

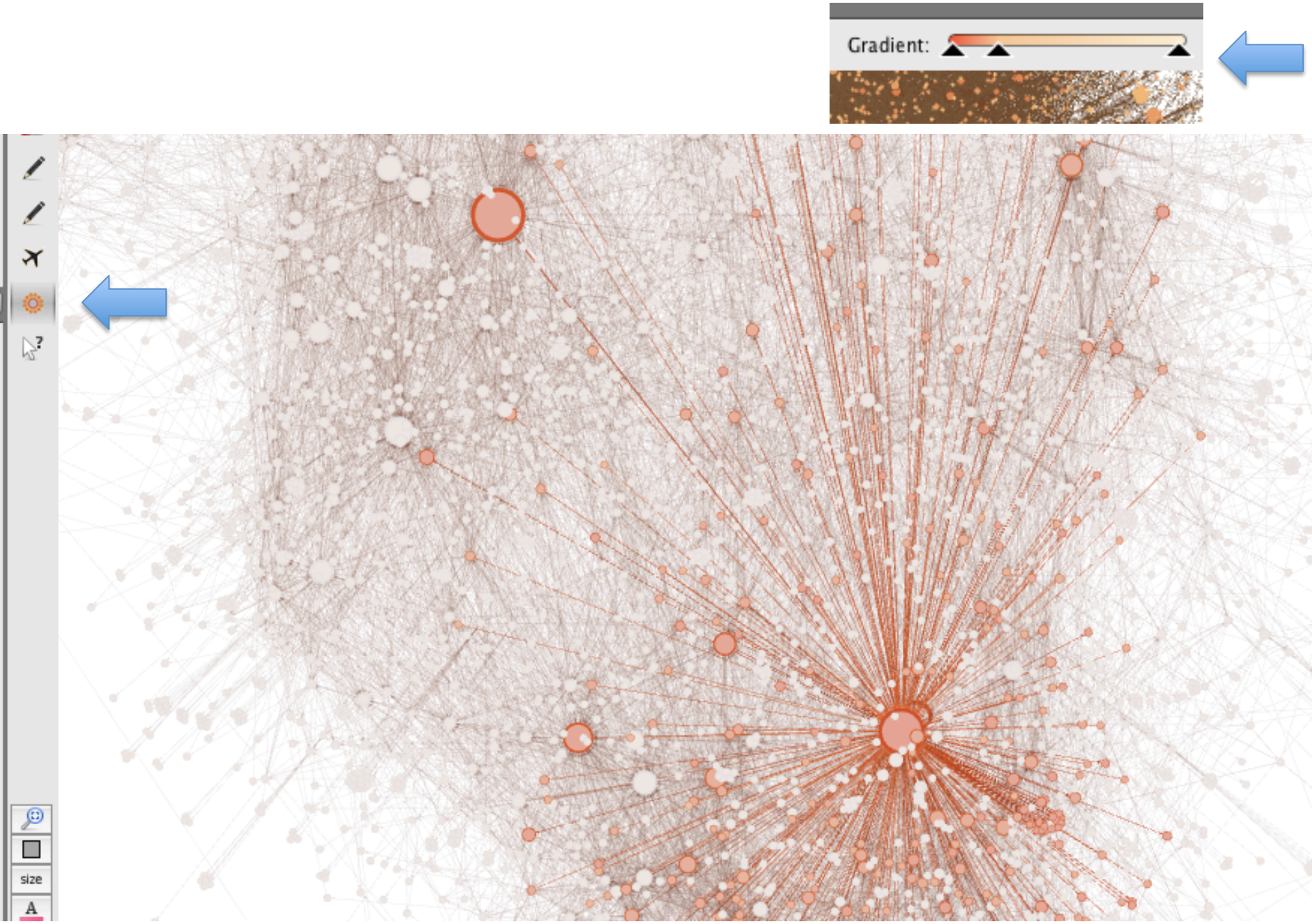
Workspace 1

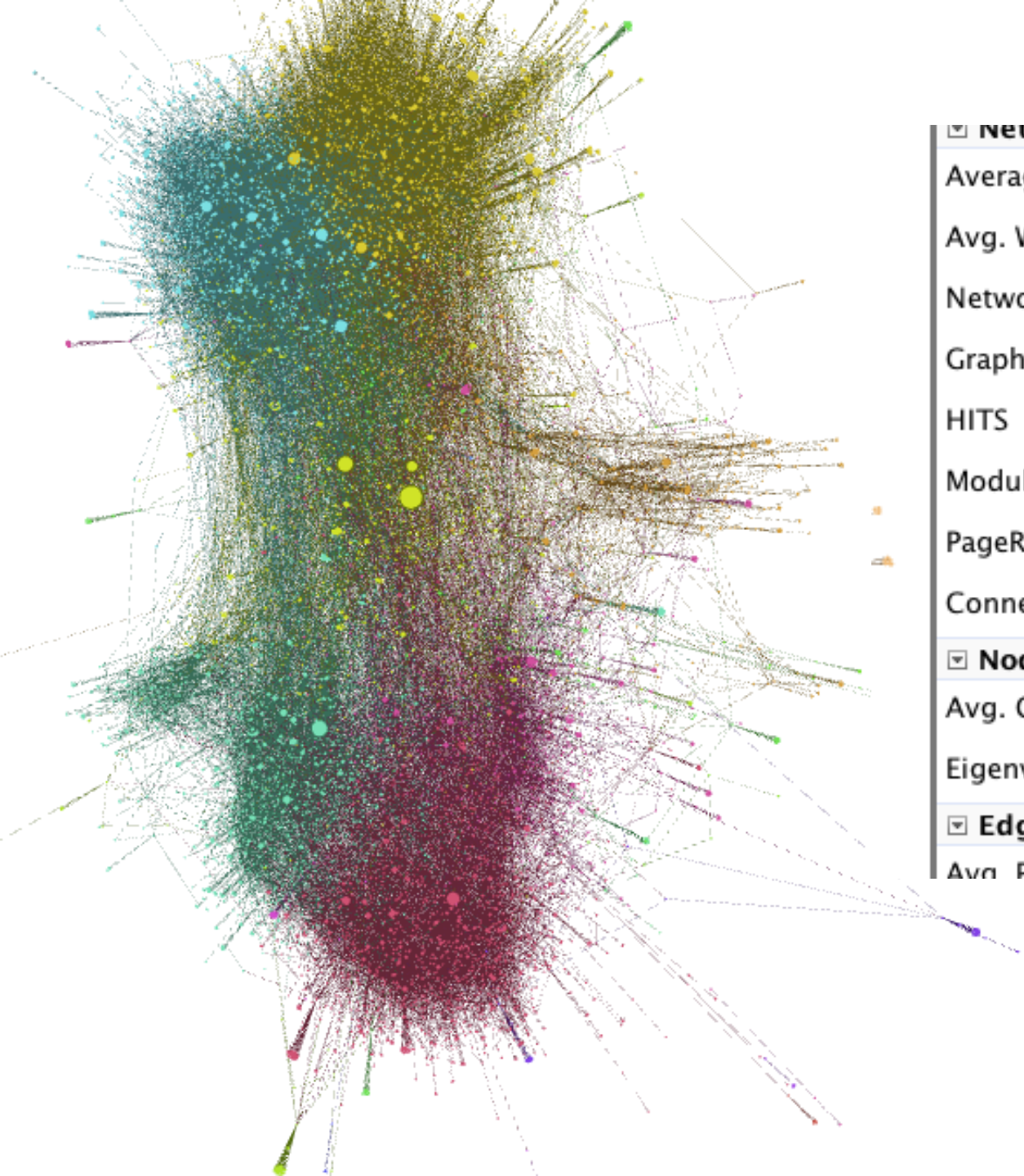
Data Table

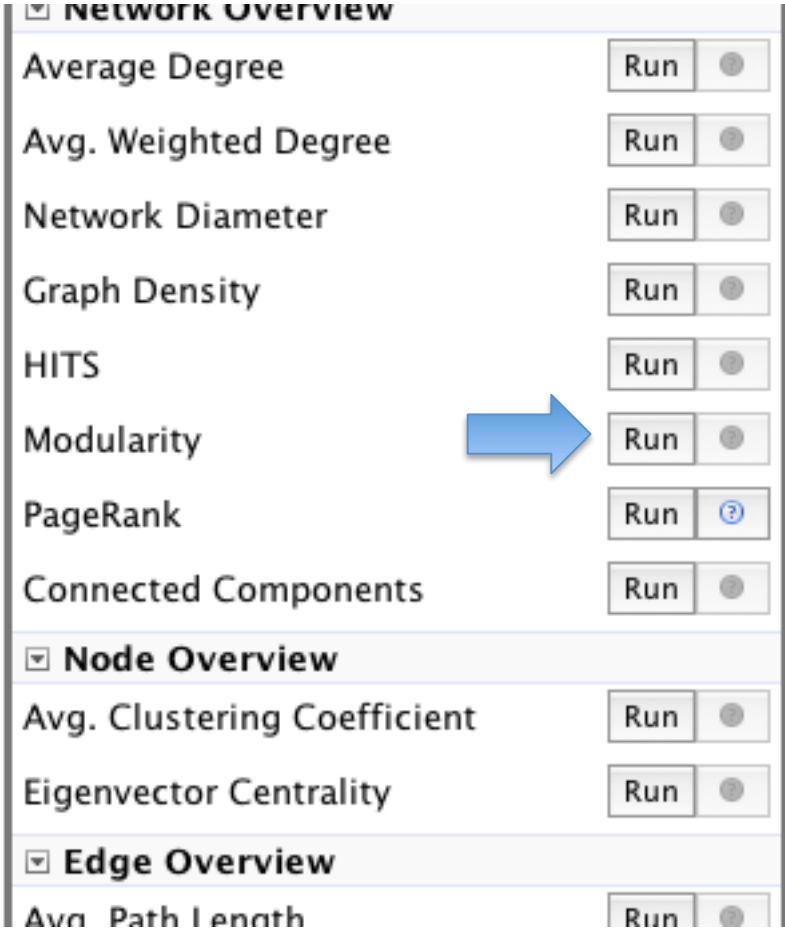
Nodes | Edges | Configuration | Add node | Add edge | Clear graph | Clear edges | Search/Replace | Import Spreadsheet | Export table | Filter

Nodes	Id	Label	PageRank
Living_people	Living_people	Living_people	0.006
Mathematics	Mathematics	Mathematics	0.004
Computer_science	Computer_science	Computer_science	0.004
Statistics	Statistics	Statistics	0.003
Java_%28programming_language%29	Java_%28programming_language%29	Java_%28programming_language%29	0.003
C_%28programming_language%29	C_%28programming_language%29	C_%28programming_language%29	0.003
Programming_language	Programming_language	Programming_language	0.003
Compiler	Compiler	Compiler	0.003
Object-oriented_programming	Object-oriented_programming	Object-oriented_programming	0.003
Year_of_birth_missing_%28living_people%29	Year_of_birth_missing_%28living_people%29	Year_of_birth_missing_%28living_peop	0.002
C%2B%2B	C%2B%2B	C%2B%2B	0.002
Probability_theory	Probability_theory	Probability_theory	0.002
Real_number	Real_number	Real_number	0.002
Set_%28mathematics%29	Set_%28mathematics%29	Set_%28mathematics%29	0.002
Data_type	Data_type	Data_type	0.002
Computer	Computer	Computer	0.001
Computer_programming	Computer_programming	Computer_programming	0.001
Random_variable	Random_variable	Random_variable	0.001


Overview
Data Laboratory
Preview
Partition
Ranking
Edit
Hiera
Nodes
Edges
---Choose a rank parameter
Hierarchy
Graph
Mouse selection
Overview
Data Laboratory
Preview
Partition
Ranking
Edit
C_%28programming_language%29 - Properties
Size
27.594063
Position (x)
-193.48184
Position (y)
880.3036
Position (z)
0.0
Color
[153,153,153]
C_%28programming_language%29 - Attributes
Id
C_%28programming_language%29
Label
C_%28programming_language%29
PageRank
0.002981598053353604
C_%28programming_language%29
Layout
ForceAtlas 2
Run
Threads


Gradient:
size




Network Overview
Average Degree
Run
Avg. Weighted Degree
Run
Network Diameter
Run
Graph Density
Run
HITS
Run
Modularity
Run
PageRank
Run
Connected Components
Run
Node Overview
Avg. Clustering Coefficient
Run
Eigenvector Centrality
Run
Edge Overview
Avg. Path Length
Run