gravicom - a web-based tool for community detection in networks

Andrea J. Kaplan

October 31, 2013

1 Background

1.1 Networks

1.2 Visualization

1.3 Layout Algorithms

2 User Interface

2.1 Design and Functionality

2.1.1 Description

The gravicom interface is comprised of five main parts,

1. Control panel
2. Data selection
3. Connection table
4. Graph display
5. Tabset.

Each of these pieces serves as a means for the user to interact with gravicom, either through controls that allow user input to gravicom or through diagnostics and visualization of a graph. Their placement on the gravicom interface can be seen in figure 1.

Control Panel The control panel serves as the starting point for a user's session in gravicom. It contains intructions for the user, as well as the means for a user to select a dataset and the diagnostic connection table. Additionally, the control panel contains a link to the source code for gravicom, should a user be interested in the inner workings of gravicom.

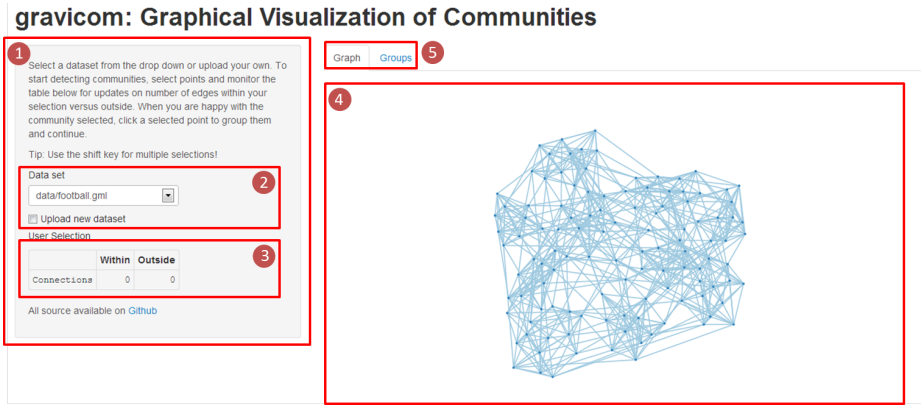


Figure 1: The components that make up gravicom, (1) Control panel, (2) Data selection, (3) Connection table, (4) Graph display, and (5) Tabset.

Data Selection The data selection piece is made up of two parts, the first being a drop down to select pre-loaded datasets to display. Currently there are two toy datasets in gravicom, a college football dataset and a karate friends dataset. From the dropdown the user can change the dataset to display in the graph. The second part of data selection gives the user the ability to upload his own dataset. Upon clicking the "Upload new dataset" checkbox, a file selection control appears which gives the user the ability to upload his own graph data to explore with gravicom. This is shown in figure 2.

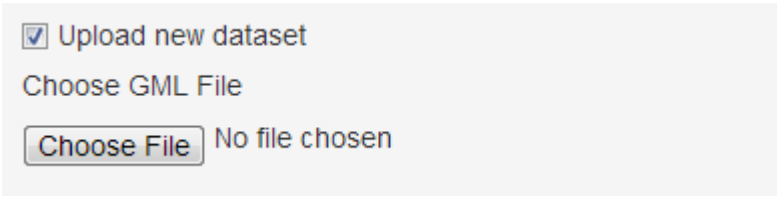


Figure 2: The data selection area upon clicking the "Upload new dataset" checkbox.

Connection Table The connection table displays the number of connection within a user's selection of nodes in the graph and the number of connections from nodes in a user's selection to nodes not in the selection. The comparison of these two numbers can give the user an idea of if what he has selected is a community or not. The idea behind the connection table is that a community will have more connections within that connections to nodes outside the community. The connection table is a diagnostic tool for the user to help determine if he has detected a community in his graph.

Graph Display The graph display shows an interactive graphical representation of a the selected (or uploaded) graph data. Upon load, the graph displays all nodes and edges in the dataset using a force-directed layout algorithm. The user has many ways to interact with the graph. He can drag, select, and group. A user can drag nodes at any time to attempt to get a better view of potential communities. However, the graph is setup with a gravity parameter that generally keeps the graph in the center of the screen. Upon dragging, the force-directed layout is then rerun, giving an altered view of the graph. Figure 3 shows a graph in the process of being dragged.

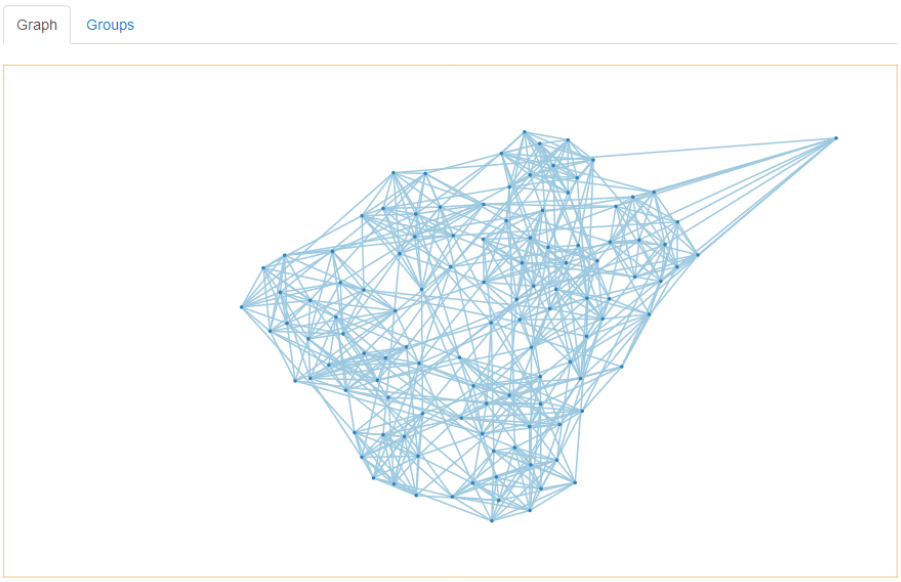
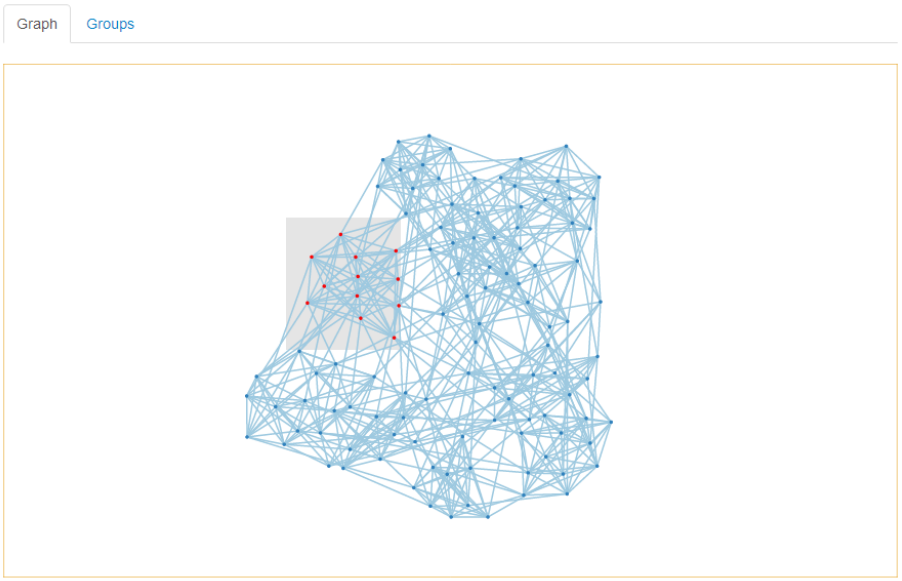


Figure 3: A graph in the process of being dragged.

The selection and grouping actions of the nodes are tied. In order to group nodes, a user must first select what he thinks is a community based on a visual appraisal of the graph. To select nodes the user clicks and drags a selection box around nodes. See figure 4 for the results of selection in the interface. The shift key can also be used for multiple selections. Upon selection, the connection table is updated and the user can evaluate their selection as a community and alter the selection if need be (the shift key selection is very useful in this step).

Once the user is happy with his selection, he can click one of the nodes selected to group the selection. Once grouped, a new node is created that comprises all grouped nodes and grouped within edges in size and charge. The force-directed layout is again run, showing the new graph with nodes grouped. This is shown in figure 5. Additionally, grouped nodes and be ungrouped by clicking on a grouped node. This process can be repeated until all nodes have been grouped or the user is satisfied that all communities have been detected.



	Within	Outside
Connections	46	43

Figure 4: A graph in the process of nodes being selected. Upon selection of nodes, the connection table updates to display within and outside edges.

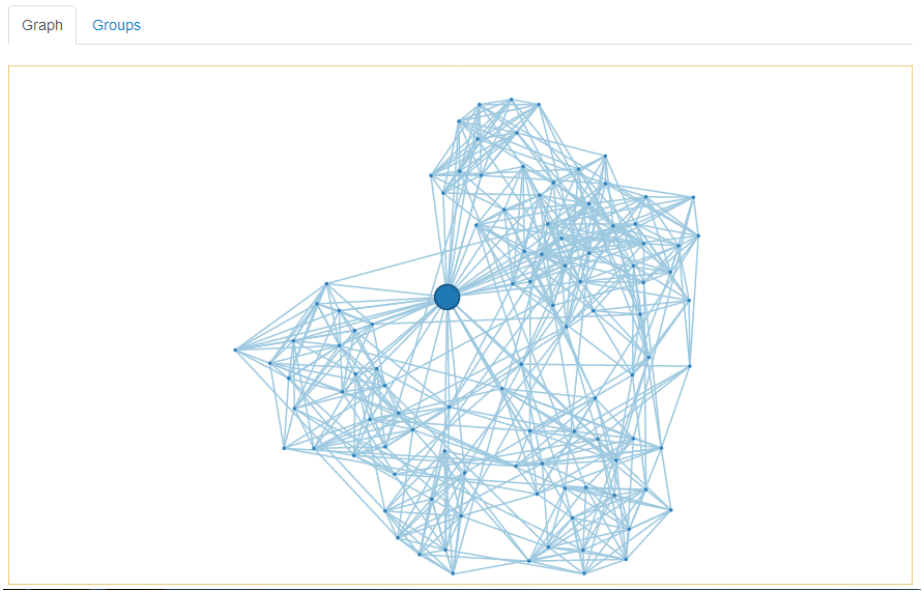


Figure 5: A graph after nodes have been grouped and the force-directed algorithm has been re-run.

Tabset The tabset allows the user to switch between two tabs on the screen. The first (and default tab) shows the graph display. The second tab shows the groups that a user has created in the graph. Each group show how many nodes are contained in that group as well as the ability to drop down and view the nodes contained in that group. If the data are equipped with node labels, these will be displayed. If there are no node labels provided, node ids will will be shown. For an example, see figure 6.

2.2 Examples

To demonstrate the use of gravicom, we present three real-world network datasets and explore their community structure.

2.2.1 College Football

The first dataset is a representation of U.S. College Football Division 1 games from the 2000 season [4]. In this network, nodes represent teams and an edge represents a regular-season game played between the two teams connected. The network as it looks on load in gravicom is presented in figure 7.

Colleges within the same football conference will play members of their conference more frequently than teams outside of their conference, making this an interesting dataset to try to detect communities in. This is also an ideal example due to the relatively small number of nodes and edges present, making a

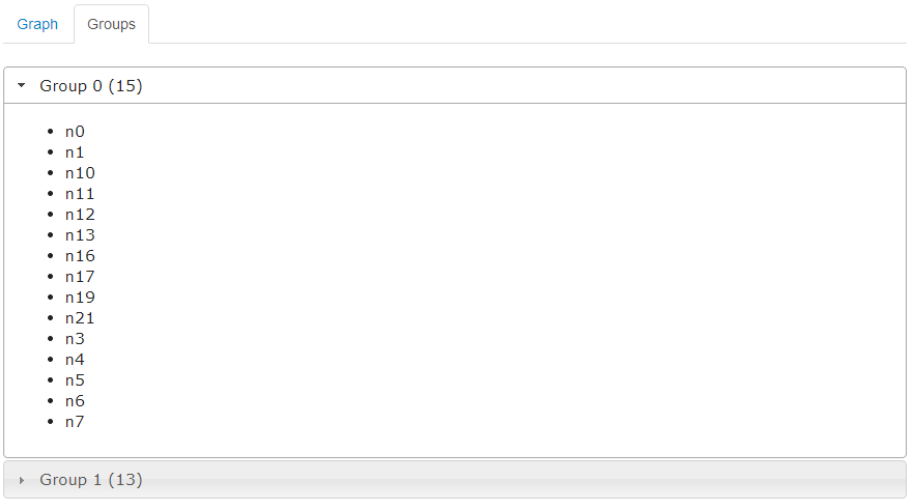


Figure 6: The groups tabset displaying which nodes have been grouped in Group 0, for example. The groups tabset also shows that Group 0 has 15 nodes, while Group 1 has 13 nodes.

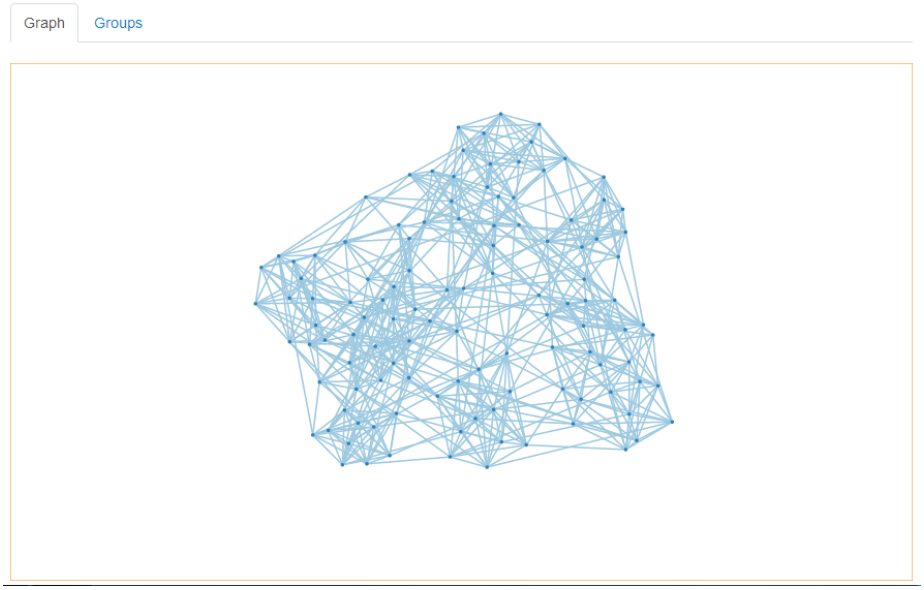
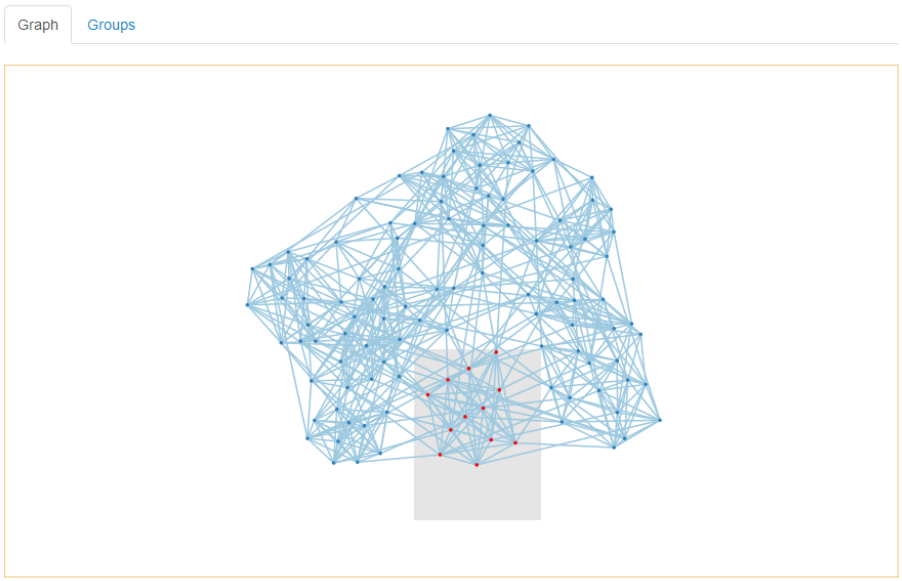


Figure 7: College Football network represented in gravicom.

graphical representation feasible. One challenge encountered with this dataset is that there are independent teams like Notre Dame that are not in a conference, and so may complicate community detection. Another complication is that small conference schools will typically play large conference schools in the beginning of the season in order to pay for their athletic programs, which could cause there to be more edges than expected between small conferences and large conferences.

Upon viewing the network in gravicom, some likely communities emerge and by selecting nodes to determine within and outside edges, we can classify colleges into conferences, as seen in figure 8.



	Within	Outside
Connections	46	43

Figure 8: Selecting a potential community in gravicom and assessing the number of within verses outside selection edges.

Once the user is satisfied that he has selected a viable community, he clicks to group those nodes and the graph will update to reflect this, allowing new communities to potentially become more easily apparent as seen in figure 9.

Additionally, the user can check which nodes were grouped in each community. In this example (figure 10), we can see the first community detected roughly corresponds to the Big 10, with independent Notre Dame being grouped in the Big 10 due to playing two Big 10 teams in 2000.

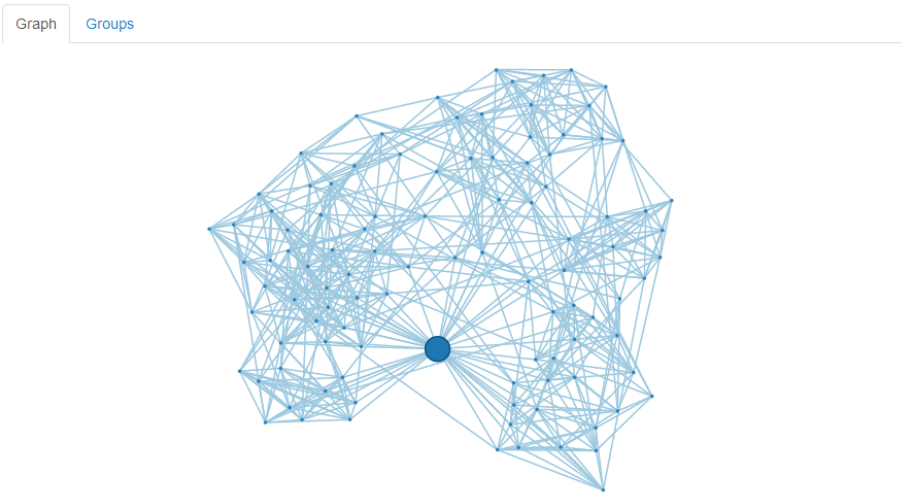


Figure 9: After the first community is detected, more communities become apparent in the network.

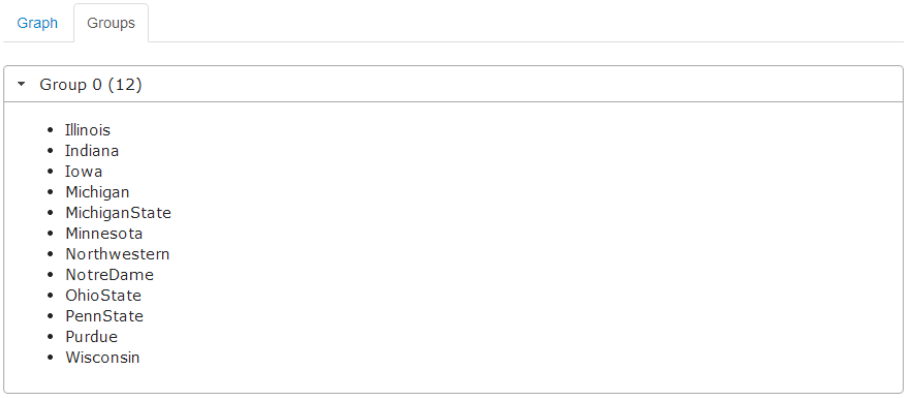


Figure 10: A drill-down of the first grouped community which roughly corresponds to the Big 10 Conference.

After the first community has been selected, another community at the top of the graph has been revealed. The user can once again, select and group this community and take a look at the nodes in the community as seen in figure 11. The second conference grouped corresponds exactly to the SEC Conference, another large college football conference. The first two communities to become evident correspond to large Division 1 conferences that play the majority of their games within conference, matching our earlier assertion that small conferences would be more difficult to detect.

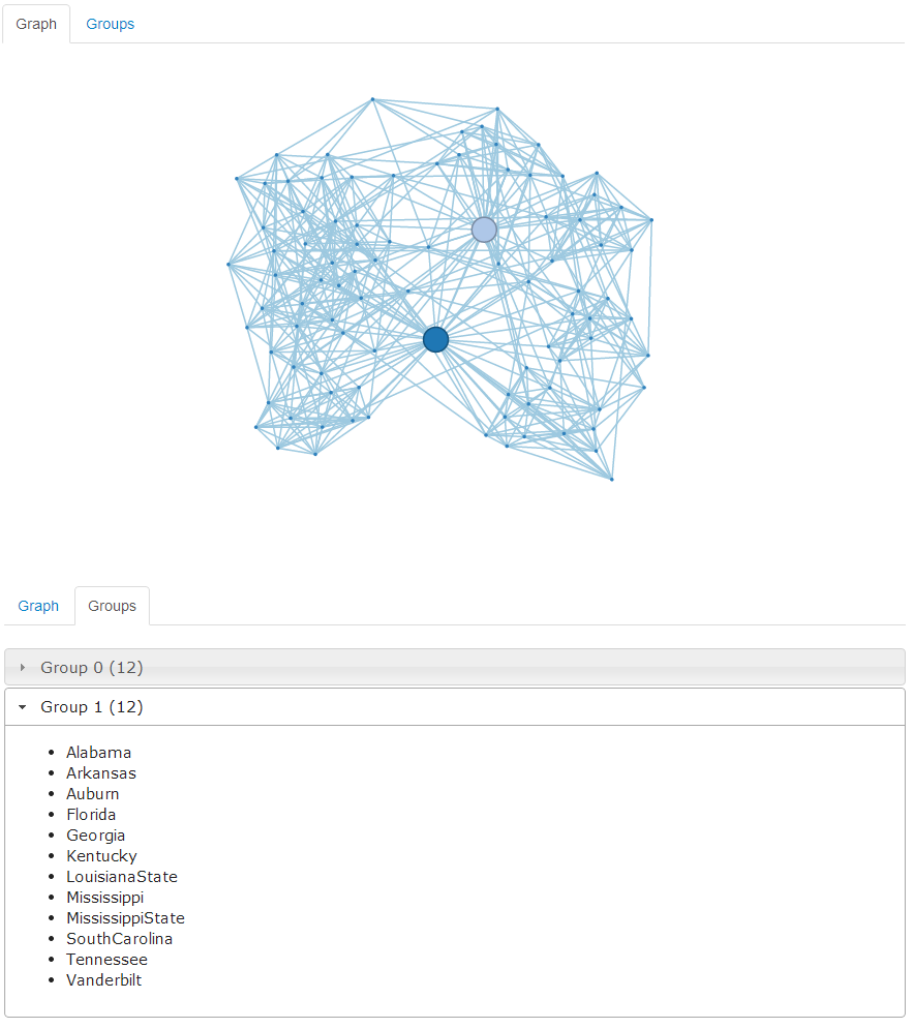


Figure 11: A second community is grouped which corresponds to the SEC Conference.

This process can be repeated until the user is satisfied with the communities

selected. The conclusion of this process is seen in figure 12. We detected visually 11 conferences in the dataset. There were 11 conferences and 6 independent schools in the dataset which roughly match with what we detected.

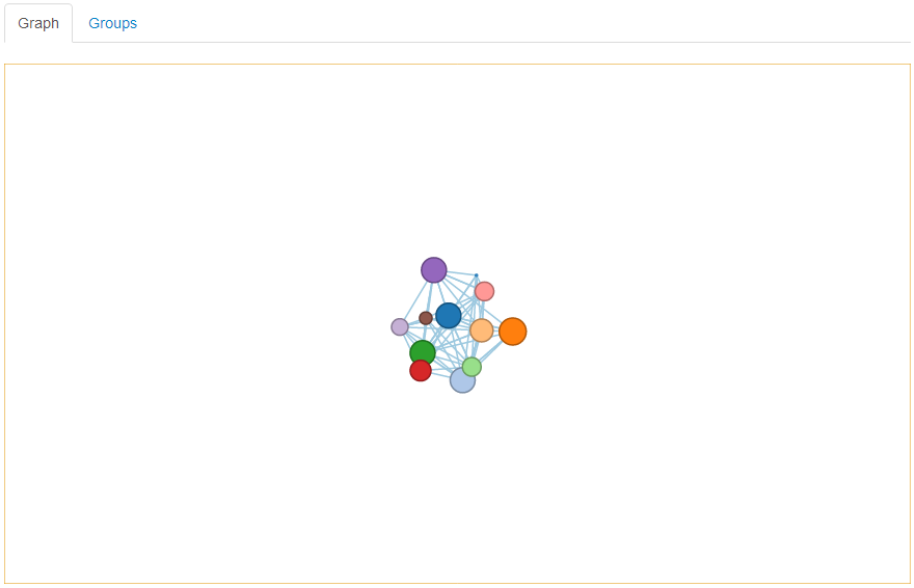


Figure 12: Final community structure detected in gravicom.

2.2.2 Dolphins

2.2.3 Political Books Sold

2.3 Technical Aspects

gravicom utilizes three main pieces of software to establish interactive user control of a random graph as sketched out in figure 13. The three pieces used are Shiny, D3, and igraph. They are used to manage server/client interaction, user interface and graph layout, and data formatting, respectively.

There are very minimal software requirements for a user of gravicom. The client simply needs to have a JavaScript enabled internet browser with HTML5 compatibility, something which almost any modern browser fulfills (an exception is IE8 and below).

The server side requirements are more extensive, but this does not affect the user of gravicom, only those wanting to host their own instance of the application. To host gravicom, a Linux server is required, with the following installed:

- Node.js (0.8.16 or later)

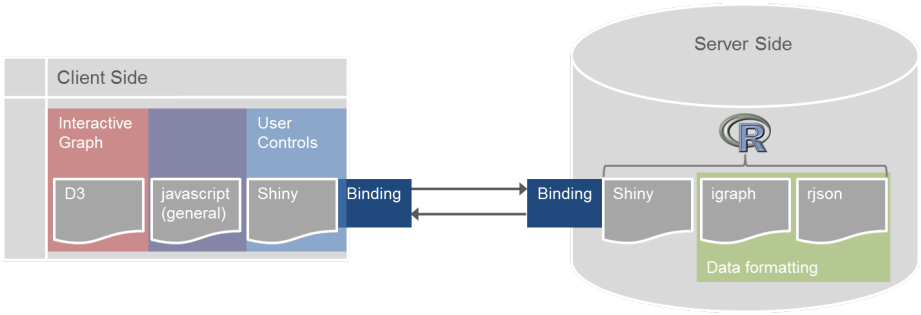


Figure 13: Relationship between client and server, specifically focusing on how data travels between the two.

- `R` (2.15 or later)
- Shiny `R` package, installed into the machine-wide site library.
- Shiny Server

2.3.1 Shiny

Shiny [5] is an `R` package created by RStudio that enables `R` users to create an interactive web application that utilizes `R` as the background engine. Through default methods to build user interface elements in HTML and a handle to the server side code, Shiny is a very simple way to turn `R` code into a website.

gravicom uses the Shiny functionality to create user controls, pass correctly formatted data to the client, and as a means to display summary information regarding the user's interactions with a graph at any point in time. In this context, Shiny serves as the translator between the formatted data and what the user sees and interacts with on their screen.

2.3.2 D3

D3 [1] stands for "Data Driven Documents" and is a JavaScript library developed and maintained by Mike Bostock with the purpose of visualizing and interacting with data in a web-based interface. It is freely available from `http://www.d3js.org`. The library facilitates manipulation of HTML elements, SVG (scalable vector graphics), and CSS (cascading style sheets) with the end goal of rendering animations and providing user interactions that are tied to the underlying data. The key idea behind the library is that Document Object Model elements are completely determined by the data. The Document Object Model (DOM) is a convention for representing and interacting with objects in HTML, XHTML and XML. So, rather than adding elements to a web page to be viewed by users, D3 allows users to see and interact with graphical representations of their data in a web framework.

gravicom uses D3 to handle all graphical displays and user interactions with the graph. The data is passed to the client and able to be used through Shiny's input bindings. It is crucial that the data has been formatted correctly at this point for the JavaScript to properly function. For this reason, we limit the file types being passed in to the tool to a robust graph-specific type.

At this point in the page lifecycle, the graph's nodes are tied to circles and the edges are tied to paths on the page. User manipulations such as selecting, dragging, and grouping are handled by D3 and data is passed back to the server via Shiny's output bindings to allow for communication between user and the `R` engine underneath. This is illustrated in figure 14. What this means is that all visualization and user interaction with the graph are accomplished using JavaScript, more specifically the library D3. Shiny and `R` serve as the framework on which the data sits, but when the user touches the data they are doing so through the JavaScript elements.

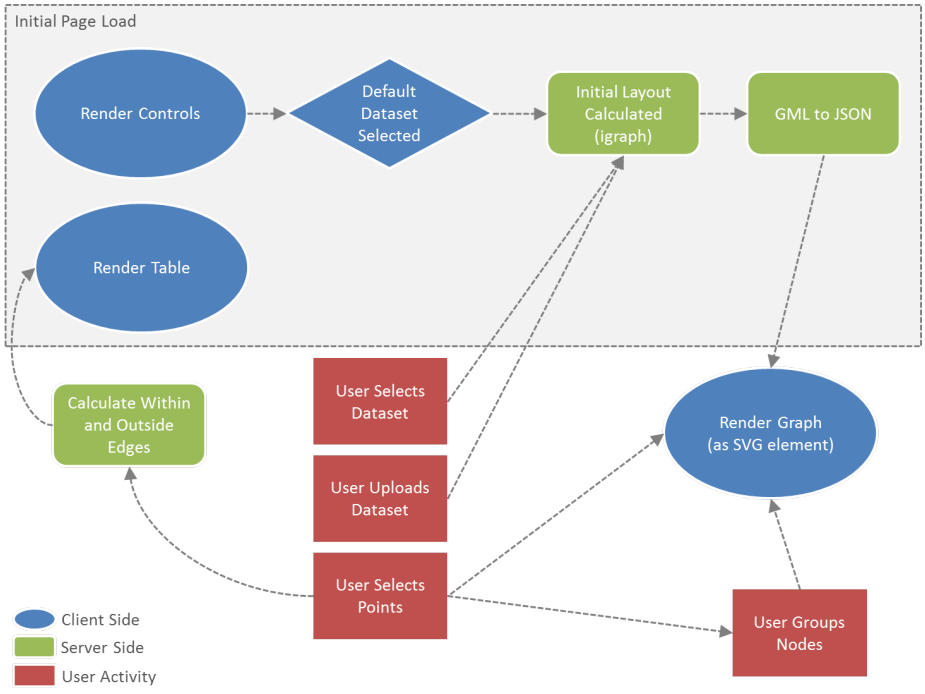


Figure 14: Page lifecycle beginning from on load. User actions are highlighted in red, server actions in green, and actions completed on the client side are highlighted in blue.

2.3.3 igraph

igraph [3] is a software package used for creating and manipulating undirected and directed graphs. It is a cross-language package available for C, R, python, and Ruby. igraph also supports multiple graph file formats and visualization of graph structures.

gravicom utilizes two parts of igraph, first is the conversion from a gml file to an XML file. The gml file format, short for Graph Modelling Language, is a hierarchical ASCII-based file format for describing graphs. Below is an example gml file of an undirected graph consisting of two nodes linked by a single edge. The important points to note are that node identifiers (id) have to be numeric. An edge consists only of source and target ids of the nodes it connects, while nodes can have other attributes, e.g. `value` in the example. For a directed graph, the parameter `directed` has to be set to 1, which will result in the edge information on target and source being evaluated accordingly.

```
## graph
## [
##   directed 0
##   node
##   [
##     id 0
##     label "Node 1"
##     value 100
##   ]
##   node
##   [
##     id 1
##     label "Node 2"
##     value 200
##   ]
##   edge
##   [
##     source 1
##     target 0
##   ]
## ]
```

For the conversion from an XML file to a JSON file we make user of the R package `rjson` [2]. JSON is the native data format used in D3, which makes working with data in the D3 library incredibly straightforward. Here is our example in the finalized JSON format:

```
## {
##   "nodes":
##   [{"id":"n0","v_id":"0","v_label":"Node 1","v_value":"100"},
```

```
##    {"id":"n1","v_id":"1","v_label":"Node 2","v_value":"200"}],
##  "edges":
##   [{"source":0, "target":1}]
## }
```

Our example data will yield the graph in figure 15.

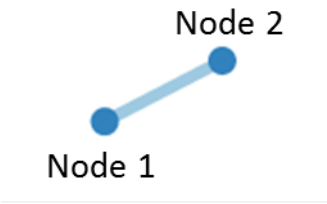


Figure 15: Graph created from sample gml file.

The second use of igraph within gravicom is to compute initial x and y coordinates for the nodes of the graph using a force-driven layout. This provides the initialization for the force-layout algorithm in D3. This reduces the computational load on the clients' side and helps minimize unnecessary movement by the nodes. This is critical as the extra movement at the loading of the pages creates an unnecessarily chaotic start to the user's experience.

3 Further Work

References

[1] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. "D3: Data-Driven Documents". In: *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)* (2011). URL: http://vis.stanford.edu/files/2011-D3-InfoVis.pdf.

[2] Alex Couture-Beil. *rjson: JSON for R*. R package version 0.2.12. 2013. URL: http://CRAN.R-project.org/package=rjson.

[3] Gabor Csardi and Tamas Nepusz. "The igraph software package for complex network research". In: *InterJournal* Complex Systems (2006), p. 1695. URL: http://igraph.sf.net.

[4] M. Girvan and M. E. J. Newman. "Community structure in social and biological networks". In: *Proceedings of the National Academy of Sciences* 99.12 (2002), pp. 7821–7826. DOI: 10.1073/pnas.122653799. eprint: http://www.pnas.org/content/99/12/7821.full.pdf+html. URL: http://www.pnas.org/content/99/12/7821.abstract.

[5] RStudio Inc. *shiny: Web Application Framework for R*. R package version 0.4.0. 2013. URL: http://CRAN.R-project.org/package=shiny.