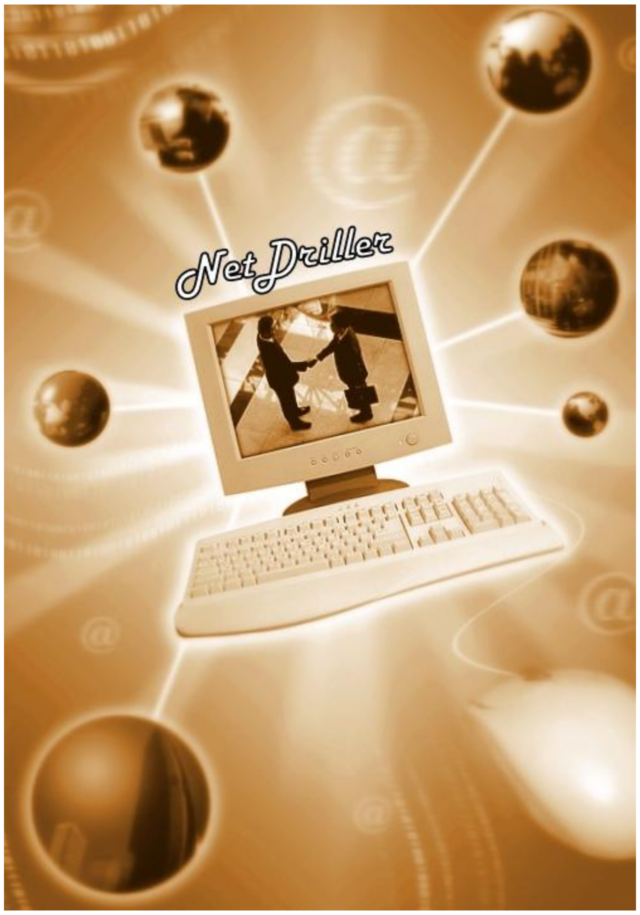
2010

NetDriller User Manual



Director: Dr. Reda Alhajj, Dr. Jon Rokne

Coordinator: Dr. Keivan Kianmehr, Negar Koochakzadeh

Developers: Alen Chia-Lung Chen (2010), Alex Chan (2010), Ali Rahmani (2010), Anthony Tam (2010), Atieh Sarraf (2010), Faisal Iqbal (2010), Fatemeh Keshavarz (2010), Ian Reinhart (2009), Jun Jiang (2009), Mona Okasha (2009), Negar Koochakzadeh (2009-2011), Omair Shafiq (2009), Sara Aghakhani (2010), Terence Ran Tang (2010)

Contents

Introduction

What is Social Network Analysis (SNA)?

The social network model was first realized in 1934 as subarea of sociology and anthropology in a trial to study the relationships between people in a small group. The analysis was conducted mainly manually to study small networks in order to identify key persons and groups within the network.

In its simplest form, a social network is a set of actors (interchangeably called individuals) and the relations between them. Actors represent nodes of a graph and relations are reflected as links. Further, social networks are classified based on the number of distinct and independent actor groups involved in the model; the number of actor groups determines the mode of the network; one-mode and two mode networks are the most commonly used and can be represented as normal graph and bipartite graph, respectively.

An example of a one-mode social network may be realized by considering students as actors and specifying links based on their involvement in courses such that two students are linked if and only if they are enrolled together in at least say three courses; on the other hand, considering students as one group of actors and courses as a second group of actors will lead to a two-mode social network where links connect students to courses; no links within the students groups and no links with the courses group.

About NetDriller

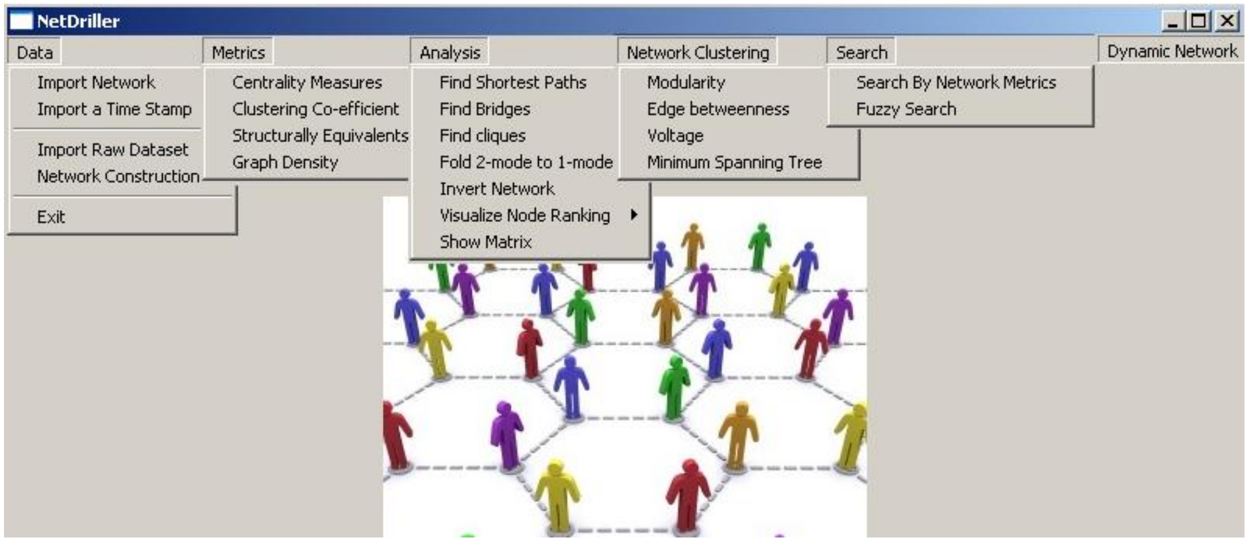
NetDriller is a tool for Social Network Analysis, being developed in University of Calgary under the supervision of Dr. Reda Alhajj. This tool helps the analysts to create a social network or open a predefined network and perform some analyses and measurements on the network.

In the first version of NetDriller, it is possible to import a predefined network from a text file. In addition, a dataset, containing transactions and list of items in each, can be imported to this tool which is then used to construct a social network of items based on the frequency of items in the transactions (association rule mining).

Various analyses can be performed on the loaded network such as: measuring node and network level metrics, filtering links, finding bridges and cliques and shortest paths, folding 2-mode network and create 1-mode network, inverting the network, clustering nodes of the network, searching the nodes based on available node level metrics (containing fuzzy search).

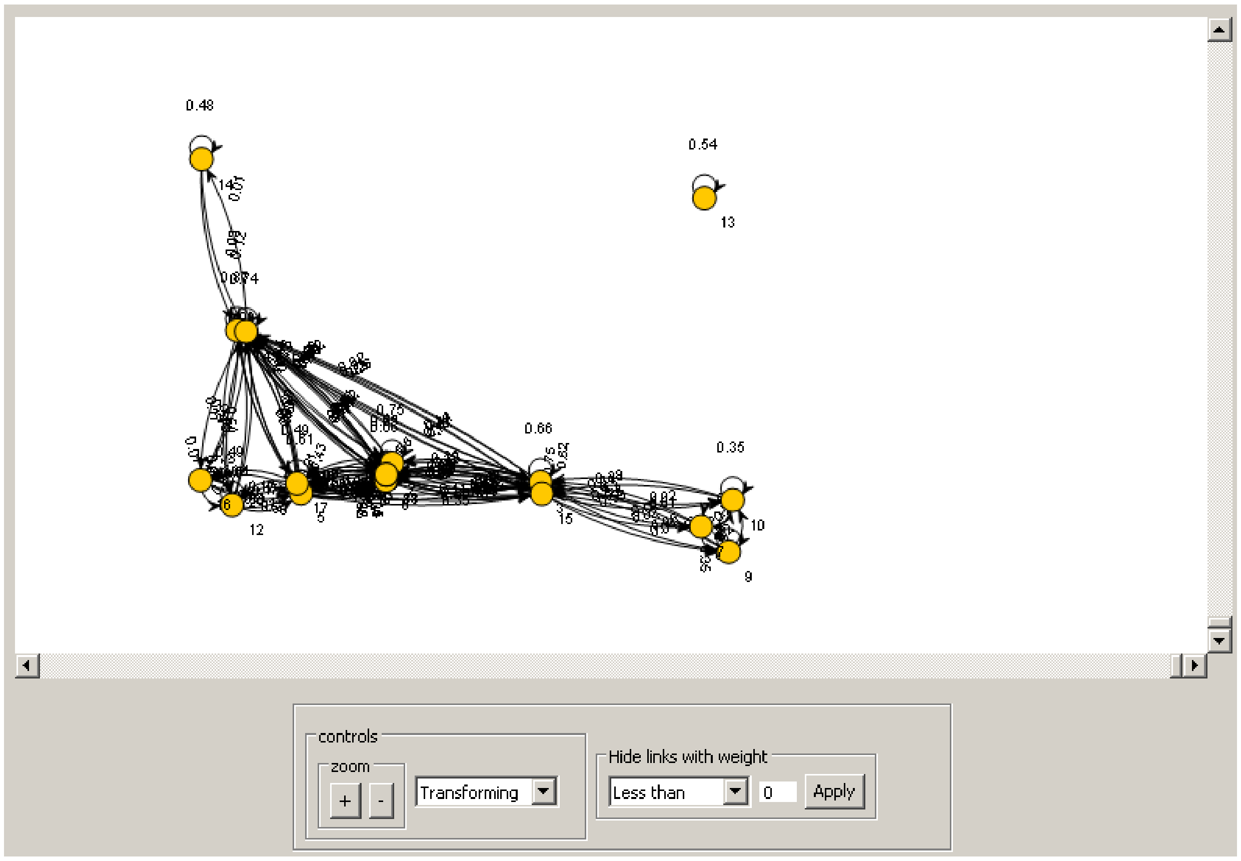
Available Functionalities in NetDriller

The main window and available menus of NetDriller is shown in the following figure:



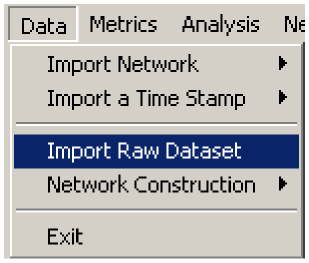
Importing Network from File

From ***Data*** menu, select ***Import Network***. In this sub menu a predefined network (1-mode or 2-mode) saved in a text file or excel format can be loaded to NetDriller. Following figure shows a sample network of *StudentsFriendship* (available in the NetDriller home page – Sample Networks). Controller box below the visualized network, helps to zoom in and out the visualizer and change the mode between picking nodes (and moving them) and transforming the whole network on the visualizer.



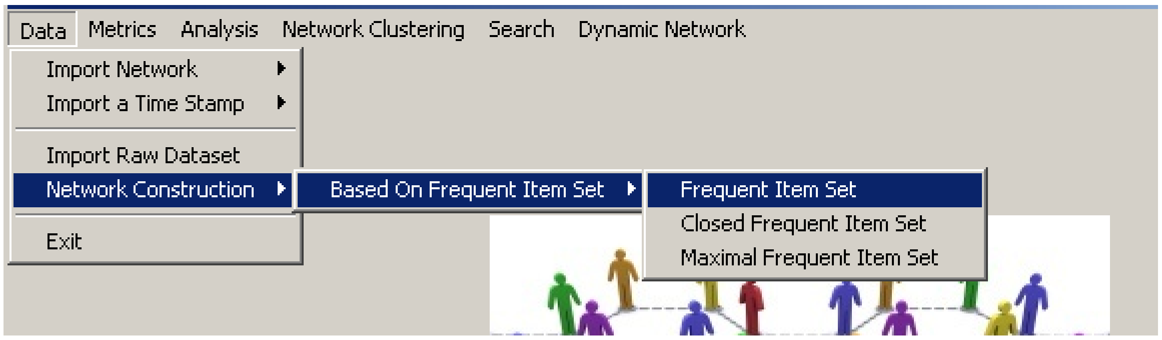
Network Construction

From ***Data*** menu, select ***Import Raw Dataset*** and then ***Network Construction***. This functionality can take the raw dataset containing transactions and list of items in each, and mine frequent patterns (also closed frequent patterns, or maximal frequent patterns) of the items. Import *Transactions* dataset (available in the NetDriller home page – Sample Networks):

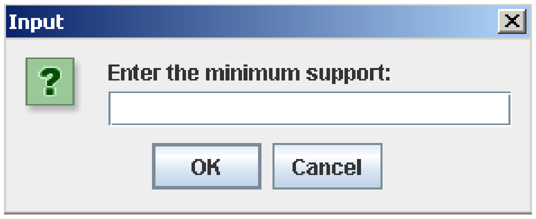


Network Construction uses the *Apriori* algorithm to find frequent patterns recursively. It uses the frequent patterns to create a network, by using the frequent patterns as links with weights indicating the number of frequent patterns the two nodes appear together in.

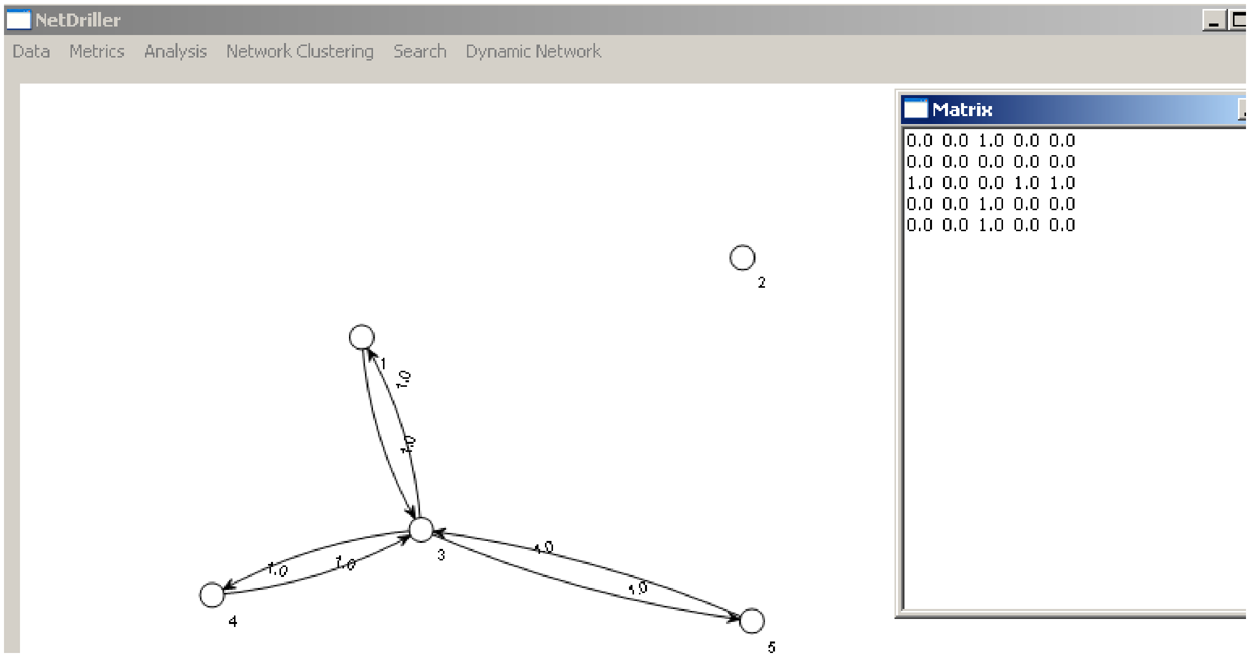
From this submenu select ***Based on Frequent Item Set*** and then select ***Frequent Item Set*** (Frequent Patterns), or ***Closed Frequent Item Set*** (Closed Frequent Patterns), or ***Maximal Frequent Item Set*** (Maximal Frequent Patterns):



Minimum support value is asked to be used in the frequent pattern mining algorithm. Enter 3 as an example:



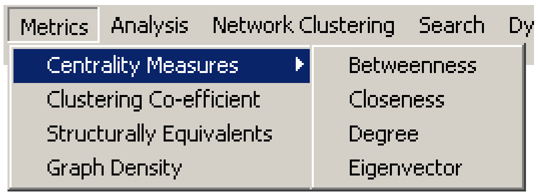
The network will be generated and the matrix will be displayed:



```
0.0 0.0 1.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0
1.0 0.0 0.0 1.0 1.0
0.0 0.0 1.0 0.0 0.0
0.0 0.0 1.0 0.0 0.0
```

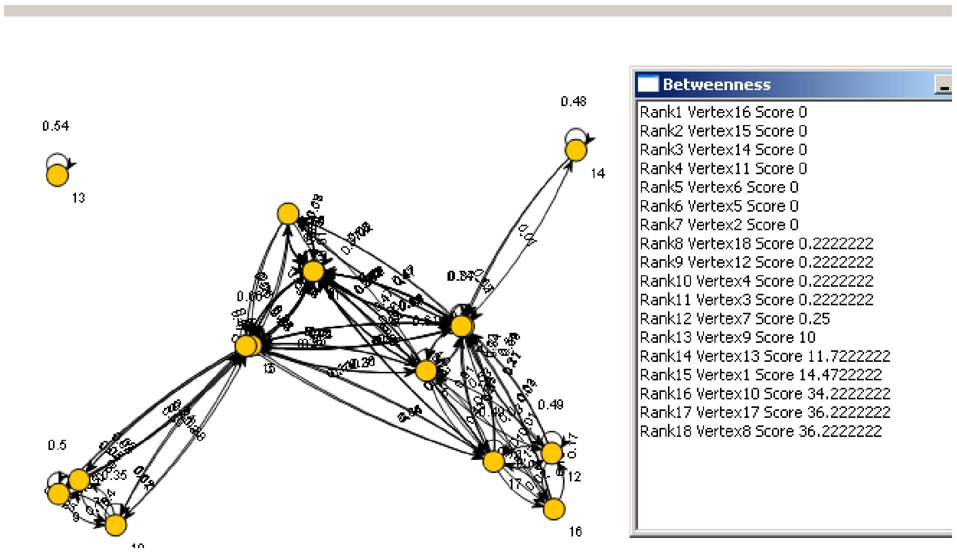
Network Metrics

From ***Metrics*** menu, select appropriate metrics to be measured on the opened network. There are three types of node level metrics (***Centrality Measures*** containing ***Betweenness***, ***Closeness***, ***Degree*** and ***Eigenvector***, ***Clustering Co-efficient***, and ***Structurally Equivalents***) and one network level metric (***Graph Density***):



Centrality

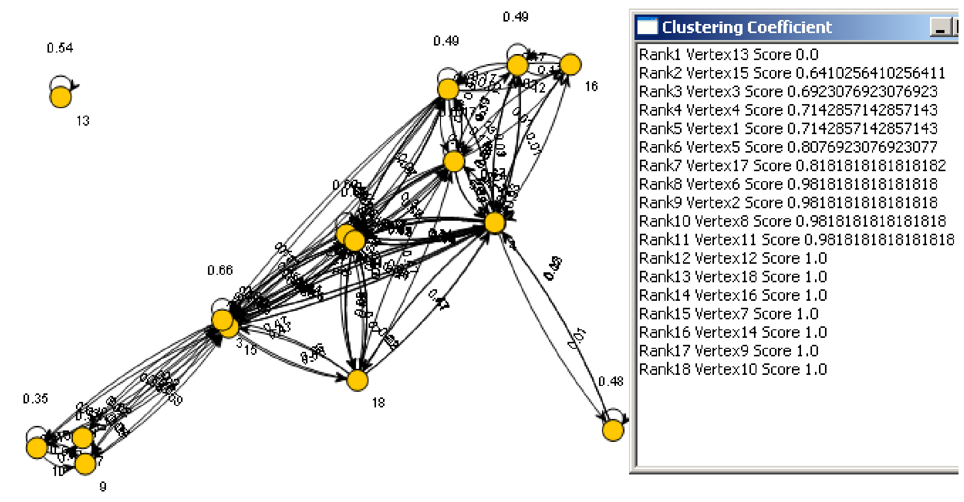
In the Centrality Measures sub menu, you can utilize options such as betweenness, closeness, degree, and eigenvector. To find each of these measures, click on it and a small box with the values will appear, with the proper ranking of the nodes based on the chosen measure. An example of finding the betweenness measure is shown in following figure:



```
Rank1 Vertex16 Score 0
Rank2 Vertex15 Score 0
Rank3 Vertex14 Score 0
Rank4 Vertex11 Score 0
Rank5 Vertex6 Score 0
Rank6 Vertex5 Score 0
Rank7 Vertex2 Score 0
Rank8 Vertex18 Score 0.2222222
Rank9 Vertex12 Score 0.2222222
Rank10 Vertex4 Score 0.2222222
Rank11 Vertex3 Score 0.2222222
Rank12 Vertex7 Score 0.25
Rank13 Vertex9 Score 10
Rank14 Vertex13 Score 11.7222222
Rank15 Vertex1 Score 14.4722222
Rank16 Vertex10 Score 34.2222222
Rank17 Vertex17 Score 36.2222222
Rank18 Vertex8 Score 36.2222222
```

Clustering Co-efficient

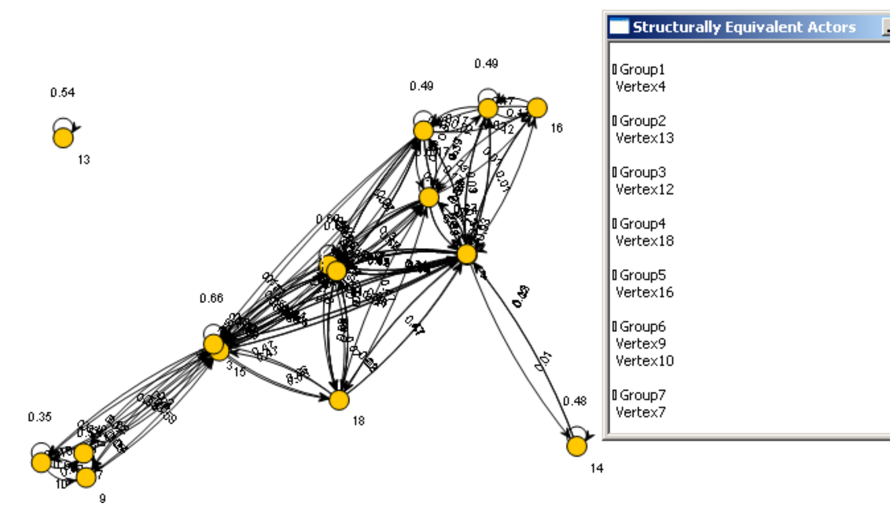
This functionality allows you to get the clustering coefficient of vertices and their rank:



```
Rank1 Vertex13 Score 0.0
Rank2 Vertex15 Score 0.6410256410256411
Rank3 Vertex3 Score 0.6923076923076923
Rank4 Vertex4 Score 0.7142857142857143
Rank5 Vertex1 Score 0.7142857142857143
Rank6 Vertex5 Score 0.8076923076923077
Rank7 Vertex17 Score 0.8181818181818182
Rank8 Vertex6 Score 0.9818181818181818
Rank9 Vertex2 Score 0.9818181818181818
Rank10 Vertex8 Score 0.9818181818181818
Rank11 Vertex11 Score 0.9818181818181818
Rank12 Vertex12 Score 1.0
Rank13 Vertex18 Score 1.0
Rank14 Vertex16 Score 1.0
Rank15 Vertex7 Score 1.0
Rank16 Vertex14 Score 1.0
Rank17 Vertex9 Score 1.0
Rank18 Vertex10 Score 1.0
```

Structurally Equivalents

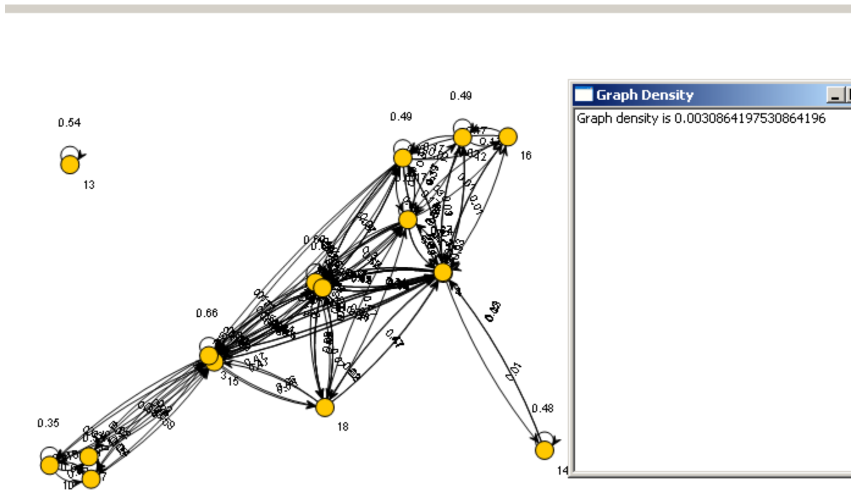
This functionality allows you to identify sets of structurally equivalent vertices in a graph. Vertices i and j are structurally equivalent if and only if the set of i's neighbors is identical to the set of j's neighbors (they connect to the exact same vertices), with the exception of i and <i>j</i> themselves. It works for both directed and undirected graphs. This algorithm finds all sets of equivalent vertices in O(V^2) time.



```
Group1
 Vertex4

Group2
 Vertex13

Group3
 Vertex12

Group4
 Vertex18

Group5
 Vertex16

Group6
 Vertex9
 Vertex10

Group7
 Vertex7
```

Graph Density

This feature allows you to find the graph density of the given network graph.

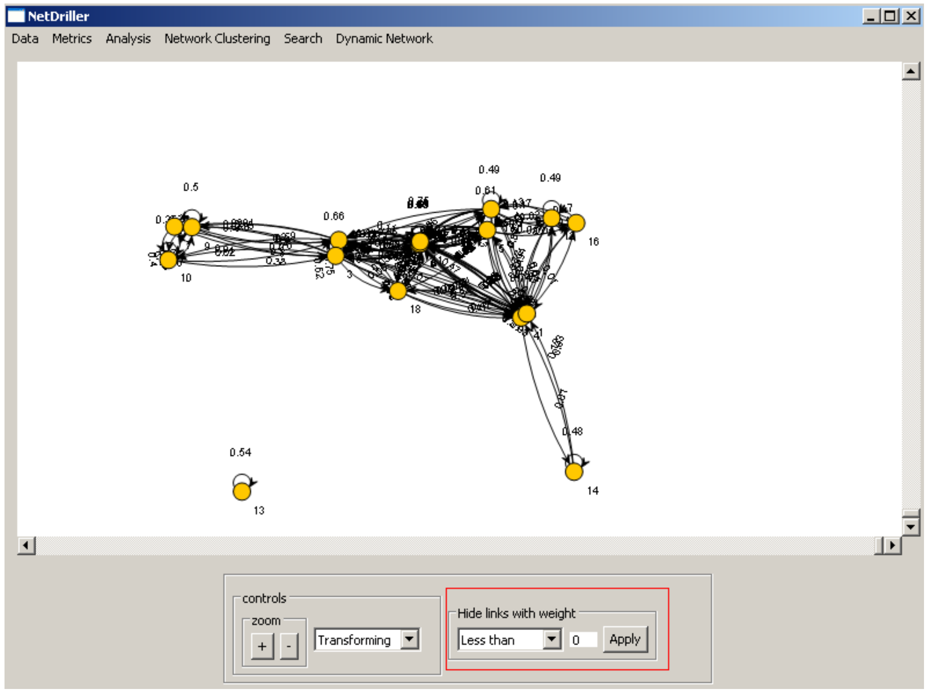

Graph Density

Graph density is 0.0030864197530864196

Analysis

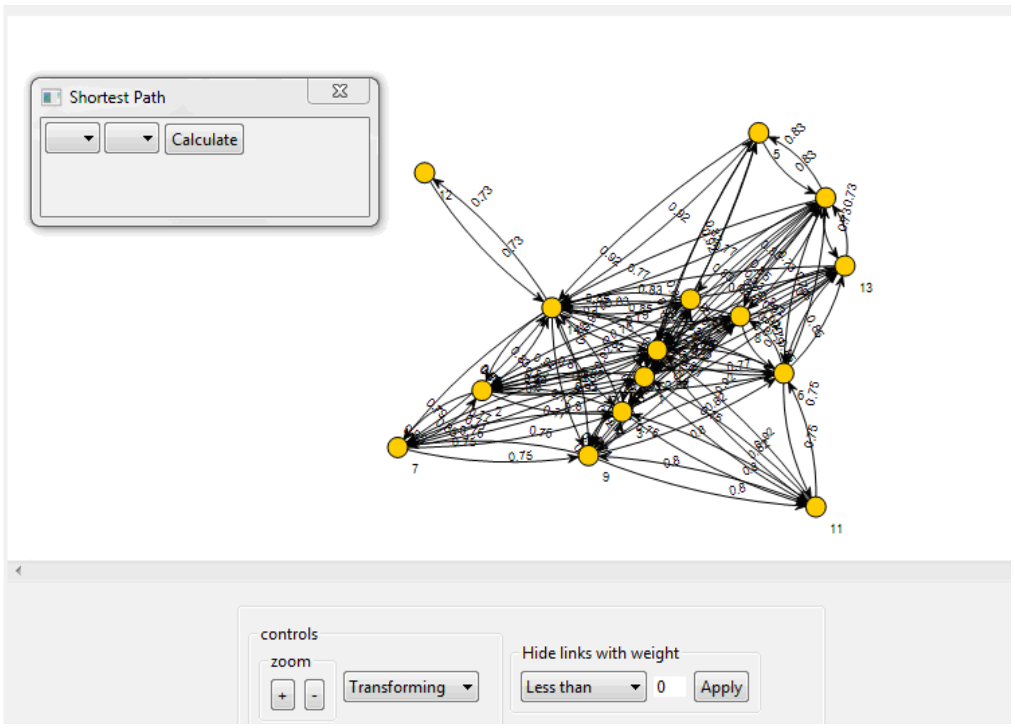
Filtering links

For hiding links with weights less than or greater than a threshold, you can use the part highlighted in the following picture after loading the network. In the first component (the combo box placed at the left side) you can define the criteria: whether you want to hide links less than a value or links greater than a value. In the second component placed at the right side, you have to enter the threshold (it can be either double or integer). Then pressing Apply you can see the result applied on the given network.

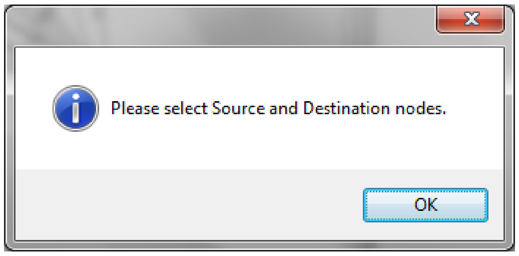


Finding Shortest Paths

From ***Analysis*** menu, select ***Find Shortest Paths***. Finding shortest paths between two nodes in a social network is not only useful in different social network analysis (SNA) measures like betweenness, but it also visually guides the user to reach from one node to another. NetDriller supports shortest path discovery in two mode networks as well as one mode network.
A new screen titled **Shortest Paths**, as shown below, will appear. First combo box represents the source node while the second represents destination node for the shortest path discovery. Nodes are sorted by their numbers.

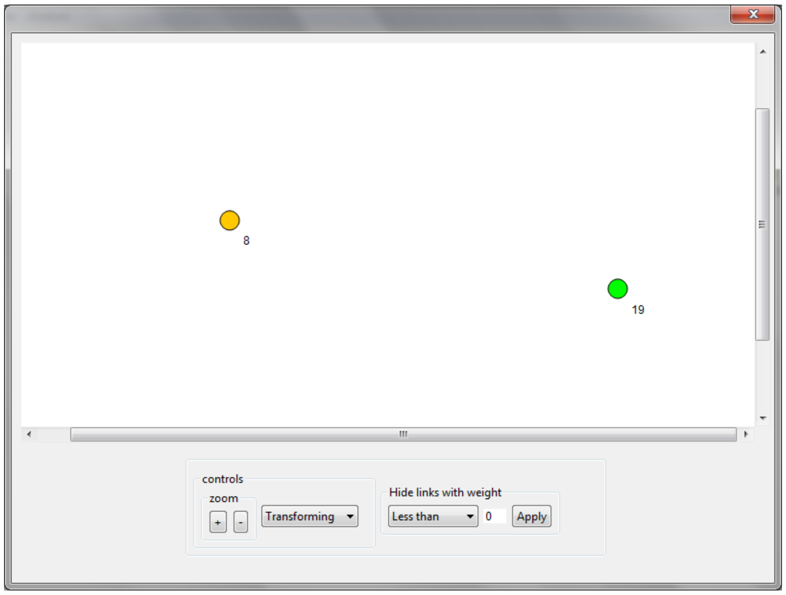


Select **source** and **destination** nodes for the shortest path and click **Calculate** button. Make sure that you both source and destination nodes are selected. NetDriller gives following error message if either the source or the destination node is not selected. Also ensure that you are not selecting the same node as source as well as destination.

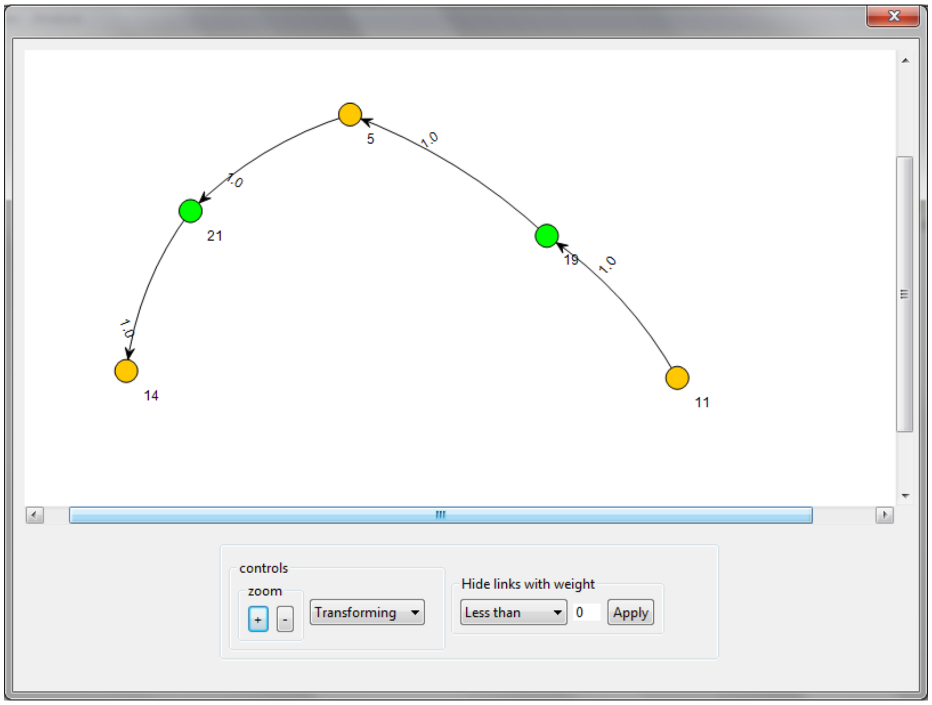

Please select Source and Destination nodes.

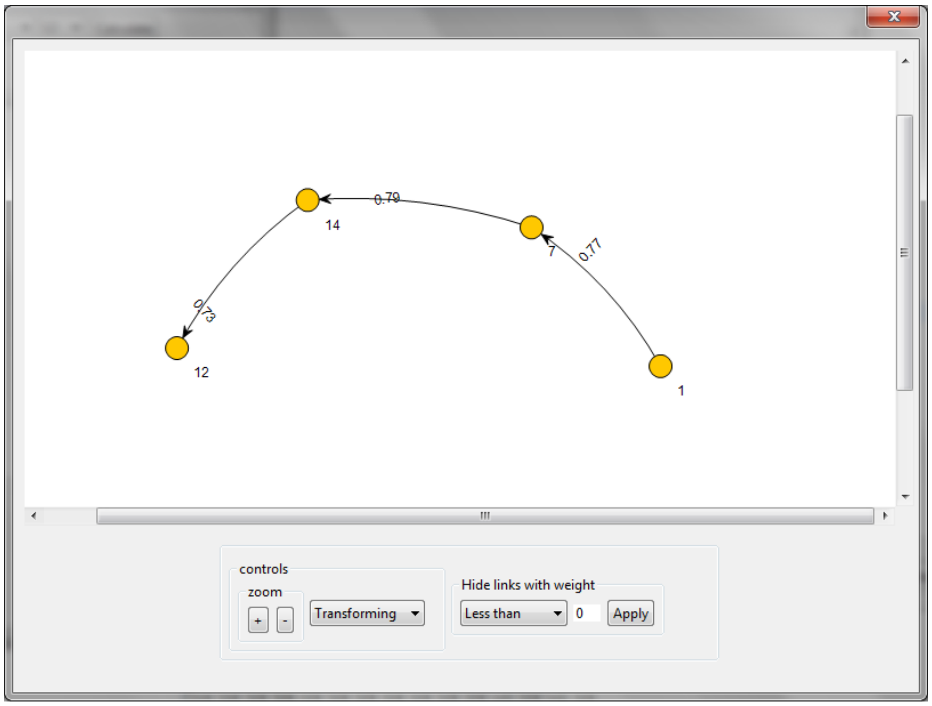
When you click **Calculate** button, a window will appear showing shortest path between the source and the destination nodes including all the intermediate nodes if any. Path is displayed as a **graph** including node names and the link weights. You can move the graph and zoom in and out of it to achieve required level of detail. Below are the two possible scenarios for shortest path graph.

a. If no shortest path exists between the two nodes, the resultant graph displays source and destination nodes with no link connecting them to each other.



b. If the shortest path exists between the nodes, the resultant graph displays this path by re-constructing the edges from source to destination node.

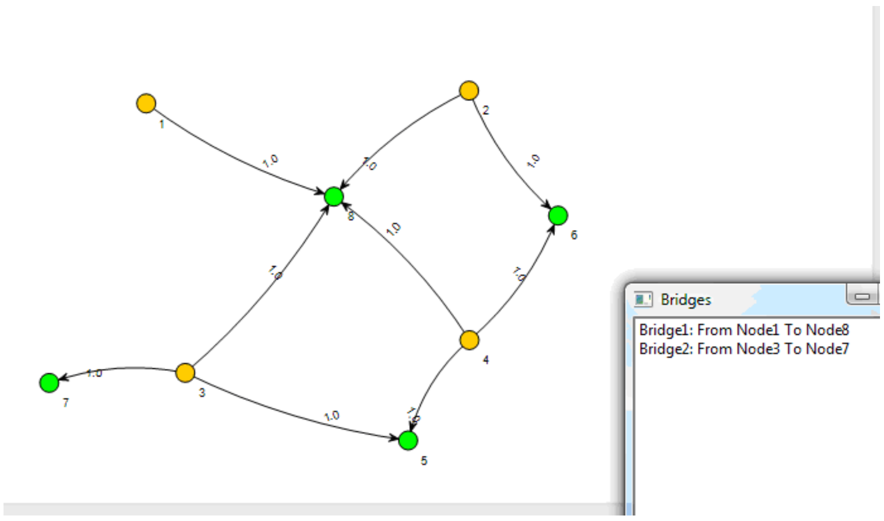




Note that the graph has single colored (yellow) nodes for a One-Mode social network and two-colored (yellow and green) nodes for a Two-Mode social network.

Finding Bridges

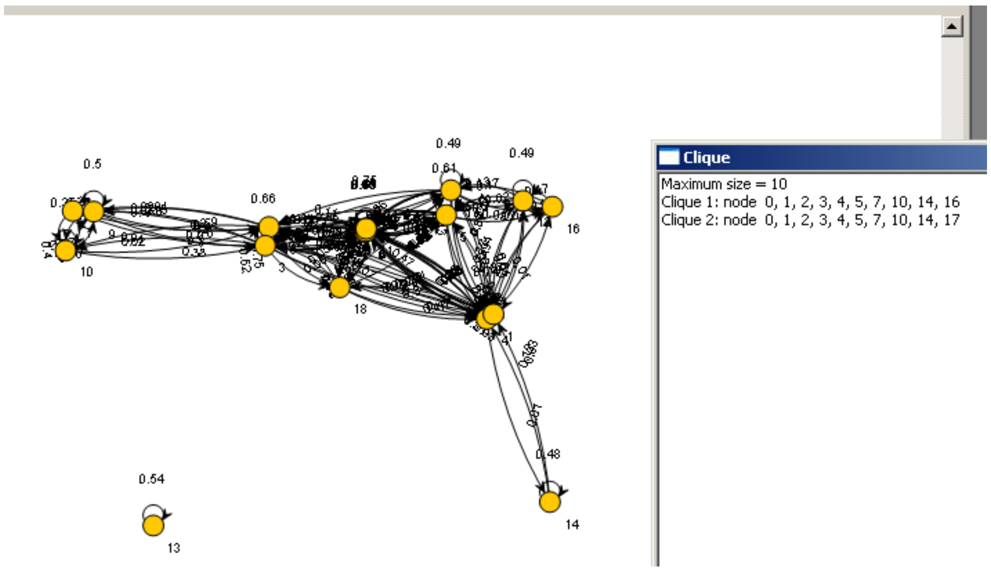
From ***Analysis*** menu, select ***Find Bridges***. This functionality allows you to find bridges in a given network. Following Figure shows an example:



Bridge1: From Node1 To Node8
Bridge2: From Node3 To Node7

Finding Cliques

From ***Analysis*** menu, select ***Find Cliques***. This functionality finds the maximum clique of a given graph. This problem is a NP-complete problem, so the main calculation of this part is to find a sequence of candidates of maximum cliques and then try all possible solutions from size = total number of node until maximum cliques is located.

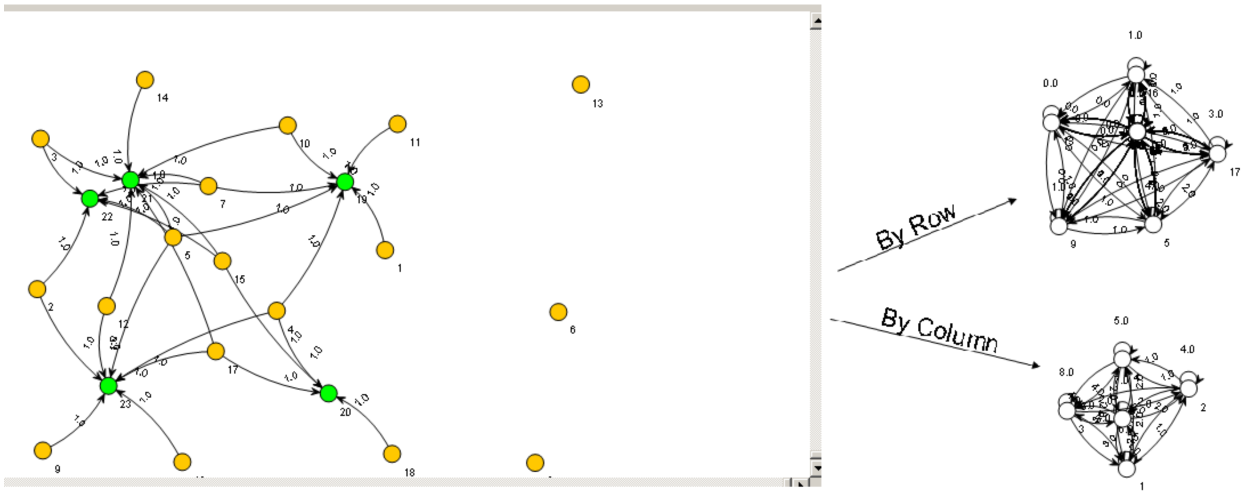


```
Maximum size = 10
Clique 1: node 0, 1, 2, 3, 4, 5, 7, 10, 14, 16
Clique 2: node 0, 1, 2, 3, 4, 5, 7, 10, 14, 17
```

Folding 2-mode to 1-mode

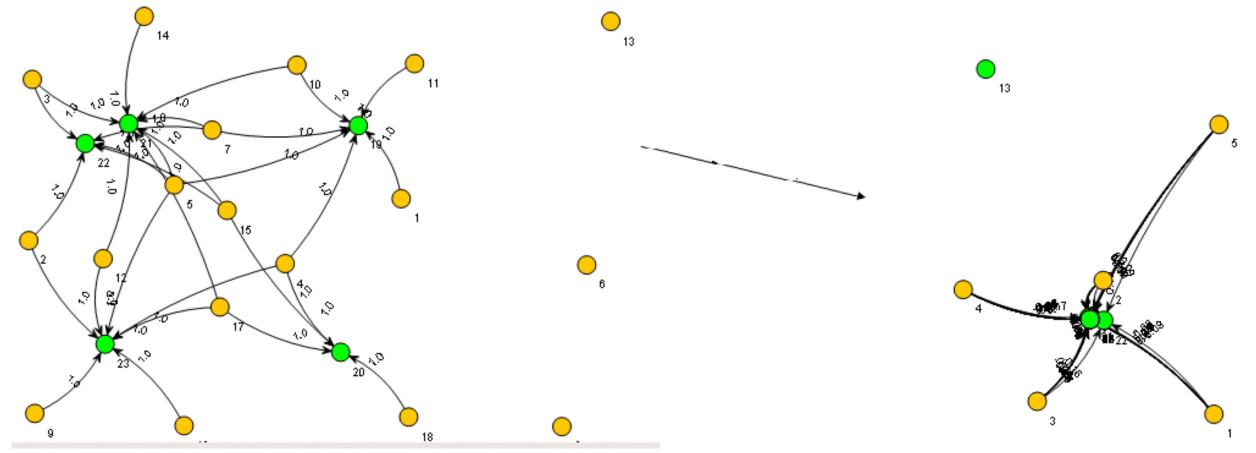
From ***Analysis*** menu, select ***Fold 2-mode to 1-mode***. This functionality provides folding option for the 2-mode networks (both by row and by column) and creating a 1-mode network.

A 2-dimensional matrix is said to be 2-mode if the rows and columns index different sets of entities (e.g., the rows might correspond to persons while the columns correspond to organizations). In contrast, a matrix is 1-mode if the rows and columns refer to the same set of entities, such as a city-by-city matrix if distances. Following figure shows *StudentCourse* example available in NetDriller home page – Sample Networks):



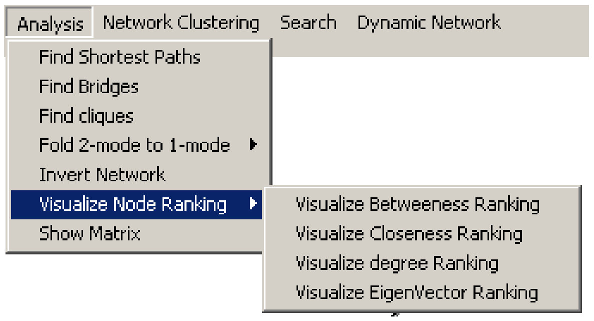
Inverting Network

From ***Analysis*** menu, select ***Invert Network***. This functionality allows you to visualize the inverse of the matrix given. If the matrix is singular, i.e. not invertible, the program will produce an appropriate error message.

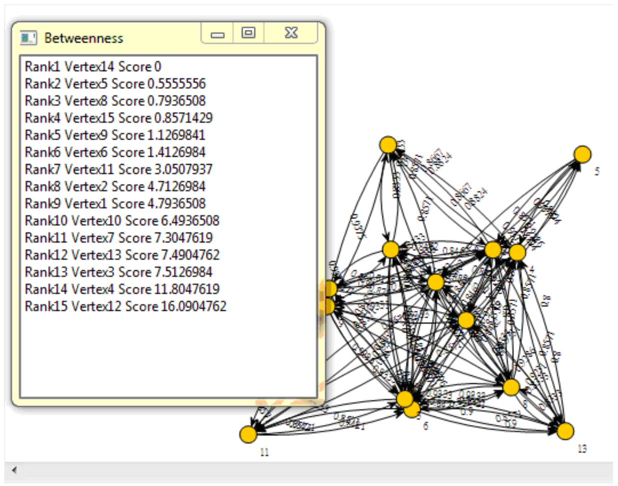


Ranking the Nodes

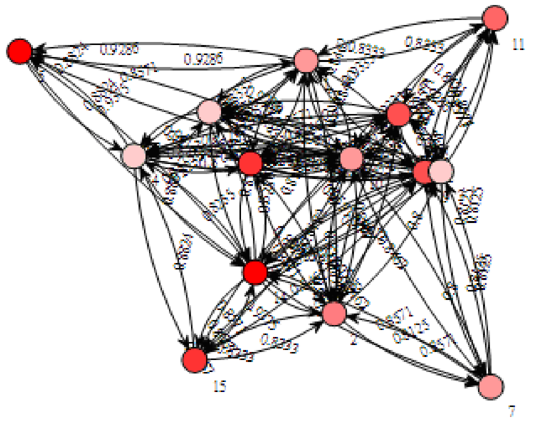
From ***Analysis*** menu, select ***Visualize Node Ranking***. From this submenu select ***Visualize Betweenness Ranking, Visualize Closeness Ranking***, ***Visualize Degree Ranking***, or ***Visualize Eigenvector ranking***:



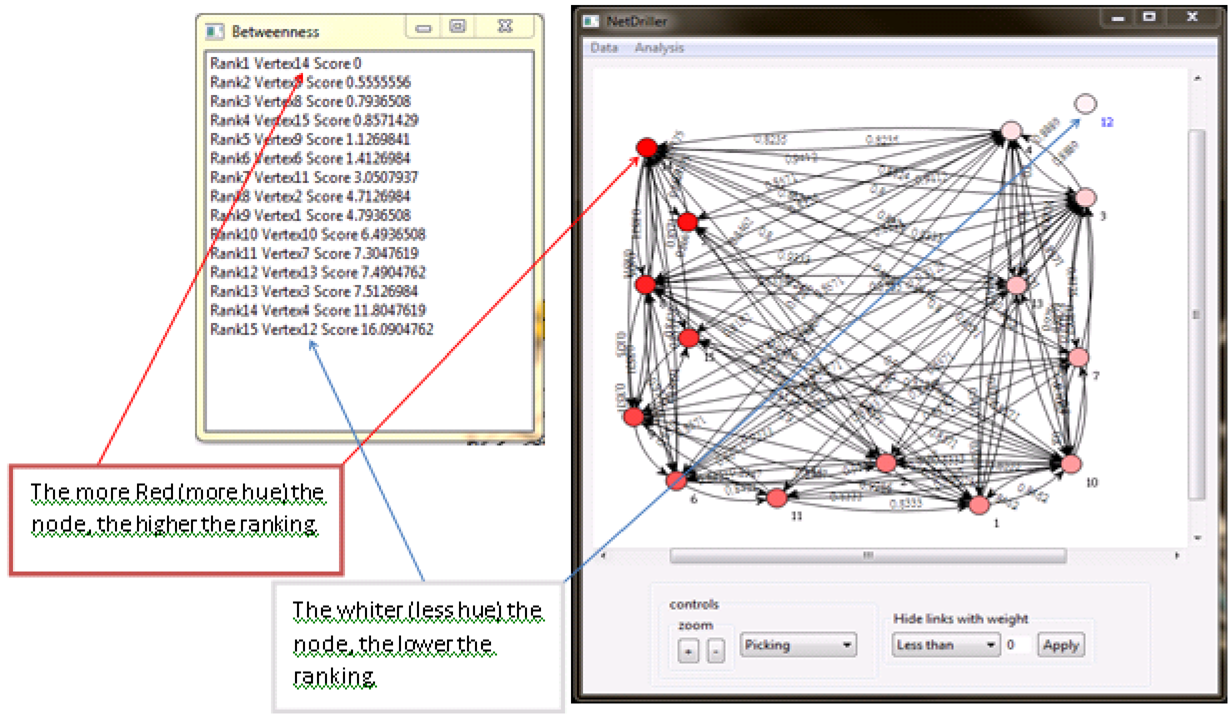
The ranking results will be show as text format in a window. By closing this window, the visualization of the ranking results will be shown



```
Rank1 Vertex14 Score 0
Rank2 Vertex5 Score 0.5555556
Rank3 Vertex8 Score 0.7936508
Rank4 Vertex15 Score 0.8571429
Rank5 Vertex9 Score 1.1269841
Rank6 Vertex6 Score 1.4126984
Rank7 Vertex11 Score 3.0507937
Rank8 Vertex2 Score 4.7126984
Rank9 Vertex1 Score 4.7936508
Rank10 Vertex10 Score 6.4936508
Rank11 Vertex7 Score 7.3047619
Rank12 Vertex13 Score 7.4904762
Rank13 Vertex3 Score 7.5126984
Rank14 Vertex4 Score 11.8047619
Rank15 Vertex12 Score 16.0904762
```



The ranking results will be expressed through red in sequential colors (differ in hues):



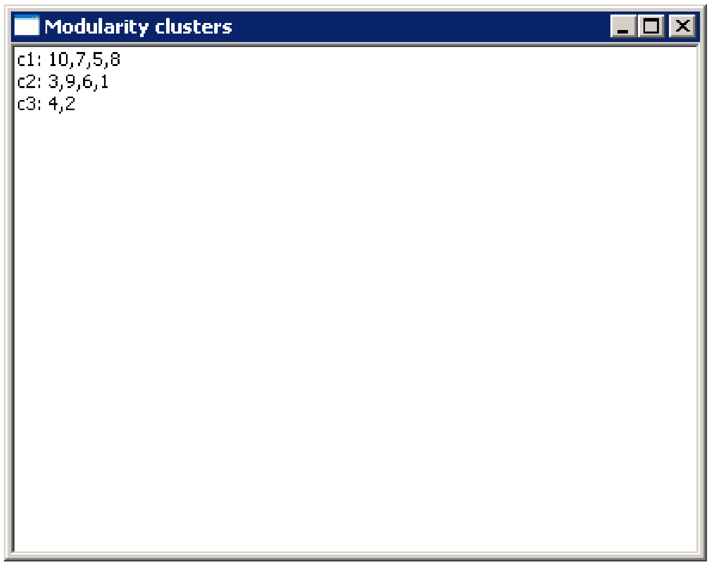
```
Rank1 Vertex14 Score 0
Rank2 Vertex5 Score 0.5555556
Rank3 Vertex8 Score 0.7936508
Rank4 Vertex15 Score 0.8571429
Rank5 Vertex9 Score 1.1269841
Rank6 Vertex6 Score 1.4126984
Rank7 Vertex11 Score 3.0507937
Rank8 Vertex2 Score 4.7126984
Rank9 Vertex1 Score 4.7936508
Rank10 Vertex10 Score 6.4936508
Rank11 Vertex7 Score 7.3047619
Rank12 Vertex13 Score 7.4904762
Rank13 Vertex3 Score 7.5126984
Rank14 Vertex4 Score 11.8047619
Rank15 Vertex12 Score 16.0904762
```

The more Red (more hue) the node, the higher the ranking.

The whiter (less hue) the node, the lower the ranking.

Clustering the Nodes

From ***Network Clustering*** menu, you can use any of the algorithms ***Modularity***, ***Edge Betweenness***, ***Voltage***, or ***Minimum Spanning Tree***, to find non-overlapping communities in the graph. Any of the clustering algorithms takes its own input parameters and finds clusters based on the input parameters. Following Figure illustrates the clusters found by Modularity clustering algorithm of *StudentsFriendships* example.



```
c1: 10,7,5,8
c2: 3,9,6,1
c3: 4,2
```

Here, we give a brief description of each of the clustering algorithms.

Modularity

In fact, our Modularity clusterer is an implementation of the Louvain method[1] which is a greedy optimization method. The method has proved to provide excellent results for a wide range of applications and is now one of the most widely used methods. The method consists of two phases. First, it looks for "small" communities by optimizing modularity in a local way. Second, it aggregates nodes of the same community and builds a new network whose nodes are the communities. These steps are repeated iteratively until a maximum of modularity is attained.

Edge Betweenness

Another commonly used algorithm for finding communities is the Girvan–Newman algorithm[2]. This algorithm identifies edges in a network that lie between communities and then removes them, leaving behind just the communities themselves. The identification is performed by employing the graph-theoretic measure betweenness, which assigns a number to each edge which is large if the edge lies "between" many pairs of nodes.

[1] V.D. Blondel, J.-L. Guillaume, R. Lambiotte and E. Lefebvre (2008). "Fast unfolding of community hierarchies in large networks". J. Stat. Mech.: P10008. doi:10.1088/1742-5468/2008/10/P10008

[2] M. Girvan and M. E. J. Newman (2002). "Community structure in social and biological networks". Proc. Natl. Acad. Sci. USA 99: 7821–7826. doi:10.1073/pnas.122653799

The Girvan–Newman algorithm returns results of reasonable quality. However, it also runs slowly, taking time $O\ (m^2n)$ on a network of n vertices and m edges, making it impractical for networks of more than a few thousand nodes.

Voltage

Voltage clustering was originally proposed by Wu and Huberman[3], based on the properties of resistor networks. Their algorithm is fundamentally a bisection algorithm, although they also give a version that will divide a network into a larger number of communities provided one knows in advance how many communities there are. The idea behind the algorithm is to consider the electrical circuit formed by placing a unit resistor on each edge of the network and then applying a unit potential difference between two vertices chosen arbitrarily. If the network divides strongly into two communities and the vertices in question happen to fall in different communities, then the spectrum of voltages on the rest of the vertices should, the authors argue, show a large gap corresponding to the border between the communities. We can thus identify the communities by finding the largest gap and dividing the vertices according to whether their voltages lie above or below it. Since the largest gap sometimes falls at the end of the spectrum, giving a highly asymmetric division of the network.

Here, we use a simplified implementation of the original algorithm. Note that the algorithm takes the number of candidate clusters as input, and the number of clusters in the output may differ from the number of candidate clusters.

Minimum Spanning Tree

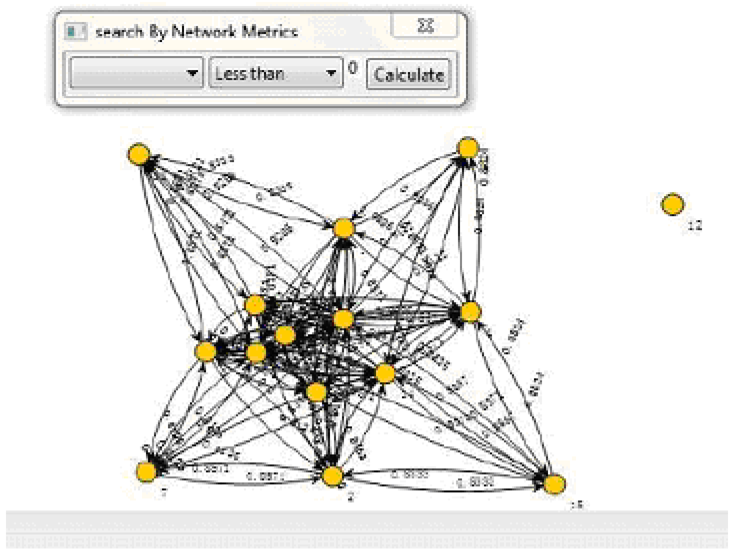
Minimum Spanning Tree (MST) clusterer first finds a minimum spanning tree in the graph. Then it removes high weight edges from the MST until it reaches the desired number of clusters which is specified by the user. The clustering result produced by this algorithm depends largely on the weight of the edges rather than the number of them.

[3] F. Wu and B. A. Huberman, Finding communities in linear time: A physics approach. Preprint condmat/0310600 (2003).

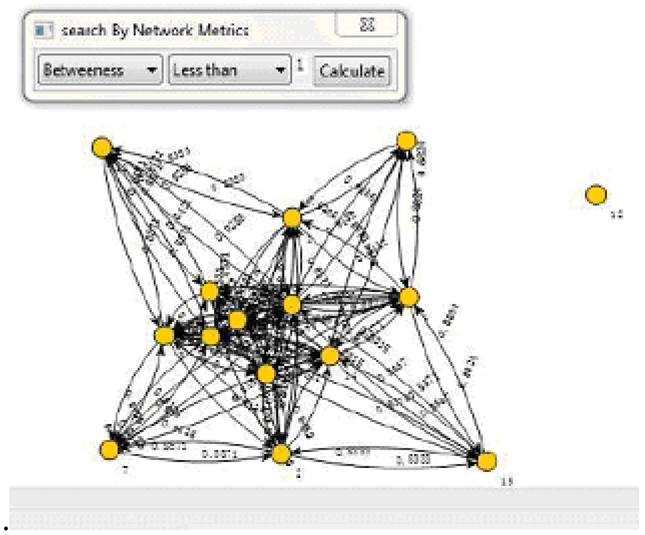
Searching the Nodes

Based on Metrics

From ***Search*** menu, select ***Search by Network Metrics***. This version of NetDriller support search only based on centrality metrics. The search by Network Metrics function box will come up.



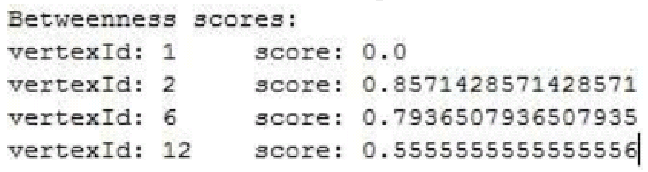
Choose one of the available options in the drop down menu (for example betweenness). Then choose the search option in the second drop down menu. Enter the values you want search:



Betweeness Less than 1

.

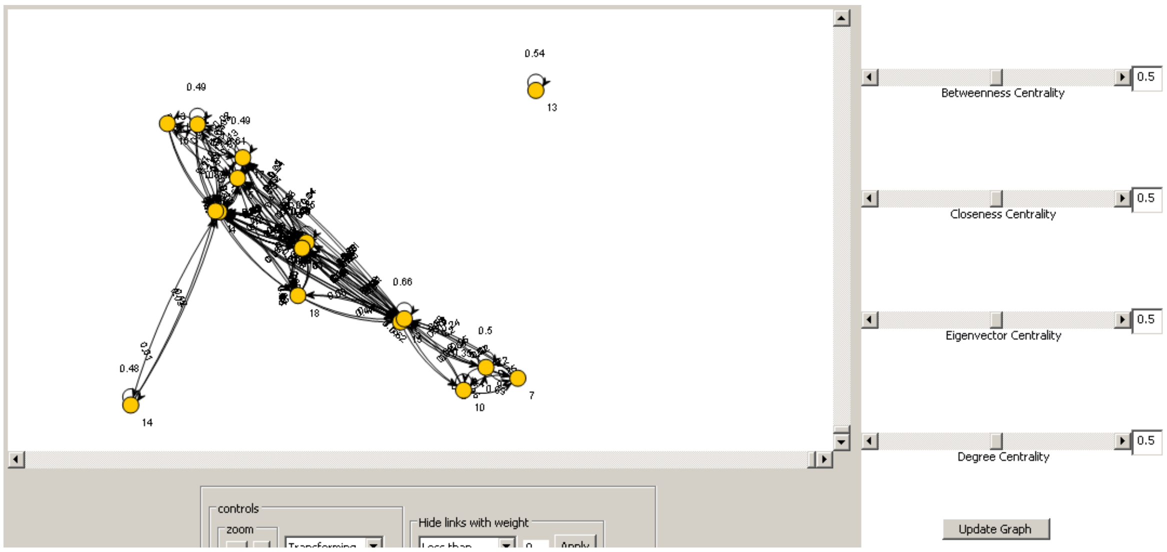
Then pass the Calculate button the result will show up.



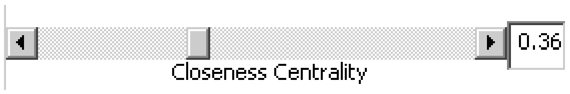
```
Betweenness scores:
vertexId: 1      score: 0.0
vertexId: 2      score: 0.8571428571428571
vertexId: 6      score: 0.7936507936507935
vertexId: 12     score: 0.5555555555555556
```

Fuzzy Searching

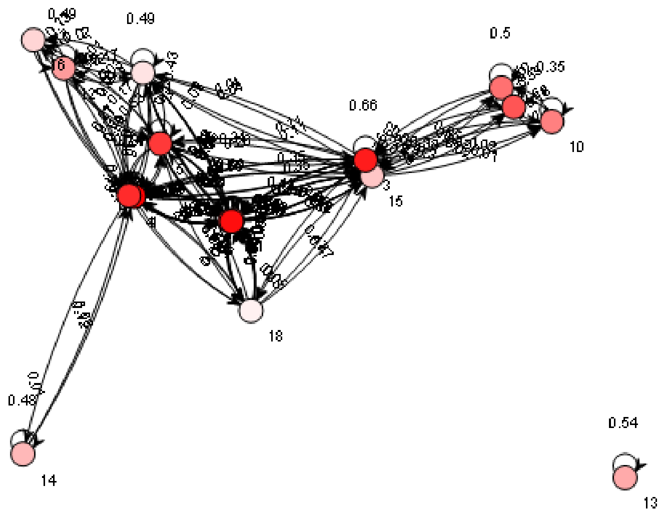
From ***Search*** menu, select ***Fuzzy Search***. This version of NetDriller support fuzzy search only based on centrality metrics. The sliders for betweenness, closeness, eigenvector and degree centralities will appear on the right side of the program:



Use the sliders to adjust the threshold values for each centrality (the default value of each centrality threshold is 0.5):

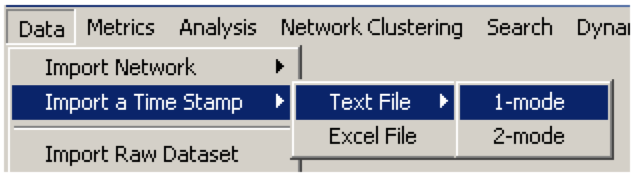


When the threshold values are all set, click the Update Graph button at the bottom, the graph will be updated. The graph will visualize the network based on the centralities threshold settings. The higher the ranking of the node, the brighter the red it has, the lower the ranking of the node, the whiter the node gets:

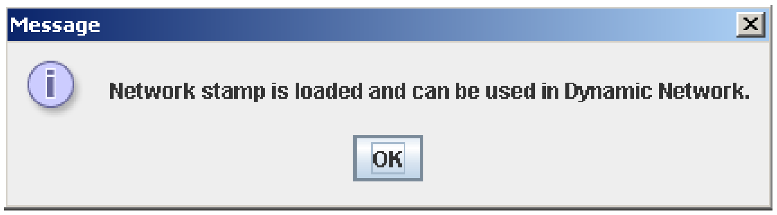


Dynamic Network

From ***Data*** menu, select ***Import a Time Stamp*** several times to import all the network stamps in time order (use *Net1*, *Net2* and *Net3* available in the NetDriller home page – Sample Networks):

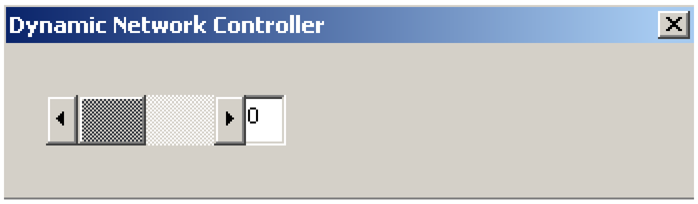


Following message shows that each network is added to the list of networks that can be used later by Dynamic Network controller.

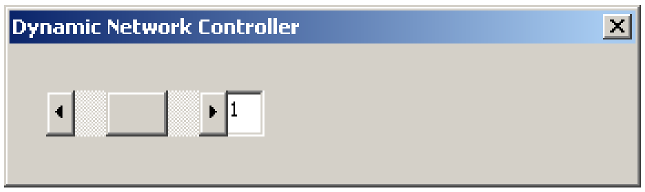


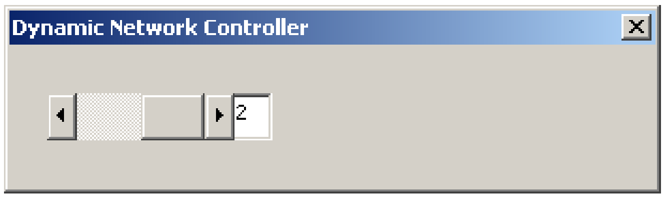
Network stamp is loaded and can be used in Dynamic Network.

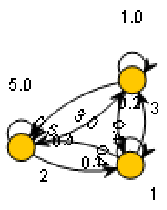
After adding all the stamps, select ***Dynamic Network*** menu. Dynamic Network controller appears:

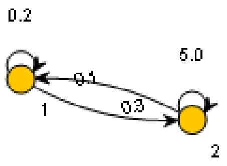


By clicking on the slider various time stamps are visualized:


Dynamic Network Controller
1


Dynamic Network Controller
2




0.2
5.0
0.1
0.3
1
2