Supplemental Material

CBE—Life Sciences Education

Chai *et al.*

Applying graph theory to examine the dynamics of student discussions in small-group learning

Supplemental material

Albert Chai[1], Joshua P. Le[1]*, Andrew S. Lee[2]*, Stanley M. Lo[1,3,4]

[1] Division of Biological Sciences, [3] Section of Cell and Developmental Biology, [4] Program in Mathematics and Science Education, University of California San Diego, La Jolla, CA 92093.

[2] Department of Computer Science, University of California Los Angeles, CA 90024.

* These authors contributed equally to this work.

Corresponding Author:

Stanley M. Lo

9500 Gilman Dr #0355

La Jolla, CA 92093-0355

Email: smlo@ucsd.edu

Phone: (858)246-1087

Subgraphs and special subgraphs

A subgraph is a smaller graph within a graph. Below are some special subgraphs useful for determining highly connected groups.

Neighborhood of a node: The immediately connected nodes

The neighborhood of a node is the set of nodes that are connected to it by a single edge (Godsil, 2001).

K-core: Subgraph with nodes of degree at least K

The K-core of a graph is the subgraph including as many nodes as possible where each node has degree at least K (Borgatti, 2013) (Figure S1).

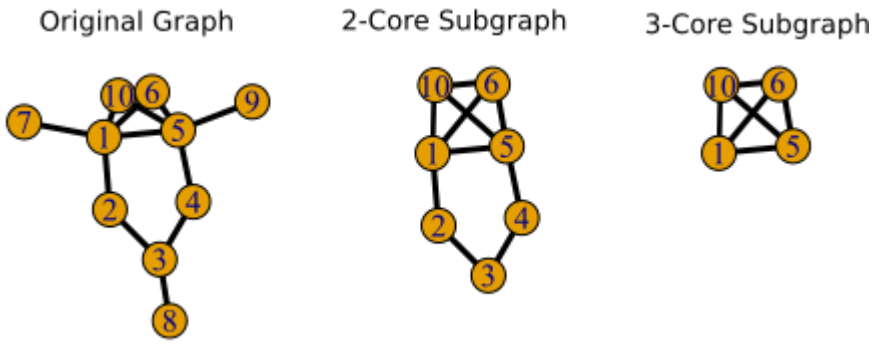


Figure S1: A graph with its 2-core and 3-core subgraphs. As the k in k-core increases, there are fewer vertices that satisfy the degree requirement. Each vertex in the k-core subgraph has degree at least k among the vertices in the subgraph.

Directed graphs, weakly and strongly connected

A directed graph is weakly connected if replacing all of its directed edges with undirected edges results in a connected graph (Godsil, 2001). A weakly connected component is a subgraph of a directed graph that is weakly connected (Figure S2).

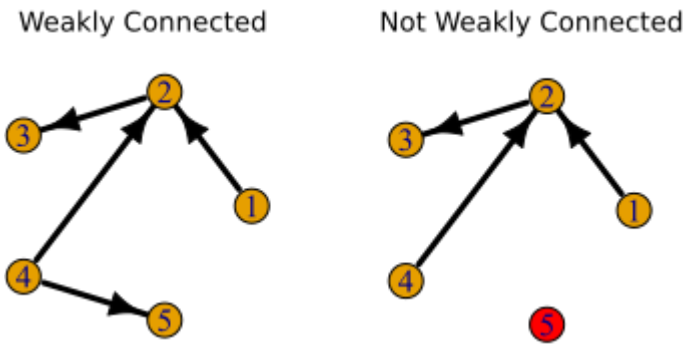


Figure S2: Two graphs, one weakly connected and the other not weakly connected. The right graph is not weakly connected because converting the directed edges to undirected does not result in a connected graph (node 5 is still not connected).

A directed graph is strongly connected if there is a directed path between any two vertices (Godsil, 2001). A strongly connected component is a subgraph of a directed graph that is strongly connected (Figure S3).

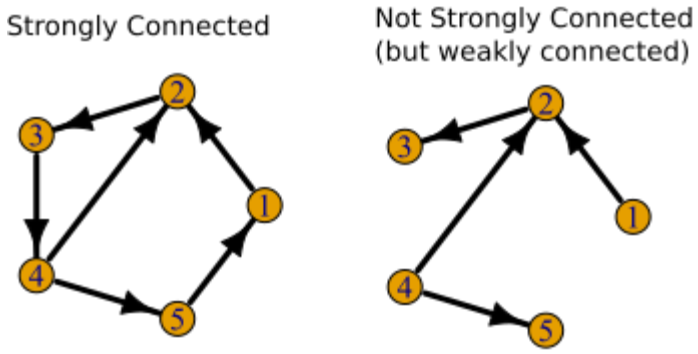


Figure S3: Two graphs, one strongly connected and the other not strongly connected. The right graph is not strongly connected because there is not a directed path from 3 to 2, from 2 to 1, 5 to 4, etc. Both graphs, however, are weakly connected.

Clique: A fully connected subgraph

A clique is a subgraph that is fully connected, meaning that all edges that could possibly exist do exist (Godsil, 2001) (Figure S4).

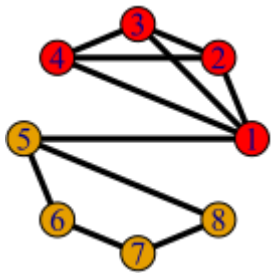


Figure S4: A graph with a clique (nodes 1, 2, 3, and 4). Nodes 5, 6, 7, and 8 are not included in the clique because nodes in a clique must all be directly connected to each other.

Graph metrics

In this section, we define a variety of metrics one can calculate about graphs, namely the parameters output by the R script mentioned in Methodology section.

Order and size: Counting

The order of a graph is the number of vertices is has, i.e. the number of elements in the vertex set V. Similarly, the size of a graph is the number of edges a it has, i.e. the number of elements in the edge set E. The order is sometimes denoted |V|, and size is denoted |E|.

Modularity: Graph divisions

Modularity is a measure of how well one can separate a graph into distinct groups. It is used to identify densely connected community structures within the entire graph. To calculate modularity, one must have a graph and a group assignment to each vertex already in mind; in other words, modularity is specific to a graph as well as how one picks the groups.

A graph and group assignment with high modularity (close to 1) indicates that there are many edges within the groups that were picked and fewer edges connecting groups. A graph and group assignment with low modularity (close to -1) indicates the opposite: fewer edges within groups and more edges connecting groups.

The modularity of a graph and group assignment is the sum over all groups of the fraction of edges within the given group minus the expected fraction of edges within the given group. Another way to visualize it is to let M be a list of all group numbers, then the modularity is given by the following formula:

$$Modularity = \sum_{m \in M} \left(\frac{\#\ of\ edges\ within\ group\ m}{\#\ of\ edges} - \frac{\#\ of\ possible\ directed\ edges\ within\ group\ m}{\#\ of\ possible\ directed\ edges\ in\ the\ graph} \right)$$

If we pick a particular group number (e.g. m = group 1), then the first fraction is “edge density” within group 1. The second fraction acts as a normalizing term; it accounts for the “edge density” within group 1 that is expected to occur if all edges in the graph were given random edge assignments. In Figure S6, the same graph is used to show the effect of different group assignments.

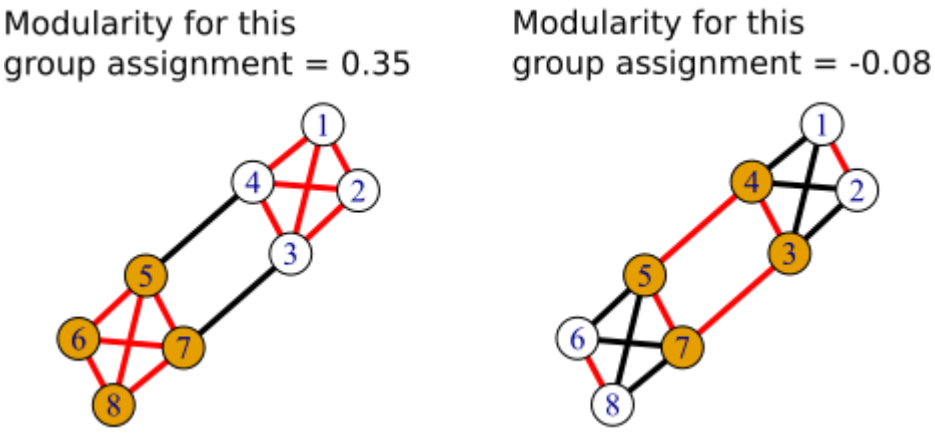


Figure S6: The same graph with different group assignments and their associated modularities. Edges colored red are within the group, and edges colored black are between groups. Group assignments that have more edges within the group and fewer edges between groups have higher modularity.

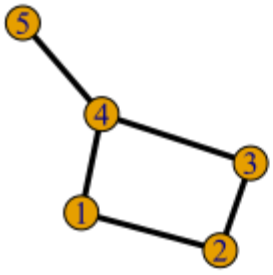
Graph representation and visualization

This section focuses on how to represent a graph (in code) and different ways graphs can be drawn for clarity.

Adjacency matrix: A graph representation

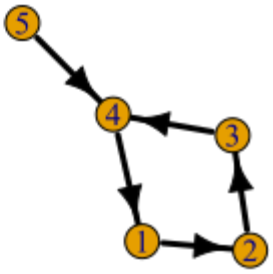
The adjacency matrix for a graph is a n by n (also written n x n) matrix where n is the number of vertices in the vertex set. Each number (or entry) in the matrix--take for example the number in the ith row and jth column--corresponds to whether or not there is an edge from vertex i to vertex j (Godsil, 2001).

Note that the adjacency matrix of an undirected graph is symmetric, meaning it is the same when flipped across the diagonal that goes from the top-left corner to the bottom-right corner. On the other hand, the adjacency matrix for a directed graph is not necessarily symmetric. See Figure S7 for examples of adjacency matrices for undirected and directed graphs.



Adjacency Matrix:

	1	2	3	4	5
1	0	1	0	1	0
2	1	0	1	0	0
3	0	1	0	1	0
4	1	0	1	0	1
5	0	0	0	1	0



Adjacency Matrix:

	1	2	3	4	5
1	0	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	0
4	1	0	0	0	1
5	0	0	0	1	0

Figure S7: Adjacency matrices for two graphs, undirected and directed. The undirected graph has a symmetric adjacency matrix while the directed graph does not.

Projections: Different ways to draw a graph

Our R script allows users to display the graphs created in various ways, which are covered in this section.

The Fruchterman Reingold and Kamada Kawai projections are force-directed layouts, meaning they use attractive and repulsive forces inspired by physics to place vertices and edges (Wild, 2016).

The Reingold Tilford projection clearly encodes the depth of a graph, which may be needed if path lengths are of interest (Kaufmann, 2003).

The Bipartite projection attempts to separate the graph into two rows of vertices with edges crossing in between rows, but not within a row (Wild, 2016).

Link to code and usage

The code used to run the analysis and usage instructions can be found at:

https://github.com/ucsd-lo-group/social-network-analysis/tree/gen1

Before running the code, you need one record of talk-turns in the question-and-response format described in the methods section.

To run the code, download the GitHub repository using the link above. There is a button labeled “Clone or download” that allows you to download a zip file of all the code. You must choose a location on your computer to save it and unzip the file.

Next, run the MATLAB code. Open MATLAB and set the working directory to the location of the unzipped code folder. Now you should be able to run “rawdatamatrixprocessor” in the console. It will guide you through the process of tabulating the talk-turns.

Next, run the R code. Open RStudio and set the working directory to the location of the unzipped code folder. Next, open the “start.R” script and click the “Source” button at the top. This script will guide you through analyzing the data and outputting statistics.

References

Borgatti, S. P., et al. (2013). Analyzing Social Networks. London, Sage Publications Ltd.

Chai, A., Lee, A.S., Le, J.P. (2018). Social Network Analysis Script. Retrieved from https://github.com/ucsd-lo-group/social-network-analysis/tree/gen1.

Godsil, C. and G. Royle (2001). Algebraic Graph Theory. New York, Springer-Verlag: 443.

Kaufmann, M. and D. Wagner (2001). Drawing Graphs: Methods and Models. Berlin, Springer, Berlin, Heidelberg.

Wild, F. (2016). Learning Analytics in R with SNA, LSA, and MPIA. Switzerland, Springer, Cham.