Synchronizability and Connectivity of Discrete Complex Systems

Michael Holroyd

November 16, 2006

Accepted for: _____

_____

_____

_____

Department of Mathematics
The College of William and Mary
Williamsburg, VA

Contents

Abstract

The synchronization of discrete complex systems is critical in applications such as communication and transportation networks, neuron respiratory systems, and other systems in which either congestion can occur at individual nodes, or system wide synchrony is of importance to proper functionality. The first non-trivial eigenvalue of a network's Laplacian matrix, called the algebraic connectivity, provides a quantifiable measure of synchronizability in a network. We study the general relationship between network topology, clustering coefficient distributions, and synchronizability, as well as the effects of degree preserving rewiring on network synchronizability. In addition, we compare the synchronizability of different network topologies, including Poisson random graphs, geometric networks, preferential attachment networks, and scale-rich networks. We also explore uses of the algebraic connectivity in the design and management of complex networks where synchronization is desired (respiration networks), or detrimental to network performance (router networks).

Acknowledgements

Special thanks to:

John Hayes, for excellent help with his simulation and helpful insight into autocorrelation analysis.

Rex Kincaid, my advisor and mentor for the project.

Christopher Del Negro, member of the commitee, for the opportunity to work with him and John on their research.

Sebastian Schreiber, member of the commitee.

Chapter 1

Background

Discrete complex system is a general term describing any system with many distinct elements coupled in a non-linear fashion. For example, in most descriptions of the Internet's topology each router is one distinct element and routers are coupled based on their physical connections around the globe. Other examples of discrete complex systems include food webs, power grids, the World-Wide-Web, social interactions, and neuron networks. Unlike simple systems, an understanding of complex systems cannot be gained by only examining the individual components. Modeling the way these components are interconnected is vital to understanding the system as a whole. *Graphs* provide a mathematical construct to study how these complex systems are connected. Consequently, the study of complex systems is closely related to the study of graphs.

1.1 Graph Theory

A *graph* (or *network*) is a set of vertices with edges between them. *Vertices*, also called nodes, are the objects that make up our system, while *edges* describe how they are connected. For example, in a model of the World-Wide-Web each website is a vertex and each hyperlink is an edge connecting two vertices. Depending on the complex system in mind, edges can be either *directed* or *undirected*. In our World-Wide-Web example edges are directed, since a link from your homepage to CNN.com does not guarantee a link from CNN.com to your homepage. In contrast, a graph of the Internet such as Figure 1.1 has undirected edges, since routers (in general) send and receive packets for all their connections. We will only consider undirected edges in this project; however, the experiments can be extended to include directed edges.

Vertex i and j are said to be *adjacent* if they have an edge connecting them. The *degree* of vertex i, denoted k_i, is the number of vertices adjacent to i (for directed graphs there is a distinction between a vertex's in-degree and out-degree). In addition to the assumption that our graphs are undirected, we will also assume our graphs are *simple*, which means our graphs do not have self-loops (edges that start and end at the same vertex) or multi-edges (more than one edge connecting two vertices). Although these may seem like further restrictions, graphs with self-loops or multi-edges can be

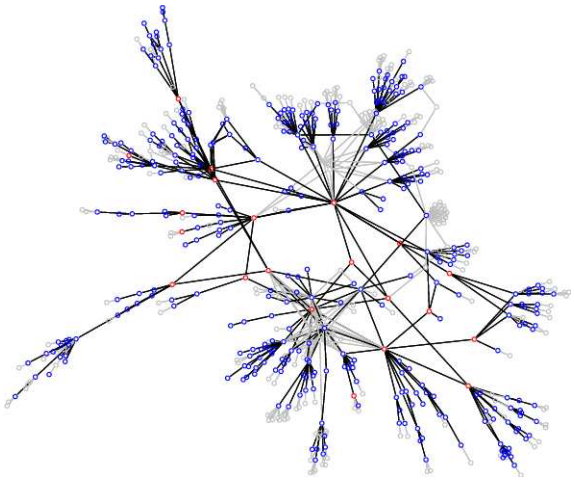


Figure 1.1: A graph of a medium-sized ISP's core network. Blue and red nodes represent ISP routers, while grey nodes represent customers. Edges show which routers are physically connected to each other, based on hop-limited packet probes. Created as part of the Scan project by Ramesh Govindan, Anoop Reddy and colleagues, at the Information Sciences Institute. (http://www.isi.edu/scan/scan.html)

made equivalent to simple graphs by duplicating the offending vertex.

It is intuitive to depict graphs visually using circles for vertices and lines for edges, but actual manipulation and storage of graph information for computers requires the use of *matrices*, which will be discussed in section 1.3.

1.1.1 Properties of graphs

Degree distributions

The simplest tools for the categorization of complex systems is identifying the associated graph's degree probability density function, or just *degree distribution*,

$$p(k) = \frac{1}{n} \sum_{\forall i | k_i = k} 1 \tag{1.1}$$

where n is the number of nodes in the graph, and k_i is the degree of the i^{th} node. In other words $p(3)$ would be the fraction of vertices with a degree of 3. By plotting this distribution, we discover that most well-formed graphs follow either Gaussian, exponential, or power-law shaped curves. We can distinguish between the most fundamentally different graphs based on which curve their degree distribution follows, and large amounts of research has been done to show what properties are specific to each distribution.

In addition, the *cumulative degree distribution* is defined to be

$$P(k) = \frac{1}{n} \sum_{i \leq k} p(i) \tag{1.2}$$

and contains the exact same information; however, as vividly illustrated in [LAT+05], the simple degree distribution in equation 1.1 can often be visually misleading compared to plots of equation 1.2. As an example, observe figure 1.2. In this example an exponential distribution could be easily mistaken for a power-law distribution if equation 1.1 were plotted. With this in mind, we will only use the cumulative distribution function in figures throughout the project.

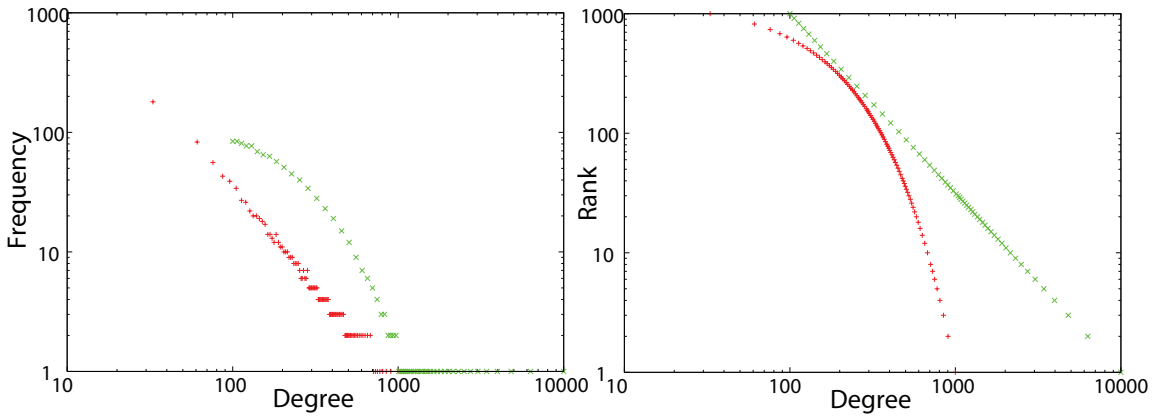


Figure 1.2: A carefully chosen exponential distribution (plotted in red +) and power-law distribution (plotted in green ×). The plot on the left is the *degree distribution*, whereas the plot on the right is the *cumulative degree distribution*. Notice that both x and y axis are in log scale, so a power-law distribution $y = x^{-\alpha}$ should appear as a straight line. Unfortunately, the exponential distribution incorrectly appears to follow a power-law in the frequency plot, while in the cumulative plot the power-law distribution can be correctly identified.

Clustering coefficients

In the search for an effective secondary characteristic to use in narrowing this broad classification, several researchers have noticed that naturally forming systems tend to display *modularity*. We consider a network to be modular if it contains groups of nodes that are very well connected amongst themselves, but loosely connected to nodes outside their group. A related measure is the *clustering coefficient* of a vertex. Specifically, *clustering* occurs when two different neighbors of a given node also happen to be connected. This occurs frequently in real-world complex systems. For example, in social networks if you are friends with two people it is very likely that they are also friends with each other. The formal definition of the clustering coefficient for vertex i is

$$c_i = \frac{t_i}{\binom{k_i}{2}} \tag{1.3}$$

where t_i is the number of times two vertices adjacent to i are connected, or visually the number of completed triangles connected to i. So, c_i is a ratio of how many of i's neighbors are connected, over how many could possible be connected. See the example in figure 1.3.

Although the average clustering coefficient gives a general sense of how much clustering is occurring in a graph, even more information can be gleaned from the *clustering coefficient distribution*. Combined with information about a graph's degree distribution, the clustering coefficient distribution provides a second tier of categorization amongst graphs. In particular, this gives us a way to distinguish between graphs which follow a strict hierarchy in their construction, and graphs that are completely unstructured.

Shortest path length

Another key measure when considering graphs of complex systems is the *average shortest path length*, which is taken over all pairs of nodes. Psychologist Stanley Milgram, after his sociology experiments in 1976, showed that the average shortest path length between two random US citizens was about 6 acquaintances, and coined this the *small-world effect*. It turns out that many other complex systems also exhibit this same property, and despite a large number of vertices, a path with a small number of edges on the order of $\log(n)$ exists between almost any two nodes (although finding this path is not always easy). In popular culture this phenomena is often referenced as the "six degrees of separation", a theory that anyone on earth can be connected to another in six links, and has been popularized by a movie of the same name as well as the "six degrees of Kevin Bacon" trivia game.

1.1.2 Types of graphs

Regular graphs

The simplest graphs have a fixed degree distribution, meaning that every vertex has the same degree. These graphs are called *regular graphs*. There are a few special cases of regular graphs, including *complete graphs*, where every vertex is connected to all other vertices, and *null graphs*, which have no edges. Another special case of regular graphs are *lattices*, which have evenly spaced vertices, each connected only to the nearest neighbors such as Figure 1.4. An interesting property of lattices is that, although they do not display the small-world property, by taking a small number of randomly chosen edges and "rewiring" them to another vertex, small-world properties emerge very quickly. [WS98]

Results on regular graphs are usually very straightforward; however, they they correspond only to oversimplifications of complex systems.

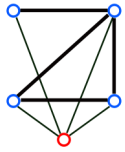


Figure 1.3: The clustering coefficient of the red vertex is $\frac{4}{6}$, since only 4 out of the possible 6 edges between the red node's neighbors exist.

Erdös-Rényi random graph

Perhaps the most studied variety of graph is the *Erdös-Rényi random graph*, or $G_{n,p}$ construction, in which we take n vertices, and between each pair of vertices create an edge with probability p. First defined by Paul Erdös and Alfréd Rényi in [ER59], random graphs were the first big step in graph theory toward modeling discrete complex systems. Many results can be obtained for these graphs in the limit for large n. In particular, this type of graph is often referred to as a Poisson random graph, since for large n the degree distribution tends toward a Poisson distribution [CNSW00]. These graphs are useful for systems with connections best described by an average degree, and mimic some properties seen in real networks such as the small-world effect. Unfortunately, they do not exhibit any clustering, and real-world complex networks only rarely have a Poisson degree distribution.

Power-law degree distribution

As the capacity for computers to manage large quantities of data increased, it became clear that the old Erdös-Rényi random graphs were not a good reflection of the complex systems researchers were interested in studying. Upon inspection, the vast majority of real-world networks display a *power-law* cumulative degree distribution of the form

$$P(k) = k^{-(1+\gamma)} \tag{1.4}$$

where γ is a constant, and in practice for most networks $2 \leq \gamma \leq 3$ [BA99][New03]. Unlike Erdös-Rényi random graphs, these networks have numerous vertices with very low degree, and far smaller numbers of high degree vertices. See figure 1.5 for an example power-law degree distribution. The most prominent explanation for why real world systems follow this pattern is called *preferential attachment*, discussed in section 2.2.2, an algorithm for naturally forming graphs with a power-law degree distribution. The actual reason for the ubiquity of power-law degree distributions is still undecided; however it has been shown that high variability self-dependent data like what we often see in complex networks is not subject to the Central Limit Theorem, and should therefore be expected to take on other forms than Gaussian [WADL04].

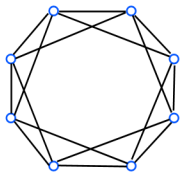


Figure 1.4: An example of a *lattice*.

1.1.3 Scale-free vs. Scale-rich

The term *scale-free* was coined in 1998 by Barabási Albert-László when they examined a portion of the World Wide Web with the help of a web-crawler, and discovered a power-law degree distribution, instead of the expected Poisson distribution. As more information and computing power became available it was quickly discovered that a majority of complex systems had a similar structure, and subsequently it has become a hot topic for research, with Barabasi's original paper receiving over 750 citations since 1999.

Several different criteria for "scale-freeness" have been proposed. Barabási's original description included any graph with a power-law degree distribution; however, many more properties have been assigned to it by other researchers. A rigorous, structural definition of scale-free was given in [LAT+05]. For a given power-law degree distribution:

$$s(g) = \sum_{(i,j)\in E} k_i k_j \tag{1.5}$$

where k_m is the degree of node m, and $(i, j) \in E$ means that vertex i and j have an edge between them. In this definition graphs with a high $s(g)$ are called *scale-free*, while graphs with a low $s(g)$ are called *scale-rich*. $S(g)$ is the normalized version of $s(g)$, but as of yet there is no algorithm for constructing the largest $s(g)$ graph.

Scale-free graphs occur when high-degree nodes connect to each other, and scale-rich graphs occur when high-degree nodes are separated and connect only to lower degree nodes (see figure 1.6). Together these two terms help to separate the set of graphs with power-law degree distributions into two separate smaller categories. In addition to their core structure of interconnected high degree nodes, scale-free networks are also *self-similar* in the same sense as fractals: a small subgraph of the network looks similar to the overall network (see figure 1.7). Finally, given any graph with a power-law degree distribution, degree-preserving random rewiring has a high probability of creating a scale-free graph as opposed to a scale-rich one. In this sense scale-free graphs are considered more random and naturally occurring than scale-rich graphs, which have to be artificially constructed and do not often arise naturally [LAT+05].

1.2 Linear Algebra

In this project we assume an elementary knowledge of linear algebra, including matrix multiplication, inverses, and determinants. For a review of this material see [Lay94]. The material presented here is the minimum needed to introduce the notion of algebraic connectivity, which will be our metric to measure the synchronizability of a graph.

1.2.1 Transpose and Hermitian adjoint

If a matrix $A = [a_{i,j}]$, then the *transpose* of A, denoted A^T, is the matrix with rows and columns switched, that is $A^T = [a_{j,i}]$. The *Hermitian adjoint* of A, denoted A^*,

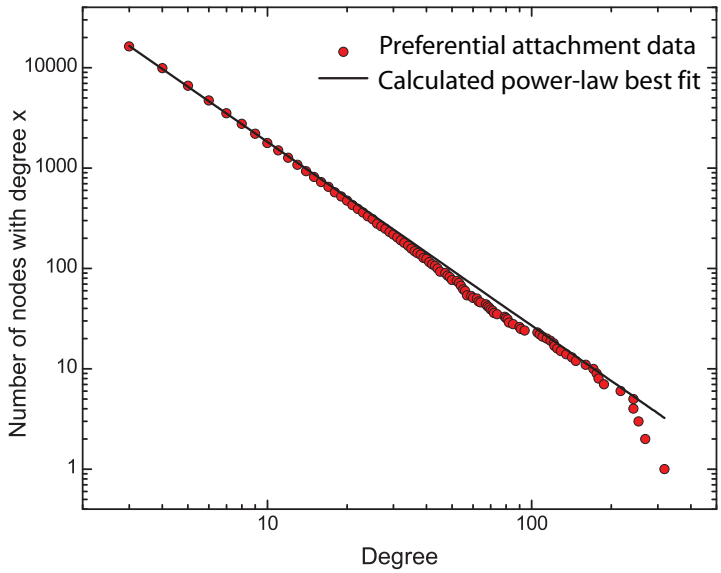


Figure 1.5: An example of a power-law degree distribution shown on a log-log plot. Data is from a random preferential attachment graph (see section 2.2.2). Solid line is a best-fit approximation power-law curve.

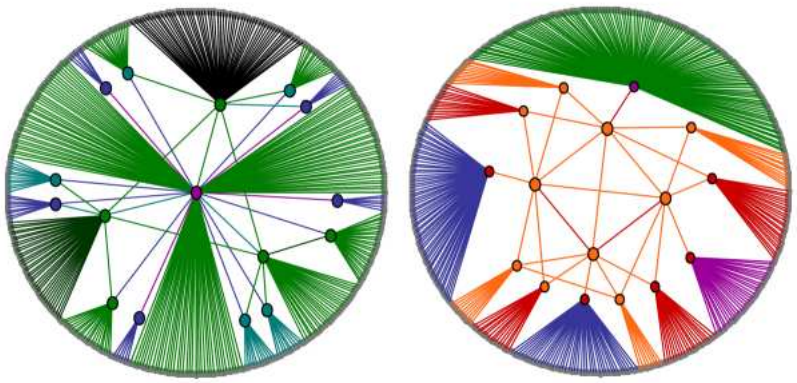


Figure 1.6: Two graphs with the same degree distribution that can be distinguished by their "scale-freeness". On the left is a *scale-free* graph, with a high degree hub. The right graph is HOTnet, a *scale-rich* graph with a low degree central core developed in [LAT+05] as an efficient Internet router topology.

is defined to be

$$A^* = \bar{A}^T = [\bar{a}_{j,i}] \tag{1.6}$$

where $\bar{x}$ is complex conjugate of x, i.e. $\overline{(a+bi)} = (a-bi)$. In other words we take the complex conjugate of each entry, and then the transpose of the matrix.

1.2.2 Eigenvalues

A vector x and scalar λ are said to be an *eigenvector* and its associated *eigenvalue* of a matrix A if $x \neq 0$ and

$$Ax = \lambda x \tag{1.7}$$

In other words, multiplying A by the vector x results in a multiple of x. For example,

$$\begin{bmatrix} 0 & 2 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 2 \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{1.8}$$

Interpreted geometrically, if we think of A as a transformation, a vector is an eigenvector if its direction is not changed after the transformation. For example, a transformation that only stretches all vectors along the x direction would have an eigenvector $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$. The requirement that $x \neq 0$ is necessary, since $A \cdot 0 = \lambda \cdot 0$ for any A and λ.

Notice that Eq. 1.7 is equivalent to

$$(A - \lambda I)x = 0 \tag{1.9}$$

which has a non-trivial solution exactly when $(A-\lambda I)$ is singular, which is equivalent to $\det(A - \lambda I) = 0$ having a non-trivial solution. This determinant is a polynomial, and so $\det(A-\lambda I)$ is called $p_A(t)$, or the *characteristic polynomial* of A, and the roots of $p_A(t)$ are the eigenvalues of A. Since every polynomial of degree n is guaranteed by the fundamental theorem of algebra to have exactly n complex roots, it also follows that any $n \times n$ matrix is guaranteed to have n complex eigenvalues. We will call these n (not necessarily distinct) eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$.

1.2.3 Hermitian matrices

A matrix is said to be *Hermitian* if $A = A^*$, in other words A is it's own Hermitian adjoint. Notice that for a matrix with all real entries $A^* = A^T$, therefore a matrix with all real entries is Hermitian if and only if $A = A^T$, the formal way of expressing that A is symmetric. Hermitian matrices have numerous useful properties, and are often viewed as a natural generalization of real numbers to matrices. A property of Hermitian matrices important to this project is that the eigenvalues of a Hermitian matrix are always real numbers. The proof of this is simple, but requires more background than is included in this project, see [HJ85]. Since each eigenvalue is a real number, it is natural to order the eigenvalues of Hermitian matrices such that $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$.

1.2.4 Positive definite matrices

A subset of Hermitian matrices with even more special properties are *positive definite matrices*. A Hermitian matrix is positive definite if and only if all its eigenvalues are greater than 0. Similarly, a Hermitian matrix is said to be *positive semi-definite* if and only if every eigenvalue is greater than or equal to 0. If Hermitian matrices are analogous to real numbers matrices, then positive definite matrices are analogous to the positive numbers.

1.3 Algebraic methods in graph theory

A useful tool for storing and manipulating graphs is the *adjacency matrix*. In an adjacency matrix, entry $a_{i,j}$ is 1 if there is an edge between vertex i and vertex j, and is 0 otherwise. This form is often the easiest to analyze and manipulate. Another related data structure is the *adjacency list*, an array containing one list for each node of all adjacent nodes. An adjacency matrix requires $O(n^2)$ space, since it includes information about every possible link regardless of whether it is present or not, while an adjacency list requires only $O(n \cdot \bar{k})$ where $\bar{k}$ is the average degree. For sparse graphs with many vertices it is more space efficient to use the adjacency list; however, the disadvantage is that in practice lists are more difficult and computationally expensive to manipulate than a matrix.

The *degree matrix* of a graph is a diagonal matrix with $a_{i,i} = k_i$ and all other entries 0.

1.3.1 Laplacian matrix

The *Laplacian matrix* of a graph is defined as $L = D - A$, where D is the degree matrix, and A is the adjacency matrix [Mer98]. Notice that because our graphs are undirected, $D - A$ is a symmetric matrix with all real entries, and therefore a Hermitian matrix. A sufficient condition for a matrix to be positive semi-definite is

$$\forall i, a_{i,i} \geq \sum_{j=0|j\neq i}^{n} |a_{i,j}| \tag{1.10}$$

and to be Hermitian [HJ85]. In our case this is always met with equality, since the diagonal entry of each row in L is the degree of vertex i, and each edge connected to i results in a -1 in the same row. So the sum of all off diagonals in a row is always $a_{i,i}$. Therefore L is a positive semi-definite matrix.

Since L is semi-definite (and therefore also Hermitian), we will adopt the ordering convention $0 \leq \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$.

A simple observation to make is that $\lambda_1 = 0$ for any Laplacian matrix, because $[1 \cdots 1]^T$ is an eigenvector. Since any given row's diagonal entry is k_i, and there is a

-1 for each connection (a total of k_i of them), we have

$$\begin{bmatrix} k_0 & & \{0, & -1\} \\ & k_1 & & \\ & & \ddots & \\ \{0, & -1\} & & k_n \end{bmatrix} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} = 0 \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \tag{1.11}$$

In fact, this follows from the more general theorem that the multiplicity of the eigenvalue 0 is equal to the number of connected components of the graph [Moh97]. Thus, a graph with 2 distinct separate connected components will have $\lambda_1 = 0$, $\lambda_2 = 0$, and $\lambda_3 > 0$.

1.3.2 Algebraic connectivity

The second smallest eigenvalue of the Laplacian matrix, λ_2, is named the *algebraic connectivity*, a term introduced by M. Fiedler in [Fie73]. Adding new edges to a graph always increases (or leaves unchanged in special situations) the algebraic connectivity, and in general higher λ_2 indicates graphs with smaller diameter and higher connectivity. [LS81] Fielder also showed that λ_2 is always less than the minimum number of vertices that would need to be removed to disconnect the graph. As mentioned earlier, in the extreme condition $\lambda_2 = 0$ for a disconnected graph. Several other inequalities have been discovered that reinforce the label "algebraic connectivity". See appendix A for an example calculation of λ_2.

It has been pointed out in [DHM04] that for general graphs with skewed degree distributions (such as those we will be considering), it is more appropriate to consider an alternate normalized Laplacian matrix when calculating the algebraic connectivity. However for *unweighted graphs* this difference is unimportant, and therefore we will use the simpler Laplacian matrix in the interest of computational efficiency.

1.4 Synchronizability

Now that the mathematical foundations are set, we introduce the primary notion we wish to discuss. The concept of synchronization is familiar informally to us already. We see it in a pair of clocks ticking together or hear it when musicians play to the same rhythm. In these examples it is clear to us why they should be synchronized, however synchronization often occurs unexpectantly as well. For example, the well documented occurrence of synchronized firefly flashing. Although these events often begin as asynchronous random noise, they converge into a recognizable pattern.

The concept behind the rigorous definition of formal synchronization is that as time progresses the difference between the state of any two vertices becomes very small. Let us denote the state of vertex i at time t by $x_i(t)$. How does the state of vertices change over time? Clearly if vertices do not take any note of their neighbors there is no chance for synchronization. Further we assume that all vertices are identical and conform to the following generic discrete time equation to determine their

next state

$$x_i(t+1) = f(x_i(t)) + \kappa \left[\frac{1}{k_i} \sum_{j|(i,j)\in E} f(x_j(t)) - f(x_i(t)) \right] \tag{1.12}$$

where $(i,j) \in E$ denotes an edge between i and j [ABJ04]. κ is called the *coupling strength* and is a scalar describing the extent to which neighboring vertices effect the state of a vertex. $f(x)$ is any differentiable function that describes the behavior of a vertex in the absence of outside influence: notice that if $\kappa = 0$, the equation is simply $x_i(t+1) = f(x_i(t))$.

Then we say that a network *synchronizes* if for all i, j

$$\lim_{t\to\infty} |x_i(t) - x_j(t)| = 0 \tag{1.13}$$

1.4.1 Measure of synchronizability

It is well established that for diffusively coupled equations such as 1.12, the algebraic connectivity is the essential measure of network synchronizability [JJ01][ABJ04]. Larger λ_2 means a graph will synchronize for a wider range of values for κ. A detailed example of this relationship has been shown in [JJ01] for the quadratic map $f(x) = 1 - ax^2$. A useful small example is shown in figure 1.8.

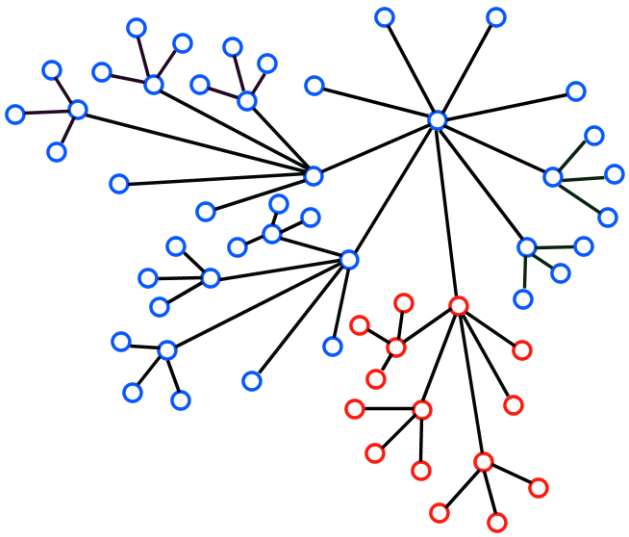


Figure 1.7: An example scale-free graph to display self-similarity. Notice that the red branch looks exactly like the whole graph if we prune degree vertices of degree one. A second pruning results in a graph similar to the next tier of branches, and pruning a third time results in a single leaf node. This internal mirroring between branches and the larger graph structure is *self-similarity*.

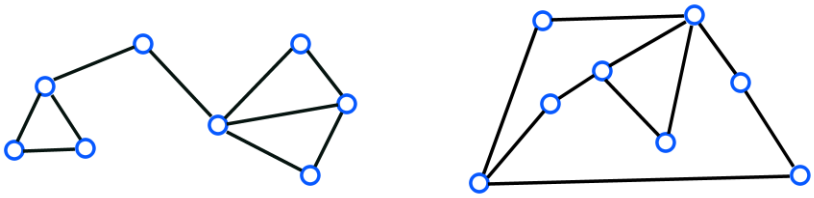


Figure 1.8: These two graphs have the same degree sequence, however notice that the graph on the left seems weakly connected. Intuitively we expect the graph on the right to be more synchronizable. On the left, $\lambda_2 = 0.238$, and on the right $\lambda_2 = 0.925$. (see appendix A for an example calculation of λ_2)

Chapter 2

Relating Network Topology to Synchronizability

The primary focus of this project is to explore the general relationship between network topology, clustering coefficient distributions, and synchronizability, as well as to specifically compare scale-free and scale-rich graph synchronization. Although there are some relationships which follow directly from properties of algebraic connectivity (for example, adding an edge always increases synchronizability), other relationships are not obvious (for example, whether regular or geometric graphs should have better synchronizability).

2.1 Sampling from a family of graphs

Obviously, we cannot pick a single graph from each topology and compare them to make any general claims. Instead we would like to take a sampling of graphs from some family [TGJ+02]. We should only compare graphs that have the same number of vertices since it would be unfair to compare a small graph of 5 nodes (which will synchronize under almost any topology) with a massive scale-rich network of 10,000 vertices. In addition to the popular dynamic graph growth models (described in Section 2.2), we would also like to explore graphs with a specific degree distribution, since this is how we usually categorize different kinds of networks. In general, even graphs with the same degree distributions can have vastly different properties [CHK+01].

It is important that we sample from our family of graphs at random and without any biases toward a certain subset of special graphs. For example, although many consider preferential attachment as the mechanism that creates real world networks with power-law distributions, this mechanism is biased toward creating scale-free networks as opposed to scale-rich ones since vertices added early in the process are far more likely to get most of the connections and become network hubs [TGJ+02].

For this reason, although we will consider some interesting growth models where appropriate, we must also use *static graph generation*.

2.1.1 Static graph generation

Unlike graphs created via some well defined growth process, in the creation of static graphs all vertices are placed and assigned degrees selected randomly from the particular degree distribution of interest. Unfortunately, static random graphs are surprisingly difficult to construct. The naive technique of connecting edges can result in disconnected graphs or leftover edges. Fortunately, the problem of quickly generating connected random graphs given a prescribed degree distribution has been recently addressed in [VL05]. The outline of the process is to first realize a simple graph with the degree distribution desired, connect the graph without changing the degrees, then finally shuffle the edges sufficiently to make it truly random (while keeping the graph simple and connected).

Realizable degree sequences

Not every degree distribution can be realized as a graph. For example, if the degree sequence chosen is {4,2,1}, it is impossible to form a graph since the degree 4 vertex only has two possible adjacent vertices (multiple edges between two nodes are not allowed). Thus, a necessary condition for a degree sequence to be realizable is that for every subset of j highest degree nodes, their total edges can be "absorbed" amongst themselves and the other vertices in the graph. Formally, for $1 \leq j \leq n-1$:

$$\sum_{i=1}^{j} k_i \leq j(j-1) + \sum_{i=j+1}^{n} \min(j, k_i) \tag{2.1}$$

Similarly, there is a necessary condition for a degree sequence to be realizable as a *connected* graph,

$$\sum_{i=1}^{n} k_i \geq 2(n-1) \tag{2.2}$$

In fact, the Erdös-Gallai theorem shows that these are both necessary and sufficient, so these are the only equations we must check to ensure a degree distribution we generate is valid [Ber73][GMZ03].

Havel-Hakimi algorithm

Given a realizable degree sequence, we can create a simple realization of a graph by using the Havel-Hakimi algorithm [Hak62][Hak63].

Step 1. Imagine for each degree in the degree sequence creating a vertex with k "stubs", or edges connected to the vertex with their end point not yet assigned. Now, pick the node with smallest number of stubs, A, and connect each of its stubs to the nodes with highest number of stubs (using up one stub from node A and one from the other node). Repeat until all stubs are used up. This ensures that the necessary condition in equation 2.1 is met at each iteration. This graph is not randomly chosen and need not even be connected, but it gives us a suitable graph with the required degree distribution.

Step 2. If the graph from step 1 is connected, we can proceed directly to the final step, otherwise we must connect the graph. Although difficult in practice, conceptually it is simple to take an unconnected graph and make it connected. Equation 2.2 guarantees that one of the disconnected components contains a cycle. We can merge two separated components by taking an edge in the cycle and an edge in the disconnected component, and then swapping their end points. This joins the two components and does not alter the degree of any vertex. After repeating the process as many times as needed, we arrive at a simple connected graph. In practice, it is usual easier to simply shuffle the graph randomly (see step 3) until the graph happens to connect, but this is not necessarily reliable.

Step 3. The final step is to shuffle the edges of the graph in order to arrive at a truly random instance from the space of all possible graphs with this degree distribution. Consider the following procedure. Repeatedly pick two edges at random, say (a, b) and (c, d) with distinct endpoints. If (a, c) and (b, d) don't already exist, then we remove the original edges and add these new edges. Notice that degrees remain unchanged under this process. A theorem of Taylor has shown that by this process, any connected graph can be transformed to any other connected graph with the same degree sequence [Ber73]. It follows then from Markov chain theory that this process converges to a uniform distribution, i.e. if we repeated this procedure an infinite number of times we would have a perfect uniformly selected random graph. The question of how long the process needs to run to be nearly a uniform distribution is still a very difficult problem. Algorithms such as CFTP or Fill's algorithm are theoretically capable of providing an estimate, but require remodeling the Havel-Hakimi algorithm explicitly as a Markov chain simulation [CLR01][Dim00]. Instead in this project we choose to include the entire path of rewirings, with the understanding that our datasets (generally over 500,000 datapoints) are sufficient to provide a fair depiction of the family.

2.2 Models of network growth

The concept of the Havel-Hakimi algorithm is to take properties we observe in a network, specifically the degree distribution, and create graphs in order to meet these observations; however, if our graphs are representative of real complex networks, it does not explain why the network displays these properties, or by what process the network was constructed. Several models have been proposed that attempt to answer these questions.

2.2.1 Small-world model

Watts and Strogatz propose a model of network growth which begins from a ring shaped lattice (as in figure 1.4), each node connected to its k closest neighbors. Next, a percentage p of the links are randomly rewired by disconnecting an end and

reconnecting it to a random node, thus allowing the model to be controlled between a regular lattice ($p = 0$) and a highly disordered graph ($p = 1$). Watts and Strogatz's interesting observation was that even for very small values of p, this model exhibits the small-world effect described in section 1.1.1 [WS98].

2.2.2 Preferential attachment

The *preferential attachment model* was originally discovered in 1965 by Derek de Solla Price and called cumulative advantage, then recently rediscovered by Barabási and Albert in 1998 where it received its current name. In the preferential attachment model, nodes are added to the graph one at a time and are more likely to connect to nodes that already have a high degree than to low degree nodes. This results naturally in a power-law degree distribution, and it has been proposed that this is the primary explanation for why power-law networks are so abundant in the real world. There are several critiques of this model, most importantly the *winner takes all* effect, in which a single node tends to grow so rapidly in degree that it is essentially connected to every node, resulting in star-shaped networks.

2.2.3 Geometric graph

Geometric graphs are a combination of random graphs and lattices. Vertices are each assigned random locations (usually inside a fixed shape on the xy plane), and rather than connecting to each other randomly, vertices are connected if they fall within a fixed radius of each other. Geometric graphs generally form Poisson distribution for large n if you ignore boundary effects caused by constraining the random points to a finite space [Pen03][IA05]. Some interesting work has been done to extend the concept of geometric works into arbitrary degree distributions [RCbAH02], but we will not discuss them here. An example geometric graph inside the unit circle is shown in figure 2.1.

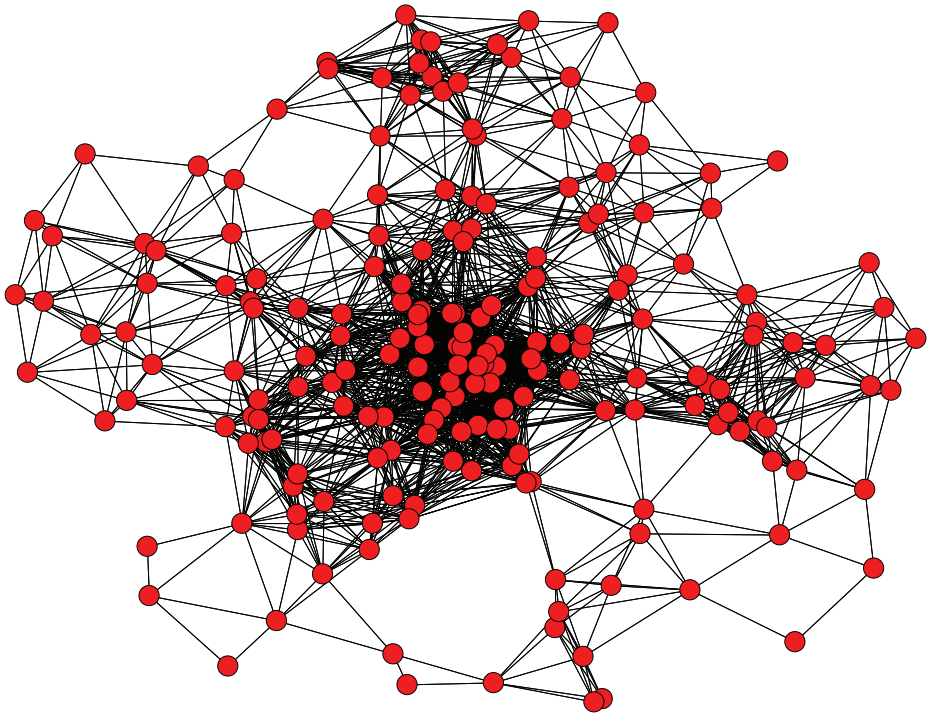


Figure 2.1: A geometric graph inside the unit circle with 200 nodes. The connection distance r is 0.29.

Chapter 3

Results

3.1 Erdös-Rényi graph data

We first present our results about the synchronization of random Erdös-Rényi graphs as a baseline from which we will measure other graphs. Our simulation created an empty graph of n nodes, and assigned each possible connection an edge with probability p as discussed in section 1.1.2. Any disconnected graphs created were discarded. Separate simulations were run for $n = \{100, 200, 300\}$ and for $p = \{\frac{2}{n}, \frac{2.5}{n}, \cdots, \frac{5}{n}\}$. The results are shown in figures 3.1 and 3.2.

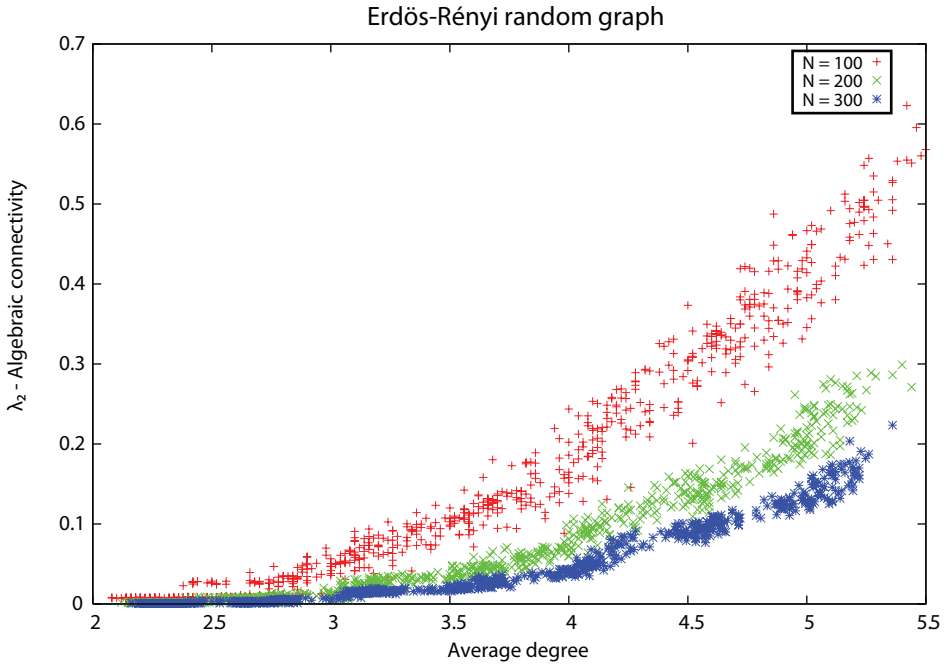


Figure 3.1: Plot depicting the relationship between average degree and λ_2 for randomly generated Erdös-Rényi graphs. n is the number of nodes in the network. Each of the 3 sets contains 1400 data points.

As the average degree of nodes in the graph increases, both λ_2 and the average clustering coefficient increases (see figure 3.2). This is expected, as adding more edges to a graph is guaranteed to increase λ_2, and also provide more opportunities for triangles to form in the network. Perhaps not as obvious is that while keeping the average degree constant, increasing the number of nodes in the network is damaging to both properties. In fact, since the network is connected completely at random, increasing the number of nodes makes it more unlikely for neighbors of a node to connect by chance, and therefore decreases the clustering coefficient. By reducing the number of tightly coupled clusters in the network and creating a larger loosely connected network, the ability of the network to synchronize is drastically reduced.

3.2 Geometric graph data

Our simulation created geometric graphs by randomly assigning (r, θ) polar coordinates to each vertex inside the unit circle (for an interesting observation of edge effects due to using the unit square instead, see appendix B). Edges were then added between any nodes within a radius r of each other. Separate simulations were run for $n = \{100, 200, 300\}$ and for $r = \{0.25, 0.27, 0.29, 0.31, 0.33\}$. The results are shown in figures 3.3 and 3.4. An example graph for $n = 200$ and $r = 0.29$ is shown in figure 2.1.

As expected, even for this different topology the fundamental relationship that higher average degree predicts a large average clustering coefficient and large λ_2 is still true. As we increase n, we are forcing more nodes into the same confined space, so it is not surprising that the despite using the same values for r in each experiment, the simulation with 300 nodes has significantly higher average degree than the simulations with less nodes. Despite this, it is clear that for any given average degree large numbers of nodes still reduce the λ_2 and the clustering coefficient.

For nodes distributed evenly in a geometric graph, we would expect a graph of 100 nodes to have the same average degree as a graph of 200 nodes when the connection area is double (or equivalently, that the radius is $\sqrt{2}$ longer). As a result, for a given average degree as we increase n, the distance between connected nodes r decreases, and it requires crossing more nodes to traverse from one side of the unit circle to the other. This impacts the ability of a network to synchronize negatively, since signals are more difficult to pass across the network.

In fact, compared against the Erdös-Rényi graphs, we see that the geometric graph synchronizability is remarkably worse for a given average degree. Signals in the network have to propagate across physical space from node to node since there are no random links that cross the entire network. This presents a fundamental difficulty for complex systems confined to a physical setting to synchronize, and shows us that a network created by linking closest neighbors will be highly resistant to synchronization.

In contrast, geometric graphs have remarkably high average clustering coefficient. Clearly, if two of your neighbors are inside the connection radius r, then it is very likely that they will also be within that radius of each other. For the special case

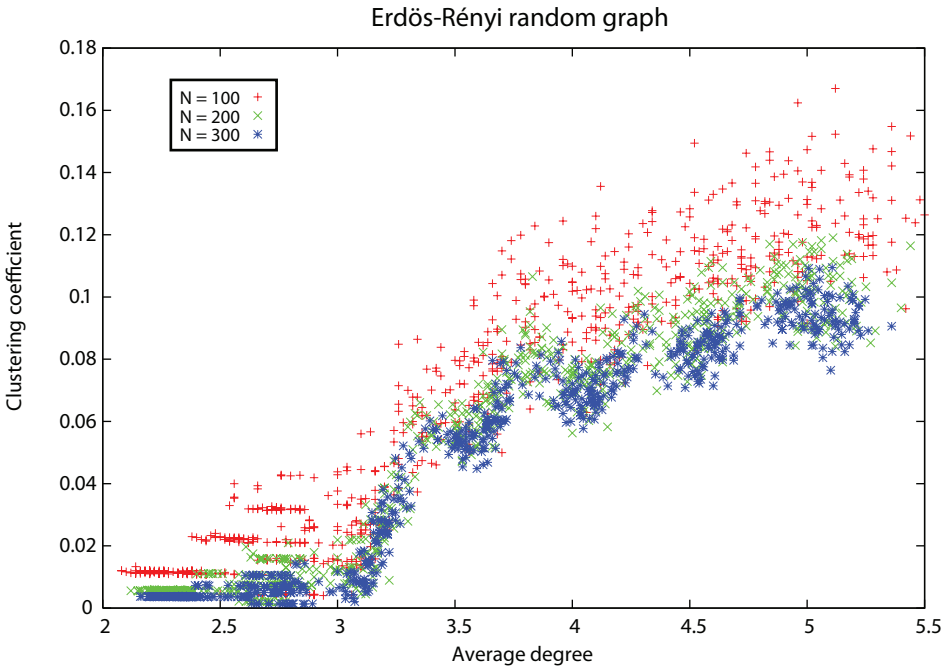


Figure 3.2: Plot depicting the relationship between average degree and clustering coefficient for randomly generated Erdös-Rényi graphs. n is the number of nodes in the network. Each of the 3 sets contains 1400 data points. (Clustering of data points is due to an uneven sampling of p).

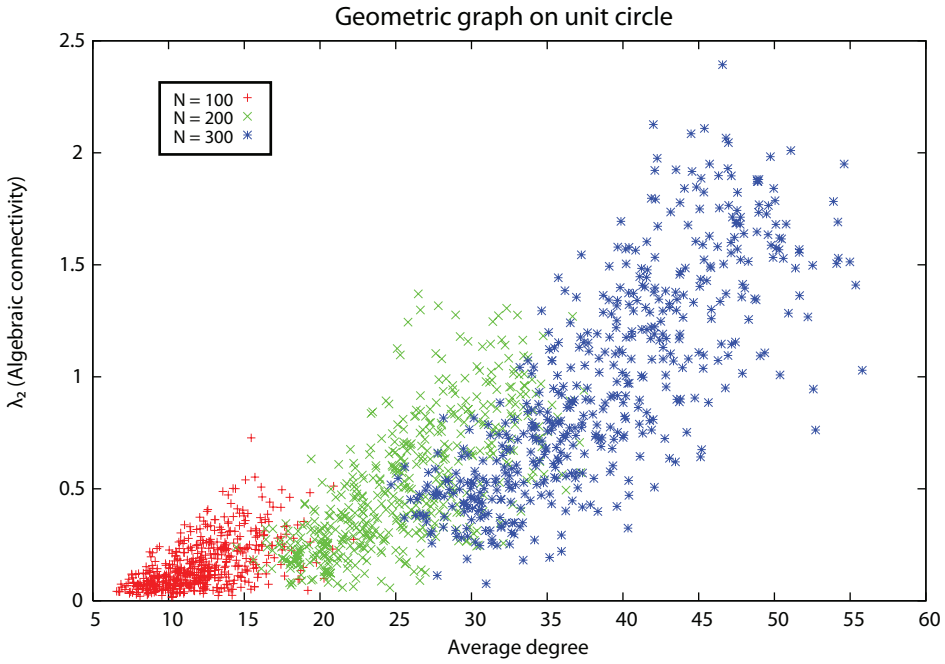


Figure 3.3: Plot depicting the relationship between average degree and clustering coefficient for randomly generated geometric graphs on the unit circle. n is the number of nodes in the network. Each of the 3 sets contains 700 data points.

of geometric graphs, if we ignore edge conditions we can use simple calculus to show analytically the expected clustering coefficient value. First we simplify the problem by assuming the connection radius $r = 1$, since the answer is independent of r. Consider the node of interest to be placed at $(0, 0)$, and it's two neighbors (r_0, θ_0) and (r_1, θ_1). Due to symmetry, we can rotate this picture to place (r_0, θ_0) at $(r_0, 0)$ without changing the problem. Now we can imagine two unit circles, one centered at $(0, 0)$ and one centered at $(r, 0)$. The second node must be inside the first circle (since it is a neighbor of our node of interest), but if the second node is placed inside the intersection of the two circles, then all three will be within a distance of 1 and form a triangle. So, we simply calculate the area of the intersection (the inside integral of equation 3.1) and divide by the total area of the circle, π. Integrating this over r_0 from 0 to 1 gives us the final answer.

$$\int_{r_0=0}^{1} \left(\frac{4 \cdot \int_{x=\frac{r_0}{2}}^{1} (\sqrt{1-x^2})}{\pi}\right) = \frac{16 - 9\sqrt{3} + 4\pi}{6\pi} = 0.68846 \tag{3.1}$$

We can see the data approaching this value from beneath, due to edge effects (nodes on the edge of the circle do not have neighbors outside the circle to connect with, and thus have worse clustering coefficients and lower degree). Since nodes are actually placed at random positions, there is large variance in the clustering coefficient for any one graph, but this upper bound becomes increasingly accurate as n increases.

3.3 Preferential attachment graph data

In Barabasi's preferential attachment model, nodes are added one at a time with a fixed number of incoming connections k. Then, for each incoming stub, a random end of an edge is chosen from the existing graph, and the node at this end is linked to the new node. In other words, a node with degree 6 is twice as likely to receive one of the new connections as a node of degree 3. Despite the popularity of Barabasi's original preferential attachment model it has several disadvantages, one of which is that it can only generate graphs with average degree of exactly $2k$. We instead choose to use a slight variation on this basic model, in which the number of incoming connections for a new node is uniformly random from 1 to k. This results in a wider variety of graphs with different average degrees centered around $k + 1$. Separate simulations were run for $k = \{2, 3, 4, 5, 6\}$ and for $n = \{100, 200, 300\}$. The results are shown in figures 3.5 and 3.6.

We can see an unexpected phenomena occurring in figure 3.5, resulting in some clustering of datapoints along the y axis, especially just beneath $\lambda_2 = 0.4$. By observing specific graphs on opposite sides of this divide, it became clear that this strange behavior was due to another shortcoming of Barabasi's preferential attachment model called the "winner takes all" scenario. In this scenario an early node in the network growth by chance gains an unusually large number of connections very quickly. Since new nodes prefer to connect with high degree nodes, a single node of very high degree can cause nearly all new nodes to connect with it (making the situation even worse for

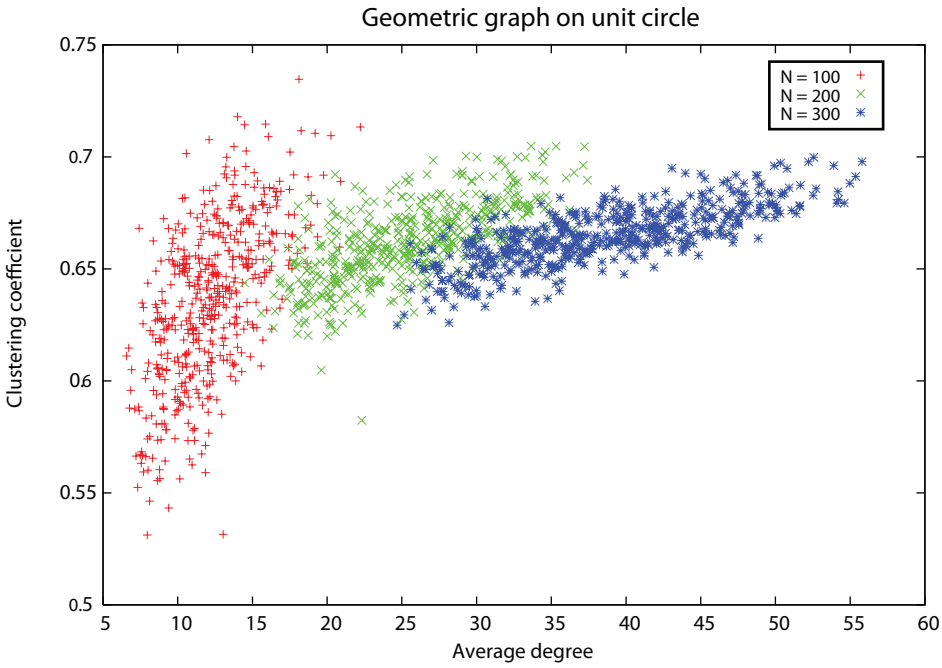


Figure 3.4: Plot depicting the relationship between average degree and clustering coefficient for randomly generated geometric graphs on the unit circle. n is the number of nodes in the network. Each of the 3 sets contains 700 data points.

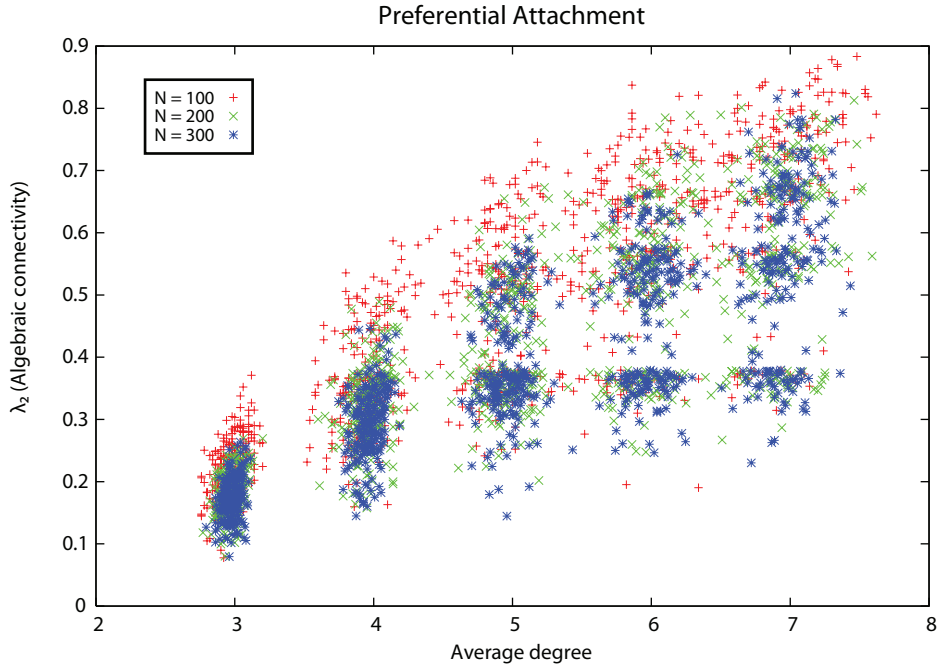


Figure 3.5: Plot depicting the relationship between average degree and λ_2 for randomly generated networks using the preferential attachment model. n is the number of nodes in the network. Each of the 3 sets contains 1000 data points. Clustering along the x axis is due to sampling from integer values of k. Clustering along the y axis is discussed in section 3.3.

the next new node). This occurrence is one of the major criticisms of Barabasi's preferential attachment model, and as we can see creates an uneven sampling of graphs. See figure 3.7 for an example comparison of graphs on each side of the connectivity gap.

This provides interesting information about the synchronizability of power-law networks. In particular, a network which relies on a single highly connected hub is less synchronizable than a network centered around a core of several nodes. This is not immediately apparent, since one might expect that one central node could better influence an entire network than several uncoordinated nodes. Apparently a single node receiving many conflicting message from individual nodes is more difficult to synchronize than a group of nodes each controlling a smaller portion of the network.

It is also of interest that although we still see the behavior that increasing n decreases λ_2, it is less pronounced than in the other examples. Regardless of the number of nodes in the network, the preferential attachment model always creates high degree hub nodes to which the rest of the network is connected. This structure remains basically unchanged regardless of n, the only difference in λ_2 as n increases results from the increased number of stray low degree nodes extending further away from the hubs.

3.4 Graphs with fixed degree distributions

We now turn our interest to statically generated graphs. In particular, we would like to see how $s(G)$, the scale-freeness of a graph, and λ_2, the algebraic connectivity or synchronizability, are related given a specific power-law degree distribution. We used the Havel-Hakimi algorithm discussed in section 2.1.1 to generate an even unbiased sampling from the space of possible graphs with the same degree distribution.

As can be seen in figure 3.8, it is difficult to make any general claims about λ_2 based on $s(g)$. Although the highest λ_2 values occur for relatively scale-free graphs, plenty of scale-free and scale-rich graphs alike have very poor synchronizability. Similarly, the most scale-free and scale-rich graphs do not have especially noteworthy λ_2. The relation between λ_2 and the average clustering coefficient is shown in figure 3.9, and is even less well defined.

Given a specific degree distribution, synchronizability is a property completely independent of scale-freeness and clustering coefficient, and nothing can be assumed based on these measures alone. It is especially important in applications to realize that the number of connections to each node is not important to network synchronization, rather the configuration of the network is of greatest importance.

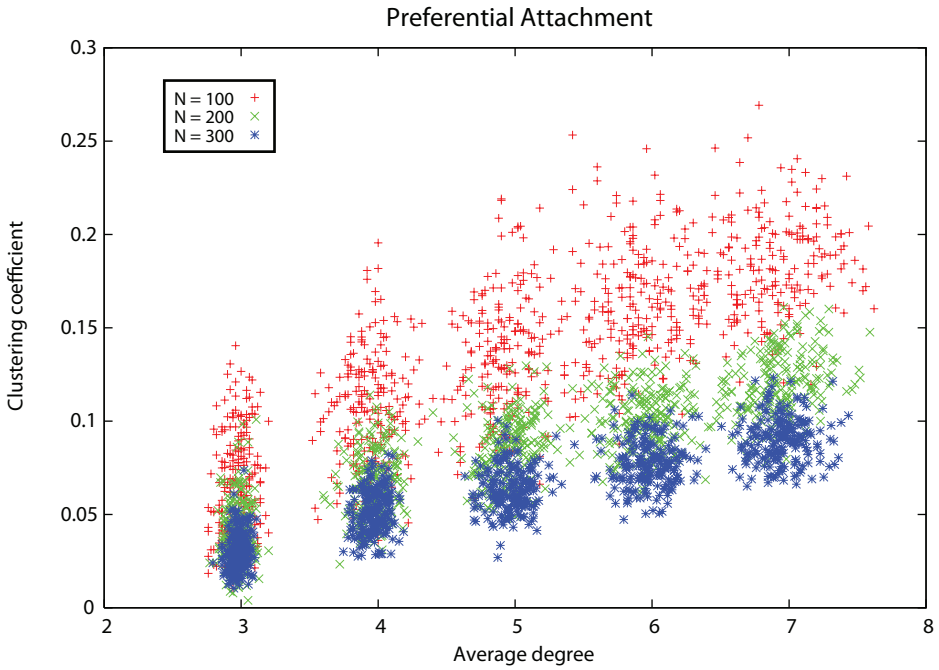


Figure 3.6: Plot depicting the relationship between average degree and clustering coefficient for randomly generated networks using the preferential attachment model. n is the number of nodes in the network. Each of the 3 sets contains 1000 data points. Clustering along the x axis is due to sampling from integer values of k.

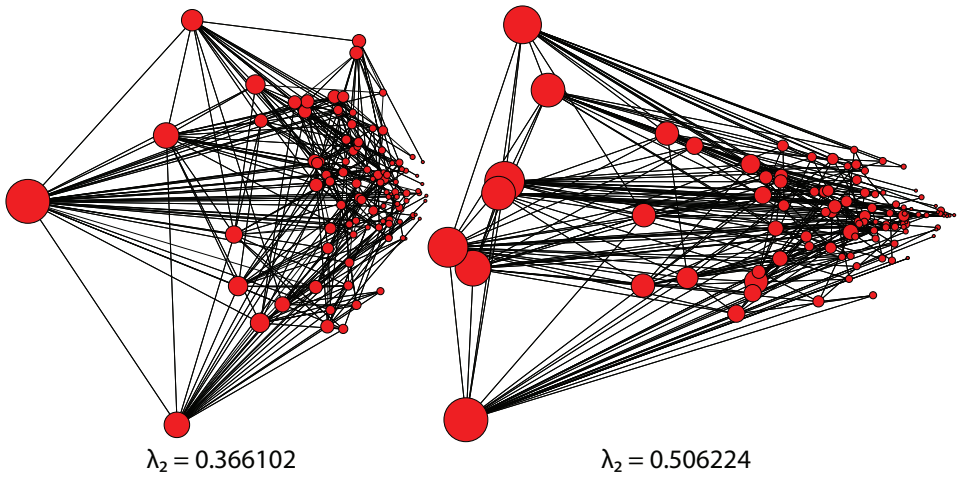


Figure 3.7: A comparison between the "winner takes all" scenario, and another graph generated through the preferential attachment method. Both graphs were generated using the same method, with $k = 5$ and 100 nodes in the network. The size of a node and its position along the x-axis are proportional to its degree. Also notice that "winner take all" networks have significantly worse algebraic connectivity than their counterparts.

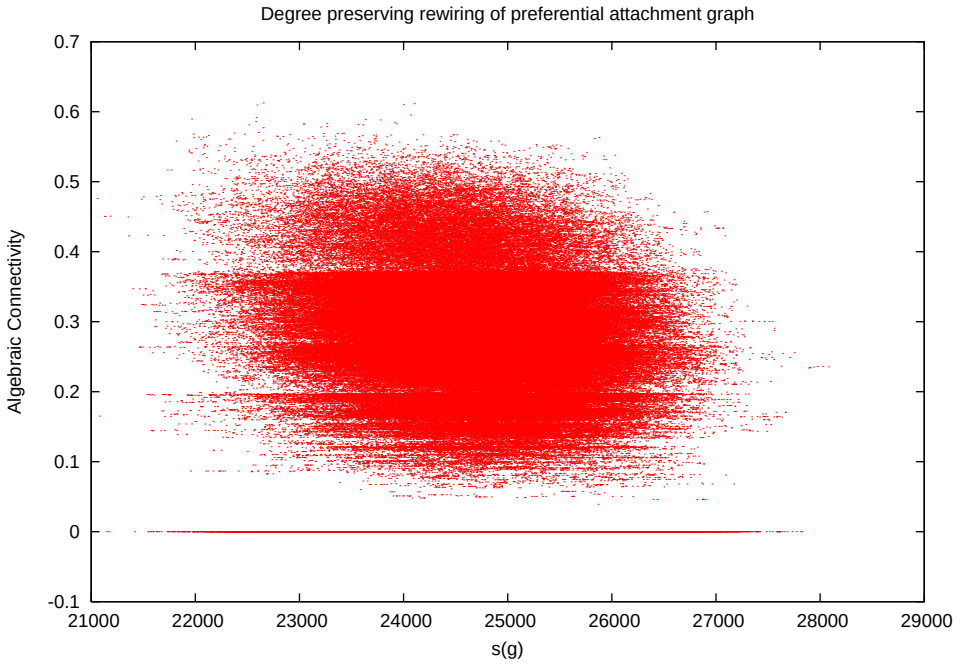


Figure 3.8: Starting from a pregenerated graph, each point represents a new graph with the same degree distribution created by rewiring as described in section 2.1.1. Graphs where $\lambda_2 = 0$ are disconnected graphs.

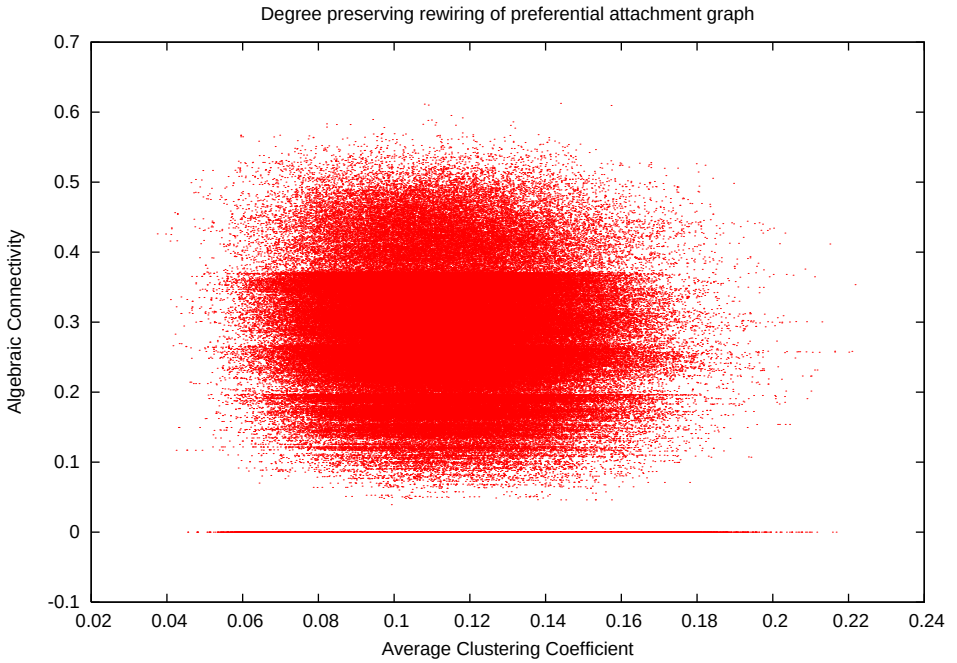


Figure 3.9: Starting from a pregenerated graph, each point represents a new graph with the same degree distribution created by rewiring as described in section 2.1.1. Graphs where $\lambda_2 = 0$ are disconnected graphs.

Chapter 4

Conclusions

The results of the previous chapter can be applied to interesting problems in various fields that deal with discrete complex systems. In many communications and transportation networks, such as the Internet, cellphone networks, airline travel, synchronization can result in congestion at the nodes, resulting in lowered efficiency, or loss of data. In contrast, synchrony is highly desired in some self-organizing such as P2P networks, the global positioning system, and neuron respiratory networks. An understanding of the synchronizability characteristics of these networks is important to identifying potential problems, as well as providing insight into the best techniques for modifying or extending the networks as they grow.

4.1 Communication and information networks

The Internet has been a hot topic for research recently. Computers around the world send packets into the network, and it is up to the relatively disorganized routers across the globe to propagate these signals to their proper location. An important topic in computer science specific to networking is how to avoid the inevitable congestion on the Internet. 95% of packets sent over the Internet are sent using transmission control protocol Reno (TCP Reno), which provides congestion control by halving its output rate whenever it sees packet loss occuring, but is always slowly increasing its output rate. While other TCP implementations (most notably TCP Vegas) have been proposed, they have not been accepted at large, and as a result congestion resulting in packet loss is constantly occuring at all routers around the world, resulting in retransmissions of packets and wasted bandwidth. When overwhelmed with traffic, naively implemented routers will drop all packets exceeding their queue length, and all connections have a probability of seeing lost packets. As a result, synchrony occurs as all connections half their speed and then slowly increase, only to overwhelm the router again shortly. Although this is inevitable on routers connected directly to computers, it can be avoided by connecting routers in such a way as to create a very low λ_2.

Although there is large debate about what the exact structure of the Internet is, models similar to that proposed in [LAT+05] have a central core of powerful routers

and appear to generate networks with low λ_2, which may explain why the Internet has been successful despite the inevitable congestion, since it is unable to propagate further into the network because of its structure.

4.2 Airline networks

Recently, some airlines have begun to push away from the older centralized hub networks toward newer decentralized systems, and have seen improvements in congestion related delays [MSed]. It has been proposed that this is due to the basic differences in network structure.

Our analysis of the random Erdös-Rémyi random graphs compared with the preferential attachment model suggests this is true (see figures 3.1 and 3.5). Randomly chosen power-law degree distributions such as those of a hub airline's network on average have much better synchronization properties than randomly connected networks. Airlines which have one single major hub may be at an advantage however, as we saw that the "winner takes all" scenario resulted in a smaller λ_2, similar to the random networks. This generalization must be made carefully. We saw in section 3.4 that merely moving from a hub network to a decentralized one is not sufficient to decrease synchronizability. Networks must be constructed with synchronizability specifically in mind, since the degree distribution is not sufficient to determine synchronizability. Despite the general properties of these two different topologies, individual cases have a wide variance and we cannot make a general guarantee without a specific network in mind.

4.3 Neuron rhythmogensis

In mammals, a small group of neurons in the brainstem called the pre-Bötzinger complex is responsible for generating a regular rhythmic output to motor cells that initiate a breath. Disconnected, these neurons are unable to provide enough output to activate the motor neurons, but their interconnected network structure allows them to synchronize without any external influence and produce regular bursts. An example of a typical neuron's output is in figure 4.1.

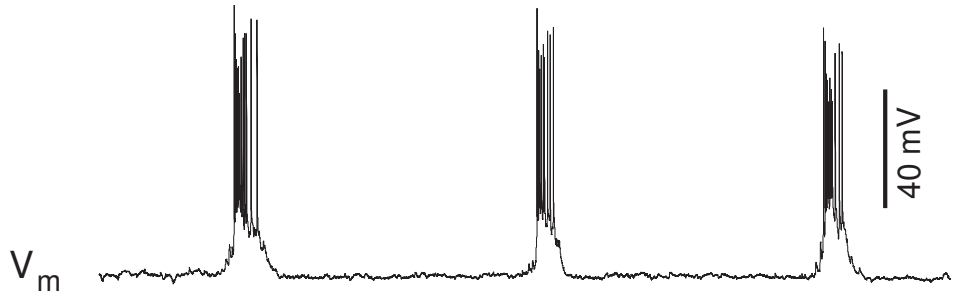


Figure 4.1: Example output from an individual neuron in the PreBötzinger complex. This function corresponds to $f(x)$ in equation 1.12.

Using a detailed simulation written by John Hayes at the College of William and Mary, we were able to experiment with how different network topologies control the effectiveness of the pre-Bötzinger complex. Intitally we ran the simulation with the smallest and largest λ_2 networks from figure 3.8. Unfortunately, both graphs were sufficiently good synchronizers, and it was difficult to visually distinguish the two from each other. We instead ran a similar experiment as in section 3.4, but starting from a geometric graph. The results of the two simulations can be seen in figure 4.2, and are very compelling evidence. Turning back to the original power-law graphs, we used autocorrelation analysis as in [KPV04] to statistically detect better synchronization in the higher λ_2 network. The results are shown in figure 4.3, and confirm that although the difference is undetectable at a glance, the higher λ_2 value statistically has better synchronization. The difference in autocorrelation is largest during the refractory (non-spiking) period, indicating that the two graphs have similar behavior during the spikes, but not between spikes.

4.4 Open problems

4.4.1 More practical measure

Although λ_2 is an powerful tool for estimating networking synchronizability, in practice computing the eigenvalues of large matricies is an expensive computation. Mathematical techniques for computing only λ_2 without the other eigenvalues exists, but ideally we could find a measure strongly correlated with λ_2 that is easier to calculate. We had initially hoped $s(g)$ would be such a measure, but as shown in section 3.4 this turned out not to be the case.

4.4.2 Algorithm for largest and smallest λ_2 graphs

Given a specific degree distribution, it is not clear how to construct the graphs with maximum and minimum λ_2. It has been shown in [ABJ04] that as n increases, graphs can be constructed with arbitrarily small λ_2; however no algorithm has been found for fixed n and this algorithm does not gaurentee the smallest possible graph. The discovery of this algorithm would provide a much deeper insight into what structural properties effect λ_2.

4.4.3 Synchronizability under targeted attacks

Graphs with power-law degree distribution are well known to be resilient to random node failure, but highly vulnerable to targeted attacks. This comes as a result of a small number of high degree nodes controlling network synchronization. We suspect that graphs with high λ_2 may be more resilient against these targeted attacks than those with low λ_2, due to increased redundancy in the network structure, but this remains to be verified experimentally.

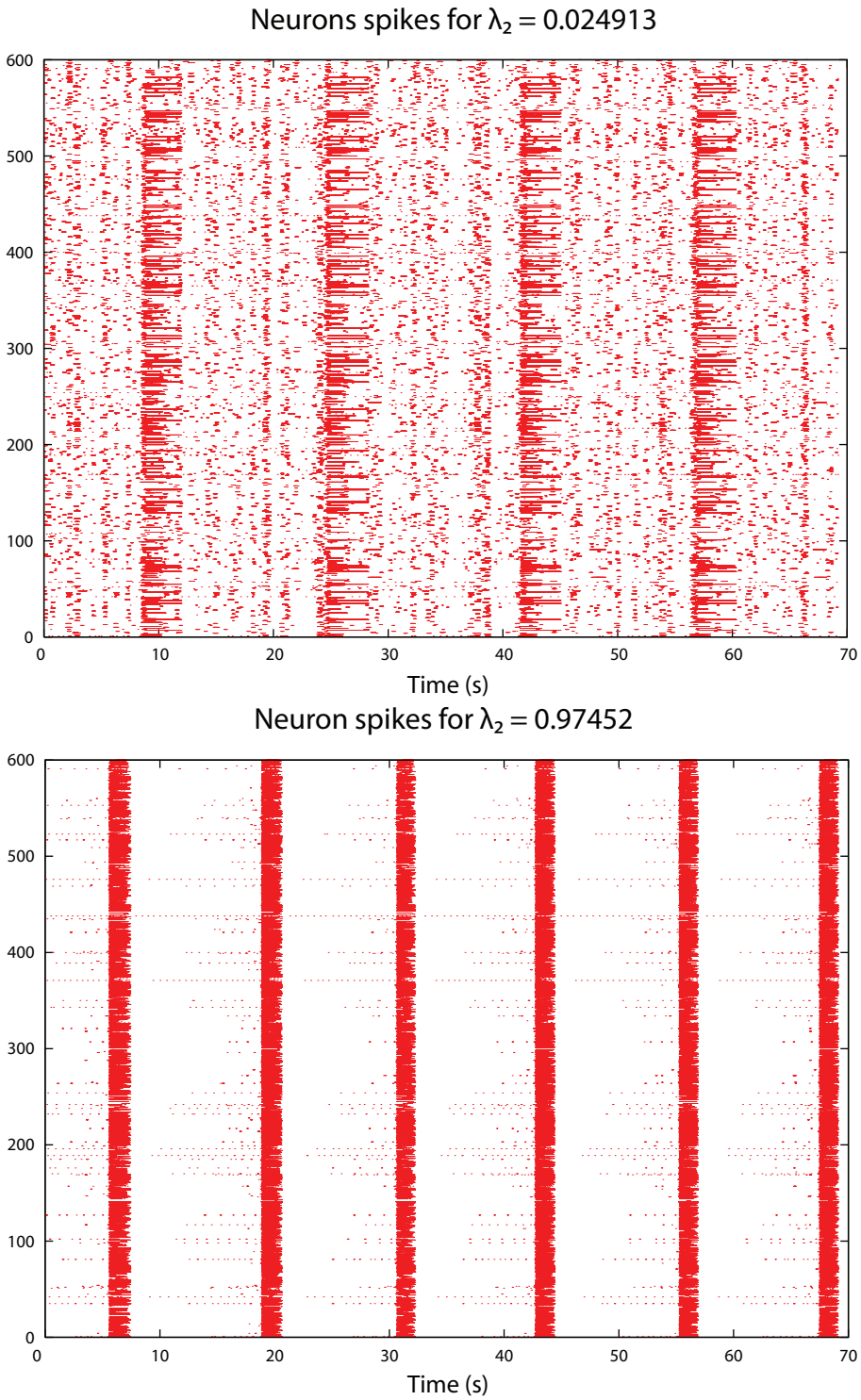


Figure 4.2: Raster plot of neuron output for two sample networks with extreme λ_2 values. A point at (x, y) indicates neuron x is spiking at time y. The higher λ_2 network displays much stronger synchronization amongst all nodes as predicted, as well as a quicker breath frequency.

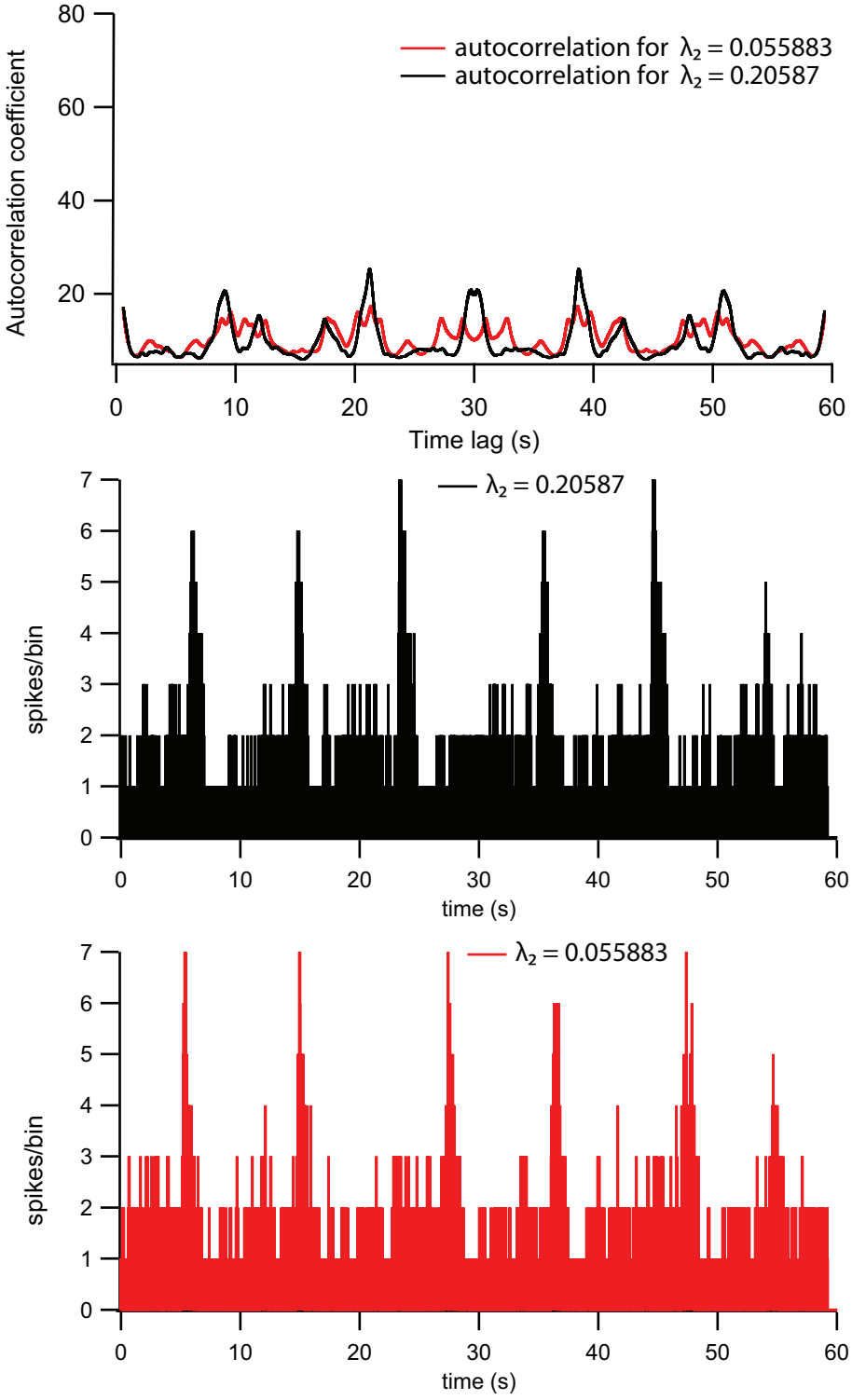
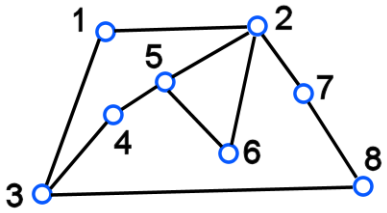


Figure 4.3: An autocorrelation plot of pre-Bötzinger complex synchronization on two graphs with the same degree-distribution, but with extreme λ_2 values. Although the two spike summaries seem indistinguishable (bottom two figures), an autocorrelation analysis (top) shows that the higher λ_2 graph displays statistically better synchronization.

Appendix A

Example λ_2 Calculation

Consider the graph:



The numbered labels on the graph correspond to the associated row and column in our matrices. The Laplacian matrix is equal to the diagonal degree matrix, minus the adjacency matrix (see section 1.3.1 for details).

$$L = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} =$$

$$= \begin{bmatrix} 2 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & 0 & 0 & -1 & -1 & -1 & 0 \\ -1 & 0 & 3 & -1 & 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & -1 & 3 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 & 2 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 2 & -1 \\ 0 & 0 & -1 & 0 & 0 & 0 & -1 & 2 \end{bmatrix}$$

Then we compute the eigenvalues of L. In our experiments these were computed using the Householder algorithm described in [WR71]; however, this can also be

accomplished by computing the characteristic polynomial of L, $p_L(\lambda)$, explained in section 1.2.2. $p_L(\lambda) = \det(L - \lambda I) = \lambda^8 + 20\lambda^7 + 163\lambda^6 + 698\lambda^5 + 1688\lambda^4 + 2298\lambda^3 + 1629\lambda^2 + 464\lambda$. Solving for the roots $p_L(\lambda) = 0$ we get

$\lambda(L) = [0, 0.925, 1.30, 1.68, 2.58, 3.76, 4.50, 5.25]$. Specifically, $\lambda_2 = 0.925$.

Appendix B

Edge-effects

Edge effects occur in geometric graphs due to constraining the (x, y) coordinates of each point to a specific shape in the plane. Vertices which are placed in the corners of the square are unlikely to have adjacent vertices to connect with, and therefore have lower than expected degree and clustering coefficient. These outlier poorly connected vertices also decrease λ_2. See figures B.1 and B.2.

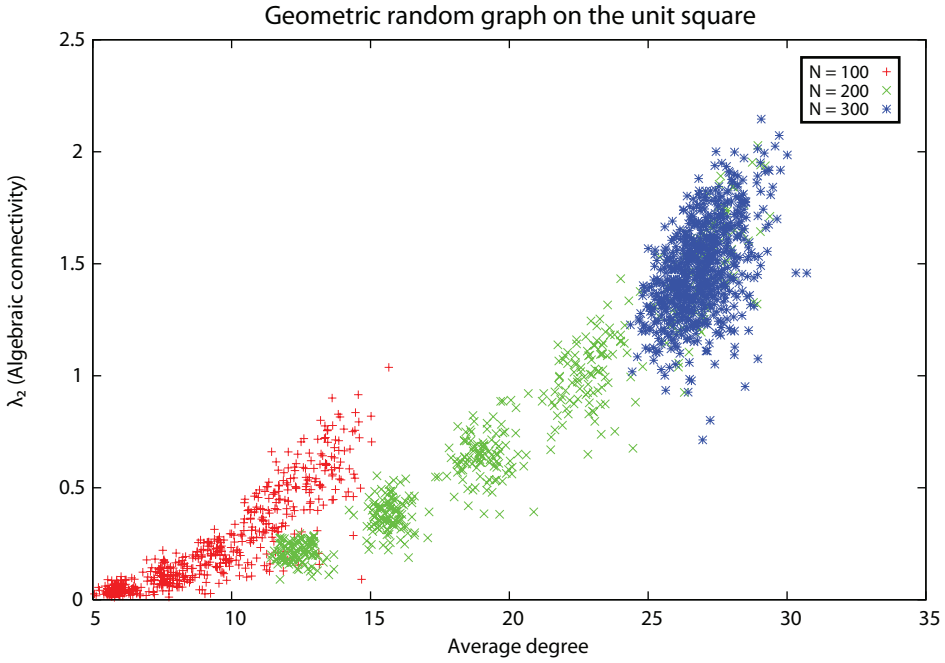


Figure B.1: Plot depicting the relationship between average degree and λ_2 (algebraic connectivity) for randomly generated geometric graphs on the unit square. N is the number of nodes in the network. Each of the 3 sets contains 700 data points.

By constraining vertices to the unit circle instead of the unit square, edge effects are spread evenly around the entire edge of the circle rather than localized in corners. Another option is to make the constraint shape much larger than the area where we will collect data (the sample space). If the constraint shape is larger than the sample

space by at least r (the connection radius) then edge effects are essentially removed. Disadvantages of this approach are that the number of nodes in the sample space is no longer constant, and simple facts about graphs like the $\sum_{i=0}^{n} k_i$ always being even are no longer true, since links exist to nodes outside the sample space. A third alternative is to instead constrain the vertex placement to a smooth 3-dimensional object like a sphere. This removes any edge effects, and does not have the problems associated with the previous option. Unfortunately, it is rarely appropriate to consider complex discrete systems on a sphere. For example, transportation networks such as power grids are usually represented by geometric networks on a 2-dimensional map. In contrast, a sphere might be an appropriate constraint space for global shipping or international airline networks.

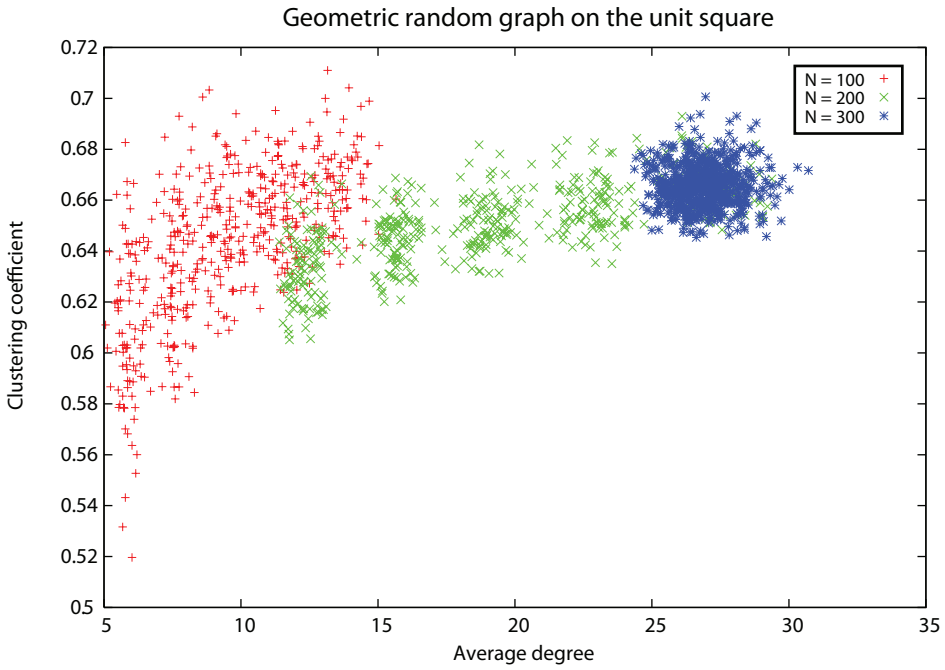


Figure B.2: Plot depicting the relationship between average degree and clustering coefficient for randomly generated geometric graphs on the unit square. N is the number of nodes in the network. Each of the 3 sets contains 700 data points.

Appendix C

Code

C.1 clustercoeff.cc

```
#include "newmat10/newmatio.h"
#include "newmat10/newmatap.h"

float ClusteringCoefficient(SymmetricMatrix A, int node)
{
    int N = A.Nrows();

    int total = 0;
    int sum = 0;

    for(int i = 0; i < N; i++)
    {
        if(!A.element(i,node) || (i == node))
            continue;

        for(int j = i+1; j < N; j++)
        {
            if(j == node)
                continue;

            if(A.element(j,node))
            {
                ++total;

                if(A.element(i,j))
                   ++sum;
            }
        }
    }
    if(total == 0) return 0;
    else return (float)sum/(float)total;
```

```
}

float ClusteringCoefficient(SymmetricMatrix A)
{
    int N = A.Nrows();

    float sum = 0.0;

    for(int i = 0; i < N; i ++)
        sum += ClusteringCoefficient(A,i);

    return sum/(float)N;
}
```

C.2 geometric.cc

```
#include <iostream>
#include "include/rngs.h"
#include "include/rvgs.h"
#include "newmat10/newmatio.h"
#include "newmat10/newmatap.h"

using namespace std;

#define N 200
#define Dist 0.29

extern float ClusteringCoefficient(SymmetricMatrix A);

int main()
{
    //Initialize PRG
    PlantSeeds(-1);
    cout << setprecision(0);

    //Initialize adjacency matrix
    SymmetricMatrix Adj(N);
    SymmetricMatrix L(N);
    DiagonalMatrix D(N);
    DiagonalMatrix Eigens(N);
    double r[N];
    double theta[N];

    for(int repeat = 0; repeat < 1; repeat++)
    {
        Adj = 0;
        D = 0;

        // Geometric Graph
        for(int i = 0; i < N; i++)
        {
            r[i] = Random();
            theta[i] = Uniform(0,360);
        }

        for(int i = 0; i < N; i++)
        {
            for(int j = i+1; j < N; j++)
            {
                double distance = sqrt(r[i]*r[i] + r[j]*r[j]
                    - 2*r[i]*r[j]*cos( theta[i]-theta[j]) );
```

```
        if( distance <= Dist )
        {
            Adj.element(i,j) = 1;
            D.element(i,i)++;
            D.element(j,j)++;
        }
    }
}

//Build the Laplacian
L = Adj*-1 + D;

EigenValues(L,Eigens);

if(Eigens(2,2) < 0.000001)
{
    repeat--;
    continue;
}

cout << "*Node data\n";
cout << "ID\n";
for(int node = 0; node < N; node++)
{
    cout << node << endl;
}
cout << "*Node properties\n";
cout << "ID x y\n";
for(int node = 0; node < N; node++)
{
    double x = r[node]*cos(theta[node]);
    double y = r[node]*sin(theta[node]);
    cout << setprecision(6) << node << " "
         << 1000*x << " " << 1000*y << endl;
}
cout << "*Tie data\n";
cout << "from to\n";

for(int row = 0; row < N; row++)
{
    for(int col = 0; col < N; col++)
    {
        if(Adj.element(row,col))
        {
            cout << setprecision(0)
            cout << row << " " << col << endl;
        }
```

```
                }
            }

        }
    }
```

C.3 geoshuffle.cc

```
#include <iostream>
#include "include/rngs.h"
#include "include/rvgs.h"
#include "newmat10/newmatio.h"
#include "newmat10/newmatap.h"

#include <signal.h>

using namespace std;

#define N 600
#define Dist 0.12

extern float ClusteringCoefficient(SymmetricMatrix A);

int s_A;
double lambda_A;
int s_B;
double lambda_B;
SymmetricMatrix A;
SymmetricMatrix B;

void printProgressA(int type)
{
    cout << "==========" << endl;
    cout << setprecision(5) << "s_A = " << s_A << endl;
    cout << "lambda_A = " << lambda_A << endl;
    cout << setprecision(0) << A;
}

void printProgressB(int type)
 {
    cout << "==========" << endl;
    cout << setprecision(5) << "s_B = " << s_B << endl;
    cout << "lambda_B = " << lambda_B << endl;
    cout << setprecision(0) << B;
}

// takes the Laplacian of a graph and calculates
// the scale-free metric
int s(SymmetricMatrix L)
{
    int sum = 0;

    for(int i = 0; i < N; i++)
    {
```

```
        for(int j = 0; j < N; j++)
        {
            if(L.element(i,j) == -1)
            {
                sum += (int)(L.element(i,i)*L.element(j,j));
            }
        }
    }

    return sum;
}

int a = 0;
int b = 0;
int c = 0;
int d = 0;

int main()
{
    //Initialize PRG
    PlantSeeds(-1);
    SelectStream(0);
    cout << setprecision(0);

    signal(SIGUSR1, printProgressA);
    signal(SIGUSR2, printProgressB);


    //Initialize adjacency matrix
    SymmetricMatrix Adj(N);
    SymmetricMatrix L(N);
    DiagonalMatrix D(N);
    DiagonalMatrix Eigens(N);


    /*
    //Read from input
    while(!cin.eof())
    {
        int i, j;
        cin >> i >> j;
        Adj.element(i,j) = 1;
        D.element(i,i) += 1;
        D.element(j,j) += 1;
    }
    */
    Eigens = 0;
```

```
while(Eigens(2,2) < 0.000001) //rounding errors
{
    Adj = 0;
    D = 0;
    double r[N];
    double theta[N];
    A = 0; B = 0;
    s_A = 0; s_B = 0;
    lambda_A = 10; lambda_B = 0;

// Geometric Graph
    for(int i = 0; i < N; i++)
    {
        r[i] = Random();
        theta[i] = Uniform(0,360);
    }

    for(int i = 0; i < N; i++)
    {
        for(int j = i+1; j < N; j++)
        {
            double distance = sqrt(r[i]*r[i] + r[j]*r[j]
                - 2*r[i]*r[j]*cos( theta[i]-theta[j]) );

            if( distance <= Dist )
            {
                Adj.element(i,j) = 1;
                D.element(i,i)++;
                D.element(j,j)++;
            }
        }
    }

    //Reconnect stage for isolated nodes.
    for(int i = 0; i < N; i++)
    {
        double backupDist = Dist;
        while(D.element(i,i) == 0)
        {
            for(int j = 0; j < N; j++)
            {
                if(i==j)
                    continue;

                double distance = sqrt(r[i]*r[i] + r[j]*r[j]
                     - 2*r[i]*r[j]*cos(theta[i]-theta[j]) );
```

```
                    if( distance <= backupDist )
                    {
                        Adj.element(i,j) = 1;
                        D.element(i,i)++;
                        D.element(j,j)++;
                        cerr << setprecision(3)
                        cerr << "backupDist = " << backupDis
t << endl;
                    }
                }
                backupDist += 0.01;

            }
        }

    //Build the Laplacian
    L = Adj*-1 + D;

    EigenValues(L,Eigens); // Calculate eigenvalues
}
    cerr << "Initial condition satisfied.\n";

    SelectStream(2);
    while(1)
    {
        //SHUFFLE
        do{
            a = Equilikely(1,N);
            b = Equilikely(1,N);
        } while ( (a == b) || (L(a,b) == 0) );

        do{

            c = Equilikely(1,N);
            d = Equilikely(1,N);
        } while ( (c == d) || (L(c,d) == 0) );

        if( (a==c)||(a==d)||(b==c)||(b==d) )
            continue;

        if(L(a,d) || L(b,c))
            continue;

        L(a,b) = 0; L(c,d) = 0;
        L(a,d) = -1; L(b,c) = -1;
        //END SHUFFLE
```

```
        EigenValues(L,Eigens);

        /*
        if(Eigens(2,2) < 0.0000001)
        {
            //UNSHUFFLE
            cerr << "Unshuffling...\n";
            if(a==0)
                cerr << "unshuffle ERROR.\n";
            L(a,d) = 0;  L(b,c) = 0;
            L(a,b) = -1; L(c,d) = -1;
            a=0;b=0;c=0;d=0;
            //UNSHUFFLE
            continue;
        }
        */

        if((Eigens(2,2) < lambda_A)&&(Eigens(2,2) > 0.0001))
        {
            A = D-L;
            s_A = s(L);
            lambda_A = Eigens(2,2);
        }
        if(Eigens(2,2) > lambda_B)
        {
            B = D-L;
            s_B = s(L);
            lambda_B = Eigens(2,2);
        }
    }

}
```

C.4 poisson.cc

```
#include <iostream>
#include "include/rngs.h"
#include "include/rvgs.h"
#include "newmat10/newmatio.h"
#include "newmat10/newmatap.h"

using namespace std;

#define N 500
#define p (5.0/(float)N)
int main()
{
    //Initialize PRG
    PlantSeeds(-1);
    cout << setprecision(0);

    //Initialize adjacency matrix
    SymmetricMatrix Adj(N);
    SymmetricMatrix L(N);
    DiagonalMatrix D(N);
    DiagonalMatrix Eigens(N);

    for(int repeat = 0; repeat < 200; repeat++)
    {
        Adj = 0;
        D = 0;

        //Random ER (Poisson) Graph

        for(int i = 0; i < N; i++)
        {
            for(int j = i+1; j < N; j++)
            {
                if(Random() < p)
                {
                    Adj.element(i,j) = 1;
                    D.element(i,i)++;
                    D.element(j,j)++;
                }

            }
        }

        //Build the Laplacian
        L = Adj*-1 + D;
```

```
        EigenValues(L,Eigens);

        if(Eigens(2,2) < 0.000001)
        {
            repeat--;
            continue;
        }

        //cout << "Algebraic Connectivity: ";
        cout << setprecision(6) << Eigens(2,2) << endl;
    }
}
```

C.5 prefattach.cc

```
#include <iostream>
#include "include/rngs.h"
#include "include/rvgs.h"
#include "newmat10/newmatio.h"
#include "newmat10/newmatap.h"

using namespace std;

#define N 300
#define k 6

extern float ClusteringCoefficient(SymmetricMatrix A);

int main()
{
    //Initialize PRG
    PlantSeeds(-1);
    cout << setprecision(0);

    //Initialize adjacency matrix
    SymmetricMatrix Adj(N);
    SymmetricMatrix L(N);
    DiagonalMatrix D(N);
    DiagonalMatrix Eigens(N);

    cout << "\n\n# N=" << N << " k=" << k << endl;

    for(int repeat = 0; repeat < 200; repeat++)
    {
        Adj = 0;
        D = 0;
        int degree = 0;

        //Create starting complete graph
        for(int i = 0; i < k; i++)
        {
            for(int j = i+1; j < k; j++)
            {
                Adj.element(i,j) = 1;
                D.element(i,i)++;
                D.element(j,j)++;
                degree += 2;
            }
        }

        //Preferential attachment
```

```
        for(int i = k; i < N; i++) //For all the remaining nodes
        {
            int numlinks = Equilikely(1,k);
            for(int j = 0; j < numlinks; j++) //make links k times
            {
                int link = Equilikely(1,degree);
                int node = 0;
                for(node = 0; link > 0; node++)
                    link -= (int)D.element(node,node);
                node--;

                if(Adj.element(i,node))
                {
                    --j;
                    continue;
                }
                else
                {
                    Adj.element(i,node) = 1;
                    D.element(i,i)++;
                    D.element(node,node)++;
                    degree += 2;
                }
            }
        }

        //Build the Laplacian
        L = Adj*-1 + D;

        EigenValues(L,Eigens); // Calculate eigenvalues
        cout << setprecision(6) << D.Sum()/N << " "
          << Eigens(2,2) << " " << ClusteringCoefficient(Adj) << endl;
    }
}
```

C.6 Makefile

```
prefattach: prefattach.cc cdh ddh
	g++ -g -Wall clustercoeff.cc prefattach.cc include/rngs.c
include/rvgs.c -o prefattach -L./newmat10 -lnewmat -lm

poisson: poisson.cc cdh ddh
	g++ -g -Wall poisson.cc include/rngs.c -o poisson -L./newmat10
-lnewmat -lm

geometric: geometric.cc
	g++ -g -Wall clustercoeff.cc geometric.cc include/rngs.c
include/rvgs.c -o geometric -L./newmat10 -lnewmat -lm

geoshuffle: geoshuffle.cc clustercoeff.cc
	g++ -Wall -O2 geoshuffle.cc include/rngs.c include/rvgs.c
clustercoeff.cc -o geoshuffle -L./newmat10 -lnewmat -lm
```

Bibliography

[ABJ04] Fatihcan M. Atay, Tuerker Biyikoglu, and Juergen Jost. Synchronization of networks with prescribed degree distributions, May 29 2004. Comment: v2: A new theorem and a numerical example added. To appear in IEEE Trans. Circuits and Systems I: Fundamental Theory and Applications.

[BA99] Albert-Laszlo Barabasi and Reka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, October 1999.

[Ber73] C. Berge. *Graphs and Hypergraphs*. North Holland, 1973.

[CHK+01] D. S. Callaway, J. E. Hopcroft, J. M. Kleinberg, M. E. J. Newman, and S. H. Strogatz. Are randomly grown graphs really random? *Phys. Rev. E*, 64:041902, 2001.

[CLR01] George Casella, Michael Lavine, and Christian P. Robert. General — explaining the perfect sampler. *The American Statistician*, 55(4):299–305, 2001.

[CNSW00] D. S. Callaway, M. E. J. Newman, S. H. Strogatz, and D. J. Watts. Network robustness and fragility: Percolation on random graphs, October 19 2000. Comment: 4 pages, 2 figures.

[DHM04] Dasgupta, Hopcroft, and McSherry. Spectral analysis of random graphs with skewed degree distributions. In *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 2004.

[Dim00] Xeni K. Dimakos. A guide to exact simulation, January 11 2000.

[ER59] Paul Erdos and A. Renyi. On random graphs I. *Publicationes Mathematicae*, 6:290–297, 1959.

[Fie73] M. Fiedler. Algebraic connectivity of graphs. *cmj*, 23:298–305, 1973.

[GMZ03] Christos Gkantsidis, Milena Mihail, and Ellen W. Zegura. The markov chain simulation method for generating connected power law random graphs. In *ALENEX*, pages 16–25, 2003.

[Hak62] S. L. Hakimi. On realizability of a set of integers as degrees of the vertices of a linear graph. I. *Journal of the Society for Industrial and Applied Mathematics*, 10(3):496–506, September 1962.

[Hak63] S. L. Hakimi. On realizability of a set of integers as degrees of the vertices of a linear graph II. uniqueness. *Journal of the Society for Industrial and Applied Mathematics*, 11(1):135–147, March 1963.

[HJ85] Roger A. Horn and Charles A. Johnson. *Matrix Analysis*. Cambridge University press, Cambridge, 1985.

[IA05] Shalev Itzkovitz and Uri Alon. Subgraphs and network motifs in geometric networks. *Physical Review E*, 71:026117, 2005.

[JJ01] J. Jost and M. P. Joy. Spectral properties and synchronization in coupled map lattices, October 19 2001. Comment: 10 pages with 15 figures (Postscript), REVTEX format. To appear in PRE.

[KPV04] E. K. Kosmidis, O. Pierrefiche, and J. F. Vibert. Respiratory-like rhythmic activity can be produced by an excitatory network of non-pacemaker neuron models. *Journal of Neurophysiology*, 92:686–699, 2004.

[LAT+05] Lun Li, David Alderson, Reiko Tanaka, John C. Doyle, and Walter Willinger. Towards a theory of scale-free graphs: Definition, properties, and implications (extended version), October 18 2005. Comment: 44 pages, 16 figures. The primary version is to appear in Internet Mathematics (2005).

[Lay94] D. C. Lay. *Linear Algebra and Its Applications*. Addison–Wesley, Reading, MA, 1994.

[LS81] L. Lovasz and Vera T. Sos, editors. *Algebraic Methods in Graph Theory*, volume 1. North-Holland, 1981.

[Mer98] Russell Merris. Laplacian graph eigenvectors. *j-LINEAR-ALGEBRA-APPL*, 278(1–3):221–236, July 1998.

[Moh97] Bojan Mohar. Some applications of laplace eigenvalues of graphs, July 28 1997.

[MSed] Christopher Mayer and Todd Sinai. Network effects, congestion externalities, and air traffic delays: Or why all delays are not evil. Zell/Lurie Center Working Papers 393, Wharton School Samuel Zell and Robert Lurie Real Estate Center, University of Pennsylvania, undated. available at http://ideas.repec.org/p/wop/pennzl/393.html.

[New03] Newman. The structure and function of complex networks. *SIREV: SIAM Review*, 45, 2003.

[Pen03] M. Penrose. *Random Geometric Graphs*. Oxford University Press, Oxford, 2003.

[RCbAH02] Alejandro F Rozenfeld, Reuven Cohen, Daniel ben Avraham, and Shlomo Havlin. Scale-free networks on lattices. *Physical Review Letters*, 89:218701, 2002.

[TGJ+02] Hongsuda Tangmunarunkit, Ramesh Govindan, Sugih Jamin, Scott Shenker, and Walter Willinger. Network topology generators: degree-based vs. structural. In *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 147–159, New York, NY, USA, 2002. ACM Press.

[VL05] Fabien Viger and Matthieu Latapy. Fast generation of random connected graphs with prescribed degrees, February 22 2005.

[WADL04] Walter Willinger, David Alderson, John C. Doyle, and Lun Li. More "normal" than normal: Scaling distributions and complex systems. In *Winter Simulation Conference*, pages 130–141, 2004.

[WR71] J. H. Wilkinson and C. Reinsch, editors. *Linear Algebra*, volume II of *Handbook for Automatic Computation*. Springer-Verlag, Berlin, 1971.

[WS98] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.