Estimating Models of Social Network Formation

ESTIMATING MODELS OF SOCIAL NETWORK FORMATION

BY
MATTHEW OLIVER HOM

A Thesis Submitted to The Honors College

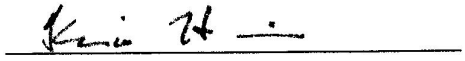
In Partial Fulfillment of the Bachelor's Degree
With Honors in

Economics

THE UNIVERSITY OF ARIZONA

May 2012

Approved by:



Keisuke Hirano
Department of Economics

STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for a degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Signed: Matthew Hom

ESTIMATING MODELS OF SOCIAL NETWORK FORMATION

MATTHEW HOM

Abstract. We study a dynamic model of network formation introduced by Matthew Jackson and Brian Rogers in the paper "Meeting Strangers and Friends of Friends: How Random Are Social Networks?" The model is unique, because nodes are allowed to form multiple links at the same time through random and network-based meetings. We produce MATLAB code which simulates the dynamic network formation process, and then develop likelihood free Markov Chain Monte Carlo (MCMC) techniques to estimate and extend the Jackson-Rogers model. By using simulation techniques, we are able to avoid having to use mean-field approximations to estimate the parameters of the model. In addition, we circumvent the need for explicit evaluation of the likelihood function, and are able to match more features of the model to the data.

Acknowledgements

I would like to thank Dr. Keisuke Hirano, my honors thesis advisor, for his endless guidance, patience, and support throughout the course of this project. I am also grateful to the faculty and staff of the University of Arizona Economics Department, particularly Dr. John Drabicki, Dr. Martin Dufwenberg and Dr. Ronald Oaxaca for their support and mentorship. I would also like to thank the Flinn Foundation and the National Elks Foundation for their generous financial support throughout my undergraduate career.

Finally, I would also like to thank my parents, my friends, and my brother, Conrad, for their love and support.

Contents

1. Introduction 5
2. The Jackson - Rogers model 6
3. Simulating the Jackson - Rogers Model with Matlab 8
4. Analysis of the Jackson - Rogers Model 9
5. Iterative Least Squares 10
6. Simulated Minimum Distance 11
7. Likelihood Free Computation (aka Approximate Bayesian Computation) 12
7.1. Rejection Sampler 13
7.2. MCMC 14
8. Results 17
8.1. Results from Simulated Minimum Distance Estimation 18
8.2. Results from the Rejection Sampler and the Metropolis Hastings Algorithm 18
9. Conclusions, Extensions and Further Work 21
10. Appendix 23
References 35

1. Introduction

Sociologists have long recognized that people make decisions based on the behavior of their family, friends, coworkers, and associates. Studies have empirically verified the influence of social networks on voting patterns, employment opportunities, consumption spending, public health and even one's weight. Researching how networks form and influence behavior is of great importance to society.

Social network analysis has become an increasingly important field of research in economics. This has been attributed to two main reasons. First, network structures influence economic activity. Many classic economic models "abstract away from patterns of interaction leav[ing] certain phenomena unexplained" (Jackson, 2010). For example, traditional job search models used in labor economics literature have made limited use of social networks, despite evidence that information about job opportunities typically comes from friends and acquaintances. Secondly, traditional economic tools like decision theory, game theory, and information economics can be used to study network formation, and develop new insights about patterns of social interaction (Jackson, 2010).

Over the years, sociologists, applied mathematicians, and economists have proposed a variety of models for network formation. These models come in two flavors. The first uses game theoretic tools and is used primarily in the realm of economics. The second is rooted in the random graph literature and is mechanical in nature, modeling network formation as a stochastic process. Although both types of models have been successful at capturing some of the empirical regularities exhibited by large networks, neither has been able to reproduce all of the features (Jackson and Rogers, 2007). Recent work has made a promising push towards combining elements of both approaches to consistently model network formation. This has created a need for new econometric techniques to analyze network data.

The goal of this paper is to use likelihood free Markov Chain Monte Carlo (MCMC) techniques to estimate and extend a model of network formation introduced by Jackson and Rogers (2007). (Hereafter JR.) The JR model is unique in that it is based in the random graph literature, but can be used to "deduce efficiency characteristics of networks" (Jackson and Rogers, 2007). Unlike other models, the JR model allows multiple links to form at the same time through random and network-based meetings. This lies in contrast with models like the one proposed by Christakis et al (2010), which do not allow for multiple links to form concurrently, but use network and degree-dependent utility functions to determine link formation. By varying the ratio of random to network-based meetings, the JR model can be used to stochastically order degree distributions. This novel feature can provide insights into how the network formation process affects the average utility in a network (Jackson and Rogers, 2007).

A limitation of the JR model is that it does not yield closed form solutions for its distributions. The JR model uses mean field approximations to circumvent this issue. This is a characteristic shared by many other network formation models, so developing simulation-based estimation techniques has broad applications. I have been working on relaxing this dependence on mean field approximations.

Jackson and Rogers estimate their model using an ad-hoc estimation technique that matches individual features of the JR model to data in a sequential fashion. While this represents an important first step towards estimating their model, Jackson and Rogers admit there are other methods of estimation. "Hypothetically, one could estimate [the models parameters] using Maximum Likelihood techniques. That would entail ... [deriving] a formula for the likelihood of observed data as a function of the parameters ... This is usually intractable" (Jackson and Rogers, 2007). In this paper I use MCMC techniques to circumvent the need for explicit evaluation of the likelihood function. Using simulation techniques, it is possible to replace "evaluation of the likelihood ... by an estimate of the proximity of a simulated dataset ... to the observed data set" (Sisson and Fan, 2011). By appealing to MCMC methods, I am able to match more features of the model to the data. This generates new estimators of the model's parameters, and obviates the need for mean field analysis.

2. The Jackson - Rogers model

Jackson and Rogers model networks as directed graphs, encapsulating information about the network within matrices (Jackson and Rogers, 2007). Given a network made up of N individuals (aka nodes), we associate a matrix G of size $N \times N$ comprised entirely of 0s and 1s. The g_{ij} entry of the G matrix contains information about the nature of the relationship between nodes i and j. If $g_{ij} = 1$, then i is linked with j. If $g_{ij} = 0$, then i is not linked with j. Nondirected graphs are defined similarly, however we require that $g_{ij} = g_{ji}$. In the case of a nondirected graph, the G matrix is said to be symmetric.

An example of a social network with a directed graph is contained in The National Longitudinal Study of Adolescent Health (Add Health). Add Health is a longitudinal study of adolescents in grades 7 through 12. It tracks the friendships and romantic relationships that develop among thousands of students at multiple high schools over several years. Suppose that i and j are two student participants in the Add Health study. It is not uncommon for $g_{ij} \neq g_{ji}$. For example, suppose

$g_{ij} = 1$ and $g_{ji} = 0$. In this scenario, i considers j to be a friend, but j does not reciprocate the feeling. An example of a nondirected social network would be Facebook. If i is friends with j on Facebook, then $g_{ij} = g_{ji}$, because Facebook treats friendships as reciprocal.

Example: Suppose we have a friendship network made up of three individuals: Adam, John and Rachel.

$$G = \begin{array}{c} \\ \text{Adam} \\ \text{John} \\ \text{Rachel} \end{array} \begin{array}{c} \begin{array}{ccc} \text{Adam} & \text{John} & \text{Rachel} \end{array} \\ \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array}$$

According to the directed graph, Adam and John are mutual friends. John and Rachel are also mutual friends. However, Adam and Rachel are not mutual friends. Rachel considers Adam a friend, but Adam does not consider Rachel to be his friend.

At this point, we define some additional terminology. The neighborhood of node i, denoted by n_i, is the set of all nodes j, which are linked to by node i. In set notation, this is expressed as $n_i = \{j \in N | g_{ij} = 1\}$. The in - degree of node i, denoted by d_i, is the number of unique nodes that link to i. We can express this in set notation as $d_i = |\{j \in N : g_{ji} = 1\}|$.

In the JR model, the network evolves over time. Links form at positive integer time values or "dates". At date t, a new node named t is born. The network existing at the end of date t is denoted by $g(t)$, and the set of nodes existing at the end of date t is denoted by $N_t = \{1, 2, ...,\text{t}\}$.

When t is born at date t, it searches randomly through N_{t-1} in search of m_r nodes to potentially link up with. These m_r nodes are called t's parent nodes. When determining whether to link up with a parent node, t weighs the marginal benefits against the marginal costs of forming a link with the parent node. If the marginal utility is positive, a "random" link is established. Otherwise, no link is formed. In the JR model, the probability a random link forms is equal to a constant, p_r. Note that p_r is independent of t.

In addition to meeting nodes at random, t also meets nodes through network-based meetings. After picking its parent nodes, t identifies m_n nodes at random from its parents' collective neighborhoods. In other words, nodes are picked uniformly at random from the union of t's parents' neighborhoods, excluding t's parents. t proceeds to weigh the marginal benefits of forming a link against the marginal costs of forming a link. If the marginal utility is positive, a "network based link" is said to form. In the JR model, the probability of a network based link is equal to a constant, p_n.

While the JR model is admittedly stylized, it can successfully replicate many of the empirical regularities exhibited in real world social networks. This is a unique feature of the JR model. Many other popular models of social network formation can not reproduce all of the following features (Jackson and Rogers, 2007).

(1) The average "distance" between pairs of nodes is small, as is the maximum "distance" between any two pairs of nodes (diameter). Here, distance is defined to be the minimum path length between the two nodes of interest.
(2) Clustering, the tendency for linked nodes to share common neighbors, is typically higher in social networks relative to networks in which links are generated randomly.

(3) The distribution of degrees typically exhibits fat tails. Nodes with relatively high and low degrees appear more frequently than nodes with medium degrees.
(4) Degrees of linked nodes are positively correlated. So nodes with high degrees are more likely to be linked with other high degree nodes. Similarly low degree nodes are more likely to be linked with low degree nodes.
(5) The clustering among neighbors of a given node tends to be inversely related to a node's degree. So neighbors of higher degree nodes are less likely to be linked to each other relative to neighbors of a lower-degree node.

3. Simulating the Jackson - Rogers Model with Matlab

We can simulate the JR model using Matlab code. Consider the following algorithm:

Rangraph Algorithm

Inputs: m_r, m_n, p_r, p_n and the size of the network dim
Output: A directed graph matrix G.

(1) Initialize parameters m_r, m_n, p_r, p_n.
(2) Pick a k such that $m_r + m_n \leq k \leq$ dim
(3) Generate a zero matrix G of dimension $k \times k$
(4) Populate G with $1s$ such that each node from $1, 2, ...k$ has at least $m_r + m_n$ neighbors. In other words, create the initial graph.
For i from 1 to k
   (a) Decide how many neighbors i will have.
   Pick a random number $r_i \in \mathbb{N}$ such that $m_r + m_n \leq r_i \leq k - 1$.
   (b) Identify i's neighbors.
   Pick r_i numbers at random from $\{1, ..., i-1, i+1, ...k\}$. Go to the ith row and place 1s in the corresponding indices.
   (c) end the for loop

Everything should be initialized at this point. G is now a matrix of $0s$ and $1s$ which represents our initial adjacency matrix. We now can simulate the birth of a new individual

(5) For $i = k + 1 :$ dim
   (a) Identify the parent set of node i.
   Define a new set: parents_$i = \{\}$.
   Draw m_r numbers at random from $\{1, ..., i-1\}$. Place them in parents_$i = \{\}$.
   (b) Determine which parent nodes will be linked to.
   For each of i's parent nodes.
   Draw a random number x from $[0, 1]$.
   If $x < p_r$ then let $G(i, \text{parent}) = 1$, otherwise leave $G(i, \text{parent})$ alone.
   end the for loop
   (c) Identify the union of i's parents neighborhoods
   For each of i's parent nodes, identify its neighborhoods.

   For instance, if j is a parent of i, go to the jth row of i. Record which entries equal 1. Each of these entries identifies a node in the

neighborhood of j. Take the union over all parents, and call this set nghbdparents_i.

(d) Loop through nghbdparents_i and form links
Pick m_n numbers at random from nghbdparents_i.
For each of the m_n numbers, draw a random number x from $[0, 1]$. If $x < p_n$ then go to the ith row and put 1 in the appropriate coordinate. Otherwise, leave the ith row alone.
end the for loop

The algorithm necessarily glosses over some of the more technical issues involved with simulating the JR model. Please refer to the appendix for fully documented MATLAB code.

4. Analysis of the Jackson - Rogers Model

Jackson and Rogers rely on mean field approximation to simplify the analysis of their model. They use the following differential equation to model the evolution of the in-degree of node i over time.

$$\frac{dd_i(t)}{dt} = \frac{p_r m_r}{t} + \frac{p_n m_n d_i(t)}{tm}$$

where $m = p_r m_r + p_n m_n$ is the expected number of links that a new node forms (Jackson and Rogers, 2007).

The intuition for the differential equation comes from considering the probability that node i with in-degree $d_i(t)$ gets a new link in period $t+1$. This probability is approximately equal to

$$\underbrace{\frac{p_r m_r}{t}}_{\text{random}} + \underbrace{p_n \frac{m_r d_i(t)}{t} \frac{m_n}{m_r(p_r m_r + p_n m_n)}}_{\text{network}}$$

(1) $\frac{p_r m_r}{t}$ represents the probability that node i is found at random by the new node and then linked to. This formula makes intuitive sense, because there are t existing nodes at time t, from which m_r are chosen at random, each having probability p_r of linking up with i.

(2) $p_n \frac{m_r d_i(t)}{t} \frac{m_n}{m_r(p_r m_r + p_n m_n)}$ represents the probability that node i is found through a network based meeting and then linked to. This probability is the product of three terms. The term $\frac{m_r d_i(t)}{t}$ represents the probability that some node, call it j, which is linked to i is subsequently chosen as a parent. The term $\frac{m_n}{m_r(p_r m_r + p_n m_n)}$ is the probability that i is found, given that j is a parent. Here, $p_r m_r + p_n m_n$ represents the expected number of links that a new node forms, and $m_r(p_r m_r + p_n m_n)$ represents the expected size of the union of the parents' neighborhoods. The final term p_n is the probability that a link occurs after a network based meeting (Jackson and Rogers, 2007).

Setting the in-degree of each newly born node to d_0, Jackson and Rogers solve the differential equation and get

$$d_i(t) = (d_0 + rm)\left(\frac{t}{i}\right)^{1/(1+r)} - rm$$

where $r = \frac{p_r m_r}{p_n m_n}$ represents the ratio of the number of links formed uniformly at random to the number of links formed via network based meetings. The interpretation of r is application dependent. For example, r might "capture the inherent randomness of the environment in terms of the meeting process, or a search algorithm that nodes choose to follow in meeting with other nodes" (Jackson and Rogers, 2007).

Using the mean field analysis framework, Jackson and Rogers are able to derive a suite of interesting properties for the JR model. Here are two notable highlights of their analysis, both involving the ratio r (Jackson and Rogers, 2007).

- Theorem 1: The in degree distribution of the mean field process above has a cumulative distribution function of

$$F_t(d) = 1 - \left(\frac{d_0 + rm}{d + rm}\right)^{1+r}, d \geq d_0$$

- Theorem 6: Consider the distribution function shown above with any fixed $m > 0$ and let $d_0 = 0$. If $r' > r$, then F' strictly second order stochastically dominates F, where F' and F are distribution functions corresponding to r' and r respectively.

  Jackson and Rogers (2007) note that Theorem 6 has important implications. "If agents' utilities can be expressed as a concave function of their degree, then we can order the total utility of a network with respect to r." In this context, concavity of utility with respect to degree implies that nodes experience diminishing marginal utilities with respect to additional links.

  Corollary 1 of Theorem 6 is: "Suppose the expected utility of a node in a network is a concave function of the node's degree and the network's degree distribution is described by Theorem 1 with $d_0 = 0$, Then, for any given m, if $r' > r$, the average expected utility of agents in the network with r' is weakly higher than that under r, with a strict ranking if the expected utility function is strictly convex in degree" (Jackson and Rogers, 2007).

Real world networks exhibit a variety of characteristics. The analysis of Jackson and Rogers (2007) suggests that the ratio of random to network based meetings can have a significant impact on the network formation process, and the overall efficiency of a network. It is therefore worthwhile to develop robust estimates of r.

5. Iterative Least Squares

Jackson and Rogers use a simple calibration process to fit their model to a variety of datasets and estimate r. Here are the steps outlined in the paper (Jackson and Rogers, 2007):

(1) Compute m from the data by calculating the average in degree for the network.
(2) Fix m and set $d_0 = 0$
(3) Use Iterative Least Squares to estimate $-(1 + r)$. Note that r enters F in the expression $(d + rm)^{-1+r}$. Beginning with an initial guess r_0 for r, plug into the expression to get

$$(d + r_0 m)^{-(1+r)}$$

Regressing $ln(1-F(d))$ on $ln(d+r_0 m)$ we obtain an estimate for $-(1+r)$, which we set equal to r_1. We can iterate this process until the estimate for r converges to a fixed point.

Iterative Least Squares is a natural first step for estimating r, however it is contingent on the assumption of mean field analysis being close to the original model. Mean field analysis is commonly used when working with complex dynamic systems, but "relatively little is known about the circumstances where such analyses result in accurate approximations...[the] simulations of the model result in characteristics consistent with the approximations, but there are no results proving ... [they] are tight" (Jackson and Rogers, 2007).

It is possible to estimate r using other techniques. Jackson and Rogers (2007) note that "Hypothetically, one could estimate [the models parameters] using Maximum Likelihood techniques. That would entail [deriving] a formula for the likelihood of observed data as a function of the parameters...This is usually intractable." However, by appealing to likelihood-free computation (aka approximate Bayesian computation) it is possible to circumvent the need for explicit evaluation of the likelihood while simultaneously matching more features of the model to the data. It is also important to note that the JR approach does not provide standard errors or other measures of parameter uncertainty. This is due to both the lack of an explicit likelihood, and the fact that data on a single network are necessarily dependent, so that i.i.d. sampling theory cannot be applied. I will be able to address both issues with likelihood-free computation. I devote the remainder of this paper to developing simulation based techniques to estimate r.

6. Simulated Minimum Distance

Given data on a social network, the goal is to estimate r, the ratio of links formed at random to those formed through network based meetings. One intuitive way to estimate r is to simulate the model under different parameter vectors $\theta = (m_r, m_n, p_r, p_n)$ and to pick the parameter vector which generates the simulated network "closest to reality". The choice of metric is admittedly arbitrary, and there are a variety of choices. Here's one approach:

Let $F(n)$ be the CDF of the in-degree distribution for the observed network. Analogously, let $\widehat{F(n|\theta)}$ be the CDF of the in-degree distribution for the simulated network, conditional on the choice of θ as a parameter vector. Define $\hat{h}(\theta)$ as follows:

$$\hat{h}(\theta) = \begin{pmatrix} F(0) - \widehat{F(0|\theta)} \\ F(1) - \widehat{F(1|\theta)} \\ \vdots \\ F(n) - \widehat{F(n|\theta)} \end{pmatrix}$$

If we define the distance between a simulated dataset and the actual dataset to equal $\hat{h}(\theta)'\hat{h}(\theta)$, the goal becomes finding the parameter vector which minimizes $\hat{h}(\theta)'\hat{h}(\theta)$. We can do this in two ways.

6.0.1. *Grid Approach.* Fixing the random number seed, simulate the JR model while cycling through different combinations of m_r, m_n, p_r, p_n. Ideally, m_r and m_n will be precalibrated using information specific to the application at hand. If we

make the simplifying assumption that $m_r = m_n = 1$, we can think of the parameter grid as a matrix.

Compute the in-degree distribution associated with each parameter combination. After each run store the results. Find the value of θ which minimizes $\hat{h(\theta)}' \hat{h(\theta)}$. We can now back out r from m_r, m_n, p_r, p_n values.

Example: Suppose that $m_r = m_n = 1$ and let p_r and p_n vary over the following mesh of values $[0, .05, .10, .15, \ldots, .85, .90, .95, 1.00]$. The theta vector, $\theta = (m_r, m_n, p_r, p_n) = (1, 1, p_r, p_n)$. If we suppress the m_r, m_n values, we can express this as $\theta = (p_r, p_n)$. Consider the following matrix

$$\begin{array}{cc} P_r \backslash P_n & \begin{array}{cccc} 0 & .05 & \ldots & 1.0 \end{array} \\ \begin{array}{c} 0 \\ .05 \\ \vdots \\ 1.0 \end{array} & \begin{pmatrix} \hat{h}(0,0)'\hat{h}(0,0) & \hat{h}(0,.05)'\hat{h}(0,.05) & & \hat{h}(0,.1.0)'\hat{h}(0,1.0) \\ \hat{h}(.05,0)'\hat{h}(.05,0) & \hat{h}(.05,.05)'\hat{h}(.05,.05) & & \hat{h}(.05,1.0)'\hat{h}(.05,1.0) \\ \vdots & \vdots & \ddots & \vdots \\ \hat{h}(1.0,0)'\hat{h}(1.0,0) & \hat{h}(1.0,.05)'\hat{h}(1.0,.05) & & \hat{h}(1.0,1.0)'\hat{h}(1.0,1.0) \end{pmatrix} \end{array}$$

To back out r, simply locate the minimal element of the matrix, and compute $\frac{p_r}{p_n}$.

6.0.2. *Numerical Optimization.* Alternatively, instead of using a grid search, we can program Matlab to use numerical optimization techniques to find the optimal values of m_r, m_n, p_r, p_n. This problem is an application of constrained nonlinear optimization and requires use of the fmincon command in Matlab. Interested readers should visit the Mathworks website for additional information on the fmincon command.

7. Likelihood Free Computation (aka Approximate Bayesian Computation)

In classical, frequentist statistics, the parameter vector θ is thought to be an unknown, albeit fixed, quantity. Knowledge about θ is obtained via random samples drawn from a population indexed by θ. In contrast, the Bayesian approach treats θ as a "quantity whose variation can be described by a probability distribution (called the prior distribution)...A sample is then taken from a population indexed by θ and the prior distribution is updated with this sample information [via Bayes' rule]. The updated prior is called the posterior distribution" (Casella and Berger, 2001).

If we let $\pi(\theta)$ represent the prior distribution for the parameter vector, and let $\pi(y|\theta)$ denote the sampling distribution, the posterior distribution, or conditional distribution of θ based on the data vector y is given by

$$\pi(\theta|y) = \frac{\pi(y|\theta)\pi(\theta)}{m(y)} \qquad m(y) = \int \pi(y|\theta)\pi(\theta)d\theta$$

where $m(y)$ is the marginal density of the data vector y (Casella and Berger, 2001). In application, we often drop $m(y)$ and write

$$\pi(\theta|y) \propto \pi(y|\theta)\pi(\theta)$$

With the exception of a few simple cases, computing the posterior distribution requires numerical simulation methods. In many applications, computing the likelihood function $\pi(y|\theta)$ is analytically impossible or computationally intractable. The

term likelihood free computation (aka approximate Bayesian computation) refers to the set of algorithms designed to circumvent the need for explicit evaluations of the likelihood function while performing Bayesian inference. Approximate Bayesian Computation (ABC) is a recent development in computational Bayesian statistics, and is summarized in Sisson and Fan (2011) .

The underlying principle of likelihood-free computation is as follows (Sisson and Fan, 2011). Given a candidate parameter vector θ', generate a dataset x from the model. In other words, let $x \sim \pi(x|\theta')$. If the simulated and the actual/observed dataset are similar [1], then θ' is a likely candidate to have generated the observed data from the given model. θ' is retained and forms a part of the samples from the posterior distribution. If the simulated and observed datasets are different from each other, then θ' is an unlikely candidate to have generated the observed dataset, and θ' is discarded.

Likelihood-free algorithms are classified according to the manner by which they generate candidate parameter vectors. In this paper, we study three well known sampling algorithms: the rejection sampler, which draws candidate vectors directly from the prior distribution, and the Metropolis and Metropolis Hastings algorithms, which draw candidate vectors from an auxiliary distribution, called the proposal distribution. Depending upon what information is known about the posterior distribution, it may be preferable to use one over the other two. In the context of the Jackson Rogers model, it is not clear a-priori which method yields the best results, so we use all three approaches.

7.1. **Rejection Sampler.** The rejection sampler is the simplest likelihood-free algorithm. While it is easy to implement, it can be inefficient, especially if the prior and likelihood are quite different. In contrast, an MCMC algorithm may be better, but it requires work to find good candidate distributions. The rejection sampler consists of three steps (Sisson and Fan, 2011):

(1) Generate $\theta' \sim \pi(\theta)$ from the prior
(2) Generate dataset x from the model $\pi(x|\theta')$
(3) Accept θ' if $x \approx y$

In the JR model, there are four main parameters of interest: m_r, m_n, p_r, p_n. Recall that m_r represents the number of nodes met uniformly at random upon birth. p_r represents the probability of becoming friends during a random encounter. m_n, p_n are defined analogously for network based encounters.

In this context we can think of θ as $\theta = (m_r, m_n, p_r, p_n)$ where m_r, m_n are random positive integers and $p_r, p_n \in [0, 1]$. A-priori all values of p_r and p_n seem equally plausible, so I propose using the bivariate uniform distribution as my prior for p_r and p_n. [2]

Algorithm:

(1) Generate two random numbers between $[0, 1]$. Use them for the values of p_r, p_n. Generate two random positive integers and use them for m_r and m_n.
(2) Run the RanGraph function with parameter values m_r, m_n, p_r, p_n. This will generate a graph G.

[1] This is defined somewhat arbitrarily, and varies depending upon the application
[2] We can use other priors e.g. the bivariate normal distribution

(3) Extract the degree distribution of G. Compute the simulated distance between G's degree distribution and the degree distribution of the actual data. If the distance is "small enough" accept (m_r, m_n, p_r, p_n) as a plausible parameter vector. Otherwise, reject. [3]

At the end of the day, the set of $\theta' s$ retained may be considered a set of sample draws from $\pi(\theta|y)$

7.2. **MCMC.** The goal of Markov Chain Monte Carlo methodology is to sample from complicated probability distributions, by constructing a Markov chain that has the desired distribution as its equilibrium distribution. Our desired distribution in this application is $\pi(\theta|y)$, the posterior distribution. After a sufficiently long burn in period, the Markov Chain can be treated as a sample of the target distribution. The quality of the sample generally improves as a function as a number of the trials/steps.

There are a variety of MCMC methods. In this paper, I focus on the Metropolis and the Metropolis Hastings algorithm. However, there are other samplers that might be appropriate for application to the Jackson Rogers model, most notably the Gibbs sampler.

7.2.1. *Metropolis Algorithm.* The Metropolis Algorithm is composed roughly of the following four steps (Sisson and Fan, 2011) (Walsh):

(1) Pick a starting parameter value (vector) θ_0 such that θ_0 represents a likely draw from the posterior distribution.
(2) Using the current θ value, sample a candidate parameter vector θ^* from some proposal/candidate-generating/ jumping distribution $q(\theta_1, \theta_2)$. The jumping distribution gives the probability of going from θ_1 to θ_2. It is important to note that the Metropolis Algorithm requires that the proposal distribution be symmetric: i.e. $q(\theta_1, \theta_2) = q(\theta_2, \theta_1)$.
(3) Given a candidate point θ^*, compute the ratio

$$\alpha = \frac{\pi_\epsilon(y|x', \theta')\pi(\theta')}{\pi_\epsilon(y|x_t, \theta_t)\pi(\theta_t)}$$

$\pi_\epsilon(y|x, \theta)$ is called the kernel density, and is used to approximate the sampling distribution, $\pi(y|x, \theta)$. For more information, please consult Sisson and Fan (2011) .
(4) If $\alpha > 1$, accept the candidate point and set $\theta_t = \theta^*$. Return to step 2 and loop through. If $\alpha < 1$, then with probability α accept the candidate point, else reject and return to step 2.

The Metropolis sampling algorithm generates a Markov Chain $(\theta_0, \theta_1, ..., \theta_k, ...)$ because the transition probabilities from θ_t to θ_{t+1} depend only on θ_t and not on the previous values $\theta_0, ..., \theta_{t-1}$. After a burn in period of say k steps, the chain approaches a stationary distribution, at which point samples from $(\theta_{k+1}, ..., \theta_{k+n})$ will represent samples from the posterior distribution (Walsh).

[3]Figuring out what constitutes "small" is arbitrary. In this paper, we use a simple euclidian metric to compare the CDF of the in-degree distribution for the actual data with the CDF of the in-degree distribution for the simulated data.

7.2.2. *Metropolis Hastings Algorithm.* The Metropolis Hastings Algorithm is a slight variation of the Metropolis algorithm. The only major difference is that the proposal distribution is not required to be symmetric. Here are the steps involved (Sisson and Fan, 2011):

(1) Initialize (θ_0, x_0) and ϵ. Let $t = 0$.
(2) Generate $\theta' \sim q(\theta_t, \theta)$ from a proposal distribution
(3) Generate $x' \sim \pi(x|\theta')$ from the model given θ'
(4) With probability

$$\min\left\{1, \frac{\pi_\epsilon(y|x', \theta')\pi(\theta')q(\theta', \theta_t)}{\pi_\epsilon(y|x_t, \theta_t)\pi(\theta_t)q(\theta_t, \theta')}\right\}$$

set $(\theta_{t+1}, x_{t+1}) = (\theta', x')$. Otherwise, $(\theta_{t+1}, x_{t+1}) = (\theta_t, x_t)$
(5) Update time. Set $t = t + 1$.

7.2.3. *Proposal Distributions.* There are two general approaches when picking proposal distributions: random walks and independent chain sampling (Walsh). When using a random walk based proposal distribution, the new value y equals current value x plus a random variable z.

That is,

$$y = x + z$$

So $q(x, y) = g(y - x) = g(z)$, the density associated with the R.V. z. If the density for the R.V. z is symmetric, which occurs when using the normal distribution, or the uniform distribution, then one should use the Metropolis sampling algorithm since $\frac{q(x,y)}{q(y,x)} = \frac{g(z)}{g(-z)} = 1$ (Walsh). [4]

The second way to generate a proposal distribution is to use an independent chain. When using an independent chain, the probability of jumping to point y is independent of the current position of the chain. This implies that $q(x, y) = g(y)$. In other words, the candidate value is drawn from a distribution of interest, and independent of the current value. In general, the proposal distribution is not symmetric, because $g(x) \neq g(y)$ and the Metropolis Hasting algorithm must be used (Walsh).

7.2.4. *Implementing the Metropolis Algorithm with a Normal proposal to estimate the JR model:*

- y represents the CDF of the in-degree distribution for the actual data.
- x represents the CDF of the in-degree distribution outputted from the RanGraph algorithm.
- $\Gamma = (\gamma_r, \gamma_n, m_r, m_n)$ represents the parameter vector.
- $F(z)$ is the inverse logit function. That is

$$F(z) = \frac{e^z}{1 + e^z}$$

- The prior distribution are as follows: $\pi(p_r, p_n)$ is a bivariate normal distribution: $\mathcal{N}(\underline{0}, \tau I_2)$. $\pi(m_r)$ and $\pi(m_n)$ are discrete uniform distributions over a set of positive integers. We assume that $\gamma_r, \gamma_n, m_r, m_n$ are independent.

[4]The variance of the proposal distribution, which is called the tuning parameter, can be varied to get better mixing results (Walsh).

- $\pi_\epsilon(y|x,\theta) = 1$ if the simulated euclidian distance between x' and $y' \leq \epsilon$ and 0 otherwise. [5]

(1) Set $t = 0$ and initialize ϵ to desired value. Here ϵ represents a pre specified tolerance level. Pick a random draw Γ^* from $\pi(\Gamma)$. That is, $\Gamma^* \sim \pi(\Gamma)$. If $\pi_\epsilon(y|x,\Gamma^*) = 1$, then set $\Gamma_0 = \Gamma^*$. Otherwise, continue sampling from $\pi(\Gamma)$ until an appropriate Γ_0 is found.

At step t:

(2) Generate $\Gamma' \sim q(\Gamma_t, \Gamma)$ where

$$\begin{pmatrix}\gamma_r' \\ \gamma_n'\end{pmatrix} = \begin{pmatrix}\gamma_r \\ \gamma_n\end{pmatrix} + \mathcal{N}\left(\begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}\sigma_1 & 0\\ 0 & \sigma_2\end{pmatrix}\right)$$

Apply F to the individual components of Γ' to get p_r and p_n.

$$\theta' \equiv \begin{pmatrix}F(\gamma_r)' \\ F(\gamma_n')\end{pmatrix} \equiv \begin{pmatrix}p_r' \\ p_n'\end{pmatrix}$$

Finally, draw new values for m_r' and m_n'. For simplicity, we use a uniform jump distribution for m_r and m_n. [6]

(3) Run the RanGraph function using m_r', m_n', p_r', p_n' as parameters. Extract the CDF of the in-degree distribution, x'.

(4) With probability $\min\left\{1, \dfrac{\pi_\epsilon(y|x',\Gamma')\pi(\Gamma')}{\pi_\epsilon(y|x_t,\Gamma_t)\pi(\Gamma_t)}\right\}$ set $(\theta_{t+1}, x_1) = (\theta', x')$. Otherwise, set $(\theta_{t+1}, x_1) = (\theta_t, x_t)$.

(5) Increment $t = t + 1$ and go back to step 2.

7.2.5. *Implementing the Metropolis Hastings Algorithm with a Beta proposal to estimate the JR model.*

- y represents the CDF of the in-degree distribution for the actual data.
- x represents the CDF of the in-degree distribution outputted from the RanGraph Function.
- $\theta = (p_r, p_n, m_r, m_n)$ represents the parameter vector.
- The prior distributions are as follows: $\pi(p_r, p_n) \sim$ Bivariate Uniform Distribution, $\pi(m_r)$ and $\pi(m_n)$ are discrete uniform distributions over some set of positive integers. We assume that p_r, p_n, m_r, m_n are independent.
- Define $\pi_\epsilon(y|x,\theta) = 1$ if the simulated euclidian distance between x' and $y' \leq \epsilon$ and 0 otherwise. [7]

(1) Set $t = 0$ and initialize ϵ to desired value. Here ϵ represents a pre specified tolerance level. Pick a random draw θ^* from $\pi(\theta)$. That is, $\theta^* \sim \pi(\theta)$. If $\pi_\epsilon(y|x,\theta^*) = 1$, then set $\theta_0 = \theta^*$. Otherwise, continue sampling from $\pi(\theta)$ until an appropriate θ_0 is found.

At step t.

[5] This is often referred to as the uniform kernel density.
[6] Note that $q(\Gamma', \Gamma_t')$ is symmetric and the proposal distribution is a random walk.
[7] This is often referred to as the uniform kernel density.

(2) Generate $\theta' \sim q(\theta_t, \theta)$
Let $p'_r \sim \beta\left(\frac{2p_{r,t-1}}{1-p_{r,t-1}}, 2\right)$, $p'_{n,t-1} \sim \beta\left(\frac{2p_{n,t-1}}{1-p_{n,t-1}}, 2\right)$ [8] and let m'_r and m'_n be drawn from a discrete uniform jump distribution. [9]
(3) Run the RanGraph algorithm using m'_r, m'_n, p'_r, p'_n as parameters. Extract the CDF of the in-degree distribution, which we denote by x'.
(4) With probability

$$\min\left\{1, \frac{\pi_\epsilon(y|x', \Gamma')\pi(\Gamma')q(\Gamma', \Gamma_t)}{\pi_\epsilon(y|x_t, \Gamma_t)\pi(\Gamma_t)q(\Gamma_t, \Gamma')}\right\}$$

set $(\theta_{t+1}, x_1) = (\theta', x')$. Otherwise, set $(\theta_{t+1}, x_1) = (\theta_t, x_t)$.
(5) Increment $t = t + 1$ and go back to step 2.

8. Results

Table 1. Ham Radio Operators Network

Degree	Number of Operators	Degree	Number of Operators	Degree	Number of Operators
0	3	9	3	18	0
1	11	10	2	19	0
2	5	11	2	20	1
3	2	12	1	21	1
4	1	13	1	22	1
5	0	14	0	23	0
6	2	15	0	24	0
7	2	16	1	25	1
8	3	17	0	26	0
				27	1

I applied the estimation techniques referenced earlier to a dataset used in the Jackson and Rogers paper. The dataset pertains to a network of Ham Radio operators and can be found at the following url: "http://www.stanford.edu/ jackson-m/Data.htm". Preliminary results suggest there may be a great deal of parameter uncertainty when estimating the JR model.

The table above represents the degree distribution of a network of amateur ham radio operators. A link signifies that two operators had a radio conversation during a one month period. The data was collected by Killworth and Bernard in 1976 for their paper, "Informant accuracy in social network data" (Jackson, 2008).

Using Iterative Least Squares to fit the model to their data, Jackson and Rogers (2007) estimate an r value of 5. I obtained a different set of estimates when using simulation based estimation techniques.

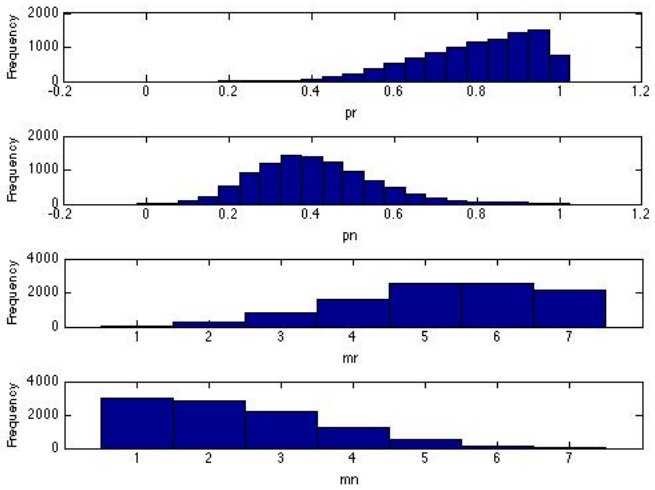
[8] The expected value of a Beta distribution with parameters α, β is equal to $\frac{\alpha}{\alpha+\beta}$. This implies that p'_r and p'_n are drawn from Beta distributions centered at $p_{r,t-1}$ and $p_{n,t-1}$ respectively. One can easily modify this technique for any desired value of β. In general, larger values of β will result in a larger variance, and hence a higher degree of mixing for the Markov Chain.

[9] The beta distribution is not symmetric, which implies that we must use the Metropolis Hastings algorithm.

8.1. **Results from Simulated Minimum Distance Estimation.** The results suggest that this method is sensitive to initial conditions. Assuming that $m_r = m_n = 1$ [10], the p_r, p_n combination that minimizes simulated distance is $[.93, .98]$. This suggests an estimate of $\hat{r} = .9490$. This value is much lower than the ILS estimate of 5. Note that simulated minimum distance estimation yields a point estimate, in contrast with the rejection sampler and MCMC, which produce approximations of the posterior distribution for θ.

8.2. **Results from the Rejection Sampler and the Metropolis Hastings Algorithm.** The results from the rejection sampler and the Metropolis Hastings Algorithm suggest that the ratio of randomly formed links to network based links is high for the Ham Radio network. This implies that the majority of the ham radio conversations observed by Killworth and Bernard occurred randomly between strangers.

FIGURE 1. Rejection Sampler, 10^4 trials, $\epsilon = .2$.



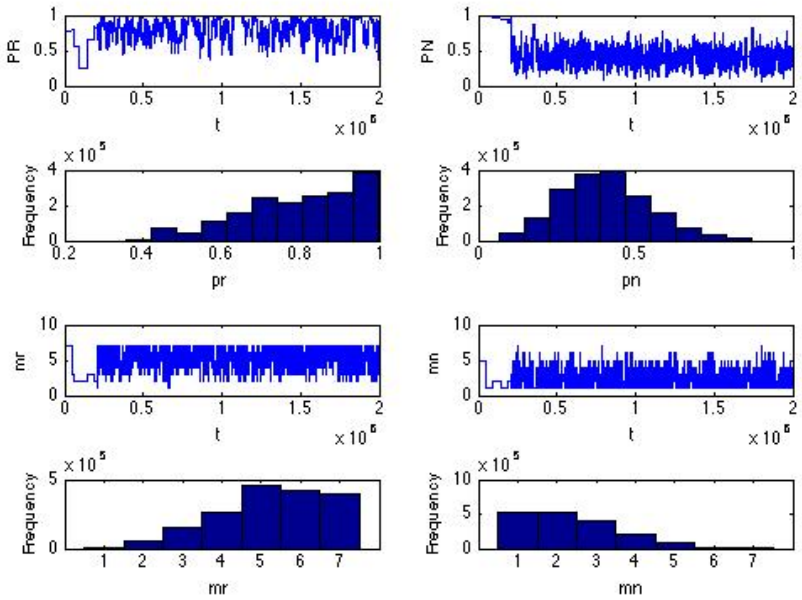
Here are the summary sample statistics for a Rejection Sampler over ten thousand trials, at a tolerance level of $\epsilon = .2$. [11] Note that because I chose to use a uniform prior, we can interpret the rejection sampler as simulating the (approximate) likelihood.

Rejection Sampler	p_r	p_n	m_r	m_n	r
sample median	.8218	.3952	5.0000	2.0000	4.9073
sample mean	.7970	.4091	5.2553	2.3964	6.1828
sample standard error	.1454	.1490	1.3631	1.2753	4.3720

[10] This is not reflective of reality, and we relax this assumption for the rejection sampler, and the Metropolis Hastings algorithm

[11] m_r and m_n were drawn from a discrete uniform distribution over $\{1, 2, ..., 7\}$.

FIGURE 2. Metropolis Hastings Algorithm, β proposal, $2 * 10^6$ trials, $\epsilon = .2$



Here are the summary sample statistics for a Metropolis Hastings based algorithm using a β proposal, over 2 million trials, at a tolerance level of $\epsilon = .2$. I account for a burn in value of $.25 * 10^6$ when computing the sample median, sample mode and sample standard deviation. Note that because I chose to use a uniform prior, we can interpret the Metropolis Hastings algorithm as simulating the (approximate) likelihood.[12]:

Metropolis Hastings	p_r	p_n	m_r	m_n	r
sample median	.8158	.3936	5.0000	2.0000	5.1123
sample mean	.7933	.4049	5.2519	2.3607	6.5966
sample standard deviation	.1504	.1429	1.3843	1.2236	5.6540

The marginal distributions of p_r under both approaches are rather diffuse, with more weight distributed towards large values of p_r. Note that they exhibit a non-trivial positive slope and appear to have a peak close to one. Furthermore, the marginal distributions of p_r are not normal - looking due to the boundary. In contrast, the marginal distributions of p_n are fairly symmetric, and centered around values near .4.

The marginal distributions of m_r and m_n are fairly skewed, with the marginal distributions of m_r placing the majority of their weight on high values of m_r, and the marginal distributions of m_n placing the majority of their weight on low values

[12] m_r and m_n were drawn from a discrete uniform distribution over $\{1, 2, \dots 7\}$.

of m_n. Overall, the data do not appear to sharply pin down the parameter values for p_r, p_n, m_r, m_n and this leads to a great deal of uncertainty when estimating r. This is useful to note because the JR approach does not reveal this insight.

FIGURE 3. Distribution of r using the Rejection Sampler, 10^4 trials, $\epsilon = .2$

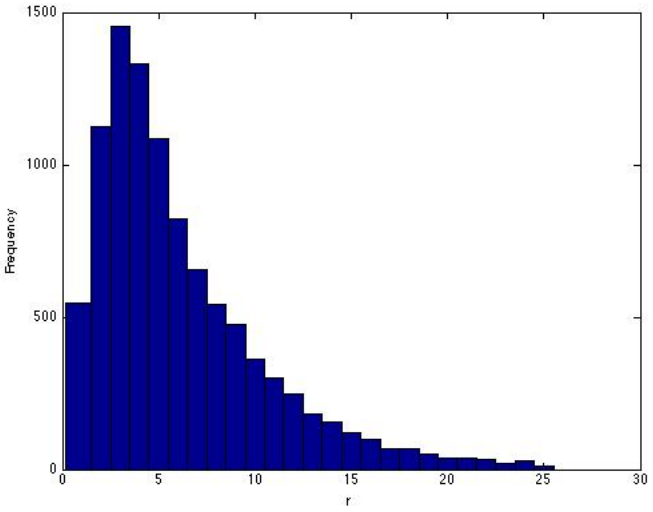
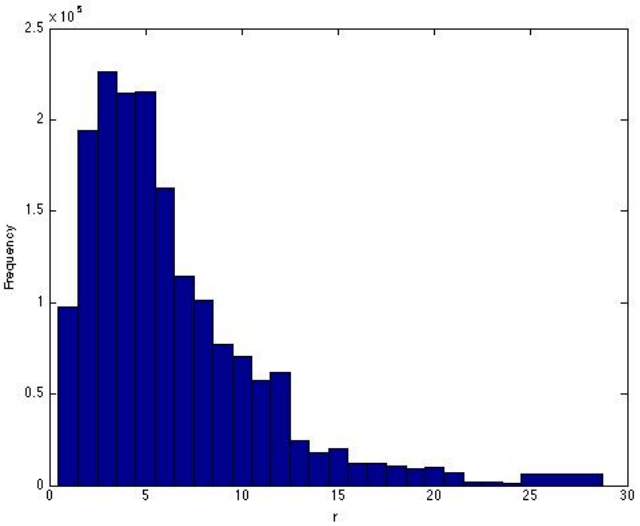


FIGURE 4. Distribution of r using the Metropolis Hastings Algorithm, β proposal, $2 * 10^6$ trials, $\epsilon = .2$



Unlike ILS, which generates a point estimate of r, we can also approximate the distribution of r using the output generated from the rejection sampler and the Metropolis Hastings Algorithm (see figures 3 and 4). Note that the simulated distributions of r are fairly diffuse, and exhibit a strong positive skew, concentrating a significant portion of their weight on low values of r. The median value of r under the rejection sampler is 4.9073 and if we exclude significant outliers, the standard deviation of r is 4.3720. [13] The median value of r under the Metropolis Hastings algorithm is 5.1123 after accounting for the burn in, and the standard deviation of r is 5.6540.

9. Conclusions, Extensions and Further Work

The results of this paper demonstrate that approximate Bayesian computation can be successfully applied to the estimation of models of social network formation. In the context of the JR model, we were able to eliminate dependence on mean field approximation, generate approximations of the parameters' distributions, and develop measures of parameter uncertainty. We discovered that in the case of the Ham Radio dataset, ILS point estimation does not capture the presence of parameter uncertainty, a feature we were able to detect using likelihood-free methods. [14] By relying on likelihood-free methods, we were also able to circumvent the need for explicit evaluation of the likelihood function, and in some cases even simulate the approximate likelihood. [15] Finally, we were able to match features of the model to the data, which ILS overlooked. [16] In the process of developing insights about the JR model, we developed techniques and estimators that can in principle be modified and adapted to work with other models of social network formation.

The JR model of network formation is admittedly stylized and there are multiple directions one could go in future research. The relaxation of dependence upon mean field approximation remains a top priority. Many of the novel results discovered by Jackson and Rogers, particularly regarding the relationship between r and network efficiency, hinge upon the assumption of mean field approximation. The results of this paper demonstrate that one can approximate the likelihood function to a high degree of accuracy. However, it remains to be seen if one can establish tight, analytical arguments without relying on mean field approximations.

Another possible direction, would be to explore the issue of parameter uncertainty. The data do not appear to sharply pin down the parameter values for p_r, p_n, m_r and m_n. I suspect that this may have to do with the marginal utility structure embedded in the link formation process.

The JR model sets the probabilities of link formation to random fixed constants (e.g. p_r, p_n) that are independent of the structure of the network. This is not ideal for modeling the formation of real world networks, in which the marginal utility associated with link formation varies from individual to individual, and is sensitive to heterogeneous tastes and preferences. Jackson and Rogers readily concede that "in many contexts, the utility would instead be network - and degree - dependent.

[13] Of the 10^4 trials, 9860 value of r fell within the range of 0 to 25. We omit outliers in figure 3.

[14] Further work needs to be done to see if this holds true in general.

[15] The posterior is equal to the likelihood when you use a uniform prior.

[16] We were able to compare the CDF of the simulated in-degree distribution with the CDF of the in-degree distribution for the actual data

It might be that nodes benefit from indirect connections, and thus might be more willing to link to nodes that have larger degrees" (Jackson and Rogers, 2007). For example, consider a network of high-school relationships like the ones studied in the Add Health Dataset. Students might receive a higher marginal utility from forming friendships with popular students as opposed to non-popular students, perhaps due to a "cool by association" effect.

Jackson and Rogers propose a simple extension of their model, which incorporates a degree dependent utility structure. Instead of letting the marginal utility be a random number, they set the marginal utility of linking with node j to

$$u_{ij}d_j - c$$

where u_{ij} is a random number between $[0, u]$ and $0 < c < u$. u_{ij} might represent the compatibility of node i and j (Jackson and Rogers, 2007). While this represents a more realistic depiction of reality, I believe it's possible to extend this logic further, and incorporate the methodology of discrete choice problems.

The decision to form a new link is an example of a binary choice problem. A node either chooses to form a link, or to not form a link. Let u_{ij} represent the utility to node i of linking with node j. If we can relate u_{ij} with characteristics of node i in some manner, say

$$u_{ij} = \beta s_n + \epsilon_n$$

such that a link is formed when $u_{ij} > 0$, we can model this process using a logit or probit. If we wanted to make the model even more sophisticated, we could model the search for parents and friends of parents with a multinomial choice model.

Another big assumption of the JR model is that once a node or link is formed, it remains intact forever. This assumption is critical to the analysis of Jackson and Rogers mean field approximation, but it reflects a unrealistic rigidity. In real world networks, links can be broken and nodes can die. The inclusion of these processes may significantly alter the dynamics of the JR model. It is worth pursuing this line of thought to see whether these processes affect the JR model's ability to replicate empirical regularities exhibited in real world social networks.

10. Appendix

MATLAB: RanGraph Code

```
function [G] = RanGraph(mr,mn,pr,pn,dim)
% Revised Feb 10th
% dim represents the dimension of the final G matrix, after everything has
% been initialized
% The inputted dim argument must exceed mr + mn + 1.
k = randint(1,1,[mr+mn+1,dim]);
G = zeros(dim,dim);


%Create the initial graph
for i = 1:k
    %Pick a random integer r between mr+mn and k-1. This indicates how many
    %neighbors i will have.

    r=randint(1,1,[mr+mn,k-1]);

    % Pick i's neighbors.
    % Generate a random permutation of {1,...,k-1}.
    % Identify the elements that are less than r+1. There should be
    % r of them. The location of those indices denotes the identity of
    % i's neighbors.

    h=randperm(k-1);
    g=h<(r+1);

    % Extract the identities of i's neighbors from the previous step. Go
    % to the ith row of G and place 1s in the appropriate locations.
    % We use the logical mask trick here.

    mask=[ones(1,k),zeros(1,dim-k)];
    mask(i)=0;
    G(i,logical(mask))=g;
end

% Now that we've generated the initial graph, we need to fill in the rest
% of the rows. We begin with row k+1 and loop till we hit the last row.

for i = k+1:dim
    % We need to find node i's parents.
    % We can the random perumtation trick used to generate the intial graph.
    % Instead of generating a random permutation of {1,...,k-1}, we now
    % generate a random permutation of {1,...,i-1}. We then identify the
    % elements with values less than mr+1 (this is equivalent to drawing mr
    % numbers at random from {1,...,i-1}).

    h=randperm(i-1);

    % Using the find command, we get the locations of the mr+1
```

```
parents= find(h<(mr+1));

% Parents now contains the identities of i's parents. For instance, if
% parents = [2,3,4] then i's parents are nodes 2, 3 and 4.

% Now we need to decide which parents i will link up with.
% Loop through the parents vector. For each parent, we "flip" a coin
% weighted (heads = pr, tails = 1—pr). If we get "heads" then i links
% up with the parent. Otherwise, i does not link up with its parent.

for j = 1:length(parents)
    heads=coinflip(pr);
    if heads == 1;
            G(i,parents(j))=1;
        else
            G(i,parents(j))=0;
    end
end

% Note that i can also link up with its parents' neighbors. We need to
% identify the union of i's parents' neighborhoods.

% Let nghbdparents represent the set of all nodes that belong to the
% neighborhood of at least one of i's parents. We initialize
% this vector to a row of zeros.

nghbdparents=zeros(1,dim);

% Sum the row vectors of i's parents. (e.g. if i's parents are 1,3,4.
% Take the sum of rows 1,3,4 from G.) Note that every nonzero element
% in the sum vector represents an element of nghbdparents.

for j=1:length(parents)
    nghbdparents=nghbdparents+G(parents(j),:) ;
end

% Use the logical command to reduce nghbdparents to a row vector
% entirely of 1s and 0s.
% (i.e. nghbdparents = [0,1,0,0,1]. So nodes 2 and 5 belong to the
% nghbd of at least one of i's parents)

nghbdparents = logical(nghbdparents);

% Use the find command to extract the identity the nodes within
% (i.e. find(nghbdparents) = find([0,1,0,0,1]) returns 2,5)

indexnghbdparents = find(nghbdparents);

% Now that we've identified nghbdparents, we need to select mn of them.
% Use the random permutation trick.

% Example:
```

```
    % Suppose indexnghbdparents = [2,4,5,7] and dim = 9.
    % We take a random permutation of {1,2,3,4}, say [2,1,4,3]
    % So l=[2,1,4,3] and that mn = 2.
    %    m=[1,1,0,0]
    % nonzeroindm = [1,2]
    % So my candidates to link up with are [2,4]

    l=randperm(length(indexnghbdparents));
    m=l<(mn+1);
    nonzeroindm = find(m);

    % Once we've identified the mn candidates, we loop through our list,
    % and flip a weighted coin each time (pn, 1— pn). Whichever ones turn
    % up heads, we allow i to link up with. Whichever ones don't, we leave
    % alone.
    for j=1:length(nonzeroindm)
        heads=coinflip(pn);
        if heads==1 & G(i,indexnghbdparents(j))==0;
            G(i,indexnghbdparents(j))=1;
        elseif heads==1 & G(i,indexnghbdparents(j))==1;
            G(i,indexnghbdparents(j))=1;
        elseif heads==0 & G(i,indexnghbdparents(j))==0;
            G(i,indexnghbdparents(j))=0;
        elseif heads==0 & G(i,indexnghbdparents(j))==1;
            G(i,indexnghbdparents(j))=1;
        end
    end
    G;
end
```

MATLAB: Degree Distribution Code

```
function [CDF] = DegreeDistribution( G )
% Matthew Hom
% Last edited: Feb 9, 2012
% Purpose: Given a G matrix, this program will generate the corresponding
% degree distribution.


% Find out how many nodes there are in the final neighborhood
l = length(G);

% Sum up the columns of G. The ith element in the InDeg vector represents
% the indegree of the ith node.
InDeg = sum(G);

% Turn the in degree vector into a column vector. Add one to the indegree
% of each element (this will come into play when using the accumarray
% command).
InDeg = (InDeg+1)';

% Use the accumarray command. This will create a column vector of
```

```
% frequencies. The ith element of the degdist vector represents the number
% of nodes with indegree equal to i—1.

% e.g. InDeg = [1,2,3,4,4]. Running accumarray(InDeg,1), I  get [1,1,1,2].

% The accumarray assigns a value of 0 to any number which does not appear
% in the input vector.

% e.g. InDeg = [1,2,5,4,4,5]. accumarray(InDeg,1) = [1,1,0,2,2]

DegDist = accumarray(InDeg,1);

% Find the maximum value in the InDeg vector to obtain an upperbound on the
% length of the DegDist and CDF vector. To be precise, there exists at
% least on element with indegree equal to m—1 in the original G matrix.
m = max(InDeg)

% Create the CDF vector. Initialize to zero. Set length equal to l. There
% are at maximum l distinct values that can occur for the CDF vector.
CDF = zeros(l,1);

% Recursively generate the CDF vector. Initialize the first value.
% Set CDF(1) = F(0) = # of elements with indeg 0/ # number of elements G.
% Use a for loop to fill in the rest of the vector. Note the for loop
% ranges from 1 to m—1 and not from 1 to m. We omit CDF(m+1) because we
% ultimately want to work with log(1—CDF) and log(1—1) = log(0) = — infty.

% e.g. CDF (2) = F(2—1) = F(1) = CDF(1) + DegDist(2)/length(G).
CDF(1) = DegDist(1)/length(G);
for i=1:m—1
CDF(i+1)=CDF(i)+DegDist(i+1)/length(G);
end

CDF

% Latex Code to Generate Table Formatted Degree Distribution Vector
fout = fopen('DegDist.tex','w','n');
fprintf(fout,'\\documentclass[12pt]{article} \n');
fprintf(fout,'\\usepackage{fullpage} \n');
fprintf(fout,'\\begin{document} \n');
fprintf(fout,'\\begin{table} \n');
fprintf(fout,'\\centering \n');
fprintf(fout,'\\begin{tabular}{c c c} \n');
fprintf(fout,'Degree & Number of Operators & CDF\\\\ \n');
for j = 0:m—1
    fprintf(fout,'%d & \t%d & \t%f \\\\ \n',j,DegDist(j+1),CDF(j+1));
end
fprintf(fout,'\\end{tabular} \n');
fprintf(fout,'\\end{table} \n');
fprintf(fout,'\\end{document} \n');
fclose(fout);

end
```

MATLAB: Simulated Minimum Distance Code (Grid Approach)

```
function [ratio,H] = SimulMOMGrid(m,sizenetwork)
Data = [3, 14, 19, 21, 22, 22 , 24 ,26, 29, 32, 34, 36, 37, 38, 38, 38,...
    39, 39, 39, 39, 40, 41, 42, 42, 42, 43, 43, 44,0,0,0,0,0,0,0,0,0,0,...
    0,0,0,0,0,0]'/44
% Objective: Estimate the ratio of random to network based
% friendships using simulated method of moments.

% Input: Actual degree distribution of real world network
% Output: r = ratio of random to network based friendships

% m denotes mesh size. 0 < m < 1.

% evaluate the RanGraphFunction using different combinations of pr,
% pn

% mr = mn = 1; pr, pn are allowed to vary.


r = [0:m:1];
n = [0:m:1];
H = zeros(length(r),length(n))

% Declare an H_pr_pn matrix of dimension length(pr) by length(pn).
% Idea will be to find the minimum value in the H matrix.

for k = 1: length(r)
    for j=1:length(n)
    s = RandStream('mcg16807','Seed',0)
    RandStream.setDefaultStream(s)

        eval(sprintf('[F_%i_%i] = AverageRanGraph(%f,%f)'';',k,...
            j,r(k),n(j)));
        eval(sprintf('h_%i_%i = Data-F_%i_%i;',k,j,...
            k, j));
        eval(sprintf('H(%i,%i) = h_%i_%i''*h_%i_%i;',k,j,k,...
            j,k,j));

    end
end

H

% Find the location of the smallest element in the H matrix.
% Compute r from the ratio of pr/pn
[p,q] = find(H==min(H(:)))
ratio = r(p)/n(q)


end
```

MATLAB: Simulated Minimum Distance Code (Numerical Optimization Approach)

```
function [] = SimulMoM(a)
% Define the CDF vector
CDF = [3, 14, 19, 21, 22, 22 , 24 ,26, 29, 32, 34, 36, 37, 38, 38, 38,...
    39, 39, 39, 39, 40, 41, 42, 42, 42, 43, 43, 44]'/44;

s = 'fminbnd(@(a) '
% Define the minimization problem
eval(sprintf('s = strcat(s,''( %f-g(%d,a,3.5))^2 '')',CDF(1),0));
for i = 2:28
eval(sprintf('s = strcat(s,''+ (%f-g(%d,a,3.5))^2 '')',CDF(i),i-1));
end
eval(sprintf('s = strcat(s,'', %f,%f)'' )',0,1))
eval(s)

end
```

MATLAB: Euclidian Distance Code

```
function [sd] = SimulatedDistance(G)
Data = [3, 14, 19, 21, 22, 22 , 24 ,26, 29, 32, 34, 36, 37, 38, 38, 38,...
    39, 39, 39, 39, 40, 41, 42, 42, 42, 43, 43, 44,0,0,0,0,0,0,0,0,0,0,...
    0,0,0,0,0,0]'/44;



% Find out how many nodes there are in the final neighborhood
l = length(G);

% Sum up the columns of G. The ith element in the InDeg vector represents
% the indegree of the ith node.
InDeg = sum(G);

% Turn the in degree vector into a column vector. Add one to the indegree
% of each element (this will come into play when using the accumarray
% command).
InDeg = (InDeg+1)';

% Use the accumarray command. This will create a column vector of
% frequencies. The ith element of the degdist vector represents the number
% of nodes with indegree equal to i-1.

% e.g. InDeg = [1,2,3,4,4]. Running accumarray(InDeg,1), I  get [1,1,1,2].

% The accumarray assigns a value of 0 to any number which does not appear
% in the input vector.

% e.g. InDeg = [1,2,5,4,4,5]. accumarray(InDeg,1) = [1,1,0,2,2]

DegDist = accumarray(InDeg,1);
```

```
% Find the maximum value in the InDeg vector to obtain an upperbound on the
% length of the DegDist and CDF vector. To be precise, there exists at
% least on element with indegree equal to m-1 in the original G matrix.
m = max(InDeg);

% Create the CDF vector. Initialize to zero. Set length equal to l. There
% are at maximum l distinct values that can occur for the CDF vector.
CDF = zeros(l,1);

% Recursively generate the CDF vector. Initialize the first value.
% Set CDF(1) = F(0) = # of elements with indeg 0/ # number of elements G.
% Use a for loop to fill in the rest of the vector. Note the for loop
% ranges from 1 to m-1 and not from 1 to m. We omit CDF(m+1) because we
% ultimately want to work with log(1-CDF) and log(1-1) = log(0) = - infty.

% e.g. CDF (2) = F(2-1) = F(1) = CDF(1) + DegDist(2)/length(G).
CDF(1) = DegDist(1)/length(G);
for i=1:m-1;
CDF(i+1)=CDF(i)+DegDist(i+1)/length(G);
end

sd = [CDF-Data]'*[CDF-Data];
CDF;
end
```

MATLAB: Rejection Sampler Code

```
function [ParameterArray,PR,PN,MR,MN] = RejectionSamplerWithSampleSizeVersion2(mrmax,mnmax,...
    epsilon,draws)

%% Set the seed
seed = 1; rand('state',seed); randn('state',seed);

%% The parameter array stores all the (pr,pn) pairs used in the rejection
%  sampler.
ParameterArray = zeros(draws,4);
ParameterArray(1,1) = rand;
ParameterArray(1,2) = rand;
ParameterArray(1,3) = randi(mrmax,1);
ParameterArray(1,4) = randi(mnmax,1);
%% Initialize the Parameter Array
while SimulatedDistance(RanGraph(ParameterArray(1,3),ParameterArray(1,4),...
        ParameterArray(1,1),ParameterArray(1,2),44)) > epsilon
        ParameterArray(1,1) = rand;
        ParameterArray(1,2) = rand;
        ParameterArray(1,3) = randi(mrmax,1);
        ParameterArray(1,4) = randi(mnmax,1);
end
%% Run the rejection sampler

% Initialize i = 2
```

```
i=2;

while i ≤ draws;

% Draw new values for the theta parameters.
pr = rand;
pn = rand;
mr = randi(mrmax,1);
mn = randi(mnmax,1);



% Now compute the distance between the CDF for G and compare with the CDF
% for the actual data. If the simulated distance is smaller than epsilon,
% we accept, and (mr,mn,pr,pn) enters our set of possible parameter values.
% If not, we reject (mr,mn,pr,pn)


    if SimulatedDistance(RanGraph(mr,mn,pr,pn,44)) ≤ epsilon
        ParameterArray(i,1) = pr;
        ParameterArray(i,2) = pn;
        ParameterArray(i,3) = mr;
        ParameterArray(i,4) = mn;
        i=i+1;
    else
       i = i;
    end
end

%% Graph the results
x = [0:.05:1];
PR = ParameterArray(:,1)';
PN = ParameterArray(:,2)';
MR = ParameterArray(:,3)';
MN = ParameterArray(:,4)';

subplot(4,1,1),hist(PR,x),xlabel('pr'),ylabel('Frequency')
subplot(4,1,2),hist(PN,x),xlabel('pn'),ylabel('Frequency')
subplot(4,1,3),hist(MR,x),xlabel('mr'),ylabel('Frequency')
subplot(4,1,4),hist(MN,x),xlabel('mn'),ylabel('Frequency')

end
```

MATLAB: Metropolis Algorithm with Multivariate Normal Proposal

```
function [ prob ] = Metropolis( T,epsilon,sigma1,sigma2,sigma_1,sigma_2 )
% T = number of iterations

%% Intialize the Metropolis Sampler
gamma = zeros(2,T)';
seed = 1; rand('state',seed); randn('state',seed); %set the random seed
gamma(1,1) = 1;
```

```
gamma(1,2) = 1;
mu = [0,0];
SIGMA = [sigma1, 0; 0, sigma2];
SIGMA2 = [sigma_1,0;0, sigma_2];


%% Start Sampling
t = 1;
while t < T % Iterate until we have T samples
    t = t+1;
    % Propose a new value for pr and pn using a multivariate normal
    % proposal density

    [disturb] = mvnrnd(mu,SIGMA);
    [gamma_star] = gamma(t-1,:)+disturb;

    [pr_star] = inverselogit(gamma_star(1));
    [pn_star] = inverselogit(gamma_star(2));
    [pr] = inverselogit(gamma(t-1,1));
    [pn] = inverselogit(gamma(t-1,2));

    % Calculate the acceptance ratio
    if  SimulatedDistance(RanGraph(1,1,pr_star,pn_star,44)) <= epsilon;
        weightgamma_star = 1;
    else
        weightgamma_star = 0;
    end

    if  SimulatedDistance(RanGraph(1,1,pr,pn,44)) <= epsilon;
        weightgamma = 1;
    else
        weightgamma = 0;
    end

    alpha = weightgamma_star*...
        mvnpdf(gamma_star,mu,SIGMA2)/mvnpdf(gamma(t-1,:),mu,SIGMA2);

%    alpha = min([1 ...
%         (weightgamma_star*mvnpdf(disturb))/(weightgamma*mvnpdf(-disturb))]);
    u = rand;
    if u < alpha
        gamma(t,:) = gamma_star;
    else
        gamma(t,:) = gamma(t-1,:);
    end

end

%% Display histogram of samples
prob = exp(gamma)./[1+exp(gamma)];
PR = prob(:,1)';
PN = prob(:,2)';
```

```
stairs(PR);
x = [0:.025:1];
subplot(2,2,1),stairs(PR),xlabel('t'),ylabel('PR')
subplot(2,2,2),stairs(PN),xlabel('t'),ylabel('PN')
subplot(2,2,3),hist(PR,x),xlabel('pr'),ylabel('Frequency')
subplot(2,2,4),hist(PN,x),xlabel('pn'),ylabel('Frequency')



end
```

MATLAB: Metropolis Hastings Algorithm with Beta Proposal

```
function [ theta ] = MetropolisBetaProposal( T,epsilon,mrmax,mnmax)
% T = number of iterations
%% Intialize the Metropolis Sampler
theta = zeros(4,T)';
seed = 1; rand('state',seed); randn('state',seed); %set the random seed
theta(1,1) = .5;
theta(1,2) = .5;
theta(1,3) = randi(mrmax,1);
theta(1,4) = randi(mnmax,1);

%% Find initial theta values which yield a simulated distance smaller ...
%  than epsilon

while SimulatedDistance(RanGraph(theta(1,3),theta(1,4),...
        theta(1,1),theta(1,2),44)) > epsilon
        theta(1,1) = rand;
        theta(1,2) = rand;
    end
%% Start Sampling
t = 1;
while t < T % Iterate until we have T samples
    t = t+1;
    % Propose a new value for pr and pn using an
    % evolving beta proposal distribution

    [pr] = theta(t-1,1);
    [pn] = theta(t-1,2);
    [pr_star] = betarnd(2*pr/(1-pr),2);
    [pn_star] = betarnd(2*pn/(1-pn),2);
    [mr] = theta(t-1,3);
    [mn] = theta(t-1,4);
    [mr_star] = randi(mrmax,1);
    [mn_star] = randi(mnmax,1);

    theta_star = [pr_star, pn_star,mr_star,mn_star];


    % Calculate the acceptance ratio
    if  SimulatedDistance(RanGraph(mr_star,mn_star,pr_star,pn_star,44)) ≤ epsilon;
```

```
        weighttheta_star = 1;
    else
        weighttheta_star = 0;
    end

%     if  SimulatedDistance(RanGraph(mr,mn,pr,pn,44)) ≤ epsilon;
%         weighttheta = 1;
%     else
%         weighttheta = 0;
%     end

    alpha = min(1,[weighttheta_star*betapdf(pr,2*pr_star/(1-pr_star),2)...
        *betapdf(pn,2*pn_star/(1-pn_star),2)]/[betapdf(pr_star,2*pr/(1-pr),2)...
        *betapdf(pn_star,2*pn/(1-pn),2)]);


    u = rand;
    if u < alpha
        theta(t,:) = theta_star;
    else
        theta(t,:) = theta(t-1,:);
    end

end

%% Display histogram of samples

PR = theta(:,1)';
PN = theta(:,2)';
MR = theta(:,3)';
MN = theta(:,4)';
stairs(PR);
x = [0:.025:1];
subplot(4,2,1),stairs(PR),xlabel('t'),ylabel('PR')
subplot(4,2,2),stairs(PN),xlabel('t'),ylabel('PN')
subplot(4,2,3),hist(PR,x),xlabel('pr'),ylabel('Frequency')
subplot(4,2,4),hist(PN,x),xlabel('pn'),ylabel('Frequency')
subplot(4,2,5),stairs(MR),xlabel('t'),ylabel('mr')
subplot(4,2,6),stairs(MN),xlabel('t'),ylabel('mn')
subplot(4,2,7),hist(MR),xlabel('mr'),ylabel('Frequency')
subplot(4,2,8),hist(MN),xlabel('mn'),ylabel('Frequency')




end
```

MATLAB: Average Ran Graph Function

```
function [F] = AverageRanGraph(pr,pn)
% m = mesh size
% t = number of times run RanGraph
```

```
% Run the RanGraph function repeatedly. Compute the cumulative distribution
% over multiple runs, and set that equal to F_k_j. (F_k_j represents the
% "avg CDF" of a sequence of randomly generated graphs with the same pr, pn
% values.

% DegDist is a column vector of length 44. DegDist(1) = # of elements with
% degree 0, DegDist(i) = # of elements with degree i — 1. DegDist is
% computed from a total of t runs of the RanGraph function. I add up the
% DegDist vector over all the different runs, and divide every element
% by 44*t. I then compute out the CDF vector.

DegDist = zeros(44,1);
for i = 1: 50
    eval(sprintf('G_%i = RanGraph(1,1,pr,pn,44);',i));
    eval(sprintf('DegDist_%i = ModifiedDegreeDistribution(G_%i);',i,...
            i));
    eval(sprintf('DegDist = DegDist + DegDist_%i;',i));

end
CDF(1) = DegDist(1)/(44*50);
for i=1:43
CDF(i+1)=CDF(i)+DegDist(i+1)/(44*50);
end
F = CDF;
F

end
```

MATLAB: Transitive Triple Function

```
function [ fractt] = TransTrip(G)
l = length(G);
H = nchoosek(1:1:length(G),3);
denominator = 0;
numerator = 0;

for a=1:length(H)
i = H(a,1);
j = H(a,2);
k = H(a,3);

%ijk, ikj, jik, jki, kij, kji

denominator = denominator + G(i,j)*G(j,k) + G(i,k)*G(k,j)+G(j,i)*G(i,k)+...
    G(j,k)*G(k,i) + G(k,i)*G(i,j)+G(k,j)*G(j,i);
numerator = numerator + G(i,j)*G(j,k)*G(i,k) + G(i,k)*G(k,j)*G(i,j)+...
    G(j,i)*G(i,k)*G(j,k)+ G(j,k)*G(k,i)*G(j,i) + G(k,i)*G(i,j)*G(k,j)+...
    G(k,j)*G(j,i)*G(k,i);
end

numerator;
```

```
denominator;
fractt = numerator/denominator;

end
```

References

[1] N. A. Christakis, J. H. Fowler, G. W. Imbens, and K. Kalyanaraman, "An Empirical Model for Strategic Network Formation," *NBER Working Paper Series*, vol. w16039, 2010.

[2] S. A. Sisson and Y. Fan, *Likelihood-free Markov chain Monte Carlo*. Chapman and Hall/CRC Press, 2011, pp. 319–341.

[3] B. Walsh, "Markov Chain Monte Carlo and Gibbs Sampling," April 2004, Lecture Notes for EEB 581.

[4] M. O. Jackson and B. W. Rogers, "Meeting Strangers and Friends of Friends: How Random Are Social Networks?" *American Economic Review*, vol. 97, no. 3, pp. 890–915, June 2007.

[5] M. O. Jackson, "Research Opportunities in Social and Economic Networks," September 2010.

[6] M. Jackson, *Social and Economic Networks*. Princeton University Press, 2008.

[7] G. Casella and R. L. Berger, *Statistical Inference*, 2nd ed. Duxbury Press, June 2001.