

Jeffrey Goett

Final Research Paper, Fall 2003

Professor Madey

19 December 2003

Abstract:

Recent observations by physicists have led to new theories about the mechanisms controlling the growth of networks. Albert-László Barabási and his collaborators have also determined that networks can be mathematically transformed into a Boson Gas for purposes of modeling, leading to new ways of characterizing states of a network. The Open Source Software Group at Notre Dame found that open source software community at sourceforge.net also obeys the rules of network formation. Computer simulations have helped study this community. Recently these simulations have been converted from Swarm to Repast. To ensure a proper migration, the group went through a series of computer iterations to dock the Repast migration to the original Swarm programs. Results of this process have almost docked three of the four programs on the variables of developer count vs. time, project count vs. time, and degree distribution. Both simulations produce distributions with the same average, but with different standard deviations, indicating subtle differences remaining between the Swarm and Repast implementations.

Introduction:

Many different phenomena that occur in the world can be modeled as a network, a group of nodes with links between them symbolizing some sort of relationship between those two nodes. Movie actors can be fit into this theory: links connect two actors who have co-starred in the same movie. Social relationships between people can be modeled as a network: links connect acquaintances. Manmade systems, such as the power grid in the southwest or the World Wide Web can be seen as networks. Interestingly, networks seen in many different domains all display a power law degree distribution. Each node has a degree equivalent to its number of links. Many different networks all have degree distributions that follow the formula:

$$P(k) = k^{-\gamma}$$

k : *degree*

γ : *constant*

Earlier models of network formation cannot explain these observations. Paul Erdős and Alfred Rényi suggested a model where nodes randomly connected with each other. Such a model, however, produces an exponential degree distribution. Another model, the Duncan Watts and Steven Strogatz model, begins with each node connected to its nearest neighbor. In particular systems, the physical meaning of nearest neighbor depends on the circumstances. In a generalized graph, it means the node ordered before it and after it. Next, a few nodes randomly disconnect and then reconnect to random nodes. Doing such produces a small world effect: the minimum number of jumps between any two random nodes is much less than the total number of nodes in the system. The degree distribution of this graph, however, does not follow a power law.

Albert-László Barabási and Réka Albert have suggested a model of network formation that does explain the power law distribution and create networks with small world properties. Networks begin with a few nodes. As time progresses, more nodes enter the system and connect to the networks. The probability that a new node will connect to a particular node in the graph is proportional to that node's degree. Thus, highly connected nodes will tend to gain new links while less connected nodes tend not to. Over time this results in a "rich get richer" phenomenon: there are a few nodes very highly connected, known as the hubs. There are more nodes that are moderately connected. Finally, there are lots of nodes will low connectivity. As this suggests, such a model creates networks that follow a degree distribution.

In such a network the older nodes tend to gain the most links. Younger nodes have fewer opportunities to gain links, hence this model prevents them from having the most links. It has been observed empirically, however, that younger nodes sometimes will quickly gain a large number of links. The search engine Google illustrates such a "young upstart" phenomenon. Google's superior performance over other search engines such as Yahoo and Excite helped it quickly become the target of more links on the web than these other more established search engines.

Barabási and Albert modified their theory to account for the role of differing “utilities” in the node connection process. To do so, each node is given a relative fitness, meant to capture its utility to the entire system in comparison with other nodes. Now, the probability that a node will gain a link is proportional to its degree and its fitness:

$$\frac{\partial k_i}{\partial t} = \frac{n_i k_i}{K}$$

where

$$K = \sum_{j=1}^l n_j k_j$$

k : degree

Such a model can still create a network with a power law degree distribution but also allows for “young upstart” nodes to enter the system and quickly become highly connected.

Ginestra Bianconi and Albert-László Barabási have also modeled such networks into as gases. Different states this gas can be in has physical significance in the corresponding network. Hence, by looking at a network’s corresponding gaseous state, one can characterize significant properties of the network’s degree distribution.

A Boson gas is a gas with discrete energy levels that follows the Boson, as opposed to the Boltzman, energy level distribution. Normally when speaking about gases, the number of molecules in each state follows the Boltzman distribution:

$$P(E) = C e^{-(E/KT)}$$

In quantum physics, boson particles have a certain affinity for each other, based on their quantum numbers and the general rules of quantum mechanics. Thus, they follow a different energy level distribution:

$$P(E) = \frac{1}{e^{-(E/KT)} - 1}$$

A given network with fitness can be mapped to a gas with energy levels in the following way. Each fitness maps to a corresponding energy level:

$$\varepsilon_i = -\frac{1}{\beta} \log n_i$$

ε : energyLevel

n : fitness

There is a particle in this energy level for every link to a node with the corresponding fitness. If there are 5 nodes of fitness 3 with degrees 4, 3, 5, 6, and 4, then there are 22 particles in the energy level corresponding to fitness 3.

If one maps the degree distribution of such a network into the gas, it is found that the particles normally follow a Bose distribution. With certain properties, however, the math transformation will not work. In the network, this occurs when a node gains and maintains a finite fraction of the links in the network. In the corresponding gas, Bose-Einstein condensation is occurring: a constant fraction of the particles are located in the lowest energy level (mapping to the highest fitness). This analogue helps to point an important state of a network, namely one where a single node maintains a fraction of the links over time. (There probably is quite a bit more mathematical insight here, but I do not understand it.)

Method/Process

The Open Source Software Group at the University of Notre Dame studies the Open Source Software community in order to better understand the dynamics of a movement that is quickly changing the landscape of the computer world. One feature of this project involves studying the formation of the social network at sourceforge.net, an Open Source Software Community. This community can be modeled as a network in two different ways. One way has developers modeled as nodes. Links exist between developers working on the same project. The second way has the projects as the nodes: two projects are linked if they have at least one developer in common.

Yongqin Gao's empirical studies of sourceforge found both instances of a network to follow a power law degree distribution. Taking it a step further, simulations of the sourceforge.net community were written in the agent based modeling language Swarm. These programs applied Barabási and Albert's network formation theory to the sourceforge.net community. Four slightly different models were implemented. All models followed the same procedure. Each day, a given number of developers would join the sourceforge.net community. They would then join one project of their choosing. Also, each day, developers already in the community could do one of four actions: remain idle, form and join a new project, join another project, or abandon one of the

projects they were working on. Each model varied, however, on how a developer picked a project to join. In the “ER” model, a developer picked a random project: there was no “preferential attachment” towards highly connected. In the “OSS” model, there was preferential attachment: the chance that a project will be connected to is proportional to that project’s degree. In the constant fitness model, each node is assigned a random fitness. The probability that a developer connects to a project is proportional to the product of the project’s degree and fitness. Similarly, in the dynamic fitness model, a project’s chance of being connected to is proportional to the product of its fitness and degree. In this model, however, a project’s fitness is decremented every month by one, unless its fitness is already one. In software development community like sourceforge.org, a project’s originality and newness plays a large part in its appeal to developers. This decreasing fitness models a project’s fading appeal with time.

Recently, the researchers at the University of Chicago have ported the Swarm package into a Java, creating a package name Repast. Repast allows users to benefit from the ease of use of Java as a programming language. To take advantage of the benefits of working with a pure implementation in Java. Jin Xu rewrote the four models in Repast. Whenever doing a port, one must make certain that migration has been done cleanly without changing the results of the simulation. To do so, one undergoes a docking process, whereby results of corresponding simulations are compared. Programmers search for the source of discrepancies between simulations and correct them. Doing so, small differences between respective software packages can be revealed. Earlier this year, the docking process between the Swarm and Repast simulations discovered that Swarms time counter begins at $t=0$, while Repast’s time counter had been set to one.

After this past summer, the two simulations remained undocked: one of the goals of OSS group this semester was to dock each model’s implementation in Repast with its corresponding implementation in Swarm.

In order to dock two simulations, many iterations of each simulation had to be run. Hence, the first task of the semester was to maximize the speed and ease with which the many iterations could be run. Originally, the OSS group wrote the Swarm and Repast simulations with a GUI. A GUI, however, slows down a simulation. Additionally, with

Swarm and Repast if a GUI is closed, the computer kills the simulation. For ease of use, the group wanted to be able to launch a sequence of simulations to be run, logout of the computer, and let the simulation server run the simulations for a few days. Hence, one of the first talks for this semester involved turning off the GUI. In Repast, doing this is relatively easy, one just needs to give add the “-b” flag to the command line when executing program, telling the Repast simulation engine to run the simulation in batch mode. The Repast program, however, did instantiate an object that needed an X11 client to run. If the X11 client closed, the simulation was killed. Jeff commended out the instantiation of this object, which was only needed for the GUI.

In Swarm, turning off the GUI involved more modification. The upper level class, StartOssSwarm, which coordinates the instantiation and execution of simulation, had to be modified to turn off the GUI. In addition, calls in the class OssModelSwarm to GUI objects had to be commented out.

Statements and queries to the SQL database, where developer and project data is stored, is another speed bottleneck in these simulations. In order to communicate with the database, the program uses an object from the class connection and an object from the class statement. Originally, each time the simulation had to communicate to the database, the program created a new connection and statement object. The computer code was changed: only one connection and statement object was created for each simulation. It was declared a public variable. Pointers to these variables were passed to methods in the simulation that needed them. This was done in both Swarm and Repast.

Finally, the Swarm simulations were originally written to use Swarm’s built in random number generator. Jeff noted this semester, however, that the program was not passing the seed to the random number generator. In addition, the group desired to have both the Swarm and the Repast simulations running with the same random number generator. Thus, Jeff modified the Swarm simulations to use Colt, the random number generator developed at CERN. He modified the Swarm programs so that a seed passed to the program through the command line would be used to initialize the random number generator.

Originally, data analysis on each simulation would give four different sets of output data: developer count vs. time, project count vs. time, degree distribution, and

diameter. Over the course of the semester, however, two problems emerged with the diameter calculations. The diameter calculating algorithm had to find developers working on the same project. Doing such a search has a computational order of n^2 . With 40,000 developers created over the course of the simulation, this algorithm would have taken many days to calculate all the diameter statistics needed. In addition, the diameter algorithm used an approximation that may not hold for the networks being studied:

$$d = \frac{\ln(N/Z_1)}{\ln(Z_2/Z_1)} + 1$$

where

$$N \gg Z_1$$

$$Z_2 \gg Z_1$$

In our network simulations, however, $Z_2 \sim Z_1$.

Thus the diameter result was temporarily dropped. For the docking this semester, only 3 data outputs were compared.

With these modifications in place, iterations of the Swarm and Repast simulations were run. After each iteration, the results of corresponding models were compared. The results of the last set of iterations will be presented in the next section. For the first few iterations, the primary goal was to dock the number of developers over time. The creation of developers did not depend on any other process in the simulation. The number of projects, on the other hand, depended on the number of developers. Similarly, the degree distribution is affected both by the number of developers and the number of projects. Thus, fixing any discrepancies in developer vs. time between Swarm and Repast simulations proved relatively easy. After implementations were docked on this variable, one can turn attention towards docking on the results projects vs. time and degree distribution.

Results

Graphs comparing the data output for the Swarm and Repast implementations of each model are presented in the Appendix (color version can be viewed at <http://www.nd.edu/~jgoett/docking/>). Visually, all appear to match up very well, with the

exception of the dynamic fitness project vs. time, where the Repast and Swarm project counts diverge as time approaches 24 months.

It is noted, however, that at each month the developer and projects counts have a distribution. This is expected: each iteration of the model used a different seed for the random number generator. The mean of the Swarm and Repast appear to dock. Consistently, however, the Repast distribution has a larger deviation about its mean than the Swarm distribution. Further, corresponding iterations in Swarm and Repast were given the same seed. Even with the same seed (and hence the same distribution of random numbers), the output of corresponding Swarm and Repast simulations are slightly different. There is still some discrepancy between the two programs. Whether this is significant still needs to be determined.

Conclusion

Over the course of this semester, important progress has been made in the docking and optimization of the Swarm and Repast programs. All simulations now can be run in a batch mode, helping to speed up simulation time. All simulations minimize the number of database connection and statement objects created. In addition, where possible, commands to search for the highest project or developer id, an expensive process, were replaced by calls to database sequences coordinating the assignment of developer and project ids. These steps all helped optimize the programs. Still, however, the Swarm implementation of the ER model runs slowly.

After several iterations, progress has been made in docking corresponding Swarm and Repast models. Resulting degree distributions match between models. For the developer count vs. time and the project count vs. time, the average of Swarm and Repast distributions match. Distributions created by the Repast implementation, however, consistently have a larger standard deviation than distributions created by Swarm implementations. One notable exception is the project vs. time results for the dynamic fitness model: the project counts for Swarm and Repast diverge as time approaches 24 months.