An Investigation of Network Fractality[*]

Vishesh Gupta[†], Cayman Simpson[‡], Bharad Raghavan[§]

8 December 2014

Abstract

In the last decade many generative models that claim to accurately emulate real-world, scale-free networks have emerged. Many of these are constructed through some notion of self-similarity. However, validation of these models generally occurs by a well-known trifecta of degree distribution, clustering coefficients, and pagerank scores. Every published model shows charts of success under these measurements which makes it appear they can be applied to any graph with success. There currently exists no way to quantifiably and more objectively determine the accuracy of these models with respect to the original network they attempt to emulate.

We propose using the box-counting method and community size distributions viewed over reductions to more thoroughly validate these fractal-like models. These methods allow us to probe the fractality of graphs by looking for self-similarity at multiple levels, giving additional insight into which models emulate the actual recursive generation of the network, not just the network properties themselves.

We found that these methods of analyses improve our understanding of the many multifractal network-generating models we scrutinized. Preliminary analysis on a diverse set of datasets confirm our hypothesis: certain models, which appear effective on a superficial level, fail to emulate the real network under these new techniques. These methods allow us to determine which fractal network-generating models is best suited for a given network.

1 Introduction

Within the last 10 years, multifractal graphs and scale-free networks have gained traction in the research world. The intuition of network fractality is simple: micro-level structures are also exhibited on a macro-level (or vice versa). This intuition seems to apply across many kinds of networks - biological, social, and systems [13]. The implications of this finding is significant: simple models based on a few initial parameters can infer much more about a network than previously thought possible. There is significant value of this kind of model in terms of extrapolation, simulation, null-model, and sampling and predictive analysis in the network and graph domains.

However, the fractal network domain is relatively new compared to other network domains, and there is little to no standard for validating the self-similarity of these models. With an increasing number of self-similar network-generation models, we found it difficult to determine the accuracy of these models with respect to the original network they attempt to emulate. Current methods simply state the degree distribution and clustering coefficient distribution, and leave it at that. At best, a model will cover a suite of metrics of on the original graph, but will not attempt to probe any kind of deeper structure.

That isn't to say there haven't been new tools developed alongside these models - analogies of the fractal box counting method have been developed[13], but never applied to validate a model, only to investigate a real world network. There have also been proposals of fractal skeletons [3] and bifurcation parameters and hurst exponents. We propose first that self-similar structure analysis requires the use of reductions on the network. Reductions are computed by partitioning the graph into boxes, then turning each box into a super node and forming links between super nodes based on whether the containing nodes have a link. Then, at each reduction level, we look at the fractal dimension and community sizes distribution. This allows for a deeper look at the structure of the networks. We hypothesize that these new methods will differentiate models that appear similar with respect to the basic metrics.

2 Summary of Current Literature

Because the fractality of scale-free networks has only recently come into focus, there is little academic literature in this domain. We are unable to cover all relevant research, as general fractal networks and their properties touch many areas of network and fractal research.

We summarize all the research relevant to our goal of analyzing fractal graph models that we have found in two parts: (2.1) current accepted models used to generate scale-free networks and (2.2) statistics proposed to describe and validate these fractal networks.

[*]code at: bitbucket.org/visheshg/fractal-graphs
[†]BS Candidate in CS, '16
[‡]BS Candidate in MCS '16
[§]Masters Candidate in CS, '16

2.1 Fractal Graph Models

In our analysis, we found three suitable accepted network generating models - Kronecker Graphs, Forest Fire Graphs and Hyperbolic Geometric Graphs, pioneered by Leskovec et al.[6], Leskovec et al. [7] and Krioukov et al. [5] respectively.

2.1.1 Kronecker Graphs

Leskovec et al. [6] designed and studied Kronecker Graphs [2]. These are scale-free graphs that closely model real-world networks and can be generated scalably, in O(|E|) time (|E| denoting the number of edges in a graph). The Kronecker product ($\otimes$) is a matrix operation where: $A \otimes B = \begin{pmatrix} a_{11}B & ... & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & ... & a_{mn}B \end{pmatrix}$. To generate a graph, take a small 'seed' adjacency matrix and repeatedly perform the kronecker product with itself. The resulting matrix will recursively present the original adjacency matrix, and the graph created from this adjacency matrix will display self-similarity and fractal properties.

Leskovec et al. devised an algorithm called KRONFIT [6] which estimates the parameters of a Kronecker graph. This algorithm uses maximum likelihood estimation in order to return the best set of parameters after permuting over various different parameters in the rows and columns of the seed matrix. KRONFIT is used to generate seed matrices for Kronecker graphs.

Leskovec et al. further devised another algorithm, KRONEM , that estimates the Kronecker parameters of real world networks [4]. KRONFIT tends to underestimate the seed matrix parameters when presented with real world networks, which generally have many nodes or edges missing. To address this, KRONEM takes in a real-world network and considers the graph to be part of a larger Kronecker graph which it has no knowledge of. By considering the missing nodes and edges as latent variables, KRONEM uses the Estimation-Maximization algorithm in order to estimate the seed parameters for the initiator matrix.

2.1.2 Forest Fire

Leskovec et al. [7] also pioneered a new fractal network-generating model called the Forest Fire model. This model takes two parameters, a forward burning probability p_f and a backward burning ratio p_b whose roles will be described below:

Consider a node v joining the network at time t ¿ 1, and let Gt be the graph constructed thus far. (G1 will consist of just a single node.) Node v forms outlinks to nodes in Gt according to the following process.

1. v first chooses an ambassador node w uniformly at random and forms a link to w.

2. We generate two random numbers, x and y, that are geometrically distributed with means $\frac{p_f}{1?p_f}$ and $\frac{p_b}{1?p_b}$, respectively. Node v selects x outlinks and y in-links of w incident to nodes that were not yet visited. Let $w1, w2, ..., wx + y$ denote the other ends of these selected links. If not enough in- or outlinks are available, v selects as many as it can.

3. v forms outlinks to $w1, w2, ..., wx + y$, and then applies step (ii) recursively to each of $w1, w2, ..., wx + y$. As the process continues, nodes cannot be visited a second time, preventing the construction from cycling.

Thus, the burning of links in the FF model begins at w, spreads to w_1, ... , $wx + y$, and proceeds recursively until it dies out.

This iterative and size-independent process of constructing the network results in several fractal properties: rich-get-richer attachment processes (scale-free degree distributions), densification power-law, and a shrinking diameter.

2.1.3 Hyper-Geometric Model

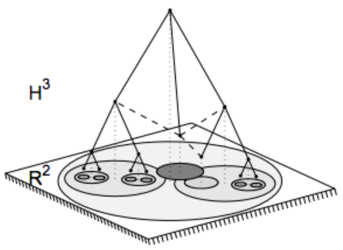

Figure 1: hyperbolic space with hierarchical structure

Kriokov et al. [5] developed a network-generating algorithm (HG) based upon mapping nodes to a Hyperbolic Geometric Space. HG generates edges between nodes in three steps. First, this model calculates the radius of the largest hyperbolic disk occupied by the nodes by mapping the nodes to a hyperbolic plane. Then, it creates two distributions (based on two attachment strength parameters) and samples two types of coordinates, radial and angular in a hyperbolic space per node. Finally, HG connects each node pair with a probability which is a function of the hyperbolic distance between the two nodes. The HG thus simulates fractality because the node coordinates are mapped and sampled in a tree-like (self-similar) fashion; they are added as sub-nodes to tree nodes whose hyperbolic distance is greater than the distance between the sub-nodes (Figure 1). In this way, the tree, and therefore the sampling distributions created, are hierarchical and self-similar on each level. Furthermore, the tests and validation measures attempted in this paper are promising enough to integrate HG in our research project. A more mathematically concise definition can be found in Appendix 8.2.

2.2 Graph Analysis

A primary motivation of this research is to investigate the fractality of these graphs models in relation to their real-world counterparts. To this end, we discovered two relatively recent techniques for discovering whether or not the graph is fractal, and what the fractal dimension should be.

Song et al. [12] developed the box-covering algorithm to calculate the fractal dimension of scale-free networks, analogous to the box counting method for mathematical fractals. Song reduces the box-covering algorithm into the well-known graph coloring problem in order to apply more scalable algorithms to calculate the fractal dimension of the graph. There are two ways of looking at this - first, the ”overall” box counting method suggests that the plot of number of boxes vs the box size should be a power law. Second, at each box size, the histogram of the box mass should also follow a power law.

The “fractality” is seen when the graph is reduced - after the box counting method is applied, each box is turned into a super node, and super nodes are connected if their nodes have a link between them. This results in a similarly structured graph, but with lower number of nodes and edges. It’s analogous to “zooming out” in a Mandelbrot Set visualization.

K.-I Goh et al. [3] explored skeletons, or communication kernels, of fractal networks. A skeleton is defined as the minimum spanning tree in priority order of the highest betweenness centrality edges. They found that these skeletons also exhibit fractal like properties if the original graph is a fractal graph, albeit with different fractal dimension.

When extended and applied to our network comparison analysis, both these methods allow us to not only simplify networks for examination through network reduction, but also help verify that the networks exhibit similar structures as you “zoom in” and “zoom out”. It is worth remarking that both the box-counting and skeleton analysis methods were published prior to the papers on KRONFIT, KRONEM, HG and FF. Therefore, it is notable that none of these algorithms were tested against these metrics. These two methods of validation institute stricter definitions and quantifiable metrics for what it means to be “scale-free” by establishing that many real-world networks (Section 4.2) are actually self-similar at many scales of reduction, at many box sizes and with respect to their corresponding skeletons and their respective reductions.

3 Hypothesis

In regards to the current models for scale-free graphs that we are analyzing, we hypothesize that the scale-free graphs generated by the Kronecker Graph model will better resemble real world networks than both the HG and FF models with respect to our datasets.

This belief is based upon the work of Leskovec et al’s paper [2], which validated the Kronecker graphs generated by KRONFIT across a variety of basic statistics (degree distribution, clustering coefficient distribution, shrinking diameter, node-triangle participation, etc). This hypothesis is also based upon the fact that both HG and FF were not extensively tested on most of the same statistics. Furthermore, we hypothesize that because the Kronecker model generates a fractal in the most literal sense, Kronecker graphs’ reductions and structure should be fractal when analyzed through box-counting and reductions.

However, we acknowledge that because there exists no serious investigation of definitions of fractal dimension and their invariance to renormalization, our research is more exploratory and investigative; it does not lend itself to prior bias. We hope that our tools and methods have the power to discern how well these models hold up under deeper structural analysis.

4 Methods

Our process followed the steps below:

1. Model Selection - we used the three models we discussed in the literature review, and here we explain why.
2. Data Selection - what kinds of graphs are being tested and fitted and why.
3. Analysis Process - what the analysis process looks like, which statistics we chose to look at, and why.
4. Implementations - which algorithms we had to implement and how we did so, explained for the purpose of being replicated by others.
5. Pipeline/Harness - a more technical discussion of what we ran and where all the statistics came from.

4.1 Model Selection and Fitting

The models we chose had to have parameters that we could model from a real graph and use to generate the equivalent "model graph." Furthermore, they had to be runnable on a laptop with a limited 4-5GB of RAM, and they had to complete in some reasonable amount of time (an absolute max cap at 10 hours, but we needed most things to be much shorter).

4.1.1 KRONFIT/KRONEM

Fitting the real world network to a Kronecker graph means, assuming our real-world network is very similar to a Kronecker graph, finding the seed matrix parameters that this graph would most likely be generated from. In our project, we fit all our graphs to a 2x2 seed matrix using both KRONFIT and KRONEM.

Because KRONEM was built to estimate the seed matrix for real-world networks (and not just Kronecker graphs), KRONEM was used to estimate 2x2 seed or initiator matrix for all our networks. But KRONEM requires an initial seed matrix as a starting point before it can use gradient descent and many iterations of Estimation-Maximization to arrive at the final seed matrix. We discovered that the initial seed matrix had a large impact on the final seed matrix value in KRONEM, which took an innumerable amount of iterations to reach convergence. Because we did not have the time to run KRONEM to convergence, we decided KRONEM would be run for a fixed number of iterations, and instead we would try and choose a initial seed matrix intelligently.

Due to the linear runtime of KRONFIT, which estimates a seed matrix for Kronecker graphs, we decided to run KRONFIT on each of our real-world networks for a large number of iterations, and use the resulting seed matrix as our the initial seed matrix for KRONEM, in hopes of getting a more reliable seed matrix in the end for our real world networks. KRONFIT, however, also requires an initial seed matrix. Instead of choosing a random seed matrix for KRONFIT, we opted to let the initial seed matrix for KRONFIT across all networks to be [0.9 0.7; 0.5 0.2]. This initial seed matrix was found by Leskovec et al [4] to be a reliable approximation for many Kronecker Graphs, thus prompting our selection.

Thus, we fit all our real world networks to a 2x2 seed matrix (which Kronecker Graphs are generated from), first by using KRONFIT to find a reliable starting point, and then applying KRONEM to get our final, more precise estimation.

4.1.2 FOREST FIRE

Unfortunately, there currently exists no way to fit the FF to a real-world network. In order to approximate it, we came up with a modified brute force algorithm.

We first examined the properties of the forest fire model (in Appendix 1). We put first priority on number of edges in the graph; we want the average degree to best emulate that of the real-world network. Because average degree experimentally seems to be constant with respect to the parameters pf and pb, we 'climbed up' the diagonal, keeping pf = pb until we found the average degree within .01% accuracy. Then, we looked within a +/- 10% of both pf and pb (at intervals of .02 for a total of 400 attempts) to find the most optimal pf and pb that best matched the clustering coefficient within that area. Because of limited computational resources, we only fit on those two parameters, but we found them experimentally sufficient as they modeled the diameter and overall degree distribution well.

We further optimized this process so that it is scalable. We noticed that, when generating FF networks, the average degree and clustering coefficient are independent of number of nodes the network is generating (Appendix 1). Therefore, if the graphs were too large to be computationally reasonable (>10K nodes), we trained and fit the parameters of the FF model on a graph of 1K nodes, as a near equivalent average degree and clustering coefficient result from the graph with the original node count. Additionally, we capped both pf and pb at 40% because larger burning probabilities resulted in infeasible computational time. With either pf and pb being greater than 40%, computation time increased exponentially; the "burning edge process" took longer to die out, so it typically created an excess of edges (the clustering coefficient would typically range from an unrealistic 80-99%).

It is also worthy to note that in order to compute the FF network for some of our larger graphs (>100K nodes), we modified the C++ SNAP implementation of the FF model and removed the time-limit for the network generation.

4.1.3 HYPERBOLIC GEOMETRIC

The HG model takes 4 parameters, the number of nodes to be generated, the average degree, the power-law coefficient α and the parameter γ representing a metric called "Temperature". The first three parameters can be determined easily enough, with the small caveat that the HG model accepts a minimum α value of 2. Temperature, or γ, is closely related to the clustering coefficient of the network. We used experimental results to relate them with the following function: $f(x) = 0$ if $x >= .6$ else $29.167x^2 - 34.167x + 10$.

4.2 Data Choice and Selection

We carefully selected the dataset which we use to validate KRONFIT, FF and HG. In collecting our dataset, we were careful to maintain a distinct portfolio, maintaining diversity in both network structure and their real-world manifestations. For example, we included biological networks like E.Coli, social networks, internet routers, Les Miserables character co-occurrence, subsets of Facebook and more. There are fewer levels of self-similarity in smaller graphs, which limits the number of times we can reduce a graph, but they are still valid and interesting cases to analyze. We feel our techniques would generalize well to large graphs, and exhibit many of the same parameters simply at more levels of reduction. All of our datasets came from either SNAP, GEPHI or Hernan Makse's (one of the inventors of box counting) homepage.

It is important to note, however, that we encountered restrictions in the datasets we could analyze, and therefore our dataset selection is not completely representative of all real-world networks. For example, HG is limited to generating only undirected graphs, as are most other current and relevant fractal network-generating models such as the one by Song, Havlin, and Makse. Therefore, for this analysis, we are limited to validating and comparing KRONFIT, FF and HG to solely undirected graphs.

For each graph, we had to generate the model (HG and KronEM run in O(N^2) time), then generate about 40 reductions of the graph (4 each at length scales 2-11) and the skeleton (which is O(N^3)), and then the community structure (see section on Analysis Process) through Infomap and Big Clam algorithms. We had to develop all the glue to put these components together along with our own algorithmic implementations, which meant that in the end it took many many iterations and man hours to get right.

Because of this runtime limitation, our datasets are further restricted to those with less than 300K nodes and/or approximately 1 million edges. Despite these restrictions, we feel that the diversity of size and type of networks that we looked at are representative enough of real-world networks and phenomena to gain meaningful insights from our work. In total we chose 9 representative types of networks.

4.3 Analysis Process

Now that we had our models and our data, we had to come up with a process for analyzing them. For the most part, the structure of a graph is defined by the shape of various distributions.

First, as a baseline, we used snap to generate degree distributions, node-triangle participation, page rank, and clustering-degree distributions. These are basic components of any graph and any model seeking to emulate a graph should be able to perform well when viewed under the lens of these statistics.

Then, we added our own new metrics - first, the box count fractal dimension, which is a plot of box size vs number of boxes, another plot which is count of each box "mass" (how many nodes are in the box) versus box size, and finally the infomap community size plot at the lowest level. The idea behind using community structure is that box size is a fixed radius and we're asking, how many boxes do we need? How massive are they? With the community detection, we ask a similar question, but with a more fluid definition for box boundary. Communities (in the partition style of infomap) are very natural “boxes” for nodes to fit into.

Of course, all of this is for one ‘graph’. There are 40 graphs generated from each base graph through box counting and we ran this pipeline on each one of them - giving us 6 charts to look at. Big Clam had an unreasonable runtime and we were unable to run it on all the reductions. We have an overall chart for it, however.

So in total, we have for each graph:

- 1 plot of BIGCLAM on the real graph and the model graphs.
- 1 plot of Infomap community sizes distribution for the real graph and the model graphs
- 4 plots of infomap community sizes “at a a glance” showing consistency of structure within each model.
- 4 plots of fractal dimension “at a glance” showing consistency of the fractal dimension within each model.
- up to 40 rows, each corresponding to a particular reduction level at a particular box size (4 reduction levels, 10 box sizes) with each row having 6 plots on it (degree distribution, node triangles, clustering-degree, page rank, boxcount fractal dimension, infomap community structure). Here we see the consistency of fractal dimension across models.

4.4 Algorithmic Implementations

For our analysis of these fractal networks, we used many already implemented algorithms through snap.py. However, we also implemented many fractal analysis algorithms ourselves, described in relevant literature (Section 2.2). Chief among these are the box counting algorithms from Song et al. [?], the Skeleton-Creation algorithm from Goh et al. [?] and PageRank (a reimplementation), in addition to the box renormalization scheme for various radii. We also used additional community detection algorithms such as BIGCLAM and Infomap. In this section, we will describe, on a high-level, the purpose and pseudocode of these more uncommon algorithms.

4.4.1 Standard Algorithms

For most of our basic analysis, we used already-implemented functions from snap.py. Specifically, we used snap to calculate the diameter, the average clustering coefficient, the maximum weakly connected component, the maximum strongly connected component, node-triangle participation and the clustering coefficient by degree. However, these are the basic and common statistics that are typically used to compare fitness of modeled networks.

4.4.2 Skeleton

The communication kernel of a graph (skeleton) is defined to be the minimum spanning tree constructed with the highest betweenness centrality edges. In order to compute this we:

1. Use snap to generate an ordered list of edges by betweenness centrality
2. run kruskal’s spanning tree algorithm to generate the final skeleton graph, with the edges evaluated in order of betweenness centrality.

Betweenness centrality is $O(N^3)$ to compute and it takes 22 minutes on a 20K node graph (as an example).

4.4.3 Box-Counting

The box counting algorithm is an extension of the same notion applied to graphical fractals. The idea is to quantify the fractal dimension of a graph - how ”space filling” a graph is. It comes in two forms - the slope of box length vs number of boxes (which requires that we compute a separate covering for each box length) or simply the power law coefficient of the box masses at a particular box size. There are 4 main algorithms - Greedy, Random Burn, CBB Burn, MEMB. A discussion of the pros and cons of each is provided in Song’s paper [12].

Eventually we settled on the greedy box counting algorithm as a compromise of speed and accuracy. This conclusion was also reached by another researcher trying to compute the fractal dimensions of software networks[9], so we felt confident with this decision.

Song’s team provides a good conceptual algorithm:

1. Assign a unique id from 1 to N to all network nodes, without assigning any colors yet.
2. For i in range(1,N).
   (a) Calculate the distance l_{ij} from i to all the nodes in the network with id j less than i.

(b) For l_B in range(1, l_{maxB}): Select one of the unused colors $c_{jl_{ij}}$ from all nodes $j < i$ for which $l_{ij} >= l_B$. This is the color $c_{i_{l_B}}$ of node i for the given l_B value.

But this tells us nothing about how to actually implement box counting. How do you select an unused color? how do you store the variables? What's the most efficient way to compute all nodes with distance less than l_B for each l_B? In other words, it's not transcribable pseudocode; you have to implement it yourself. Here is an implementation in pseudo-python with an explanation of each numbered step.

```
color(set colors): the position of the first missing number in the set of positive numbers
restrictedset(dict lcolormap, set nodes): reduce(OR, get(lcolormap, nodes))
boxcount(graph, mbox):
  (1) onids = pick a node, and order nodes by distance from that node.
  (2) colormap = {l:{onids[0]:ibs([0])} for l in range(MIN_BOX,mbox+1)}
  for i in range(1,len(onids)):
    colors = map(
        (4) lambda t: (t[0], color(restrictedset(colormap[t[0]], t[1])) ),
        zip(range(MIN_BOX,mbox+1),
                (3) drop(MIN_BOX-1, remgen(omap, onids[i], mbox, ibs(onids[:i]))) ))

    (5) for (l,c) in colors: colormap[l][onids[i]] = ibs([c])
  (6) return {l: {color: set(ids)} }
```

1. Pick a random node. For each 'ring' of a BFS, add nodes in that ring to a list. This is a heuristic so that the restricted ids always fall into some contiguous region of the graph (i.e., it grows from 1 node following the bfs).
2. colormap is a map of box size to a map of id: color. Initialize all colors of the first node to 0.
3. remgen takes a set of restricted nodes at length 1 (i.e, the first i-1 ids of on ids), and does a BFS from node i, and computes a list of sets, where each set says what nodes node i must have a different color from. The great thing about the heuristic established at the first step is that once the restricted set is exhausted, we can just break and return empty sets, and if we go in order of a BFS, the chances for that are very high.
4. For each restricted set, compute the set of colors spanned by the ids. Then, pick the first missing positive number in the set of colors. This is the color for node i at that box size.
5. Set the colormap to include the new colors we've just determined
6. For each box size, we have a map(id:color). Invert this map so that it is map(color:set(id)) and return.

It's hard to reason about the runtime of this algorithm, since there are many finite size bfs calls and the structure of the graph will greatly impact how this algorithm runs, but it's much faster than brute force computing the distance to all other nodes, since at every step we're only interested in expanding until we've covered i-1 nodes, which could be only a couple hops. Our implementation can be found in the file **boxcount.pyx** in the repo linked at the top of this paper. The profile took 30 seconds to run on 10K nodes, and it grows approximately quadratically.

4.4.4 PageRank

On a high level, the PageRank of a graph is the principle eigenvector of the adjacency matrix of that network. Each value of this eigenvector corresponds to the importance, or centrality, of each node.

We implemented and computed the PageRank of networks using the Monte Carlo simulation. We used the Monte Carlo simulation for the combination of its speed, limited memory usage and its algorithmic simplicity. The Monte Carlo implementation simulates random walks from every node to other nodes. This simulation then normalizes the number of visits each node receives by a constant computed from the initial parameters, making the return vector stochastic and reducing the expectation of visits of each node to the node's actual PageRank value. Pseudocode is in the Appendix.

4.4.5 Community Detection Algorithms

To analyze community size distribution of networks and their corresponding reductions, we used two existing community detection algorithms, BIGCLAM and Infomap.

BIGCLAM is a overlapping community detection algorithm that uses non-negative factorization to determine which communities nodes belong to. It determines the number of communities using Maximum Likelihood Estimation, and then creates probabilities describing how likely it is that each node belongs to each community. It then goes through every node and community pair, deciding whether the node belongs to that community.

Infomap is a partitioning (non-overlapping) community detection algorithm using hierarchical clustering similar to Louvain's. It initializes each node in it's own community, then in a random and sequential order moves nodes to a neighboring community that results in the largest decrease of the map equation (described below) following a random walker. If no move of a node will result in a decrease of the map equation, the node stays in its module and the random walk ceases. This process is repeated iteratively in a new random sequential order until the map equation is minimized. The resulting communities at the end of this process are the hierarchically-nested communities, and we pick the deepest level of those communities.

4.5 Testing Harness

We built a testing harness and developed a pipeline that takes in a network and generates the appropriate models, reductions, and statistics to properly compare all 3 fractal models with the real-world network. It outputs a stats.dat file, which is a mathematica-formatted list that we then visualize to produce the list in the Analysis Process section.

5 Results

There are approximately 5 pages of charts for each of the 9 graphs we tested, and therefore 45 pages of data in total. We choose not to replicate that much data here, as it would be too much to parse. NB: Global color scheme: **real** = **blue**, **forest fire** = **red**, **hypergeometric** = **orange**, **kronecker** = **green**.

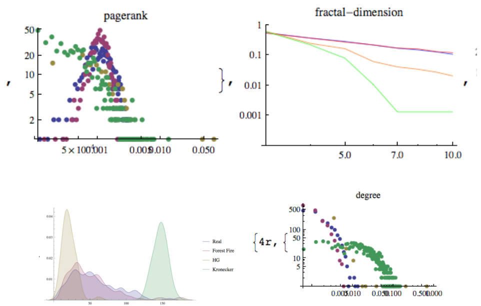

Figure 2: power grid plots

eColi: HG had the most stable overall results. Forest Fire performed well under community detection this time, although none of the models fit perfectly. This was an unusual graph because the protein interactions of eColi have no triangles, and the clustering coefficient is 0. However, it's still undoubtedly fractal, as it exhibits a clear power law degree distribution across reductions. Kronecker performed poorly in this instance - the fractal dimension was very steep meaning that the graph was too dense.

power grid: Forest Fire was clearly the best model for this graph. It had the most accurate at pagerank distribution and it had the lowest error in the fractal dimensions' slopes (fractal-dimension plot). Furthermore the overall communities were very accurate (third plot). Let it also be known that kronecker fails to retain its power law degree distribution when reduced (at whatever length scale - example in the 4th plot), while the other models do not have this problem.

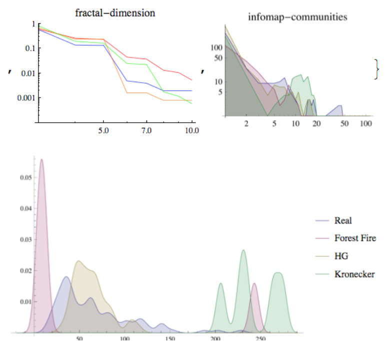

Figure 3: facebook plots

as20000102: No model performed decisively well on this network. HG seems to have had the most success via the box counting method as the fractal dimension error is lower than the other models, but forest fire is comparable. Community structure-wise, forest fire performs the best while HG performs only decently. The Kronecker fails everywhere - the fractal dimension is too steep and the communities are closer to a bell curve than power law.

facebook combined: HG performs the best. It's important to note that network has a different real world manifestation than the others - it is a social network, albeit a small one. Note how there's a steep drop in fractal dimension from 5-6 box size that is mimicked by the model, and the communities are similarly well-modeled.

oregon1: No model fits the big clam communities. HG has the best fractal dimensions, while kronecker is the only one that retains good degree distribution and pagerank, and forest fire models the infomap communities well.

internet routers: As with the other autonomous systems, we see forest fire is clearly the best model - both in terms of fractal dimension and communities. Hypergeometric emulates the fractal dimensions well (better than forest fire half the time), but does not accurately community structure.

6 Discussion

Our primary aim was to examine and evaluate analysis techniques that could describe the deep structures of scale-free, fractal-like networks. To measure the "fractality" of a real network and fractal-like graphs generated from a set of models, we reduced these graphs multiple times with the box-counting method, and checked to see if the degree distribution (the standard metric used in previous research), fractal dimension, and community size distribution matched those of the real network's.

After looking over our data, we realized that there was no one model that performed well across all the graphs. This was expected: we chose a diverse set of graphs and the expectation is that each model would work well on some and do poorly on others. Our approach to the project was also validated, as our diversity in models and graphs enabled us to examine the strengths and weaknesses of each model in a case-by-case manner. Our findings supported the notion that there is no universal scale-free network model. This means our technique is useful for telling the three models apart.

6.1 The Forest Fire (FF) model

The Forest Fire model showed promise in modeling the technological interaction networks (as20000102,internet routers, powergrid), displaying similar community structures and fractal dimensions to those networks in multiple reductions across all box sizes. This suggests that the development of technological interaction networks is similar to dynamics of the Forest Fire model. In addition, the Forest Fire model did comparatively well in modeling community structure of the E.Coli network (a biological network). Finally, the Forest Fire model did not perform well on the `facebook_combined` network (which is a social network). This suggests that Forest Fire is not an appropriate model for biological and social networks. For the non-technological networks, the fractal dimension of the FF model was consistently higher than that of those real world networks. However, in terms of degree distribution, the Forest Fire model generated similar degree distributions to those of all the real world networks.

6.2 The Hyper Geometric Model

Another model that showed promise. It modeled the social network (`facebook_combined`) particularly well, with highly similar estimations of the degree distribution, fractal dimensions, and the community structure of the actual network. The HG model was the overall best at mimicking the fractal dimension. The fractal dimensions matched well with different box lengths across many reductions. Additionally, the Hyper Geometric model also generated similar degree distributions to those of all the real world networks. Finally, in terms of community structure, the community structures of the Hyper Geometric graphs contained a larger number of smaller communities when compared to the real-world community distributions. This indicates that the Hyper Geometric graphs have more layers and a more modular structure than that of our real-world fractal networks. Overall, this model had middling performance in all the networks (apart from the social network); it was never completely off base when estimating the desired features, yet apart from `facebook_combined`, it never generated features that were particularly close to the real world features. The shapes of its community size distributions, for example, had significant variation (more so than that of the real world networks).

6.3 Kronecker

In general, while the Kronecker model had similar degree distributions to the real world networks, it failed in generating community distributions or matching fractal dimension trends. Under both the BigClam or InfoMap metrics, the Kronecker community distribution did not resemble the real world network. Specifically, for InfoMap communities, for half the graphs, Infomap lumped the entire network together as 1 community, and for 2 other graphs it nearly shattered the graph (so that almost every node was its own community), and only 2 graphs (as20000102 and powergrid) had a varied community distribution. Partitioning the network into non-overlapping communities is somewhat of an all-or-nothing venture. Additionally, the 2 graphs for which the Kroneckers did have a somewhat varied InfoMap distribution were the two larger networks.

One possible explanation for these results could be the initial seed matrix ([0.9 0.7; 0.5 0.2]), which reflects our assumption that all large scale networks adhere to the core-periphery model as specified in lecture. Even after modifying this seed matrix with KRONFIT and KRONEM, the final seed matrices arent likely to deviate significantly from our initial seed matrix. Thus its possible the Kronecker Graphs we generate adhere to this core periphery paradigm, resulting in a graph with a large core component and many small periphery components. This would explain InfoMaps tendency to describe a large majority of graphs either as 1 giant community or hundreds of small communities (of size 1 or 2). When looking at the BigClam (overlapping communities), the distributions of the Kroneckers and the real world networks more resemble each other, but for all the graphs, there exist noticeable differences, indicating a clear inability for the Kronecker Graph to model community structure. In this case, the majority of distributions for both real world networks and the Kronecker graphs were shaped as bell-curves, but a large majority of the Kronecker graphs had a higher mean community size than the real world network, and the variances of the Kronecker networks were less than or equal to the variances of the real world network. Once again, the core-periphery structure that the Kronecker graphs adhere to could also explain why in the BigClam distributions, the mean community size for Kronecker Graphs is higher than those of the real-world networks due to the presence of one or more of these core communities which many nodes (even those in the periphery) would be connected to.

7 Further Work

7.1 Network Models

Going onwards, we hope to test more fractal, scale-free models such as the one proposed by Song et al. (Reverse Box Counting Method) in order to check their viability and comprehensiveness with respect to modeling real-world networks. Especially with the success of the Forest Fire model on certain types of networks, we think our work suggests that there exists an optimal multifractal network-generating model/process for social networks; this would be an extremely valuable discovery across many domains of research. We had attempted to ameliorate this deficiency by further obtaining and compiling the Geometric Protean model [11], but unfortunately its run time was O(N3), which was unfeasible given our computation and time constraints.

We also acknowledge that because the parameters for all the models we analyzed were not deterministic, our analyses may have suffered from random chance. In the future, with more computation power and time, we would like to replicate our results with multiple versions of each model we tested in order to ensure accurate results and reduce model variation.

7.2 Network Analysis Methods

We did not solidify an objective and quantifiable metric for comparing community structures aside from plotting their size distribution. The bifurcation parameter may be worth examining further. Finally, we also would like to confirm our findings through the adoption of a more traditional fractal measure like the Hurst exponent. There's also the question of reduction by lowest-order community structure. If we see a similar community structure after reducing, then that could be one more definition of fractality. We also computed skeletons of graphs, but did not do much with them. In general, the work we found suggested many redundant and similar conclusions found by applying the box counting algorithm to a skeleton. However, we would like to try looking at the distribution of subtree sizes at each level and see if there is anything self-similar there. The bifurcation parameter would also be relevant here.

We also need a quantifiable metric for error for all these charts and plots. Looking at them is definitely required, but it would help a lot to have a metric of which model is the closest to the real distribution. However, there are two components to the distribution - shape and range, and we would need intelligent ways of including both. For things like fractal dimensions, it would be easier to just do a least squares error since the domain is exactly the same.

7.3 Expand Selection of Datasets

Due to the time constraint of our project, we had to limit our datasets in size, which limited the variety of networks we can analyze. Certainly, there were more interesting and unique networks in the world that were possible to extract and analyze. This is especially important because of our discovery that the Forest Fire model fit well a certain kind of real-world network. This suggests that there may be a specific type of network that we might have not tested which resonates with one or more of our models. Given more time and/or computing power, we would have also liked to process and analyze much larger and diverse graphs, as size was a limiting factor when choosing networks to analyze.

7.4 Further Investigation of Forest Fire and Autonomous Systems

Because the Forest Fire model fit technological systems interactions so well, we would like to further investigate if its fitness is generalizable to all technology-interaction networks. Additionally, because the Forest Fire networks were so consistently fit across these networks, we think it is worthwhile to investigate why the Forest Fire model performs so well on these interactions: do technological systems interact and form connections in the same way the Forest Fire networks are generated?

Acknowledgements

We'd like to thank the CS224W staff for a wonderful quarter and Professor Leskovec for all his help. Special thanks to Paris Syminelakis for all his help with our project and suggesting models.

References

[1] Anthony Bonato, Jeannette Janssen, and Pawe Prat. Geometric protean graphs. *Internet Math.*, 8(1-2):2–28, 2012.

[2] GEPHI. Datasets. `https://wiki.gephi.org/index.php/Datasets`, 2014.

[3] Dong-Hee Kim, Jae Noh, and Hawoong Jeong. Scale-free trees: The skeletons of complex networks. *Phys. Rev. E*, 70:046126, Oct 2004.

[4] Myunghwan Kim and Jure Leskovec. The network completion problem: Inferring missing nodes and edges in networks.

[5] Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá. Hyperbolic geometry of complex networks. *Phys. Rev. E*, 82:036106, Sep 2010.

[6] Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. Kronecker graphs: An approach to modeling networks. *J. Mach. Learn. Res.*, 11:985–1042, Mar 2010.

[7] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Trans. Knowl. Discov. Data*, 1(1), March 2007.

[8] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters, 2008.

[9] Mario Locci, Giulio Concas, and Ivana Turnu. Computing the fractal dimension of software networks. In *Proceedings of the 9th WSEAS international conference on Applied computer science*, pages 146–151. World Scientific and Engineering Academy and Society (WSEAS), 2009.

[10] Hernan Makse. Datasets. `http://www-levich.engr.ccny.cuny.edu/~hmakse/soft_data.html`, 2014.

[11] Stanford Snap. Datasets. `http://snap.stanford.edu/data`, 2014.

[12] Chaoming Song, Lazaros K Gallos, Shlomo Havlin, and Hernán A Makse. How to calculate the fractal dimension of a complex network: the box covering algorithm. *Journal of Statistical Mechanics: Theory and Experiment*, 2007(03):P03006, 2007.

[13] Chaoming Song, Shlomo Havlin, and Hernan A Makse. Self-similarity of complex networks. *Nature*, 433(7024):392–395, 2005.

8 Appendices

8.1 Mathematically Concise Definition of Hyperbolic Geometric Model

The Hyperbolic Geometric graph generation model works by mapping nodes to a hyperbolic space and connecting them with a probability as a function of their hyperbolic distance and additional other parameters. The model takes in 5 parameters:
N - Number of Nodes, **k** - Expected average degree, γ - Expected power-law exponent (Minimum: 2), **T** - Temperature; default 0; ζ - Square root of the hyperbolic plane curvature $K = -\zeta^2$

The algorithm starts out by mapping and placing each node i ? N uniformly on a circle in hyperbolic space with radius: $R = 2\ln(\frac{8N}{\pi k})$ It then iterates through each node, attaching the nodes with a probability proportional to their hyperbolic distance within the circle. Probability that an edge between node i and node j can be described by the equation: $p_{ij} = \frac{1}{e^{\omega_{ij}}+1}$ where $\omega_{ij} = \beta\frac{\zeta}{2}(x_{ij} - R)$ such that β is defined using the relation:

$$k = \frac{Ne^{-\beta\zeta R/2}}{1-\beta}(\frac{2}{\pi})^{\beta}(\frac{\gamma-1}{\gamma-2})^2$$

In this case, x_{ij} represents the hyperbolic distance between two nodes i and j, and can be represented by the equation:

$$x_{ij} = r_i + r_j + \frac{2}{\zeta} ln(\sin \frac{abs(\Theta_i - Theta_j)}{2})$$

where Θ_n and r_n are the polar coordinates of node n in hyperbolic space.

With this intuition, one can easily see that the probability of a connection between two nodes grows with decreasing hyperbolic distance of those two nodes. [8]

8.2 Mathematically Concise Definition of Stochastic Kronecker Graph Model

For the Stochastic Kronecker Graph Model, we have 2 parameters. M - the initial n x n seed matrix where all elements are between 0 and 1 inclusive k - a positive integer representing the power you want the seed matrix to be raised to.

The output is M^k, the kth Kronecker Power of matrix M, representing a probability matrix from which we can generate a Kronecker Graph.

The dimensions of the M^k (the kth Kronecker Power of M) are $nk \times nk$, and because the elements of M were between 0 and 1 (inclusive), the elements M^k will also be between 0 and 1 (inclusive). See section 2.1 for a visual explanation of the Kronecker Product.

M^k can be considered a probability matrix and an adjacency matrix where m_{ij} (m_{ij} M^k) represents the probability of an edge existing between node i and node j.

To generate a Kronecker graph K from M^k, we simply flip a weighted coin for each cell of M^k with the probability in that cell.

8.3 Concise Overview of KRONFIT

KRONFIT is an algorithm which accepts (ideally) a Kronecker Graph G and tries to estimate the a set of parameters Q (these parameters being the initial seed matrix, generating the Graph). Essentially, KRONFIT tries to find the Q that has the highest probability of generating graph G.

We are solving the expression: argmax Q P(G—Q).

While solving this rigorously for Q would have an enormous runtime, KRONFIT makes this run in linear time simply by framing it as a gradient descent problem going like this:

Initialize Seed Matrix Q while not converged do: evaluate the gradient of Q Q = Q + gradient of Q return converged Q

Evaluating the gradient of Q is the costly step, and is achieved by getting a sample permutation of a set of the nodes from G, and mapping their respective rows and columns onto Q, and essentially taking the gradient of logP(G—,Q). In short, this ends up becoming the gradient for seed matrix Q. Many steps in the process have been optimized in order to yield a linear running time for KRONFIT.

One can also end the loop after a set number of iterations, if convergence takes too long.

8.4 Concise Overview of KRONEM

KRONEM accepts any network G as input and estimates a set of parameters Q (the initial seed matrix) that would generate a Kronecker graph H, of which we assume G is a component of. In this case, we assume that network G is part of a larger Kronecker graph H, whose unknown component well call Z. Z and G together make up the Kronecker Graph H.

One objective is to identify the unknown component Z based on known component G and seed Parameters Q. But then we also need to find the Q most likely to generate Kronecker graph H (composed of Z and G), which means well need to have some knowledge of Z as well. This lends itself to the Estimation-Maximization approach (thus the name KRONEM), where we first identify the structure of the Z most likely to be generated from our current Q and G, and then use this new Z and G and try and find the Q that would generate a Kronecker graph of H (composed of Z and G).

In this case, the edges of Z are the latent variables in this EM approach. An instance of Z is sampled by using the Kth Kronecker Power of Q (Qk) and assigning a sampling permutation of rows from this Qk to compose Z. Many instances of Z are found, and based on this, we attempt to find the Q most likely to generate the many instances of H (composed of G and the many instances of Z), by using average gradient ascent and identifying a new Q.

This process repeats until convergence or for a set number of EM steps.

8.5 Forest Fire Statistics

Below are visualizations of network statistics for generated Forest Fire graphs. Figure 8.7.1 describes the clustering coefficient of Forest Fire graphs, where the x-axis is $p_f \times 10^3$ and the y-axis is $p_r \times 10^3$. Figure 8.7.2 describes the clustering coefficient of Forest Fire graphs where the x-axis is the Number of Nodes/500 and the y-axis is the clustering coefficient. We held p_f and p_r at .25 and .25, respectively. Figure 8.7.3 describes the average degree of Forest Fire graphs where the x-axis is $p_f \times 10^3$ and the y-axis is $p_r \times 10^3$. Figure 8.7.4 describes the clustering coefficient of Forest Fire graphs where the x-axis is the Number of Nodes/500 and the y-axis is the clustering coefficient. Note that while the Forest Fire graphs average degree and clustering coefficient are independent of the number nodes, both statistics seem roughly proportional to the sum of p_f and p_r. That is: avgDeg $\approx p_f + p_r$ and clustCoeff $\approx p_f + p_r$

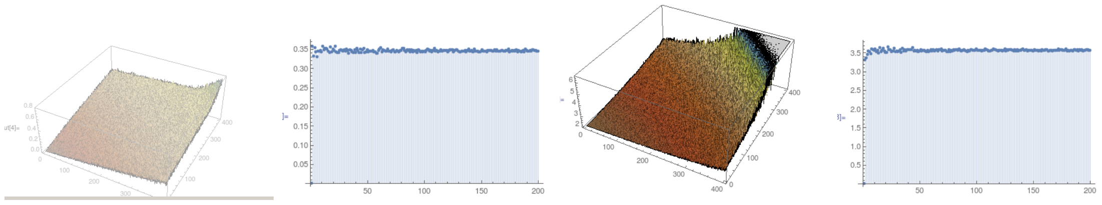

Figure 4: 8.7.1, 8.7.2, 8.7.3, 8.7.4 from top left to bottom right

8.6 Page Rank Pseudocode

```
PageRank(Beta, R, i):
visits = array[n];
for i in iterations, node in n, r in R:
  currentNode = node
  visits[currentNode]++;
  while(randProb[0,1] < Beta):
    neighbors = currentNode.getNeighbors();
    currentNode = neighbors[randProb[0,1]*neighbors.length]
    visits[currentNode]++
return visits/i*(1-Beta)/(nR)
```