Scalable Modeling of Real Graphs using Kronecker Multiplication

Jure Leskovec JURE@CS.CMU.EDU
Christos Faloutsos CHRISTOS@CS.CMU.EDU
Carnegie Mellon University

Abstract

Given a large, real graph, how can we generate a synthetic graph that matches its properties, *i.e.*, it has similar degree distribution, similar (small) diameter, similar spectrum, etc? We propose to use "Kronecker graphs", which naturally obey all of the above properties, and we present KronFit, a fast and scalable algorithm for fitting the Kronecker graph generation model to real networks. A naive approach to fitting would take super-exponential time. In contrast, KronFit takes *linear* time, by exploiting the structure of Kronecker product and by using sampling. Experiments on large real and synthetic graphs show that KronFit indeed mimics very well the patterns found in the target graphs. Once fitted, the model parameters and the resulting synthetic graphs can be used for anonymization, extrapolations, and graph summarization.

1. Introduction

Large, real graphs have a lot of structure: they typically obey power laws in their in- and out-degree distributions; they have small diameter; and they often have a self-similar structure, with communities within communities

Although several, realistic, graph generators have been proposed in the past (like the preferential attachment, the copying model, the small-world model, the forest fire model, etc.), very little work exists on how to fit the parameters of such models.

This is exactly the problem we examine here. Given a large real graph, we want to choose the most realistic generator and to estimate its parameters, so that our resulting synthetic graph matches the properties of the real graph as well as possible.

Ideally we would like: (a) A graph generation model that naturally obeys as many properties as possible, among the ones observed in real graphs. (b) The parameter fitting should be fast and scalable, so that we can handle graphs with thousands and millions of nodes. (c) The resulting set of parameters should generate realistic-looking graphs, that match the topological properties of the target, real graph.

The fitting presents several conceptual and engineering challenges: Which generator should we choose, among the many in the literature? How do we measure the goodness of the fit? How do we solve the correspondence problem (which node of the real graph corresponds to what node of the synthetic one)?

We examine the Kronecker graphs (Leskovec et al., 2005) which are based on Kronecker matrix multiplication. Kronecker model can generate graphs that obey many of the patterns found in real graphs. Moreover, we present KronFit, a fast and scalable algorithm for fitting Kronecker graphs by using maximum likelihood. When calculating the likelihood one needs to consider all mappings of nodes to the graph adjacency matrix, which becomes intractable for graphs with more than a few nodes. Even when given "true" mapping evaluating the likelihood is prohibitively expensive. We present solutions to both problems: We develop Metropolis sampling algorithm for node mapping and approximate the likelihood to obtain a *linear* time algorithm that scales to large graphs.

Once the model is fitted to the real graph, there are several benefits and applications: (a) The parameters give us information about the structure of the graph itself; (b) *Graph compression:* we can compress the graph, by just storing the model parameters, and the deviations between the real and the synthetic graph; (c) *Extrapolations:* we can use the model to generate a larger graph, to help us understand how the network will look like in the future; (d) *Sampling:* conversely, we can also generate a smaller graph, which may be useful for running simulation experiments (e.g., simulating routing algorithms in computer networks, or virus/worm propagation algorithms), when these algorithms may be too slow to run on large graphs; (e) *Anonymization:* suppose that the real graph can not be publicized, like, e.g., corporate e-mail network; customer-product sales in a recommendation system.

Appearing in *Proceedings of the 24th International Conference on Machine Learning*, Corvallis, OR, 2007.

Yet, we would like to share our network. Our work gives ways to such a realistic, 'similar' network.

2. Related Work and Background

Networks across a wide range of domains have been found to share common statistical properties. We use these properties as sanity checks, that is, our synthetic graphs should match the properties of the target graph. First we give a list of such properties, then we mention the graph generators, and finally we survey earlier attempts at graph fitting.

Graph patterns One of the most striking patterns is the power-law of the *degree distribution*: $n_k \propto k^{-a}$, where $a > 0$ is the power-law exponent, and n_k is the count of nodes with degree k. Power law (or power law tail) distributions occur in the Web (Kleinberg et al., 1999), in the Internet (Faloutsos et al., 1999), in citation graphs (Redner, 1998), in on-line social networks (Chakrabarti et al., 2004), and many more.

The second pattern is the *small diameter* (the small-world phenomenon, or 'six degrees of separation'): In real graphs, most pairs of nodes are within few hops from each other (Albert & Barabási, 2002; Milgram, 1967). *Hop-plot* extends the notion of diameter by plotting the number of reachable pairs $P(h)$ within h hops. It gives us a sense how quickly nodes' neighborhoods expand with the distance.

The spectral properties also exhibit power laws: The *scree plot* is a plot of the eigen- (or singular-) values of graph adjacency matrix, versus their rank. It often obeys a power law. The same holds for the distribution of the components of the first eigenvector ("network value" of each node) (Chakrabarti et al., 2004).

Generative models The earliest generative model for graphs is a random graph model (Erdos & Renyi, 1960)where a pair of nodes has identical, independent probability of being joined by an edge. Although heavily studied, this model fails to generate power-law degree distributions. For small diameters, there is the *small-world* generator (Watts & Strogatz, 1998). The rest of the recent ones all generate heavy-tailed degree distributions (power-law, or lognormal). An influential idea was the *preferential attachment* (Albert & Barabasi, 1999; Kleinberg et al., 1999) where new nodes prefer to attach to high-degree older nodes, which leads to power-law tails and to low diameters. There are also many variations: "copying model", the "winner does not take all" model, and the "forest fire" model. See (Chakrabarti & Faloutsos, 2006) for a detailed survey and comparison of these methods.

One should also note that most of graph generative models usually aim in modeling (explaining) just a single property of the network. For these models it is known that there are certain network properties they don't generate. Moreover, simple expressions have been derived that relate the network property (e.g., degree exponent) with the setting of (usually just a single) parameter. So, in these models there is no interesting parameter estimation and fitting.

Our work builds on the "Kronecker Graph model" (Leskovec et al., 2005), where Kronecker matrix multiplication is used to lead to realistic graphs obeying *multiple* properties of real world graphs. Kronecker graphs have a variable number of parameters, which makes them interesting for fitting. We describe them in more detail later.

Fitting graph models Most work in fitting network models comes from the social sciences, where the so-called *exponential random graph models* were introduced, also known as $p*$ (Wasserman & Pattison, 1996). The $p*$ model focuses on "local" structural features of networks (like, *e.g.* characteristics of nodes that determine a presence of an edge), while here we model a large real-world graphs as a whole. Moreover, for large graphs the number of parameters becomes large, and estimation prohibitively expensive.

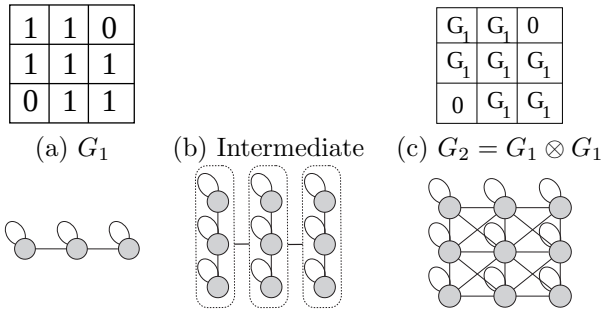
A common theme when estimating $P(G)$ is the challenge of factorially many orderings of nodes. Ordering can define the mapping to rows of adjacency matrix, or the order in which nodes were added to the network. (Butts, 2005) used permutation sampling to determine similarity of adjacency matrices, and (Bezáková et al., 2006) used it for graph model selection. Recently, an approach for estimating parameters of "copying" models was introduced (Wiuf et al., 2006), however authors also note that the class of "copying" models may not be rich enough to model real networks.

As we show later, the Kronecker Graph model has the necessary expressive power to mimic real graphs.

3. Kronecker Graphs

Kronecker matrix multiplication was recently proposed for realistic graph generation, and shown to be able to produce graphs that match many of the patterns found in real graphs (Leskovec et al., 2005). Kronecker graphs are based on a recursive construction. A procedure that is best described in terms of the *Kronecker product* of graph adjacency matrices.

Deterministic Kronecker Graphs The main idea is to create self-similar graphs, recursively. We begin with an *initiator* graph G_1, with N_1 nodes, and by recursion we produce successively larger graphs $G_2 \ldots G_n$ such that the k^{th} graph G_k is on $N_k = N_1^k$ nodes. *Kronecker product* is a perfect tool for this goal:



(a) G_1 (b) Intermediate (c) $G_2 = G_1 \otimes G_1$

Figure 1. Kronecker multiplication: Top row: structure of adjacency matrices. Bottom: corresponding graphs – "3-chain" and its Kronecker product with itself; each of the nodes gets expanded into 3 nodes, which are then linked.

Definition 1 (Kronecker product of matrices) *Given two matrices* $\mathbf{U} = [u_{i,j}]$ *and* $\mathbf{V}$ *of sizes* $n \times m$ *and* $n' \times m'$ *respectively, the Kronecker product matrix* $\mathbf{S}$ *of dimensions* $(n * n') \times (m * m')$ *is given by*

$$\mathbf{S} = \mathbf{U} \otimes \mathbf{V} \doteq \begin{pmatrix} u_{1,1}\mathbf{V} & u_{1,2}\mathbf{V} & \dots & u_{1,m}\mathbf{V} \\ u_{2,1}\mathbf{V} & u_{2,2}\mathbf{V} & \dots & u_{2,m}\mathbf{V} \\ \vdots & \vdots & \ddots & \vdots \\ u_{n,1}\mathbf{V} & u_{n,2}\mathbf{V} & \dots & u_{n,m}\mathbf{V} \end{pmatrix} \quad (1)$$

Kronecker product of two graphs is defined as Kronecker product of their adjacency matrices. We denote k^{th} Kronecker power of G_1 as $G_1^{[k]}$ (abbreviated to G_k), where $G_k = G_1^{[k]} = G_{k-1} \otimes G_1$.

Figure 1 shows the recursive construction of Kronecker graphs. We start with G_1, a 3-node chain, and Kronecker power it to obtain G_2. To produce G_k from G_{k-1}, we "expand" (replace) nodes of G_{k-1} by copies of G_1, and join the copies according to the adjacencies in G_{k-1} (see fig. 1). One can imagine this by positing that communities in the graph grow recursively, with nodes in the community recursively getting expanded into miniature copies of the community. Nodes in the sub-community then link among themselves and to nodes from other communities.

Stochastic Kronecker Graphs Here we will be working with a stochastic version of Kronecker Graphs. The difference is that now initiator matrix is stochastic: we start with a $N_1 \times N_1$ *probability matrix* $\Theta = [\theta_{ij}]$, where the element $\theta_{ij} \in [0, 1]$ is the probability that edge (i, j) is present. We compute the k^{th} Kronecker power $\mathcal{P} = \Theta^{[k]}$; And then for each $p_{uv} \in \mathcal{P}$, include edge (u, v) with probability p_{uv}.

Stochastic Kronecker Graphs are thus parameterized by the $N_1 \times N_1$ *probability (parameter) matrix* Θ. The probability p_{uv} of an edge (u, v) occurring in k-th Kronecker power $\mathcal{P} = \Theta^{[k]}$ can be calculated as:

$$p_{uv} = \prod_{i=0}^{k-1} \Theta\Big[\lfloor \frac{u-1}{N_1^i} \rfloor (\mathrm{mod} N_1) + 1, \lfloor \frac{v-1}{N_1^i} \rfloor (\mathrm{mod} N_1) + 1\Big]$$

The equation imitates recursive deepening into matrix $\mathcal{P}$, where at every level i the appropriate element of Θ is chosen. Since $\mathcal{P}$ has N_1^k rows and columns it takes $O(k \log N_1)$ to evaluate the equation.

4. Proposed Method

4.1. Preliminaries

Stochastic graph models introduce probability distributions over graphs. A generative model assigns probability $P(G)$ to every graph G. $P(G)$ is the *likelihood* that a given model generated graph G. We concentrate on Stochastic Kronecker Graph model, and consider fitting it to a real graph G. We use maximum likelihood approach, *i.e.* we aim to find parameter values Θ that maximize the $P(G)$ under the model. This presents several challenges:

Model selection Graph is a single structure, and not a set of items drawn i.i.d. from some distribution. So one can not split it into independent training and test sets. The fitted parameters will thus be best to generate a *particular* instance of a graph. Also, overfitting is an issue since more complex model fits better.

Node labeling The second issue is the node ordering or node labeling. Graph G has a set of N nodes, and each node has unique index (label). Labels do not carry any particular meaning. One can think of this as a graph is first generated and then the labels are randomly assigned to the nodes. This means that two isomorphic graphs that have different node labeling should have the same likelihood. So to compute the likelihood one has to consider all node labelings $P(G) = \sum_\sigma P(G|\sigma)P(\sigma)$, where the sum is over all permutations σ of N nodes.

Likelihood estimation Calculating $P(G|\sigma)$ naively takes $O(N^2)$ by simply evaluating the probability of each edge in the graph adjacency matrix. The challenge is averaging over the *super-exponentially* many permutations which is computationally intractable, and thus one has to reside to simulation and sampling. As we will later see for real graphs even calculating $P(G|\sigma)$ in $O(N^2)$ is infeasible.

We use sampling to avoid super-exponential sum over the node labelings. By exploiting the structure of kronecker matrix multiplication we develop an algorithm to evaluate $P(G|\sigma)$ in *linear* time $O(E)$. Since real graphs are *sparse*, *i.e.* the number of edges is of the same order as the number of nodes, this makes the fitting of the Kronecker model to large graphs tractable.

4.2. Problem Formulation

Suppose we are given a graph G on $N = N_1^k$ nodes (for some positive integer k), and a N_1 by N_1 Stochastic Kronecker Graph initiator matrix Θ. Θ is a parameter matrix, a set of parameters that we aim to estimate. For now also assume N_1 is given. Later we will show how to select it. Next, we create a Stochastic Kronecker Graph probability matrix $\mathcal{P} = \Theta^{[k]}$, where every cell p_{ij} of $\mathcal{P}$ contains a probability that node i links to node j. We evaluate the probability that G is a realization of $\mathcal{P}$. The task is to find such Θ that has the highest probability of generating G. Formally, we are solving:

$$\arg\max_{\Theta} P(G|\Theta) \tag{2}$$

A permutation σ of the set $\{1, \ldots, N\}$ defines the mapping of nodes from G to stochastic adjacency matrix $\mathcal{P}$. The node labeling is arbitrary and carries no significant information. A priori all labelings are equally likely. To evaluate the likelihood of G one needs to consider all possible mappings of N nodes of G to rows of $\mathcal{P}$. For convenience we work with *log-likelihood* $l(\Theta)$, and solve $\arg\max_\Theta l(\Theta)$, where $l(\Theta)$ is defined as:

$$\begin{aligned} l(\Theta) &= \log P(G|\Theta) = \log \sum_{\sigma} P(G|\Theta, \sigma) P(\sigma|\Theta) \\ &= \log \sum_{\sigma} P(G|\Theta, \sigma) P(\sigma) \end{aligned} \tag{3}$$

$P(G|\Theta, \sigma)$ is calculated as follows. First, by using Θ we create the Stochastic Kronecker graph adjacency matrix $\mathcal{P} = \Theta^{[k]}$. Permutation σ defines the mapping of nodes of G to the rows and columns of stochastic adjacency matrix $\mathcal{P}$. Modeling edges as Bernoulli random variables we evaluate the likelihood:

$$P(G|\mathcal{P}, \sigma) = \prod_{(u,v)\in G} \mathcal{P}[\sigma_u, \sigma_v] \prod_{(u,v)\notin G} (1 - \mathcal{P}[\sigma_u, \sigma_v]), \tag{4}$$

where we denote σ_i as the i^{th} element of the permutation σ, and $\mathcal{P}[i, j]$ is the element at row i, and column j of matrix $\mathcal{P} = \Theta^{[k]}$. The products go over all edges present in graph G, and all edges missing from G.

Ideally, we would like to compute the log-likelihood $l(G|\Theta)$ and the gradient matrix $\frac{\partial}{\partial \hat{\Theta}_t} l(\hat{\Theta}_t)$, and then use the gradient to update the current parameter estimates and move towards a better solution. Algorithm 1 gives an outline of the optimization procedure.

As the problem is introduced there are several difficulties. First, we assume gradient descent type optimization will work, *i.e.* the problem does not have (too many) local minima. Second, we are summing over exponentially many permutations in equation 3.

Algorithm 1 KronFit algorithm

Input: integer N_1, and graph G on $N = N_1^k$ nodes
Output: MLE parameters $\hat{\Theta}$ ($N_1 \times N_1$ matrix)
initialize $\hat{\Theta}_1$
while not converged **do**
  evaluate gradient: $\frac{\partial}{\partial \hat{\Theta}_t} l(\hat{\Theta}_t)$
  update parameters: $\hat{\Theta}_{t+1} = \hat{\Theta}_t + \lambda \frac{\partial}{\partial \hat{\Theta}_t} l(\hat{\Theta}_t)$
end while
return $\hat{\Theta} = \hat{\Theta}_t$

Algorithm 2 Calculating log-likelihood and gradient

Input: Parameter matrix Θ, and graph G
Output: Log-likelihood $l(\Theta)$, and gradient $\frac{\partial}{\partial \Theta} l(\Theta)$
for $t = 1$ **to** T **do**
  $\sigma^{(t)} :=$ `SamplePermutation(`G, Θ`)`
  $l_t = \log P(G|\sigma^{(t)}, \Theta)$
  $grad_t := \frac{\partial}{\partial \Theta} \log P(G|\sigma^{(t)}, \Theta)$
end for
return $l(\Theta) = \frac{1}{T} \sum_t l_t, \quad \frac{\partial}{\partial \Theta} l(\Theta) = \frac{1}{T} \sum_t grad_t$

Third, the evaluation of equation 4 as it is written takes $O(N^2)$ and needs to be evaluated $N!$ times. So, naively calculating the likelihood takes $O(N^2 N!)$.

Next, we show that all these can be done in linear time.

4.3. Summing over the Node Labelings

To maximize equation 2 using algorithm 1 we need to obtain log-likelihood gradient $\frac{\partial}{\partial \Theta} l(\Theta)$. We can write:

$$\begin{aligned} \tfrac{\partial}{\partial \Theta} l(\Theta) &= \frac{\sum_\sigma \frac{\partial}{\partial \Theta} P(G|\sigma, \Theta) P(\sigma)}{\sum_{\sigma'} P(G|\sigma', \Theta) P(\sigma')} \\ &= \frac{\sum_\sigma \left[\frac{\partial \log P(G|\sigma, \Theta)}{\partial \Theta} P(G|\sigma, \Theta) \right] P(\sigma)}{P(G|\Theta)} \\ &= \sum_\sigma \frac{\partial \log P(G|\sigma, \Theta)}{\partial \Theta} P(\sigma|G, \Theta) \end{aligned} \tag{5}$$

Note we are still summing over all permutations σ, so calculating eq. 5 is computationally intractable for graphs with more than a few nodes. However, the equation has a nice form which allows to use simulation techniques and avoid the summation over super-exponentially many node labelings. We simulate draws from the permutation distribution $P(\sigma|G, \Theta)$, and evaluate the quantities at the sampled permutations to obtain the expected values of log-likelihood and gradient. Algorithm 2 gives the details.

Next, we describe a Metropolis algorithm to simulate draws from the permutation distribution $P(\sigma|G, \Theta)$, which is given by $P(\sigma|G, \Theta) = P(\sigma, G, \Theta) / \sum_\sigma P(\sigma, G, \Theta) = \sum_\sigma P(\sigma, G, \Theta)/Z_\sigma$, where Z_σ is the normalizing constant that is hard to

Algorithm 3 `SamplePermutation`(G, Θ): Metropolis sampling of the permutations

Input: Kronecker initiator matrix Θ and a graph G on N nodes
Output: Permutation $\sigma^{(i)} \sim P(\sigma|G, \Theta)$
$\sigma^{(0)} := (1, \ldots, N)$
repeat
  Draw j and k uniformly from $(1, \ldots, N)$
  $\sigma^{(i)} :=$ `SwapElements`$(\sigma^{(i-1)}, j, k)$
  Draw u from $U(0, 1)$
  if $u > \frac{P(\sigma^{(i)}|G,\Theta)}{P(\sigma^{(i-1)}|G,\Theta)}$ **then**
    $\sigma^{(i)} := \sigma^{(i-1)}$
  end if
  i = i + 1
until $\sigma^{(i)} \sim P(\sigma|G, \Theta)$
return $\sigma^{(i)}$
Where $U(0, 1)$ is a uniform distribution on $[0, 1]$, and $\sigma' :=$ `SwapElements`(σ, j, k) is the permutation σ' obtained from σ by swapping elements j and k.

compute since it involves the sum over $N!$ elements. However, if we compute the likelihood ratio between permutations σ and σ' (eq. 6) notice that the normalizing constants cancel out:

$$\frac{P(\sigma'|G,\Theta)}{P(\sigma|G,\Theta)} = \prod_{(u,v)\in G} \frac{\mathcal{P}[\sigma_u,\sigma_v]}{\mathcal{P}[\sigma'_u,\sigma'_v]} \prod_{(u,v)\notin G} \frac{(1-\mathcal{P}[\sigma_u,\sigma_v])}{(1-\mathcal{P}[\sigma'_u,\sigma'_v])} \quad (6)$$

$$= \prod_{\substack{(u,v)\in G \\ (\sigma_u,\sigma_v)\neq(\sigma'_u,\sigma'_v)}} \frac{\mathcal{P}[\sigma_u,\sigma_v]}{\mathcal{P}[\sigma'_u,\sigma'_v]} \prod_{\substack{(u,v)\notin G \\ (\sigma_u,\sigma_v)\neq(\sigma'_u,\sigma'_v)}} \frac{(1-\mathcal{P}[\sigma_u,\sigma_v])}{(1-\mathcal{P}[\sigma'_u,\sigma'_v])} \quad (7)$$

This immediately suggests Metropolis sampling algorithm (Gamerman, 1997) to simulate draws from the permutation distribution since Metropolis is solely based on such ratios. In particular, suppose that in the Metropolis algorithm (Algorithm 3) we consider a move from permutation σ to a candidate permutation σ'. Probability of accepting the move to σ' is given by eq. 6, if $\frac{P(\sigma'|G,\Theta)}{P(\sigma|G,\Theta)} \leq 1$ or 1 otherwise.

We further speed up algorithm by using the following observation. As written the eq. 6 takes $O(N^2)$ to evaluate since we have to consider N^2 possible edges. However, notice that permutations σ and σ' differ only at two elements, *i.e.* elements at position j and k are swapped, *e.i.* σ and σ' map all nodes except the two to the same location, which means those elements of equation 6 cancel out. Thus we only need to traverse two rows and columns of matrix $\mathcal{P}$, namely rows and columns j and k, since everywhere else the mapping of nodes to the adjacency matrix is the same for both permutations. We get eq. 7 where the products go only over the two rows where σ and σ' differ.

Graphs we are working with here are too large to explicitly create and store the stochastic adjacency matrix $\mathcal{P}$ by Kronecker powering the initiator matrix Θ. Every time probability $\mathcal{P}[i, j]$ of edge (i, j) is needed the equation in section 3 is evaluated, which takes $O(k)$. So a single iteration of algorithm 3 takes $O(kN)$.

4.4. Efficiently Evaluating the Likelihood

Similarly to evaluating the likelihood ratio, naively calculating the log-likelihood $l(\Theta)$ or its gradient $\frac{\partial}{\partial\Theta} l(\Theta)$ takes time quadratic in the number of nodes. Next we show how to compute this in linear time.

We begin with observation that real graphs are sparse, which means the number of edges is not quadratic but rather linear in the number of nodes. This means that majority of elements of graph adjacency matrix are zero, *i.e.* most of the edges are not present. We exploit this fact. The idea is to first calculate the likelihood (gradient) of an empty graph, i.e. a graph with no edges, and then correct for edges that are in G.

Naively calculating the likelihood for an empty graph one needs to evaluate every cell of graph adjacency matrix. We consider Taylor approximation to the likelihood, and exploit the structure of matrix $\mathcal{P}$ to devise a constant time algorithm.

First, consider the second order Taylor approximation to log-likelihood of an edge that succeeds with probability x but does not appear in the graph: $\log(1-x) \approx -x - \frac{1}{2}x^2$. Calculating $l_e(\Theta)$, the log-likelihood of an empty graph, becomes:

$$l_e(\Theta) = \sum_{i,j=1}^{N} \log(1 - p_{ij}) \approx -\Big(\sum_{i,j=1}^{N_1} \theta_{i,j}\Big)^k - \frac{1}{2}\Big(\sum_{i,j=1}^{N_1} \theta_{i,j}^2\Big)^k \quad (8)$$

Equation 8 is holds due to the structure of matrix $\mathcal{P}$ generated by the Kronecker product. We substitute the $\log(1 - p_{ij})$ with its Taylor approximation, which gives a sum over elements of $\mathcal{P}$ and their squares. Next, we notice the sum of elements of $\mathcal{P}$ forms a multinomial series, and thus $\sum_{i,j} p_{ij} = (\sum_{i,j} \theta_{ij})^k$, where θ_{ij} denotes an element of Θ, and $\mathcal{P} = \Theta^{[k]}$.

Calculating log-likelihood of G now takes $O(N)$: First, we calculate the likelihood of an empty graph in constant time, and then account for edges that are present, *i.e.* we subtract no-edge likelihood and add the edge likelihood:

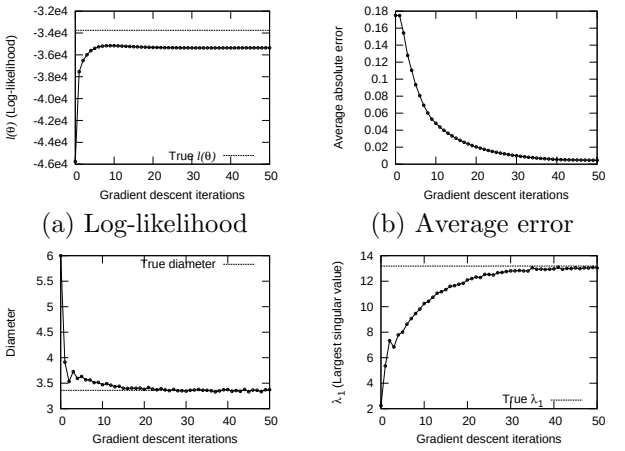
$$l(\Theta) = l_e(\Theta) + \sum_{(u,v)\in G} -\log(1 - \mathcal{P}[\sigma_u, \sigma_v]) + \log(\mathcal{P}[\sigma_u, \sigma_v])$$

Calculation of the gradient follows exactly the same pattern. We first calculate gradient if graph G would have no edges, and then correct for the edges that are present in G. We skip the details of the derivation for brevity. As in previous section we speed up the calculations of log-likelihood and the gradient by exploiting the fact that permutations σ and σ' differ at only two

positions, and thus given the log-likelihood (gradient) from previous time step one only needs to account for the swap of two rows and columns of $\mathcal{P}$ to update it.

4.5. Determining Size of the Initiator Matrix

For model selection to find the appropriate value of N_1, the size of matrix Θ, and choose the right trade-off between the complexity of the model and quality of the fit, we propose to use the Bayes Information Criterion (BIC) (Schwarz, 1978). Stochastic Kronecker Graphs model the presence of edges with Bernoulli random variables, where the canonical number of parameters is N_1^{2k}, which is a function of a lower-dimensional parameter Θ. This is then a *curved exponential family* (Efron, 1975), and BIC naturally applies: BIC $= -l(\hat{\Theta}) + \frac{1}{2}N_1^2 \log(N^2)$, where $\hat{\Theta}$ are maximum likelihood parameters under the model with $\hat{\Theta}$ of size $N_1 \times N_1$, and N is the number of nodes in G.



(a) Log-likelihood (b) Average error

(c) Diameter (d) Largest singular-value

Figure 2. Convergence of graph patterns with the number of iterations of gradient descent using synthetic dataset.

5. Experiments

We begin by investigating the convergence of sampling and gradient descent, then present results on fitting large real-world graphs.

For the experiments we considered synthetic and real graphs. Synthetic Kronecker graphs were generated using $\tilde{\Theta} = [.9, .7; .5, .3]$, and $k = 14$ ($N_1 = 16,384$). Real graphs include a graph of connectivity among Internet Autonomous systems (AS) with $N = 6,474$ and $E = 26,467$; and a who-trusts-whom type social network from *Epinions* (Richardson et al., 2003) with $N = 75,879$ and $E = 508,960$.

In general networks do not have the number of nodes be the integer power of N_1. As the removal of random nodes corrupts the degree distribution (Stumpf et al., 2005) we pad the graph with isolated nodes so that the total number of nodes is a integer power of N_1.

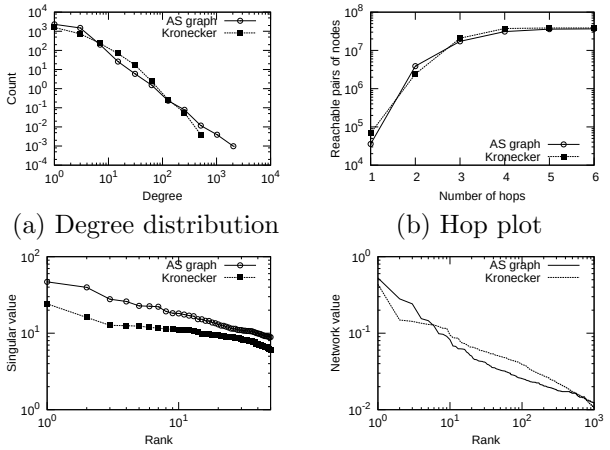
5.1. Convergence

In maximizing the likelihood we use stochastic approximation to the gradient. This adds variance to the gradient and makes efficient optimization techniques, *e.g.* conjugate gradient, highly unstable. Thus we use gradient descent, which is slower but easier to control.

Permutation sampling We examine the convergence of Metropolis permutation sampling. Starting with a random permutation we run algorithm 3, and measure convergence of likelihood and gradient to their true values. Experiments showed that one needs less than a million samples for the estimates to converge. We also measured the variance of the estimates is sufficiently small. In our experiments we start with a random permutation and use long burn-in time. Then when performing optimization we use the permutation from previous step to initialize the permutation at current step of gradient descent. The intuition is that small changes in Θ also mean small changes in $P(\sigma|G, \Theta)$.

Optimization space In Kronecker graphs permutations of the parameter matrix Θ all have the same likelihood. This means that the maximum likelihood optimization problem is not convex, but rather has several global minima. To check for the presence of other local minima where gradient descent could get stuck we run the following experiment: we generated 100 synthetic Kronecker graphs on 16,384 (2^{14}) nodes and 1.4 million edges on average, with a randomly chosen 2×2 parameter matrix Θ^*. For each of the 100 graphs we start gradient descent from a different random location Θ', and try to recover Θ^*. In 98% of the cases the descent converged to the true parameters. Many times the algorithm converged to a different global minima, *i.e.* permuted true parameter values. This suggests surprisingly nice structure of the optimization problem: it seems it behaves like a convex optimization problem with many equivalent global minima.

Gradient descent To get a better understanding of the convergence of the gradient descent we performed the following experiment. After every step t of gradient descent, we compare the true graph G with the synthetic Kronecker graph K generated using the current parameter estimates $\hat{\Theta}_t$. Figure 2 gives the convergence of log-likelihood (a), average absolute error in parameters (b), diameter (c), and largest singular value (d). Note how with iterations of gradient descent properties of graph K quickly converge to those of G even though we are not directly optimizing over



(a) Degree distribution (b) Hop plot

(c) Scree plot (d) "Network" value

Figure 3. Autonomous Systems: Overlayed patterns of real graph and the fitted Kronecker graph. Notice that the fitted Kronecker graph matches patterns of the real graph.

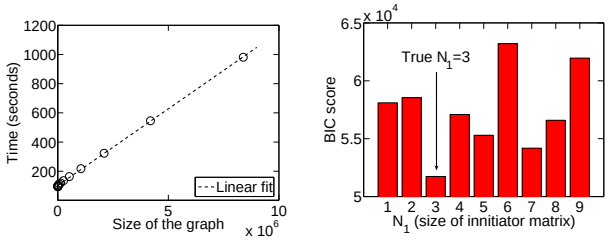
them: log-likelihood increases, average absolute error decreases, diameter and largest singular value of K both converge to G. This is a nice result since it shows that through the optimization of the maximum likelihood the graphs also match in several other properties even though we are not directly optimizing over them.

5.2. Fitting to Real-world Graphs

We also present experiments of fitting Kronecker Graphs model to real-world graphs. Given a real graph G we aim in discovering most likely parameters $\hat{\Theta}$ that ideally would generate a synthetic graph K having same properties as G. This assumes that Kronecker Graphs is a good model for real graphs, and that KronFit is able to recover good parameters. We take real graph G, find parameters $\hat{\Theta}$ using KronFit, generate synthetic graph K using $\hat{\Theta}$, and compare their properties that we introduced in section 2.

Figure 3 shows properties of Autonomous Systems graph, and compares them with the properties of a synthetic Kronecker graph generated using the fitted parameters $\hat{\Theta}$ of size 2×2. Notice that properties of both graphs match really well.

Autonomous Systems is undirected graph and the fitted parameter matrix $\hat{\Theta} = [.98, .58; .58, .06]$ is also symmetric. This means that without a priori biasing the fitting towards undirected graphs, the recovered parameters obey this. Fitting AS graph from a random set of parameters, performing gradient descent for 50 iterations and at each iteration sampling half a million permutations, took less than 20 minutes on a standard desktop PC. This is a significant speedup over (Bezáková et al., 2006), where by using a similar permutation sampling approach for calculating the likelihood of a preferential attachment model on similar AS graph took about two days on a cluster of 50 machines, while in our case finding the MLE parameters took 20 minutes on a desktop PC.



(a) Scalability (b) Model selection

Figure 5. (a) Run time to sample 1 million gradients as the graph grows. The algorithm scales linearly with the graph size. (b) BIC score for model selection. Notice it recovers the model with the true number of parameters.

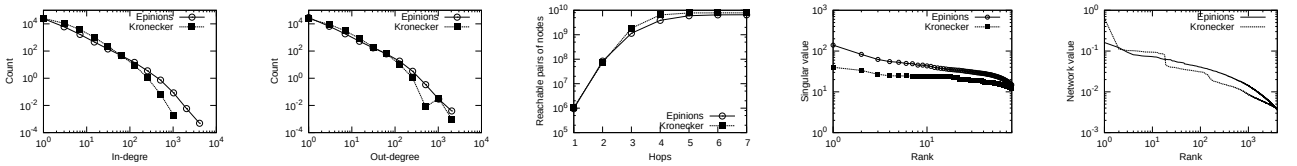
Last, we present the results of fitting Epinions graph. We performed 200 steps of gradient descent and at each step sampled 200,000 permutations. The fitting took 2.5 hours on a standard desktop. Figure 4 shows the results. Notice very good fit of all properties between the Epinions graph and the synthetic graph. Estimated parameter matrix is $\hat{\Theta} = [.99, .54; .49, .13]$. As with Autonomous Systems estimated parameter matrix $\hat{\Theta}$ is very skewed: $\theta_{11} \approx 1$, the diagonal parameters $(\theta_{12}, \theta_{21})$ are around 0.5, and θ_{22} is very small. This indicates that in the Epinions network we observe the "core-periphery" type of network structure.

5.3. Scalability

We generated a sequence of increasingly larger synthetic graphs on N nodes and $8N$ edges, and measured the time of one iteration of gradient descent, *i.e.* sample 1 million permutations and evaluate the gradients. We started with a graph on 1000 nodes, and finished with a graph on 8 million nodes, and 64 million edges. Figure 5(a) shows KronFit scales *linearly* with the size of the graph. We plot processor time vs. size of the graph. Dashed line presents linear fit to the data.

5.4. Model Selection

Last, we present result on model selection. Figure 5(b) shows BIC scores for the following experiment: We generated Kronecker graph with $N = 2,187$ and $E = 8,736$ using $N_1 = 3$ (9 parameters) and $k = 7$. For $1 \leq N_1 \leq 9$ we find the MLE parameters using gradient descent, and calculate the BIC scores. Model with lowest score is chosen. As figure 5(b) shows we recovered the true model, *i.e.* BIC score is lowest for the model with the true number of parameters, $N_1 = 3$.



(a) In-degree (b) Out-degree (c) Hop plot (d) Scree plot (e) "Network" value

Figure 4. Epinions: Overlayed patterns of real graph and the fitted Kronecker graph. Notice that the synthetic Kronecker graph generated using the fitted parameters matches patterns of the real Epinions graph.

6. Conclusion

We presented KronFit, a fast, scalable algorithm to create a synthetic graph that mimics the properties of a given real graph.

In contrast to earlier work, our work has the following novelties: (a) it is among the few that estimates the parameters of the chosen generator (b) it is among the few that has a concrete measure of goodness of the fit (namely, likelihood) (c) it avoids the quadratic complexity of computing the likelihood by exploiting the properties of the "Kronecker graphs" (d) it avoids the factorial explosion of the correspondence problem, by using Metropolis sampling.

The resulting algorithm matches well all the known properties of real graphs, as we show with the *Epinions* graph and the *AS* graph, it scales linearly on the number of edges, and it is order of magnitudes faster than earlier graph-fitting attempts: 20 minutes on a commodity PC, versus 2 days on a cluster of 50 workstations (Bezáková et al., 2006).

The benefits of fitting a Kronecker graph model into a real graph are several: *Extrapolation*: Once we have the Kronecker generator Θ for a given real matrix G (such that G is mimicked by $\Theta^{[k]}$), a larger version of G would be generated by $\Theta^{[k+1]}$. *Sampling*: Similarly, if we want a realistic sample of the real graph, we could use a smaller exponent in the Kronecker exponentiation, like $\Theta^{[k-1]}$. *Anonymization*: Since $\Theta^{[k]}$ mimics G, we can publish $\Theta^{[k]}$, without revealing information about the nodes of the real graph G.

Acknowledgements. Authors would like to thank Zoubin Gharamani, Pall Melsted, Carlos Guestrin and Larry Wasserman for discussions.

This material is based upon work supported by the NSF under grants SENSOR-0329549 IIS-0534205 and under the auspices of the DOE by UC LLNL under contract No.W-7405-ENG-48. This work is also partially supported by the PITA, an IBM Faculty Award, a Yahoo Research Alliance Gift, and generous gift by Hewlett-Packard. Jure Leskovec was partially by a Microsoft Research Graduate Fellowship.

References

Albert, R., & Barabasi, A.-L. (1999). Emergence of scaling in random networks. *Science*, 509–512.

Albert, R., & Barabási, A.-L. (2002). Statistical mechanics of complex networks. *Reviews of Modern Physics*.

Bezáková, I., Kalai, A., & Santhanam, R. (2006). Graph model selection using maximum likelihood. *ICML*.

Butts, C. T. (2005). *Permutation models for relational data*(Tech. Rep. MBS 05-02). Univ. of California, Irvine.

Chakrabarti, D., & Faloutsos, C. (2006). Graph mining: Laws, generators, and algorithms. *ACM Computing Surveys*, *38*.

Chakrabarti, D., Zhan, Y., & Faloutsos, C. (2004). R-MAT: A recursive model for graph mining. *SDM*.

Efron, B. (1975). Defining the curvature of a statistical problem (with applications to second order efficiency). *Ann. Statist.*, *3*, 1189–1242.

Erdos, P., & Renyi, A. (1960). On the evolution of random graphs. *Publication of the Mathematical Institute of the Hungarian Acadamy of Science*, *5*, 17–67.

Faloutsos, M., Faloutsos, P., & Faloutsos, C. (1999). On power-law relationships of the internet topology. *SIGCOMM* (pp. 251–262).

Gamerman, D. (1997). *Markov chain monte carlo, stochastic simulation for bayesian inference*. Chapman & Hall.

Kleinberg, J. M., Kumar, S. R., Raghavan, P., Rajagopalan, S., & Tomkins, A. (1999). The web as a graph: Measurements, models and methods. *COCOON*.

Leskovec, J., Chakrabarti, D., Kleinberg, J. M., & Faloutsos, C. (2005). Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication. *PKDD* (pp. 133–145).

Milgram, S. (1967). The small-world problem. *Psychology Today*, *2*, 60–67.

Redner, S. (1998). How popular is your paper? an empirical study of the citation distribution. *European Physical Journal B*, *4*, 131–134.

Richardson, M., Agrawal, R., & Domingos, P. (2003). Trust management for the semantic web. *ISWC*.

Schwarz, G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, *6*, 461–464.

Stumpf, M. P. H., Wiuf, C., & May, R. M. (2005). Subnets of scale-free networks are not scale-free: Sampling properties of networks. *PNAS*, *102*.

Wasserman, S., & Pattison, P. (1996). Logit models and logistic regressions for social networks. *Psychometrika*, *60*, 401–425.

Watts, D. J., & Strogatz, S. H. (1998). Collective dynamics of 'small-world' networks. *Nature*, *393*, 440–442.

Wiuf, C., Brameier, M., Hagberg, O., & Stumpf, M. P. (2006). A likelihood approach to analysis of network data. *PNAS*, *103*, 7566–7570.