
Graph Model Selection using Maximum Likelihood

keyword1, keyword2, keyword3, keyword4

keyword1, keyword2, keyword3, keyword4

keyword1, keyword2, keyword3, keyword4

Abstract

In recent years, there has been a proliferation of theoretical graph models, e.g., preferential attachment and small-world models, motivated by real-world graphs such as the Internet topology. To address the natural question of which model is best for a particular data set, we propose a model selection criterion for graph models. Since each model is in fact a probability distribution over graphs, we suggest using Maximum Likelihood to compare graph models and select their parameters. Interestingly, for the case of graph models, computing likelihoods is a difficult algorithmic task. However, we design and implement MCMC algorithms for computing the maximum likelihood for four popular models: a power-law random graph model, a preferential attachment model, a small-world model, and a uniform random graph model. We hope that this novel use of ML will objectify comparisons between graph models.

1. Introduction

A plethora of random graph models have been proposed which reproduce and explain the various properties of interesting graphs, such as the Internet topology, the link structure of the World Wide Web, citation graphs, social networks and genetic networks in biological systems (Barabási & Albert, 1999), to name a few. Such models are useful both for understanding the nature of complex graphs as well as for simulations that require such graphs. In this paper,

we rank some of these models using Maximum Likelihood (ML). The mainstream approach to comparing models for these graphs has been somewhat subjective and very application dependent – comparisons are often based on the model’s capability of reproducing a set of properties observed in the real data, such as the power-law degree distribution or the small-world phenomenon (Medina et al., 2000; Bu & Towsley, 2002; Tangmunarunkit et al., 2002).

Each graph model is specified by a set of parameters and for a fixed parameter setting, the model induces a probability distribution over graphs. Hence it is natural to use ML to rank graph models for a given data set. In particular, we compare the probabilities assigned by the different models and choose the model (and parameters) that assigns the largest probability to actual graph data. This gives a more objective ranking than previous approaches in the sense that it is less property or application specific. However, applying ML in this context is surprisingly tricky.

While ML is a standard principle in model selection, application to graph models brings up two interesting issues. The first is definitional: graphs are abstract objects that are difficult to describe without imposing an ordering on nodes. The second difficulty is computational. For some models, calculating the likelihoods is easy, while for others we design novel algorithms based on Monte Carlo Markov Chains (MCMC). To illustrate the feasibility of our approach on existing models, we implement algorithms for computing ML estimates for four natural models: a power-law random graph model, a preferential attachment model, a small-world model, and a uniform random graph model. Based on experiments on three snapshots of the Internet topology graph, we find that the preferential attachment model ranks highest, while the uniform random graph model performs the worst. We note that our algo-

rithms are applicable to other real-world complex networks, including the protein interaction networks and citation networks.

We hope that this novel application of ML will enable a more objective model comparison and the development of improved models. As a byproduct, our algorithms estimate the best parameters of such models. The need for such an estimation procedure may arise when setting parameters for a simulator based on a given model.

2. Ranking graph models

We consider stochastic graph models, each of which induces a probability distribution over graphs. In particular, a model P assigns a probability $p(G)$ to every graph G . The quantity $p(G)$ is called the *likelihood*. We score a model by the *log-likelihood* $-\log P(G)$, which is a more tractable quantity. The Maximum Likelihood Estimate prefers the model with the largest likelihood (and hence the smallest log-likelihood).

One issue that arises in such a scenario is that a graph is a single item and not a set of independent items drawn i.i.d. from some distribution, as is typically assumed in problems such as classification. It is unclear how to naturally divide data into independent training and test sets. Here the choice of likelihood is convenient, as it naturally measures how well a model *predicts* an entire data set. (Certain models attempt to *explain* how the data set could have originated. We note that our approach scores the predictive power of the model, not its explanatory power.)

A second issue is node ordering. In our datasets each vertex is specified by a unique number from the set $\{1, \dots, n\}$ but typically these labels do not bear any information relevant to the generating process. Therefore the models we consider are symmetric in the sense that they generate all vertex labelings (each of the $n!$ permutations of vertex labels) of a given graph with the same probability. Any graph model can easily be symmetrized by appending a permutation step: after the nodes and edges are generated in some order, the nodes are randomly permuted. So, if the original model had probability distribution p , the new distribution would be p' with $p'(G) = \frac{1}{n!} \sum_{\pi} p(\pi(G))$, where the sum is over all permutations of the n nodes in the graph. For a given permutation π , $p(\pi(G))$ can typically be computed easily, but computing $p'(G)$, which is the average of $p(\pi(G))$ over exponentially many permutations π , is much harder. This is where some of our new ideas for MCMC-based sampling are used.

Comparisons can be drawn between language models,

which are probability distributions over sequences of words, and graph models. Language models are most often ranked by log-likelihood (or equivalently, *cross-entropy* or *perplexity* (Chen et al., 1998)). The related Minimum Description Length principle (Rissanen, 1978) scores models by the log-likelihood, plus the description size of the model¹. Interestingly, computing values for some models, like maximum entropy models, involves Monte Carlo Markov Chain (MCMC) methods similar to the ones we use (Chen & Rosenfeld, 1999).

Language models are similarly tested on large text corpora. However, the models typically have many thousands of parameters and hence their description size cannot be ignored. As a result, they are often tested on independent held-out data. If they had as few parameters as our models, there would be no need for an independent test set, as little overfitting is possible.

3. Preliminaries

We want to generate a directed graph $G = (V, E)$ with vertex set V and edge set E , where $n = |V|$ is the number of vertices and $m = |E|$ is the number of edges. For a node v , let $\mathbf{in}(v)$ be its indegree, i.e., the number of edges pointing into that node and $\mathbf{out}(v)$ be its outdegree, i.e., the number of edges pointing out of that node.

We consider models which represent different streams of graph modeling research. We include a power-law model (PRG), a preferential attachment model (PA), a small-world model (SW), and, for comparison, the Erdős-Rényi (ER) random graph model. To be precise, let us define a power-law probability distribution with exponent β and cutoff c (possibly ∞) to be the distribution over integers that generates $i \in \{1, 2, \dots, c\}$ with probability $i^{-\beta} / \sum_{j=1}^c j^{-\beta}$.

Many graph models are unrealistic in the sense that they would assign probability 0 to most real-world graphs. In most cases, slight variations on these models are more appealing in that they retain the essential features of the model while assigning positive probability to every graph. We have modified the PA model and the SW model for this purpose.

- **PA model.** The PA model, inspired by (Mitzenmacher, 2001), simulates a graph growth process where incoming nodes connect to existing nodes with probability proportional to their degree. It

¹We note that in case of the graph models, which are usually very simple (with only few parameters), the two notions are practically equivalent.

is parameterized by probabilities p, q satisfying $p + q < 1$, and a parameter $\gamma > 0$. The graph is created iteratively. We start with a single vertex. In each iteration $i = 2, \dots, n$ a random vertex “appears” and edges between the new vertex and already existing nodes are generated by the following process. With probability p (resp. q) an edge from (resp. to) the new vertex is created and the other end-point v is chosen with probability proportional to $\mathbf{in}(v) + \gamma$ (resp. $\mathbf{out}(v) + \gamma$). The edge-generating process is then repeated (possibly resulting in multi-edges). Otherwise, with probability $1 - p - q$ the process stops and new iteration $i + 1$ begins. It can be shown that this model produces with high probability a graph whose indegree and outdegree distributions are governed by power laws with exponents $\beta_{\mathbf{in}}$ and $\beta_{\mathbf{out}}$, where $\beta_{\mathbf{in}}$ and $\beta_{\mathbf{out}}$ are functions of p, q and γ .

- **PRG model.** This model is based on the model for random graphs with power-law degree distribution proposed by Aiello, Chung and Lu (Aiello et al., 2000), which is a special case of the model for random graphs with a given degree sequence due to Bollobás (Bollobás, 1985). The motivation for this model is that many real-world networks have been observed to exhibit power-law distribution of degrees (Barabási & Albert, 1999; Siganos et al., 2003). Input parameters are $\beta_{\mathbf{in}}, \beta_{\mathbf{out}}$, the intended power-law exponents, and cutoffs $c_{\mathbf{in}}, c_{\mathbf{out}}$. First we generate the indegrees and outdegrees of each of the n vertices independently at random according to power-law distributions with exponents $\beta_{\mathbf{in}}$ and $\beta_{\mathbf{out}}$, respectively, and maximum indegrees and outdegrees $c_{\mathbf{in}}$ and $c_{\mathbf{out}}$, respectively. Then we connect the vertices randomly to match the selected indegrees and outdegrees.

To be precise, we need to specify exactly how the vertices are matched. If the sum of the indegree sequence equals the sum of the outdegree sequence, we connect the vertices as follows. For every vertex, we create $\mathbf{in}(v)$ copies of v : $v_1, \dots, v_{\mathbf{in}(v)}$. Let A denote the set of all these vertex copies. Similarly, for out-degrees we create a set B . We then pair up the vertices, one to one, randomly from these two sets. For each pair $v_i \in A$ and $u_j \in B$, we create an edge from v to u in the original graph. This process creates a directed graph (possibly a multi-graph). If the sums of the indegrees and outdegrees do not equal, we output the empty graph (a graph with n nodes and no edges).²

²It can be shown that the log-likelihoods assigned by

- **SW model.** Many real-world networks have been observed to have low diameter and high clustering co-efficient (Bu & Towsley, 2002; Barabási et al., 2000). This is known as the small-world phenomenon. Our model for graphs exhibiting the small-world phenomenon is inspired by the Watts-Strogatz and Kleinberg models (Watts & Strogatz, 1998; Kleinberg, 2000). The basic idea is to add random links to an underlying well-structured graph, in our case the grid. The parameters of our model are s , the side of an underlying grid topology, and constants α, β . There are $s \times s$ vertices arranged in a grid. For every pair of vertices u, v , an edge from u to v is added with probability $\alpha \text{dist}(u, v)^{-\beta}$, where $\text{dist}(u, v)$ is the Manhattan distance on the grid, i.e., $|u_x - v_x| + |u_y - v_y|$. All vertices with no incident edges are omitted.
- **ER model.** The Erdős-Rényi model is parameterized by a probability $p \in [0, 1]$. For every pair of vertices u, v an edge from u to v appears independently with probability p .

The PA and PRG models can technically produce graphs with multiple copies of a single edge, which do not occur in our data sets. We have two options - to consider all such graphs with multiple copies of an edge as equivalent to the corresponding simple graph, or to distinguish them. We choose the second option, i.e., interpret our data sets as multigraphs, because it makes the probability calculation significantly easier. Choosing the first option would only increase the scores of the PA and PRG models in our applications, and hence would not affect the rankings of these models relative to the other models.

4. Algorithms

In this section, for each of the models, we present an algorithm which estimates the corresponding likelihood of a given directed graph G . The input to our algorithms is the graph G together with a parameter setting of the model. The output of our algorithms is the likelihood, i.e. the negative logarithm of the probability that the model generates G . To find the best likelihood, we also need to search for the optimal settings of the parameters. Due to space constraints, we defer the proof of the correctness of the algorithms to the full version of the paper.

this model and the more natural model which generates the degree sequences until the sums match are almost identical (they differ only in the least significant digits). The advantage of our model is that we can calculate the likelihood exactly.

4.1. ER algorithm

Recall that n is the number of nodes and m is the number of edges in G . Therefore the number of non-edges is $n(n-1) - m$ and the probability that the ER model generates G is $\Pr(G) = p^m(1-p)^{n(n-1)-m}$. In this model we can not only compute the probability exactly but also compute the best parameter $p = \frac{m}{n(n-1)}$, which maximizes the above probability.

4.2. PRG algorithm

The algorithm for computing the probability of a graph according to the PRG model is given in Figure 1. Fixing the parameters $\beta_{\text{in}}, \beta_{\text{out}}, c_{\text{in}}, c_{\text{out}}$, the probability that the PRG model generates G can be computed as the product of the probability of generating the in-degree sequence, the probability of generating the out-degree sequence, and the probability of matching up the right vertices.

PRG($G, \beta_{\text{in}}, \beta_{\text{out}}, c_{\text{in}}, c_{\text{out}}$)

- $Z_{\text{in}} := \sum_{d=1}^{c_{\text{in}}} d^{-\beta_{\text{in}}}$, similarly for Z_{out} .
 - $P_{\text{in}} := \prod_{v \in V} \frac{(\text{in}(v))^{-\beta_{\text{in}}}}{Z_{\text{in}}}$, similarly for P_{out} .
 - return $P_{\text{in}} P_{\text{out}} \frac{1}{m!} \prod_{v \in V} \text{in}(v)! \text{out}(v)!$
-

Figure 1. PRG model computation

In the PRG model a vertex gets assigned an in-degree $d \in \{0, \dots, c_{\text{in}}\}$ with probability $d^{-\beta_{\text{in}}}/Z_{\text{in}}$, where $Z_{\text{in}} = \sum_{d=1}^{c_{\text{in}}} d^{-\beta_{\text{in}}}$ is the normalization factor. Therefore, the probability of generating the particular indegree sequence is P_{in} , as defined in Figure 1. For a graph with no multiple edges, the probability of any particular matching can be shown to be $\frac{1}{m!} \prod_{v \in V} \text{in}(v)! \text{out}(v)!$. We cannot easily compute the optimal parameters $\beta_{\text{in}}, \beta_{\text{out}}, c_{\text{in}}, c_{\text{out}}$, but, since the computation is fast, we quickly search the (appropriately discretized) parameter space.

4.3. PA algorithm

Given a permutation of the vertices π (the order of their appearance), it is not difficult to compute the probability that the PA model generates G : $\Pr(G|\pi) =$

$$\prod_{i=2}^n d_{\pi}(i)! r \prod_{\substack{j < i: \\ (\pi_j, \pi_i) \in E}} p \frac{\text{in}_{\pi}^{\pi}(\pi_j) + \gamma}{m_{\pi}^{\pi}} \prod_{\substack{j < i: \\ (\pi_i, \pi_j) \in E}} q \frac{\text{out}_{\pi}^{\pi}(\pi_j) + \gamma}{m_{\pi}^{\pi}},$$

where,

$$\begin{aligned} \text{in}_{\pi}^{\pi}(v) &= |\{j : j < i \wedge (\pi_j, v) \in E\}| \\ \text{out}_{\pi}^{\pi}(v) &= |\{j : j < i \wedge (v, \pi_j) \in E\}| \\ m_{\pi}^{\pi} &= \sum_{k=1}^{i-1} (\text{in}_{\pi}^{\pi}(\pi_k) + \gamma) = \sum_{k=1}^{i-1} (\text{out}_{\pi}^{\pi}(\pi_k) + \gamma) \\ d_{\pi}(i) &= \text{in}_{\pi}^{\pi}(\pi_i) + \text{out}_{\pi}^{\pi}(\pi_i) \\ r &= 1 - p - q \end{aligned}$$

In words, $\text{in}_{\pi}^{\pi}(v)$ and $\text{out}_{\pi}^{\pi}(v)$ are the in- and out-degrees of vertex v just *before* the i th node appears and $d_{\pi}(i)$ is the number of edges incident to the i th node generated before the $(i+1)$ st iteration begins (before the $(i+1)$ st node appears). These edges can be generated in any of the $d_{\pi}(i)!$ orders. An edge from π_i to π_j (for $j < i$) is generated with probability $q(\text{in}_{\pi}^{\pi}(\pi_j) + \gamma)/m_{\pi}^{\pi}$ and an edge from π_j to π_i appears with probability $p(\text{out}_{\pi}^{\pi}(\pi_j) + \gamma)/m_{\pi}^{\pi}$. Finally, with probability r we move to the next iteration. This justifies the above expression for $\Pr(G|\pi)$.

The computation of $\Pr(G|\pi)$ is relatively straightforward. However, as discussed earlier, we need to compute $\Pr(G) = \frac{1}{n!} \sum_{\pi} \Pr(G|\pi)$. Obviously, summing over all $n!$ permutations is not feasible and traditional sampling methods (pick several permutations at random and average the corresponding $\Pr(G|\pi)$'s) do not approximate the sum well as the variance of the $\Pr(G|\pi)$ can be huge. Our algorithm, summarized in Figure 2, is based on a MCMC method.

Each Self-iter procedure uses a random walk to compute the conditional probability $\Pr(\sigma_i|G, \sigma_1 \dots \sigma_{i-1})$. The procedure walks on the space of all permutations with the first $i-1$ elements fixed to $\sigma_1, \dots, \sigma_{i-1}$, where the stationary distribution of a permutation π is proportional to $\Pr(G|\pi)$. By Bayes' rule,

$$\begin{aligned} \Pr(G) &= \frac{\Pr(G|\sigma) \Pr(\sigma)}{\prod_{i=1}^n \Pr(\sigma_i|G, \sigma_1 \dots \sigma_{i-1})} \\ &= \frac{\Pr(G|\sigma)}{n! \prod_{i=1}^n \text{Self-iter}(G, \sigma, i, p, q, \gamma, T)} \end{aligned}$$

To get some intuition for the walk we use, imagine trying to shuffle a deck of n cards uniformly at random. The natural random walk to do (and the first one we tried) would be to pick an arbitrary pair of adjacent cards and swap them. Unfortunately, this would require approximately $\theta(n^3)$ steps until it is approximately random, because each card would need to be swapped $\theta(n^2)$ times to be in a random place. In contrast, imagine picking a random card, removing it, and reinserting it in a random place in the deck. The cards will be random as soon as each one has

 PA(G, p, q, γ, T)

- Let σ be the permutation of vertices sorted by the total degree $\mathbf{in}(v) + \mathbf{out}(v)$ in decreasing order.
 - return $\frac{\Pr(G|\sigma)}{n! \prod_{i=1}^n \text{Self-iter}(G, \sigma, i, p, q, \gamma, T)}$.
-

 Self-iter($G, \sigma, i, p, q, \gamma, T$) // est. $\Pr(\sigma_i|G, \sigma_1 \dots \sigma_{i-1})$.

- $\pi := \sigma$
 - est := 0
 - for $t=1$ to T
 - $v :=$ random vertex from $\{\sigma_i, \sigma_{i+1}, \dots, \sigma_n\}$
 - for $j \in \{i, i+1, \dots, n\}$,
let $a[j] := \Pr(G|\text{Shift}(\pi, v, j))$
 - choose j at random from $\{i, \dots, n\}$ with probability $a[j] / \sum_{k=i}^n a[k]$.
 - $\pi := \text{Shift}(\pi, v, j)$ // random walk step
 - for $j \in \{i, i+1, \dots, n\}$,
let $b[j] := \Pr(G|\text{Shift}(\pi, \sigma_i, j))$
 - est := est + $b[i] / \sum_{k=1}^n b[k]$ // estimation
 - return $\frac{\text{est}}{T}$.
-

 Shift(π, v, j) is the permutation obtained from π by inserting v at position j .

Figure 2. PA model computation

been touched once, which happens in approximately $\theta(n \log n)$ steps. While the steps in the latter case take $O(n)$ times longer than the steps in the first case, the second algorithm is still an order faster. Since n is large, and the algorithms are slow, this difference amounts to hundreds of hours of computation.

To search for the optimal parameters, we first found good candidates and then we searched the parameter space close to the candidate values. In particular, we noticed that for our data sets $\Pr(G|\sigma)$ is quite close to $\Pr(G)$ (their logarithms are within 5% of each other). Since the computation of $\Pr(G|\sigma, p, q, \gamma)$ is fast, we could quickly find the (candidate) values of the parameters that minimize this probability.

There are several important details to discuss. First of all, in the algorithm we do not recompute $\Pr(G|\text{Shift}(\pi, v, j))$ from scratch. Instead, such calculations can be quickly computed incrementally from the value $\Pr(G|\pi)$. This is essential to the efficiency of the algorithm. Second, we do not do estimation every

step but rather every ten steps. Since consecutive steps are dependent, the benefit of estimating *every* step is not worth the cost. And, third, the algorithm can be easily parallelized by computing, separately for each i , $\text{Self-iter}(G, i, p, q, \gamma, T)$.

4.4. SW algorithm

We do not know how to accurately (and in a reasonable time) estimate the correct likelihood by the SW model, rather we bound it from below and from above. This information is sufficient to conclude that the SW model ranks worse than the PA and PRG models but not worse than the ER model. Notice that the likelihood of the SW model is at least as big as the likelihood of the ER model. In particular, setting $\beta = 0$ gives exactly the ER model with parameter $p = \alpha$. It remains to bound the likelihood by the SW model from above.

We attempt to use a similar MCMC approach as for the PA model. If we knew the initial grid alignment of the vertices g , then computing the probability of generating G from g would be straightforward:

$$\Pr(G|g) = \prod_{(u,v) \in E} \alpha \text{dist}(u,v)^{-\beta} \prod_{(u,v) \notin E} (1 - \alpha \text{dist}(u,v)^{-\beta}),$$

where $\text{dist}(u,v) = |u_x^g - v_x^g| + |u_y^g - v_y^g|$ is the Manhattan-distance of vertices u and v in g , and (v_x^g, v_y^g) are the coordinates of vertex v in g . As before, we want to compute $\Pr(G) = \frac{1}{n!} \sum_g \Pr(G|g)$ where the sum ranges over all possible grid alignments. An MCMC approach analogous to the one presented for the PA model, a (self-reducible) walk on the grid alignments with stationary distribution proportional to $\Pr(G|g)$, converges to the correct likelihood as the number of steps goes to infinity. Unfortunately, this walk does not mix rapidly, both in practice and theoretically (consider two cliques of size $n/2$, they will position themselves on opposite sides of the grid and it will take exponentially long for them to switch).

In lieu of estimating this probability, we upper bound the probability as follows,

$$\Pr(G) = \frac{1}{n!} \sum_g \Pr(G|g) \leq \max_g \Pr(G|g)$$

In other words, the probability given the best grid is an upper bound on the average probability over all grids.

To find the best grid, we use Simulated Annealing (Kirkpatrick et al., 1983), a well-known optimization

method. It searches among a set of states S with a real-valued energy function $c(s)$. It consists of several phases, each characterized by a temperature T . For a given T a Markov chain is run with stationary distribution proportional to $e^{-c(s)/T}$. Starting with a high temperature, for which the distribution is nearly uniform, the temperature is progressively decreased, biasing the distribution towards low-energy states.

In our case, S is the set of all grid alignments and $c(g) = -\ln \Pr(G|g)$. The stationary distribution at temperature T is proportional to $\Pr(G|g)^{1/T}$. The transitions are swaps of arbitrary grid vertices. The algorithm, specified in Figure 3, is a standard annealing algorithm with initial temperature T_0 and a geometric cooling schedule.

```

SW( $G, \alpha, \beta, T_0, R, \delta$ )  // Finds  $\max_g \Pr(G|g)$ .

1.  $T := T_0$   // temperature
2.  $g :=$  random initial assignment to the grid.
3. repeat:  // until convergence of  $\Pr(G|g)$ .
    (a)  $r := 0$   // step counter
    (b) Let  $u$  and  $v$  be two vertices on grid  $g$ . Let  $g'$ 
        be the grid  $g$  with  $u$  and  $v$  swapped.
    (c) with probability  $\max \left\{ 1, \left( \frac{\Pr(G|g')}{\Pr(G|g)} \right)^{1/T} \right\}$ , do:
        •  $g := g'$ 
        •  $r := r + 1$ 
        • if  $r > R$ , then  $r := 0$ ;  $T := T(1 - \delta)$ 
    
```

Figure 3. SW model computation

We use brute force search over the (discretized) parameter space to find the optimal parameters α, β .

5. Experiments

We ran our algorithms on three publicly available snapshots of the AS-level Internet topology, data from 1997, 1999, and 2001 (NLNR, 2001). These data sets have 3,117, 6,266, and 11,080 vertices and 6,024, 13,681, and 25,485 edges, respectively. As suggested by (Gao, 2001), for each year we combined five snapshots taken in a short time span into one representative snapshot. A histogram, including error estimate (a rather large uncertainty region) for SW, is shown in Figure 4. Figure 5 describes the raw results of our algorithms, together with the optimal parameters. The numbers are reflective of the corresponding log-likelihoods divided by the number of edges in the data sets.

As can be seen in the histogram, all three data sets produced the same ranking of models. This result is more meaningful than a comparison based on a single data set alone. In all three cases, the PA model placed first, closely followed by the PRG model. The two models both attempt to reproduce the power-law property. In the PRG, it is imposed directly, while in PA, it arises naturally. Nonetheless, the two scored similarly.

The SW model ranked third. With care, comparisons between models can be interpreted as statements about the relative importance of graph properties. In this case, one could conclude that the power-law degree distribution is a more significant predictor of Internet-topology data than the small-world phenomenon.

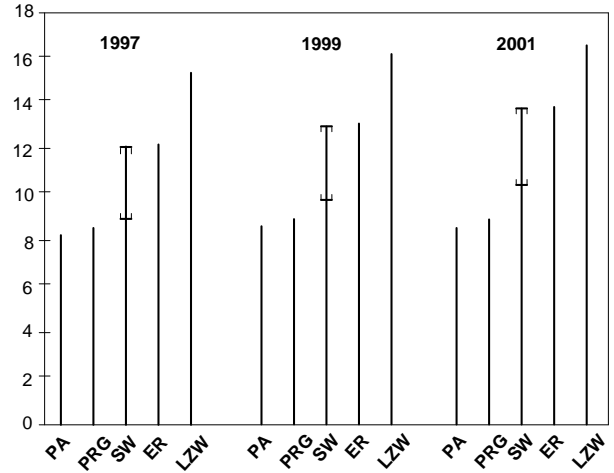


Figure 4. The log-likelihoods (per edge) of the data sets using the four models. The SW model has an uncertainty region, but we can still rank it with respect to all the others.

5.1. PA implementation

We estimate convergence time heuristically and conservatively. To upper-bound time T needed for our estimator to converge to the true value. We choose a value $T > 100,000$ so that ten independent executions of the same Self-iter converge to the same number x in time $t/4$ and for the next $3t/4$ steps the value remains within ϵ of x , for a small ϵ . We do this for a sampling of the values $i \in \{1, 2, \dots, n\}$. As expected, fewer steps T were required for convergence for later (larger i) Self-iter's.

To verify the correctness of our estimates, we first ran the algorithm with the target permutation having nodes sorted in order of increasing total degree. The difference in likelihoods was an insignificant .1%.

1997:	<table border="1"> <tr> <th>PA</th><th>PRG</th><th>SW</th><th>ER</th></tr> <tr> <td>8.30</td><td>8.60</td><td>8.96</td><td>12.10</td></tr> <tr> <td>$p = 0.58$</td><td>$\beta^{\text{in}} = 1.55$</td><td>$\alpha = 0.111$</td><td>$p = 6.2\text{e-}4$</td></tr> <tr> <td>$q = 0.08$</td><td>$\beta^{\text{out}} = 2.39$</td><td>$\beta = 1.9$</td><td></td></tr> <tr> <td>$\gamma = 0.5$</td><td>$c^{\text{in}} = 610$</td><td></td><td></td></tr> <tr> <td></td><td>$c^{\text{out}} = 69$</td><td></td><td></td></tr> </table>	PA	PRG	SW	ER	8.30	8.60	8.96	12.10	$p = 0.58$	$\beta^{\text{in}} = 1.55$	$\alpha = 0.111$	$p = 6.2\text{e-}4$	$q = 0.08$	$\beta^{\text{out}} = 2.39$	$\beta = 1.9$		$\gamma = 0.5$	$c^{\text{in}} = 610$				$c^{\text{out}} = 69$				
PA	PRG	SW	ER																								
8.30	8.60	8.96	12.10																								
$p = 0.58$	$\beta^{\text{in}} = 1.55$	$\alpha = 0.111$	$p = 6.2\text{e-}4$																								
$q = 0.08$	$\beta^{\text{out}} = 2.39$	$\beta = 1.9$																									
$\gamma = 0.5$	$c^{\text{in}} = 610$																										
	$c^{\text{out}} = 69$																										
1999:	<table border="1"> <tr> <th>PA</th><th>PRG</th><th>SW</th><th>ER</th></tr> <tr> <td>8.55</td><td>8.83</td><td>9.76</td><td>12.93</td></tr> <tr> <td>$p = 0.61$</td><td>$\beta^{\text{in}} = 1.57$</td><td>$\alpha = 0.092$</td><td>$p = 3.5\text{e-}4$</td></tr> <tr> <td>$q = 0.08$</td><td>$\beta^{\text{out}} = 2.44$</td><td>$\beta = 1.8$</td><td></td></tr> <tr> <td>$\gamma = 0.4$</td><td>$c^{\text{in}} = 1410$</td><td></td><td></td></tr> <tr> <td></td><td>$c^{\text{out}} = 172$</td><td></td><td></td></tr> </table>	PA	PRG	SW	ER	8.55	8.83	9.76	12.93	$p = 0.61$	$\beta^{\text{in}} = 1.57$	$\alpha = 0.092$	$p = 3.5\text{e-}4$	$q = 0.08$	$\beta^{\text{out}} = 2.44$	$\beta = 1.8$		$\gamma = 0.4$	$c^{\text{in}} = 1410$				$c^{\text{out}} = 172$				
PA	PRG	SW	ER																								
8.55	8.83	9.76	12.93																								
$p = 0.61$	$\beta^{\text{in}} = 1.57$	$\alpha = 0.092$	$p = 3.5\text{e-}4$																								
$q = 0.08$	$\beta^{\text{out}} = 2.44$	$\beta = 1.8$																									
$\gamma = 0.4$	$c^{\text{in}} = 1410$																										
	$c^{\text{out}} = 172$																										
2001:	<table border="1"> <tr> <th>PA</th><th>PRG</th><th>SW</th><th>ER</th></tr> <tr> <td>8.58</td><td>8.85</td><td>10.42</td><td>13.68</td></tr> <tr> <td>$p = 0.63$</td><td>$\beta^{\text{in}} = 1.57$</td><td>$\alpha = 0.088$</td><td>$p = 2.1\text{e-}4$</td></tr> <tr> <td>$q = 0.07$</td><td>$\beta^{\text{out}} = 2.5$</td><td>$\beta = 1.8$</td><td></td></tr> <tr> <td>$\gamma = 0.3$</td><td>$c^{\text{in}} = 2421$</td><td></td><td></td></tr> <tr> <td></td><td>$c^{\text{out}} = 214$</td><td></td><td></td></tr> </table>	PA	PRG	SW	ER	8.58	8.85	10.42	13.68	$p = 0.63$	$\beta^{\text{in}} = 1.57$	$\alpha = 0.088$	$p = 2.1\text{e-}4$	$q = 0.07$	$\beta^{\text{out}} = 2.5$	$\beta = 1.8$		$\gamma = 0.3$	$c^{\text{in}} = 2421$				$c^{\text{out}} = 214$				
PA	PRG	SW	ER																								
8.58	8.85	10.42	13.68																								
$p = 0.63$	$\beta^{\text{in}} = 1.57$	$\alpha = 0.088$	$p = 2.1\text{e-}4$																								
$q = 0.07$	$\beta^{\text{out}} = 2.5$	$\beta = 1.8$																									
$\gamma = 0.3$	$c^{\text{in}} = 2421$																										
	$c^{\text{out}} = 214$																										

Figure 5. Raw data results of the four algorithms on datasets (not including error bars). Bold numbers represent log-likelihood per edge, the rest are the best parameter values. Note: SW only computes a lower bound on the log-likelihood.

Second, to ensure that estimates in each phase were converging, for a sampling of the Self-iter's, we ran the walks for significantly longer and the values remained within .1%. Note that margin of error of the total estimate is actually lower than the margin of error of any individual Self-iter. This is because the total is the sum of thousands of independent random variables. Hence, as long as each estimate has the correct expectation, the margin of error of the total estimate will be significantly smaller (in percentage).

We ran the Self-iter steps in parallel. To do this we used Condor (CONDOR, 2005) and the University of Chicago Condor pool consisting of about 70 2-GHz computers, out of which about 35-50 were available at a time. Our computations for the 1997 data took around 40 hours total, compared with twice as long for the 1999 data and about a week for the 2001 data. However, as discussed above, we were very conservative in our stopping heuristics.

5.2. SW implementation

For the SW algorithm, we used $\delta = 0.01$ and $R = 1,000,000$. We chose a random grid alignment of vertices as a starting state of our algorithm. The algorithm converged in a few hours. Figure 6 graphically shows the results of annealing on the largest data set.

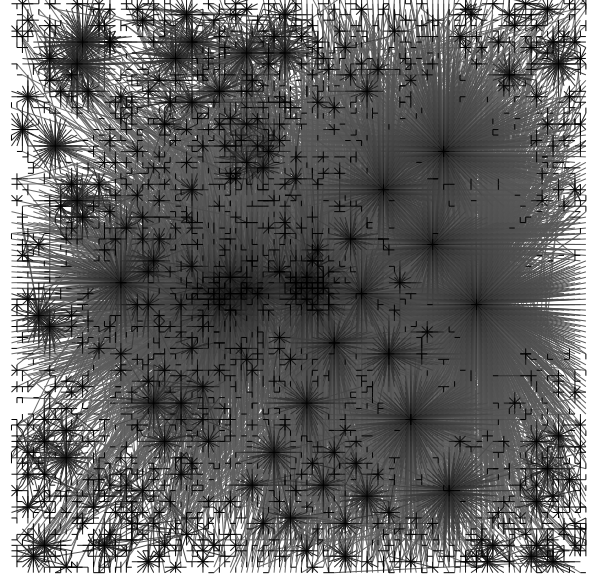


Figure 6. The result of the simulated annealing on the 2001 snapshot.

For the SW model, our ranking depends crucially on the ability of simulated annealing to come close to the true maximum $\Pr(G|g)$. We performed this test to validate the SW algorithm: We took the simple grid graph, where each interior node has degree four. We permuted the nodes randomly and ran a search for the best configuration. As seen in Figure 7, the final answer is not perfect but very close. The error in final likelihood was less than 1%. For practical purposes, we conclude that the likelihood assigned by the SW model is between 99% of the raw score of SW (which computes a bound on the SW model's performance) and the score of ER, which can never be better.

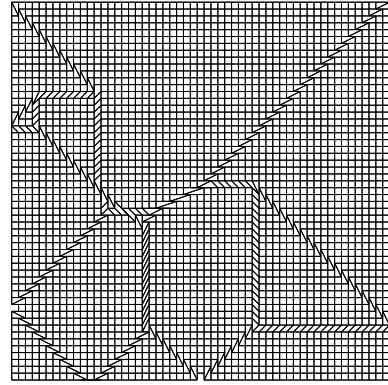


Figure 7. The results of simulated annealing on a grid graph.

6. Conclusions and future work

We have proposed using ML for ranking graph models, which involves computing the likelihood of real-world graphs with respect to each model. In contrast with previously used approaches which typically estimate the model's ability to reproduce a set of observed properties, ML is a universal criterion and can be used to rank models in an objective manner.

We have designed and implemented algorithms computing likelihoods for four natural random graph models from various parts of the modeling spectrum. We designed algorithms based on a variety of techniques, ranging from simple to complex, including an MCMC-based algorithm and Simulated Annealing, for computing the probability assigned to a graph. While the algorithms we use are not always simple, we have demonstrated the feasibility of applying the approach even to models which were not designed with this metric in mind.

For future work, one would like to design improved models for a variety of real-world data sets. Recent fast MCMC (Lovász & Vempala, 2003) techniques show a promising approach which could yield algorithms applicable to a wide range of models, and ultimately help to design better scoring models.

It is also interesting to compare our work with related work on compressing the linkage information on the web (Adler & Mitzenmacher, 2001; Bharat et al., 1998). Log-likelihood and compression are closely related, since the log-likelihood of a data set with respect to a model is the measure of how well the data can be compressed using the model (via arithmetic coding). For compression, better performance is achieved using special vertex orderings, such as when the data is sorted by URL. Since we are comparing graph models only, we take a random ordering over nodes and no URL information. However, the ML estimate could be applied to a wide range of models with other information as well.

Finally, ML could also be used for the interesting problem of predicting links in social networks (Kubica et al., 2003), and interactions in various cellular networks (Barabási & Oltvai, 2004; Bork et al., 2004).

References

- Adler, M., & Mitzenmacher, M. (2001). Towards compressing web graphs.
- Aiello, W., Chung, F., & Lu, L. (2000). A random graph model for massive graphs. *Proceedings of ACM Symposium on Theory of Computing*.
- Barabási, A., & Albert, R. (1999). Emergence of scaling in random networks. *Science*, 286, 509–512.
- Barabási, A. L., Albert, R., & Jeong, H. (2000). Scale-free characteristics of random networks: topology of the world-wide web. *Physica A*, 281, 69.
- Barabási, A.-L., & Oltvai, Z. (2004). Network biology: Understanding the cells functional organization. *Nature Reviews Genetics*, 5, 101–113.
- Bharat, K., Broder, A., Henzinger, M., Kumar, P., & Venkatasubramanian, S. (1998). The connectivity server: fast access to linkage information on the web. *Proceedings of the 7th World Wide Web Conference*.
- Bollobás, B. (1985). *Random graphs*. Academic Press.
- Bork, P., Jensen, L., von Mering, C., Ramani, A., Lee, I., & Marcotte, E. (2004). Protein interaction networks from yeast to human. *Current Opinion in Structural Biology*, June 14(3), 292–9.
- Bu, T., & Towsley, D. F. (2002). On distinguishing between internet power law topology generators. *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Society (INFOCOM-02)* (pp. 638–647).
- Chen, S., Beeferman, D., & Rosenfeld, R. (1998). Evaluation metrics for language models. *DARPA Broadcast News Transcription and Understanding Workshop*.
- Chen, S., & Rosenfeld, R. (1999). Efficient sampling and feature selection in whole sentence maximum entropy language models. *Proceedings of ICASSP '99*.
- CONDOR (2005). University of Wisconsin at Madison. <http://www.cs.wisc.edu/condor/>.
- Gao, L. (2001). On inferring autonomous system relationships in the Internet. *IEEE/ACM Transactions on Networking*, 9, 733–745.
- Kirkpatrick, S., C.D.Gellatt, & Vecchi, M. (1983). Simulated annealing. *Science*, 220.
- Kleinberg, J. (2000). The small-world phenomenon: an algorithm perspective. *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing* (pp. 163–170).
- Kubica, J., Moore, A., Cohn, D., & Schneider, J. (2003). Finding underlying connections: A fast graph-based method for link analysis and collaboration queries. *Proceedings of Twentieth International Conference on Machine Learning*.
- Lovász, L., & Vempala, S. (2003). Simulated annealing in convex bodies and an $O^*(n^4)$ volume algorithm. *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computing* (pp. 650–).
- Medina, A., Matta, I., & Byers, J. (2000). On the origin of power laws in internet topologies. *Computer Communications Review*, 30, 18–28.
- Mitzenmacher, M. (2001). A brief history of generative models for power law and log normal distributions. *Proceedings of the 39th Annual Allerton Conference on Communication, Control, and Computing* (pp. 182–191).
- NLANR (2001). National Laboratory for Applied Network Research Routing data. <http://moat.nlanr.net/Routing/rawdata/>.
- Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, 14, 265–271.
- Siganos, G., Faloutsos, M., Faloutsos, P., & Faloutsos, C. (2003). Power laws and the AS-level internet topology. *IEEE/ACM Transactions on Networking*, 11, 514–524.
- Tangmunarunkit, H., Govindan, R., Shenker, S., Jamin, S., & Willinger, W. (2002). Network topology generators: Degree-based vs. structural. *Proceedings of ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (pp. 147–160).
- Watts, D., & Strogatz, S. (1998). Collective dynamics of 'small-world' networks. *Nature*, 393, 440–442.