Fast Generation of Large Scale Social Networks While Incorporating Transitive Closures

Joseph J. Pfeiffer III[1], Timothy La Fond[1], Sebastian Moreno[1], Jennifer Neville[1,2]
Departments of Computer Science[1] and Statistics[2], Purdue University, West Lafayette, IN
{jpfeiffer, tlafond, smorenoa, neville}@purdue.edu

Abstract*—A key challenge in the social network community is the problem of network generation—that is, how can we create synthetic networks that match characteristics traditionally found in most real world networks? Important characteristics that are present in social networks include a power law degree distribution, small diameter, and large amounts of clustering. However, most current network generators, such as the Chung Lu and Kronecker models, largely ignore the clustering present in a graph and focus on preserving other network statistics, such as the power law distribution. Models such as the exponential random graph model have a transitivity parameter that can capture clustering, but they are computationally difficult to learn, making scaling to large real world networks intractable. In this work, we propose an extension to the Chung Lu random graph model, the Transitive Chung Lu (TCL) model, which incorporates the notion transitive edges. Specifically, it combines the standard Chung Lu model with edges that are formed through transitive closure (e.g., by connecting a 'friend of a friend'). We prove TCL's expected degree distribution is equal to the degree distribution of the original input graph, while still providing the ability to capture the clustering in the network. The single parameter required by our model can be learned in seconds on graphs with millions of edges; networks can be generated in time that is *linear* in the number of edges. We demonstrate the performance of TCL on four real-world social networks, including an email dataset with hundreds of thousands of nodes and millions of edges, showing TCL generates graphs that match the degree distribution, clustering coefficients and hop plots of the original networks.

I. Introduction

Due to the prevalence of 'small world', 'power law' graphs such as Facebook and the World Wide Web [1], models that attempt to generate statistics frequently observed in these networks have become a much-discussed topic in the field ([2], [3], [4], [5], [6], [7]). The defining characteristics of many real-world networks include a power law degree distribution, high clustering, and a small diameter.

The first random graph algorithm, the Erdos-Renyi model[8], independently samples a set of random connections between nodes in the graph. However, the ER model produces a Binomial degree distribution, not power law, and generally lacks clustering when generating sparse networks. As a result, Exponential Random Graph Models (ERGMs [3]) were developed as an extension of the Erdos-Renyi model, in order to represent additional statistics of the graph as parameters. The typical ERGM approach is to model the network with a Markov independence assumption throughout the graph—edges are only dependent on other edges that share the same node. Using this assumption, ERGMs define an exponential family of models using various Markov statistics of the graph then algorithms to learn the model parameters maximize the likelihood of the input graph. The algorithms for learning and generating ERGMs are resource intensive, making them inpractical for application involving networks larger than a few thousand nodes.

More recent efforts on developing statistical models of networks make scalability an explicit goal when constructing models and sampling algorithms. Notable examples include the Chung-Lu Graph Model (CL) [4] and the Kronecker Product Graph Model (KPGM) [2]. CL is also an extension of the Erdos-Renyi model, but rather than creating a *summary* statistic based on the degrees, it generates a new graph such that the expected degree distribution matches the given distribution exactly. In [5], the authors modify the initial CL approach to allow generation in time linear to the number of edges, making generation of large scale networks tractable.

In contrast to both ERGM and CL, KPGM learns a small matrix of parameters (i.e., 2x2) and samples edges according to the Kronecker product of the matrix with itself $\log n$ times. This method has shown success in generating networks with a range of global properties including a power law degree distribution and path length distributions. For large graphs, the KPGM algorithm can learn the parameters defined by the 2x2 matrix in a few hours and once the parameters of the model are learned, large (new) graphs can be sampled in minutes [2].

The CL and KPGM algorithms offer scalable algorithms for learning and generating graphs with hundreds of thousands of nodes and millions of edges. However, in order to achieve scalability, a power law degree distribution and small diameter, both models have largely ignored the *clustering* in the network. This is not an insignificant limitation, as the seminal work of Wattz and Strogatz [1] show a small world network is in part defined by the fact that it exhibits a significant amount of clustering (greater than would be expected by random chance).

The term *clustering* itself can have different interpretations. Loosely, it means how connected 'grouped' sets of nodes in the graph are. Wattz and Strogatz [1] define the global clustering coefficient as the ratio between the number of triangles to the possible number of triangles given the current wedges (paths of length two) in the graph. Similarly, the local clustering coefficient is the ratio of the number of triangles incident to a node compared to the number of wedges centered on a that node—a distribution of large local clustering coefficients implies a large global clustering coefficient. More recent work has focused on the notion of *conductance communities*, defining clustering in terms of how likely a random walk on a network is to leave or enter a set of nodes. Gleich and Seshadhri [9] recently proved that a large global clustering coefficient coupled with a heavy tailed degree distribution implies the existence of communities with good conductance scores; thus, a method which accurately generates the distribution of local clustering coefficients and models the degree distribution implicitly prdoduces networks with communities having good conductance scores.

In order to generate sparse networks which accurately capture the degree distribution, small diameter, *and* the distribution of local clustering coefficients, we propose to extend the CL algorithm in multiple ways. The first portion of this paper will show the naive fast generation algorithm for the CL model is biased, and we outline a correction to this problem. Next, we introduce a generalization to the CL model known as the *Transitive Chung Lu* (TCL). To do this, we observe the CL algorithm corresponds to a 'random surfer' style model, similar to the PageRank random walk algorithm [10]. However, while PageRank models the probability of following an edge *versus* randomly surfing, the CL generation algorithm always chooses the random surfer—thus it has no affinity for connecting nodes along transitive edges. Based on this key insight, our TCL model extends the CL algorithm by sometimes choosing to follow transitive paths and form an edge to close a triangle rather than transitioning to a random node. The probability of randomly surfing versus closing a triangle is a single parameter in our model which we can learn in *seconds* from an observed graph. The contributions of our work can be summarized as follows:

- Introduction of a 'transitive closure' parameter to the CL model.
- A correction to the 'edge collision' problem that occurs in the naive fast CL generation algorithm.
- Analysis showing TCL has an approximate expected degree distribution equal to the original input network's degree distribution.
- A learning algorithm for TCL that runs in seconds for graphs with millions of edges.
- A TCL generation algorithm that runs on the same order as the fast CL algorithm, and faster than the KPGM sampling algorithm.
- Empirical demonstration that the graphs generated by TCL match the degree and clustering coefficient distributions of the input graph more accurately than CL or KPGM.

In section II we discuss in more depth the ERGM, KPGM and CL models, while in section III we outline the basic notations used in this paper, as well as the CL model. Next, we show the fast method used for generating graphs in section IV, and our correction to it. In section V we introduce our modification to the CL model, proving the expected degree distribution and demonstrating how to learn the transitive probability, as well as analying the runtimes of our fast CL correction and TCL. We learn parameters for real world graphs and generate synthetic networks closely resembling the original graphs in section VI, ending in section VII with conclusions and future directions.

II. Related Work

Recently a great deal of work has focused on developing generative models for small world and scale-free graphs (e.g., [11], [1], [6], [12], [3], [2], [4]). As an example, the Chung Lu algorithm is able to generate a network with a provable expected degree distribution equal to the degree distribution of the original graph. The CL algorithm, like many, attempts to define a process which matches a subset of features observed in a network.

The importance of the clustering coefficient has been demonstrated by Watts and Strogatz [1]. In particular, they show that small world networks (including social networks) are characterized by a short path length and large clustering coefficient. One recent algorithm by Seshadri et al [7] matches these statistics by putting together nodes with similar degrees and generating Erdos-Renyi graphs for each group. The groups are then tied together. However, this algorithm has two parameters that must be set manually, giving no option for learning individual models for individual networks.

One method that models clustering and learns the associated parameters is the Exponential Random Graph Model (ERGM; [3]). ERGMs define a probability distribution over the set of possible graphs with a log-linear model that uses feature counts of local graph properties as summary statistics on a network. However, these models are hard to train as each update of the Fisher scoring function takes $O(n^2)$. With real-world networks numbering in the hundreds of thousands or millions of nodes, ERGMs quickly become impossible to fit.

Another method is the Kronecker product graph model (KPGM; [2]), a scalable algorithm for learning models of large-scale networks that empirically preserves a wide range of global properties of interest, such as degree distributions and path-length distributions. Due to these desirable characteristics, the KPGM is considered a state-of-the art algorithm for generation of large networks. As a result, it was selected as a generation algorithm for the Graph 500 Supercomputer Benchmark [5]. The KPGM starts with a initial square matrix Θ_1 of size $b \times b$, where each cell value is a probability. To generate a graph, the algorithm uses k Kronecker multiplications to grow until a determined size (obtaining Θ_k with $b^k = N$ rows and columns). Each edge is then independently sampled using a Bernoulli distribution with parameter $\Theta_k(i, j)$. A rough implementation of this algorithm runs in time $O(N^2)$, but improved algorithms can generate a network in $O(M \log N)$, where M is the number of edges in the network [2]. According to [2], the learning time is linear in the number of edges.

III. Notation and Chung-Lu Models

Let $G = \langle V, E \rangle$ represent a graph, where V is a set of N vertices, and $E = V \times V$ is a set of M edges between the vertices. Let A represent the adjacency matrix for G where:

$$A_{ij} = \begin{cases} 1 & (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Next, let D be a diagonal matrix such that:

$$D_{ij} = \begin{cases} \sum_k A_{ik} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The diagonal of matrix D contains the degree of each node. For ease of notation, we will use $D_i = D_{ii}$ to refer to the *observed* degree of node v_i in a given network. We then use $\tilde{D}_i^X$ to represent the random variable corresponding to the degree of node v_i in graphs drawn from the distribution represented by model X, where X will refer to various models that we discuss throughout the paper.

We define a transition probability matrix P that represents the likelihood that a random walk originating at v_i on G will transition to a node v_j along edge e_{ij} as:

$$P_{ij} = \frac{A_{ij}}{D_i} \tag{3}$$

Note that while the transition matrix in this case is defined in terms of A, in general that need not be the case.

The Chung Lu algorithm generates a graph by independently considering each pair v_i, v_j, sampling the edge e_{ij} with probability $\frac{D_i D_j}{2M}$, under the assumption that $\forall k \; D_k < \sqrt{M}$. Let $\tilde{D}_i^{CL}$ be the r.v. for the degree of node v_i in a graph sampled from the Chung Lu model. The expected degree for any node v_i is the degree of the node in the original input graph G [4]:

$$\mathbb{E}\left[\tilde{D}_i^{CL}\right] = \sum_j \frac{D_i D_j}{2M} = D_i \sum_j \frac{D_j}{2M} = D_i$$

In [5], the authors describe a fast Chung Lu (FCL) sampling algorithm which runs in $O(M)$. The FCL algorithm defines the target sampling distribution of nodes as $\pi(i) = \frac{D_i}{2M}$. The graph is constructed by sampling two nodes v_i, v_j according to π, and then an edge is created between the sampled nodes and added to the sampled graph: $E = E \cup e_{ij}$. Since the algorithm draws from π twice, the expected degree $\tilde{D}_i^{FCL}$ for node v_i in the sampled graph is:

$$\mathbb{E}\left[\tilde{D}_i^{FCL}\right] = M \cdot [\pi(i) + \pi(i)] = M \cdot 2\frac{D_i}{2M} = D_i$$

Note however, that the FCL algorithm samples the nodes with *replacement*. Specifically, the implementation creates a vector of size $2M$, which contains D_i replicates of each node v_i. To generate the graph, the algorithm samples a node v_i uniformly at random from the vector—this can be done in $O(1)$. Given a first node v_i, the algorithm then draws another vertex v_j *independently* from the vector (i.e., with replacement). This sampling method can alternatively be viewed as generating edges via a random walk in the transition matrix P^{FCL}, where the probability of transitioning from one node to another is always distributed according to π.

Definition 1. *Let P^{FCL} be a transition probability matrix such that:*

$$P_{jk}^{FCL} = \pi(k)$$

The fast Chung Lu sampling algorithm generates edges according to a random walk of length M in P^{FCL}. This stems from the fact that regardless of which v_j is drawn first, the second node v_k is sampled directly from π.

P^{FCL} differs from P in the sense that it is completely connected and the next step of a random walk does not depend on the properties of the current node.

However, we note that since the algorithm samples with replacement, the algorithm must handle *collisions* where the same edge is sampled twice. We discuss this next.

IV. Fast Chung-Lu Bias Adjustments

The fast Chung-Lu method, as described in previous work [5], has two straightforward implementations that can lead to bias in the generated networks. In the first, the algorithm attempts exactly M insertions, ignoring collisions completely. In such a scenario, the probability of an edge existing between nodes v_i and v_j is $1 - [1 - 2\pi(i)\pi(j)]^M$. This is because the edge has M opportunities to be sampled, and it can also be generated by sampling the nodes is either order.

Since M is a positive integer and $2\pi(i)\pi(j) \leq 1$, the Bernoulli Inequality [14] can be used to show that the edge probabilities in FCL are less than those of the original CL model:

$$\begin{aligned} 1 - [1 - 2\pi(i)\pi(j)]^M \leq & 1 - [1 - 2M\pi(i)\pi(j)] \\ = & \frac{D_i D_j}{2M} \end{aligned} \tag{4}$$

Since the edge probabilities do not exist with probability $\frac{D_i D_j}{2M}$, but with lower probability—they have a downward *bias*. Consequently, the expected degrees will also be biased (as they are the sum of the edge random likelihoods).

A second (and the presumed implementation used in [5]) attempts to avoid this bias by inserting precisely M edges into the network using a form of *rejection sampling*. Specifically, in this approach if an edge is sampled for a second time, it is *rejected* and another pair of nodes is drawn until exactly M unique edges are added to the network. Such a method clearly increases the exponent on the left side of equation 4, and thus the underestimated probabilities may no longer occur. However, while this implementation avoids the clear bias of the first (naive) method, two problems remain. We show below that show that the rejection sampling approach leads to a *degree* bias, and outline an alternative algorithm which corrects the bias. Next, we show that the FCL algorithm, even with a correction to the degree bias, does not match the edge probabilities of the original Chung Lu algorithm in practice—it only *approximates* the target edge probabilities.

A. Edge Collision Degree Bias

Consider the FCL algorithm that draws two nodes from π and if an edge already exists between the two nodes in the sampled graph, it rejects the pair and draws again. Notice that this implementation of the FCL algorithm samples *edges* only once (i.e., without replacement), but *nodes* may be sampled multiple times (i.e., with replacement). Since, the sampled pairs are rejected according to whether or not an *edge* already exists between the nodes, the probability of collision is higher for higher degree nodes. As a result the edges around high degree nodes are rejected more frequently than those around low degree nodes—which results in the high degree nodes being undersampled.

Proposition 1. *Let v_i, v_j be two nodes such that $D_i > D_j$. When the FCL algorithm rejects repeated samples of the same edge, $\mathbb{E}[\tilde{D}_i^{FCL}]$ will be underestimated to a greater extent than $\mathbb{E}[\tilde{D}_j^{FCL}]$.*

Proof: Let v_k be an arbitrary node that is selected by the FCL algorithm after C samples. Consider the event which corresponds to selecting node v_i or v_j as the second node:

$$p(e_{ki}|v_k) = \pi(i) \qquad p(e_{kj}|v_k) = \pi(j)$$

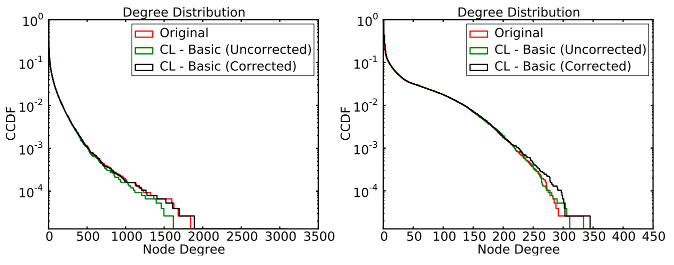
Notice that each edge is sampled according to the target degree of each node, i.e., π. However, the likelihood that the selected edge is a *collision* depends on whether the edge had already been selected during the last C samples. Thus the probability that the edge is rejected is:

$$p(collision_{ki}|v_k) = 1 - [1 - \pi(k)\pi(i)]^C$$
$$p(collision_{kj}|v_k) = 1 - [1 - \pi(k)\pi(j)]^C$$

Since $\pi(i) > \pi(j)$, $p(collision_{ki}|v_k) > p(collision_{kj}|v_k)$. While each node is sampled according to its target degree, once sampled the high degree nodes have a larger chance of being rejected based on a collision, for any arbitrary edge. The rejections lower the effective degree of the node in the sampled graph, i.e. $\mathbb{E}[\tilde{D}_i^{FCL}]$ will be underestimated. ■

One simple approach to correct this problem is to sample $2M$ nodes independently from π. These samples can then be paired together and the pairings checked for duplicates. Should any pair of nodes be chosen more than once, the entire set of nodes can be randomly permuted and rematched. This process would continue until no duplicate pairings were found.

The general idea behind this random permutation motivates our correction to the fast method. While the random permutation of all M edges is somewhat extreme, a method which permutes only a few edges in the graph—the ones with collisions—is tractable. With this in mind, we describe our straightforward solution (Algorithm 1). When the algorithm encounters a collision, we place both vertices in a waiting queue. Before continuing with regular insertions the algorithm attempts to select neighbors for all nodes in the queue. Should the new edge for a node selected from the queue also result in a collision, the chosen neighbor is also placed in the queue, and so forth. This ensures that if a node is 'due' for a new edge but has been prevented from receiving it due to a collision, the node is 'slightly permuted' by exchanging places with a node sampled later.



(a) Epinions (b) Facebook

Fig. 1. Comparison of Basic and Corrected CCDF on two datasets. The original method underestimates the degree of the high degree nodes.

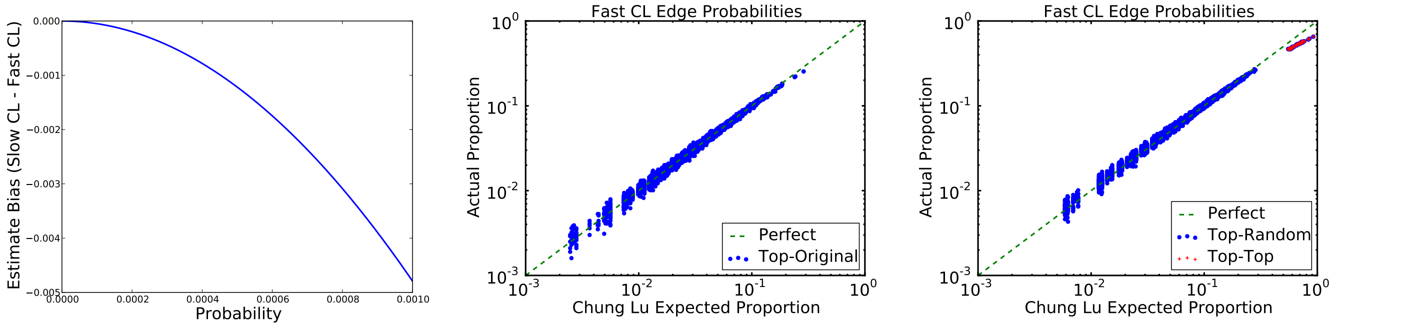
This shuffling ensures that M edges are placed in the graph (to ensure the probability of edge existence is the same as the slow Chung Lu), without affecting the degree distribution as can happen by proposition 1.

The modification to the fast CL model is correct only when there is independence between the edge placements and the current graph configuration. However, this independence only truly holds when collisions are allowed (i.e. when generating a multigraph). In practice, edge placements are not truly independent in the algorithm since the placement of edges that already exist in the graph is disallowed. The modification we have described removes the bias described in proposition 1 but is not guaranteed to generate graphs exactly according to the original π distribution. Our FCL graph generation algorithm must project from a space of multigraphs down into a space of simple graphs, and this projection is not necessarily uniform over the space of graphs. However, our empirical results show that for sparse graphs our correction removes the majority of the bias due to collisions and that the bias from the projection is negligible, meaning we can treat graphs from the modified FCL as being drawn from the original Chung-Lu graph distribution.

In Figure 1, we can see the effect of the correction on two labeled datasets, Epinions and Facebook (described in section VI). The green line corresponding to sampled degree distribution using the simple FCL sampling technique, which clearly underestimates the high degree nodes in both instances. Our FCL modification results in a much closer match to the original degree distribution, particularly on the high degree nodes. By utilizing this algorithm modification, we can generate graphs whose degree distributions are largely unaffected by collisions and we are able to generate graphs in $O(M)$.

B. Fast Chung-Lu Edge Probabilities

In this section, we consider the special case of regular graphs to show that for our 'corrected' FCL algorithm, the probability of an edge existing is exactly the same for as for the slow Chung Lu algorithm. Before doing so, we first derive the



(a) Biases (b) Original (c) Adjusted

Fig. 2. In (a), we show the theoretical bias as the probability of the edge increases. In (b) and (c), we show the bias empirically, where (b) shows the true Facebook network and (c) augmented Facebook network.

Algorithm 1 CL($\pi, N, |E|$)

```
1: $E^{CL} = \{\}$
2: initialize(queue)
3: for iterations do
4:     if queue is empty then
5:         $v_j$ = sample-pi($\pi$)
6:     else
7:         $v_j$ =pop(queue)
8:     end if
9:     $v_i$ = sample-pi($\pi$)
10:    if $e_{ij} \notin E^{CL}$ then
11:        $E^{CL} = E^{CL} \cup e_{ij}$
12:    else
13:        push(queue, $v_i$)
14:        push(queue, $v_j$)
15:    end if
16: end for
17: return($E^{CL}$)
```

expected number of times a node v_i will be selected to place an edge, according to the *corrected* Fast Chung Lu algorithm ($cFCL$).

Proposition 2. *Let v_i be a node in a regular graph, such that $\forall i,j\ D_i = D_j$. We refer to the sampled degree of v_i in the corrected FCL method as $\tilde{D}_i^{cFCL}$. After R sampled edges from π, where $0 \le R \le N^2$ and $0 \le \tilde{D}_i^{cFCL} \le N$:*

$$\mathbb{E}\left[\tilde{D}_i^{cFCL}|R\right] = \frac{2R}{N}$$

Proof: Note that the limit on R ensures that the number of sampled edges is possible to fit in the network of size N. Since $\forall i,j\ D_i = D_j$, then $\forall i,j\ \mathbb{E}\left[\tilde{D}_i^{cFCL}\right] = \mathbb{E}\left[\tilde{D}_j^{cFCL}\right]$. Additionally, the sum of degrees must always equal $2 \cdot R$, implying the expectation of the sum of degrees is $2 \cdot R$.

$$\begin{aligned} \mathbb{E}\left[\sum_{v_i} \tilde{D}_i^{cFCL} \middle| R\right] &= 2R \\ \sum_{v_i} \mathbb{E}\left[\tilde{D}_i^{cFCL}|R\right] &= 2R \\ N \cdot \mathbb{E}\left[\tilde{D}^{cFCL}|R\right] &= 2R \\ \mathbb{E}\left[\tilde{D}^{cFCL}|R\right] &= \frac{2R}{N} \end{aligned} \tag{5}$$

■

Using this, we can show that in a regular graph the probability of sampling an edge in the cFCL algorithm is the same as that of the slow CL algorithm.

Proposition 3. *Let v_i, v_j be nodes in a regular graph, such that $\forall i,j\ D_i = D_j$. Then the probability that edge e_{ij} is sampled in the cFCL is the same as the probability e_{ij} is sampled in the slow CL algorithm.*

Proof: The cFCL algorithm selects every node at random with replacement. Let $\tilde{D}_j^{cFCL}$ refer to the sampled degree of node v_j. Since the graph is regular, each node is selected with equal probability and all $\tilde{D}_j^{cFCL}$ are identically distributed (denoted $\tilde{D}^{cFCL}$ where necessary). Let $P_{cFCL}(e_{ij}|\neg e_{ik})$ refer to the transition probability from v_i to v_j when e_{ik} has already been placed (removing it from consideration for future placement). The edge placement probability $P_{cFCL}(e_{ij}|\tilde{D}_j^{cFCL} = d)$ of an edge being sampled between v_i and v_j when v_i has a current sampled degree of d is:

$$\begin{aligned} &P(e_{ij}|\tilde{D}_j^{cFCL} = d) \\ &= P_{CL}(e_{ij}) + \sum_{v_k \in V, k \neq j} P_{CL}(e_{ij}|\neg e_{ik}) P_{CL}(e_{ik}) + \dots \\ &= \frac{D_j}{2M} + \sum_{v_k \in V, k \neq j} \frac{D_k}{2M - D_k} \frac{D_j}{2M} + \dots \\ &= \frac{D_j}{2M} + \frac{D_j}{2M} \sum_{v_k \in V, k \neq j} \frac{D_k}{2M - D_k} + \dots \end{aligned} \tag{6}$$

When the graph is regular, $D_k = D \, \forall v_k$, meaning that the denominator is the same for all v_k. Furthermore, this implies the numerator sum equals the denominator. Simplifying further:

$$\begin{aligned} P(e_{ij}|\tilde{D}_j^{cFCL} = d) &= \frac{D_j}{2M} + \frac{D_j}{2M}\frac{2M-D}{2M-D} + \ldots \\ &= 2\frac{D_i}{2M} + ... \end{aligned} \tag{7}$$

More precisely, for $d \leq N$ the probability of v_j being drawn in the d position when all nodes are equally probable is $\frac{D}{2M}$ (a standard combinatorial result). Thus, after d insertions for v_i, the probability that e_{ij} has been placed is therefore $d\frac{D_j}{2M}$ for $d \leq N$. The probability of edge e_{ij} being placed is the marginalization over the degree variable $\tilde{D}_i^{cFCL}$ after M draws from π. This can be computed as:

$$\begin{aligned} P\left(e_{ij}\right) =& \sum_{d=0}^{n} P(e_{ij}|\tilde{D}_i^{cFCL} = d, M)P(\tilde{D}_i^{cFCL} = d|M) \\ =& \sum_{d=0}^{n} \frac{D_j}{2M} \cdot d \cdot P(\tilde{D}_i^{cFCL} = d|M) \\ =& \frac{D_j}{2M} \sum_{d=0}^{n} P(\tilde{D}_i^{cFCL} = d|M) \cdot d \end{aligned} \tag{8}$$

The summation is simply the expected value of $\tilde{D}_j^{cFCL}$ for a given number of draws M. Since every node has degree D, the total number of edges is $\frac{D \cdot N}{2}$. Using proposition 2:

$$\begin{aligned} P\left(e_{ji}\right) =& \frac{D_j}{2M} \sum_{d=0}^{n} P(\tilde{D}_i^{cFCL} = d|M) \cdot d \\ =& \frac{D_j}{2M} \cdot \mathbb{E}\left[\tilde{D}_i^{cFCL}|M\right] \\ =& \frac{D_j}{2M} \cdot \frac{2M}{N} \\ =& \frac{D_i D_j}{2M} \end{aligned}$$

Thus, the probability of placing an edge e_{ij} is $\frac{D_i D_j}{2M}$. $\blacksquare$

In general, we do not have a regular graph, which means the breakdown between the degrees does not have the convenient cancellation of sums as shown in equation 6. As the bias from the cFCP approximation is less for edges with lower likelihoods (see Figure 2.a), additional samplings due to collisions will bias their probabilities slightly *higher*, while the high degree edges will have likelihoods that are slightly *lower*.

For sparse graphs, we assume the proportion of degrees is close enough to one another such that the summations effectively cancel. The difference between the two probabilities is illustrated in Figure 2b-c. The dataset we use is a subset of the Purdue University Facebook network, a snapshot of the class of 2012 with approximately 2000 nodes and 15,000 edges—using this smaller subset exaggerates the collisions and their effects on the edge probabilities. We plot the edge probabilities along the x-axis as outlined by the original CL method versus a simulation of 10,000 networks for the cFCL edge probabilities. The y-axis indicates the proportion of generated networks which have the edge (we plot the top 10 degree nodes' edges).

In panel (b), we show the probabilities for the original network, where the probabilities are small and unaffected by the fast model. To test the limits of the method, in panel (c) we take the high degree nodes from original network and expand them such that they have near $\sqrt{2M}$ edges elsewhere in the network. Additionally, these high degree nodes are connected *to each other*, meaning they approach the case where $\frac{D_i D_j}{2M} > 1$. We see that the randomly inserted edges still follow the predicted slow CL value, although the probabilities are slightly higher due to the increased degrees. It is only in the extreme case where we connect $\sqrt{2M}$ degree nodes to one another that we see a difference in the realized probability from the CL probability. These account for $.05\%$ of edges in the augmented network, which has been created specifically to test for problem cases. For social networks, it is unlikely that these situations will arise.

V. Transitive Chung-Lu Model

A large problem with the Chung-Lu model is the lack of *transitivity* captured by the model. As many social networks are formed via friendships, drawing randomly from distribution of nodes across the network fails to capture this property. We propose the *Transitive Chung Lu* (TCL) model described in algorithm 2, which selects edges based on the probability of a 'random surfer' connecting two nodes across the network, along with the the additional probability of creating a new transitive edge between a pair of nodes connected by a two-hop path. In the TCL model, we incorporate the transitive edges *while* maintaining approximately the same transitional probability matrix as the original CL model, implying the TCL model has the same expected degree distribution as the original network.

The algorithm begins by constructing a graph of M edges using the cFCL model described in section IV. This gives us an initial edge set E which has the same expected degree distribution as the original data, as well as edge probabilities $P(e_{jk}) = \frac{D_j D_k}{2M}$, an assumption we hold through all proofs in this section. The algorithm then initializes a queue which will be used to store nodes that have a higher priority for receiving an edge. Next, we define an update step which replaces the oldest edge in the graph with a new one selected according to the TCL model, repeating this process for the specified number of iterations. If the priority queue is not empty, we will choose the next node in the queue to be v_j, the first endpoint of the edge; otherwise, on line 5 we sample v_j using the π distribution. With probability ρ the algorithm will add an edge between v_j and some node v_i through transitive closure by choosing an intermediate node v_k uniformly from j's neighbors, then selecting v_i uniformly from k's neighbors. In contrast, with probability $(1-\rho)$ the algorithm will use the CL method, randomly choosing v_i from the graph according to π. The TCL transition probability matrix P_{TCL} is comprised of two parts, the first part being the P_{CL} transition matrix and

Algorithm 2 TCL($\pi, \rho, N, |E|, iterations$)

```
1:  $E^{TCL} = CL(\pi, N, |E|)$
2:  initialize(queue)
3:  for iterations do
4:      if queue is empty then
5:          $v_j$ = sample-pi($\pi$)
6:      else
7:          $v_j$ = pop(queue)
8:      end if
9:      $r$ = sample-bernoulli($\rho$)
10:     if $r = 1$ then
11:         $v_k$ = sample-uniform($E_j^{TCL}$)
12:         $v_i$ = sample-uniform($E_k^{TCL}$)
13:     else
14:         $v_i$ = sample-pi($\pi$)
15:     end if
16:     if $e_{ij} \notin E^{TCL}$ then
17:         $E^{TCL} = E^{TCL} \cup e_{ij}$
18:         // remove oldest edge from $E^{TCL}$
19:         $E^{TCL} = E^{TCL} \setminus min(time(E^{TCL}))$
20:     else
21:         push(queue, $v_i$)
22:         push(queue, $v_j$)
23:     end if
24: end for
25: return($E^{TCL}$)
```

the second being the *transitive closure* transition probability matrix.

Definition 2. *Let P_{CLO} be the transitive closure transition probability matrix for the TCL algorithm. For two nodes v_j, v_k, $P_{CLO}(e_{jk})$ is the transition probability of leaving v_j and arriving at v_k:*

$$P_{CLO}(e_{jk}) \propto \sum_{D_j^{CL}=0}^{|V|} \sum_{e_{jk} \in \{0,1\}} p(w_{jk}|e_{jk}, D_j^{CL}) p(D_j^{CL}|e_{jk}) P(e_{jk})$$

Here $p(D_j^{CL}|e_{jk})$ refers to the probability of the degree of the node v_j and $p(w_{jk}|e_{jk}, D_j^{CL})$ refers to the probability of sampling the edge e_{jk} from the edges currently present in the graph. Both the degree of v_j in the graph as well as whether or not the edge exists, are marginalized out to calculate the transition probability.

This transition matrix represents a hybrid between P and P_{CL}; the transition is picked according to the edges in the network as in P, but the edges exist with probability proportional to π (similar to P_{CL}).

Either transition matrix, P_{CLO} or P_{CL}, can be used to sample an additional node for the method to use as the other endpoint. If the selected pair of nodes does not already have an edge in the graph, the algorithm adds it and removes the oldest edge in the graph. (Note the algorithm continually removes the warmup CL edges.) If the selected pair already has an edge in the graph, the selected endpoint nodes are placed in the priority queue (lines 21 and 22). The replacement operation is repeated many times to ensure that the original CL graph is mostly replaced and then the final set of edges is returned as the new graph. In practice, we find that M replacements is sufficient to remove all edges generated originally by CL and generate a reasonable sample.

In order to show that the TCL update operation preserves the expected degree distribution, we prove the following:

1) The transitive closure transition matrix $P_{CLO}(e_{jk})$ is approximately π for every v_j.
2) TCL places edges with approximately $\pi(i)\pi(j)$ probability.
3) The change in the expected degree distribution after a TCL iteration is approximately zero.

Step 1: Transitive Closure Matrix is Approximately π

We begin by defining a few quantities. The first is the state of the neighborhood around node v_j, excluding two of its possible neighbors v_{k_1}, v_{k_2}.

Definition 3. *Let v_j, v_{k_1}, v_{k_2} be nodes in V, where $e_{j*-\{k_1,k_2\}}$ is the set of* existing *edges, with v_j as one endpoint, excluding e_{jk_1}, e_{jk_2}. Define:*

$$C = |\{e_{j*-\{k_1,k_2\}}\}|$$

to be the size of the set of edges that currently exist in the generated graph from v_j to its possible neighbors (excluding v_{k_1}, v_{k_2}). Let $P(C)$ denote the probability of a particular number of existing edges in C.

Next, the ratio of transition probabilities quantifies the odds of picking an edge e_{jk_1} versus e_{jk_2} when leaving node v_j.

Definition 4. *Let v_j, v_{k_1}, v_{k_2} be nodes in V with P_X being a transition probability matrix. Define:*

$$R_X(e_{jk_1}, e_{jk_2}) = \frac{P_X(e_{jk_1})}{P_X(e_{jk_2})}$$

to be the ratio of transition probabilities when v_j and landing at v_{k_1}, compared to leaving v_j and landing at v_{k_2}.

Since P_{CL} defines the ideal transition probabilities, the transition probability ratio bias defines how far the ratio between two transition probabilities in a potential transition probability matrix P_X is from the desired ratio defined by R_{CL}.

Definition 5. *Let v_j, v_{k_1}, v_{k_2} be nodes in V with R_X being the ratio of transmission probabilities for some transition matrix P_X. Define the* transition probability ratio bias *to be:*

$$\delta_X(e_{jk_1}, e_{jk_2}) = R_X(e_{jk_1}, e_{jk_2}) - R_{CL}(e_{jk_1}, e_{jk_2})$$

In essence, δ encapsulates how far our transition probability matrix is from the desired distribution π. We now quantify how far the transition probabilities defined by P_{CLO} are from P_{CL}.

Theorem 1. *For nodes v_j, v_{k_1}, v_{k_2}, and a given C defined as in Definition 3, the transition probability ratio bias for the transition matrix defined by P_{CLO} is:*

$$\delta_{CLO}(e_{jk_1}, e_{jk_2}) = \frac{D_{k_1}\left[C + 2 - \frac{D_j D_{k_1}}{2M}\right]}{D_{k_2}\left[C + 2 - \frac{D_j D_{k_2}}{2M}\right]} - \frac{D_{k_1}}{D_{k_2}}$$

Proof: Begin by computing $R_{CLO}(e_{jk_1}, e_{jk_2})$. For a given C, Definition 2 reduces to:

$$P_{CLO}(e_{jk_1}) = P(C) \sum_{e_{jk_1} \in \{0,1\}} P(e_{jk_1}) P(w_{jk_1} | C, e_{jk_1})$$

The case where e_{jk_1} does not exist is 0 (since the walk probability is 0 in such cases). Thus, the above can be put in terms of the marginalization over the other free variable (e_{jk_2}):

$$\begin{aligned} P_{CLO}(e_{jk_1}) &= P(C)P(e_{jk_1}) \sum_{e_{jk_2} \in \{0,1\}} P(e_{jk_2}) P(w_{jk_1} | C, e_{jk_1}, e_{jk_2}) \\ &= P(C)P(e_{jk_1}) \left[[1 - P(e_{jk_2})] \frac{1}{C+1} + P(e_{jk_2}) \frac{1}{C+2} \right] \end{aligned}$$

The ratio of transition probabilities for CLO is:

$$\begin{aligned} &R_{CLO}(e_{jk_1}, e_{jk_2}) \\ &\quad = \frac{P(C)P(e_{jk_1}) \left[[1 - P(e_{jk_2})] \frac{1}{C+1} + P(e_{jk_2}) \frac{1}{C+2} \right]}{P(C)P(e_{jk_2}) \left[[1 - P(e_{jk_1})] \frac{1}{C+1} + P(e_{jk_1}) \frac{1}{C+2} \right]} \\ &\quad = \frac{P(e_{jk_1}) \left[[1 - P(e_{jk_2})](C+2) + P(e_{jk_2})(C+1) \right]}{P(e_{jk_2}) \left[[1 - P(e_{jk_1})](C+2) + P(e_{jk_1})(C+1) \right]} \\ &\quad = \frac{D_{k_1}\left[C + 2 - \frac{D_j D_{k_2}}{2M}\right]}{D_{k_2}\left[C + 2 - \frac{D_j D_{k_1}}{2M}\right]} \end{aligned}$$

Since $R_{CL}(e_{jk_1}, e_{jk_2}) = \frac{D_{k_1}}{D_{k_2}}$, the theorem holds. ■

There are two interesting cases for this transition matrix. The first is the case where the input graph is regular—where all the edge probabilities are the same.

Corollary 1. *In the case where the input network is a regular graph, the CLO transition probability ratio bias is* 0.

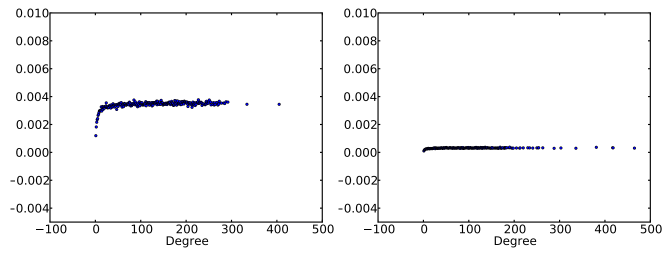
Proof: Let D_k be the degree for all nodes in the original network. From Theorem 1, we have:

$$\delta_{CLO}(e_{jk_1}, e_{jk_2}) = \frac{D_{k_1}\left[C + 2 - \frac{D_k D_k}{2M}\right]}{D_{k_2}\left[C + 2 - \frac{D_k D_k}{2M}\right]} - \frac{D_{k_1}}{D_{k_2}} = 0$$

■

The second interesting, and more practical, case is when either the edge probabilities are small, or the value of C is large. In these instances, the transition ratio bias is *nearly* zero.

Corollary 2. *As C increases or $P(e_{jk_1}), P(e_{jk_2})$ decrease,* $\delta_{CLO}(e_{jk_1}, e_{jk_2}) \approx 0$.



(a) Facebook (b) PurdueEmail

Fig. 3. Transition edge biases for two datasets. The larger dataset has considerably less bias, implying the method works better for larger datasets.

Proof: Note that as the ratio

$$\frac{C + 2 - \frac{D_j D_{k_2}}{2M}}{C + 2 - \frac{D_j D_{k_1}}{2M}}$$

approaches 1, $\delta_{CLO}(e_{jk_1}, e_{jk_2}) \approx 0$. Should C be large the numerator and denominator are both approximately $C+2$ since $P(e_{jk_1})$ and $P(e_{jk_2})$ are bounded by 1. In addition, if $P(e_{jk_1})$ and $P(e_{jk_2})$ tend to 0 then the numerator and denominator are also both approximately 1 due to the constant 2. In either case, $\delta_{CLO}(e_{jk_1}, e_{jk_2}) \approx 0$. ■

Using the above theorem and corollary it is easy to see that if the first step away from node v_j is π distributed, the second is as well. A key question is how much bias should we see in large social networks. In Figure 3 we show the average bias for every degree present in two of our real-world networks: Facebook and PurdueEmail. For each degree we sample:

- A *center* node v_c from the nodes with the desired degree
- Two neighboring v_{n_1}, v_{n_2} nodes according to π

We set C to be the expected value of the present degree missing two of the edges followed by computing $\delta_{CLO}(e_{cn_1}, e_{cn_2})$ and $\delta_{CLO}(e_{cn_2}, e_{cn_1})$. The process was repeated 10,000 times for every degree, and Figure 3 reports the averages. Both datasets results in biases under .005 for every degree, with PurdueEmail biases being much lower. This is reasonable, since it implies the method *improves* on graphs with larger size (Table I.a).

Step 2: TCL places edges with approximate probability $\pi(i)\pi(j)$

Utilizing the above theorem and corollary we show the transition matrix for the TCL algorithm is approximately π.

Proposition 4. *For two nodes v_j, v_k, the transition matrix for the TCL model $P_{TCL}(e_{jk})$ is approximately π.*

Proof: Let ρ be the probability of selecting according to two random walk steps from P_{CLO} and $(1 - \rho)$ be the probability of selecting according to P_{CL}. $P_{TCL}(e_{jk})$ is then

$$\rho P_{CLO}(e_{jk}) + (1 - \rho) P_{CL}(e_{jk}) \approx \pi(k)[\rho + (1 - \rho)] = \pi(k)$$

■

We next show the probability of placing edge e_{ij} in the graph is $\pi(j)\pi(i)$.

Theorem 2. *The TCL algorithm selects edge e_{ij} for insertion with probability: $P(e_{ij}) = \pi(i) * \pi(j)$.*

Proof: The first node is selected directly from π while the second is selected according to P_{TCL}, which was shown in Proposition 4 to be π-distributed. ■

Therefore, the inductive step of TCL will place the endpoints of the new edge according to π.

Step 3: The expected degree distribution of TCL matches CL

Corollary 3. *The expected degree distribution of the graph produced by TCL is the same as the degree distribution of the input graph.*

Proof: The inductive step of TCL places an edge with endpoints distributed according to π, so the expected increase in the degree of any node v_i is $\pi(i)$. However, the inductive step will also remove the oldest edge that was placed into the network. Since the oldest edge can only have been placed in the graph through a Chung-Lu process or a transitive closure, the expected decrease in the degree is also $\pi(i)$, which means the expected change in the degree distribution is zero. Because the CL initialization step produces a graph with expected degree distribution equal to the input graph's distribution, and the TCL update step causes zero expected change in the degree distribution, the output graph of the TCL algorithm has expected degree distribution equal to the input graph's distribution by induction. ■

This implies that the algorithm is placing edges according to $\pi(i)\pi(j)$, and the algorithm continues for M insertions. This is the same approach that the cFCL algorithm—which means that if the cFCL method matches the slow CL, then the TCL does as well. In practice, TCL and CL capture the degree distribution well (see Section VI).

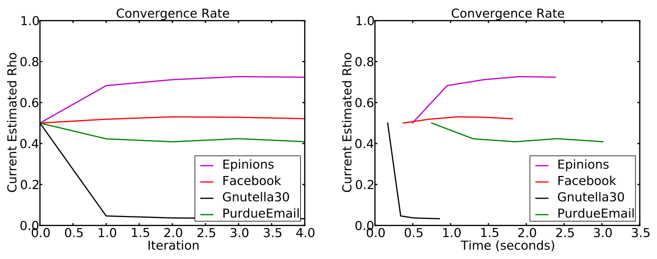
A. Fitting Transitive Chung Lu

Now that we have introduced a ρ parameter which controls the proportion of transitive edges in the network we need a method for learning this parameter from the original network. For this, we need to estimate the probability ρ by which edge formation is done by triadic closure, and the probability $1-\rho$ by which the random surfer forms edges. We can accomplish this estimation using Expectation Maximization (EM). First, let $z_{ij} \in Z$ be latent variables on each $e_{ij} \in E$ with values $z_{ij} \in \{1, 0\}$, where 1 indicates the edge e_{ij} was placed by a transitive closure and 0 indicates the edge was placed by a random surfer.

We can now define the conditional probability of placing an edge e_{ij} from starting node v_j given the method z_{ij} by which the edge was placed:

$$P\left(e_{ij}|z_{ij}=1, v_j, \rho^t\right) = \rho^t \sum_{v_k \in e_{j*}} \frac{\mathbb{I}[v_i \in e_{k*}]}{D_j} \frac{1}{D_k}$$
$$P\left(e_{ij}|z_{ij}=0, v_j, \rho^t\right) = (1-\rho^t) \cdot \pi(i)$$

Starting at v_j, the probability of the edge existing between v_i and v_j given that the edge was placed due to a triangle



(a) (b)

Fig. 4. Convergences of the EM algorithm—both in terms of time and number of iterations. 10000 samples per iteration.

closure is ρ times the probability of walking from v_j to a mutual neighbor of v_i and v_j and then continuing the walk on to v_i. Conversely, $(1-\rho) \cdot \pi(i)$ is the probability the edge was placed by a random surfer. We now show the EM algorithm.

Expectation: Note that the conditional probability of z_{ij}, given the edge e_{ij} and ρ, can be defined in terms of the probability of an edge being selected by the triangle closure divided by the probability of the edge being placed by any method. Using Bayes' Rule, our conditional distribution on Z is:

$$P\left(z_{ij}=1|e_{ij}, v_j, \rho^t\right) = \frac{\rho^t \left[\sum_{v_k \in e_{j*}} \frac{\mathbb{I}[v_i \in e_{k*}]}{D_j} \frac{1}{D_k}\right]}{\rho^t \left[\sum_{v_k \in e_{j*}} \frac{\mathbb{I}[v_i \in e_{k*}]}{D_j} \frac{1}{D_k}\right] + (1-\rho^t)\left[\pi(i)\right]}$$

And our expectation of z_{ij} is:

$$\mathbf{E}[z_{ij}|\rho^t] = P\left(z_{ij}=1|e_{ij}, v_j, \rho^t\right)$$

Maximization: To maximize this expectation, we note that ρ is a Bernoulli variable representing $P(z_{ij} = 1)$. We sample a set of edges $\mathbb{S}$ uniformly from the graph to use as evidence when updating ρ. The variables z_{ij} are *conditionally independent* given the edges and nodes in the graph, meaning the maximum likelihood update to ρ is then calculating the expectation of $z_{ij} \in \mathbb{S}$ and then normalizing over the number of edges in $\mathbb{S}$:

$$\rho^{t+1} = \frac{1}{|\mathbb{S}|} \sum_{z_{ij} \in \mathbb{S}} \mathbf{E}[z_{ij}|\rho^t]$$

We sampled the edge subsets uniformly at random from the set of all edges. This can be done quickly using the node ID vector we constructed for sampling from the π distribution. Since each node i appears D_i times in this vector, sampling a node from the vector and then uniformly sampling one of its edges results in a $\frac{D_i}{M} * \frac{1}{D_i} = \frac{1}{M}$ probability of sampling any given edge. We sampled subsets of 10000 edges per iteration and the EM algorithm converged in a few seconds, even on datasets with millions of edges. Figure 4 shows the convergence time on each of the datasets.

B. Time Complexity

The methods presented for both generating a new network and for learning the parameter ρ can be done in an efficient manner. To show this, we need to bound the expected number

Dataset	Nodes	Edges
Epinions	75,888	811,480
Facebook	77,110	500,178
Gnutella30	36,682	176,656
PurdueEmail	214,773	1,711,174

(a) Size

Dataset	CL	KPGM	TCL
Epinions	N/A	9,105.4s	2.5s
Facebook	N/A	5,689.4s	2.0s
Gnutella30	N/A	3,268.4s	0.9s
PurdueEmail	N/A	8,360.7s	3.0s

(b) Learning Time

Dataset	CL	KPGM	TCL
Epinions	20.0s	151.3s	64.6s
Facebook	14.2s	92.4s	30.8s
Gnutella30	4.2s	67.8s	7.0s
PurdueEmail	61.0s	285.6s	141.0s

(c) Generation Time

TABLE I
DATASET SIZES, ALONG WITH LEARNING TIMES AND RUNNING TIMES FOR EACH ALGORITHM

of attempts to insert an edge into the graph. Note that a node v_i with c edges has probability $\pi(i) = \frac{c}{M}$ of colliding with its own edge on the draw from π; by extension, the probability of colliding with its own edges k times is $\pi(i)^k$. This corresponds to a geometric distribution, which has the expected value of hits H (collisions) on the edges of the nodes:

$$\mathbf{E}\left[H|\pi(i)\right] = (1-\pi(i))\sum_{k=1}^{\infty}\pi(i)^{k-1} = \frac{1}{1-\pi(i)}$$

This shows the expected number of attempts to insert an edge is bounded by a constant. As a result, we can generate the graph in $O(N+M)$, the same complexity as the FCL model. The initial step of running the basic CL model takes $O(N+M)$. Next, the algorithm generates M insertions while gradually removing the current edges. This corresponds to lines 3-24 of Algorithm 2. In this loop, the longest operations are selecting randomly from neighbors and removing an edge. Both of these operations cost are in terms of the maximum degree of the network, which we assumed bounded, meaning the operations can be done in $O(1)$ time. As a result, the total runtime of graph generation is $O(N+M)$.

For the learning algorithm, assume we have I iterations which gather s samples. It is $O(1)$ to draw a node from the graph and $O(1)$ to choose a neighbor, meaning each iteration costs $O(s)$. Coupled with the cost of creating the initial π sampling vector, the total runtime is then $O(N+M+I\cdot s)$.

VI. EXPERIMENTS

For our experiments, we compared three different graph generating models. The first is the fast Chung Lu (CL) generation algorithm with our correction for the degree distribution. The second is Kronecker Product Graph Model (KPGM [2]) implemented with code from the SNAP library[1]. Lastly, we compared the Transitive Chung Lu (TCL) method presented in this paper using the EM technique to estimate the ρ parameter. All experiments were performed in Python on a Macbook Pro, aside from the KPGM parameters which were generated on a desktop computer using C++. All the datasets were transformed to be undirected by reflecting the edges in the network, except for the Facebook network which is already undirected.

[1]SNAP: Stanford Network Analysis Project. Available at http://snap.stanford.edu/snap/index.html. SNAP is written in C++.

A. Datasets

To empirically evaluate the models, we learned model parameters from real-world graphs (Table I.a) and then generated new graphs using those parameters. We then compared the network statistics of the generated graphs with those of the original networks.

The first dataset we analyze is Epinions [15]. This network represents the users of Epinions, a website which encourages users to indicate other users whose consumer product reviews they ‘trust’. The edge set of this network represents nominations of trustworthy individuals between the users.

Next, we study the collection of Facebook friendships from the Purdue University Facebook network. In this network, the users can add each other to their lists of friends and so the edge set represents a friendship network. This network has been collected over a series of snapshots for the past 4 years; we use nodes and friendships aggregated across all snapshots.

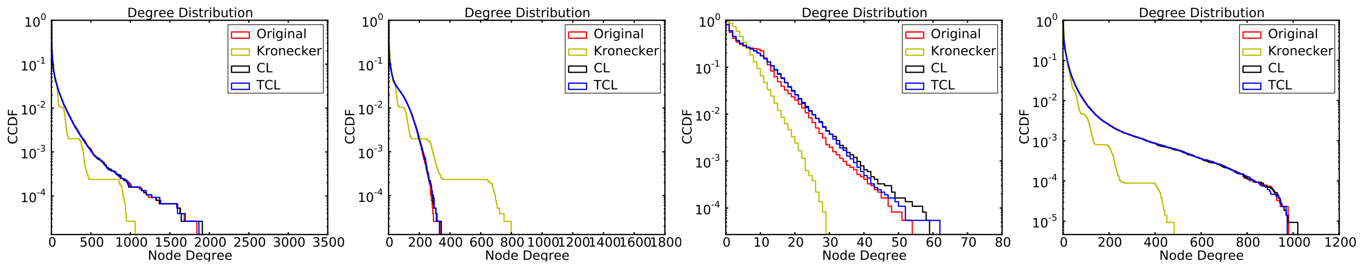
The Gnutella30 network differs from the other networks presented. Gnutella is a Peer2Peer network where users are attempting to find seeds for file sharing [16]. The user reaches out to its current peers, querying if they have a file. If not, the friend refers them to other users who might have a file, repeating this process until a seed user can be found.

Lastly, we study a large collection of emails gathered from the SMTP logs of Purdue University [17]. This dataset has an edge between users who sent e-mail to each other. The mailing network has a small set of nodes which sent out mail at a vastly greater rate than normal nodes; these nodes were most likely mailing lists or automatic mailing systems. In order to correct for these ‘spammer’ nodes, we remove nodes with a degree greater than $1,000$ as these nodes did not represent participants in any kind of social interaction.

B. Running Time

In Figure 4 we can see the convergence of the EM algorithm when learning parameter ρ, both in terms of the number of iterations and in terms of the total clock runtime. Due to the independent sample sets used for each iteration of the algorithm, we can estimate whether the sample set in each iteration is sufficiently large. If the sample size is too small, the algorithm will be susceptible to variance in the samples and will not converge. Using Figure 4.a we can see that after 5 iterations with 10,000 samples each iteration our EM method has converged to a smooth line.

In addition to the convergence in terms of iterations, in Figure 4.b we plot the wall time against the current estimated



(a) Epinions (b) Facebook (c) Gnutella30 (d) PurdueEmail

Fig. 5. Degree distribution for the Epinion, Facbook, Gnutella30 and PurdueEmail datasets.

ρ. The gap between 0 and the start of the colored lines indicates the amount of overhead needed to generate our degree distribution statistic and π sampling vector for the given graph (a step also needed by CL). The Purdue Email network has the longest learning time at 3 seconds. For the same Email network, learning the KPGM parameters took approximately 2 hours and 15 minutes, meaning our TCL model can learn parameters from a network significantly faster than the KPGM model (shown in Table I.b).

Next, the performance in terms of graph generation speed is tested, shown in Table I.c. The maximum time taken to generate a graph by CL is 61 seconds for the Purdue Email dataset, compared to 141 seconds to generate via TCL. Since TCL must initialize the graph using CL and then place its own edges, it is logical that TCL requires at least twice as long as CL. The runtimes indicate that the transitive closures cost little more in terms of generation time compared to the CL edge insertions. KPGM took 285 seconds to generate the same network. The discrepancy between KPGM and TCL is the result of the theoretical bounds of each—KPGM takes $O(M \log N)$ while TCL takes $O(M)$.

C. Graph Statistics

In order to test the ability of the models to generate networks with similar characteristics to the original four networks, we compare them on three well known graph statistics: degree distribution, clustering coefficient, and hop plot.

Matching the degree distribution is the goal of both the CL and KPGM models, as well as the new TCL algorithm. In Figure 5, the degree distributions of the networks generated from each model for each real-world network is shown, compared against the original real-world networks' degree distribution. The measure used along the y-axis is the complementary cumulative degree distribution (CCDF), while the x-axis plots the degree, meaning the y-value at a point indicates the percentage of nodes with greater degree. The four datasets have degree distributions of varying styles—the three social networks (Epinions, Facebook, and PurdueEmail) have curved degree distributions, compared to Gnutella30 whose degree distribution is nearly straight, indicating an exponential cutoff. As theorized, both the CL and TCL have a degree distribution which closely matches their expected degree distribution, regardless of the distribution shape. KPGM best matches the Gnutella30 network, sharing an exponential cutoff indicated by a straight line, but is still separated from the original network's distribution. With the social networks, KPGM has an alternating dip/flat line pattern which does not resemble the true degree distribution.

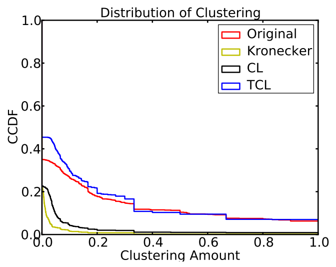
The next statistic we examine is TCL's ability to model the distribution of local clustering coefficients compared to CL and KPGM (see Figure 6). As with the degree, we plot the CCDF on the y-axis, but against the local clustering coefficient on the x-axis. On the network with the largest amount of clustering, Epinions, TCL matches the distribution of clustering coefficients well with the TCL distribution covering the original distribution. The same effect is visible for Facebook and PurdueEmail, despite the large size of the latter. The Gnutella30 has a remarkably low amount of clustering—so low that it is plotted in log-log scale—yet TCL is able to capture the distribution as well. Furthermore, the networks exhibit a range of ρ values which TCL can accurately learn.

In contrast, CL and KPGM cannot model the clustering distribution. For each network, both methods lack appreciable amounts of clustering in their generated graphs, even undercutting the Gnutella30 network which has far less clustering than the others. This shows a key weakness with both models, as clustering is an importation characteristic of small-world networks.

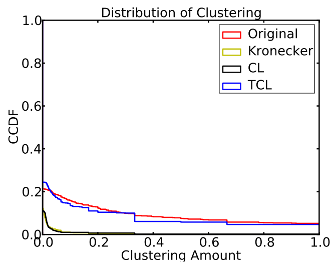
The last measure examined is the Hop Plot (see Figure 7). The Hop Plot indicates how tightly connected the graph is; for each x-value, the y-value corresponds to the percentage of nodes that are reachable within paths of the corresponding length. When generating the hop plots, we excluded any nodes with infinite hop distance and discarded disconnected components and orphaned nodes. All of the models capture the hop plots well, with TCL producing hop plots very close to those of the standard CL. This indicates that the transitive closures incorporated into the TCL model did not impact the connectivity of the graph and the gains in terms of clustering can be obtained without reducing the long range connectivity.

VII. Conclusions

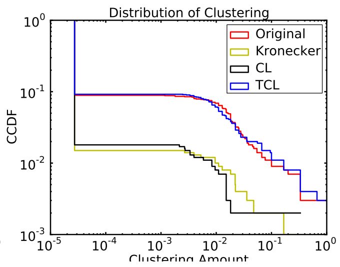
In this paper we demonstrated a correction to the fast Chung Lu estimation algorithm and introduced the Transitive Chung Lu model. Given a real-world network, the TCL algorithm can learn a model and generate graphs which accurately captures the degree distribution, clustering coefficient distribution and hop plot found in the training network, where alternative



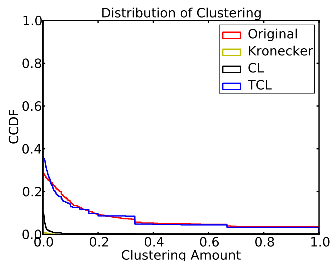
(a) Epinions



(b) Facebook

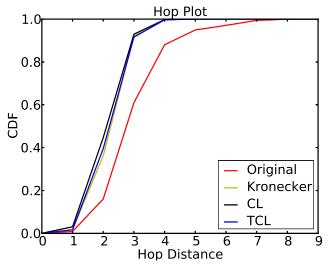


(c) Gnutella30

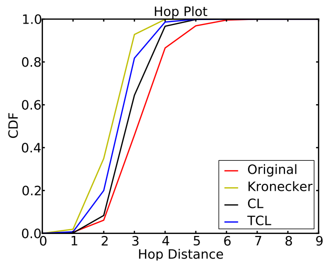


(d) PurdueEmail

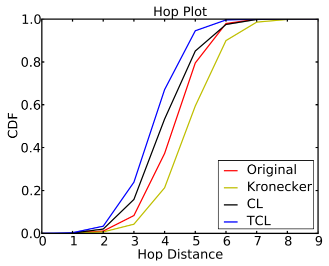
Fig. 6. Clustering Coefficient Distribution for the Epinion, Facbook, Gnutella30 and PurdueEmail datasets.



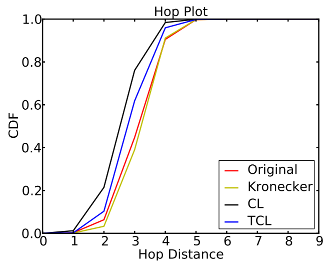
(a) Epinions



(b) Facebook



(c) Gnutella30



(d) PurdueEmail

Fig. 7. Hop plots for the Epinion, Facbook, Gnutella30 and PurdueEmail datasets.

methods fail on one or more of these characteristics. We proved the algorithm generates a network in time thats linear in the number of edges, on the same order as the original CL algorithm and faster than KPGM. The amount of clustering in the generated network is controlled by a single parameter, and we demonstrated how estimating the parameter is several orders of magnitude faster than estimating the parameters of the KPGM model. However, while our analysis of TCL shows how it can generate networks that match the degree distributions and clustering of a real-world network, usage of a transitivity parameter for clustering is still a heuristic approach. A more formal analysis of the clustering expected from such a model would be worth pursuing.

Acknowledgements

This research is supported by NSF under contract number(s) IIS-1017898 and IIS-0916686. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements either expressed or implied, of NSF or the U.S. Government. We also thank Nathan Davis and Joshua Patchus for useful discussions.

References

[1] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks." *Nature*, vol. 393, no. 6684, pp. 440–442, Jun. 1998.
[2] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, "Kronecker graphs: An approach to modeling networks," 2009.
[3] S. Wasserman and P. E. Pattison, "Logit models and logistic regression for social networks: I. An introduction to Markov graphs and p*," *Psychometrika*, vol. 61, pp. 401–425, 1996.
[4] F. Chung and L. Lu, "The average distances in random graphs with given expected degrees," *Internet Mathematics*, vol. 1, 2002.
[5] A. Pinar, C. Seshadhri, and T. G. Kolda, "The similarity between stochastic kronecker and chung-lu graph models," *CoRR*, vol. abs/1110.4925, 2011.
[6] A. Barabasi and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, pp. 509–512, 1999.
[7] C. Seshadhri, T. G. Kolda, and A. Pinar, "Community structure and scale-free collections of Erdős-Rényi graphs," arXiv:1112.3644 [cs.SI].
[8] P. Erdos and A. Renyi, "On the evolution of random graphs," *Publ. Math. Inst. Hung. Acad. Sci*, vol. 5, pp. 17–61, 1960.
[9] D. F. Gleich and C. Seshadhri, "Vertex neighborhoods, low conductance cuts, and good seeds for local community methods," in *KDD2012*.
[10] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Technical Report 1999-66, November 1999, previous number = SIDL-WP-1999-0120.
[11] O. Frank and D. Strauss, "Markov graphs," *Journal of the American Statistical Association*, vol. 81:395, pp. 832–842, 1986.
[12] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal, "Stochastic models for the web graph," in *Proceedings of the 42st Annual IEEE Symposium on the Foundations of Computer Science*.
[13] P. O'Connor, P. O'Connor, and A. Kleyner, *Practical Reliability Engineering*, ser. Quality and Reliability Engineering Series. John Wiley & Sons, 2012. [Online]. Available: http://books.google.com/books?id=V1Ttz5L_V50C
[14] D. S. G. Stirling, *Mathematical Analysis And Proof*, ser. Albion Mathematics & Applications Series. Albion, 1997. [Online]. Available: http://books.google.com/books?id=kBy3_7syd7cC
[15] M. Richardson, R. Agrawal, and P. Domingos, "Trust management for the semantic web," in *The Semantic Web - ISWC 2003*.
[16] M. Ripeanu, A. Iamnitchi, and I. Foster, "Mapping the gnutella network," *IEEE Internet Computing*, vol. 6, pp. 50–57, January 2002.
[17] N. Ahmed, J. Neville, and R. Kompella, "Network sampling via edge-based node selection with graph induction," in *Purdue University, CSD TR #11-016*, 2011, pp. 1–10.