SIAM J. SCI. COMPUT.
Vol. 36, No. 5, pp. C424–C452


# A SCALABLE GENERATIVE GRAPH MODEL WITH COMMUNITY STRUCTURE*

TAMARA G. KOLDA†, ALI PINAR†, TODD PLANTENGA†, AND C. SESHADHRI†

**Abstract.** Network data is ubiquitous and growing, yet we lack realistic generative network models that can be calibrated to match real-world data. The recently proposed block two-level Erdős–Rényi (BTER) model can be tuned to capture two fundamental properties: degree distribution and clustering coefficients. The latter is particularly important for reproducing graphs with community structure, such as social networks. In this paper, we compare BTER to other scalable models and show that it gives a better fit to real data. We provide a scalable implementation that requires only $O(d_{\max})$ storage, where $d_{\max}$ is the maximum number of neighbors for a single node. The generator is trivially parallelizable, and we show results for a Hadoop MapReduce implementation for modeling a real-world Web graph with over 4.6 billion edges. We propose that the BTER model can be used as a graph generator for benchmarking purposes and provide idealized degree distributions and clustering coefficient profiles that can be tuned for user specifications.

**Key words.** graph generator, network data, block two-level Erdős–Rényi (BTER) model, large-scale graph benchmarks

**AMS subject classifications.** 05C80, 05C82, 90B15

**DOI.** 10.1137/130914218

## 1. Introduction.

Network interaction data is now available from online social interactions, computer-to-computer communications, financial transactions, collaboration networks, telecommunications, and more. A major obstacle to working in the field of network science is that access to data is restricted due to a combination of security and privacy concerns; yet models, algorithms, software, and hardware are struggling to keep pace with increasing demands for scalability and relevance. For these reasons, network science researchers need scalable generative models for large-scale graphs. Ideally, these generative models should capture salient features of the networks being modeled.

Suppose we are given a graph representation of our data set. We introduce basic terminology for those unfamiliar with graph theory. Let $G = (V, E)$ be an undirected, unweighted graph. We let $V$ denote the set of *vertices* or *nodes* of the graph, and we let $E$ denote the set of *edges* where $(i, j) \in E$ means there is an edge between nodes $i$ and $j$ or, equivalently, that nodes $i$ and $j$ are *adjacent*. We assume that the graph is simple, meaning that the edges are undirected and unweighted and that there are no self-edges. Let $V_i = \{\, j \mid (i, j) \in E \,\}$ denote the set of nodes that are adjacent to $i$; then the *degree* of node $i$ is $d_i = |V_i|$. The *transitivity* or *clustering coefficient* of node $i$ is defined as $c_i = |\{\, (j, k) \in E \mid j, k \in V_i \,\}| / \binom{d_i}{2}$, and it is a measure of the proportion of triangles that node $i$ participates in compared to the number of possible triangles

*Submitted to the journal's Software and High-Performance Computing section March 25, 2013; accepted for publication (in revised form) March 28, 2014; published electronically September 25, 2014. This work was funded by the GRAPHS Program at DARPA and by the Applied Mathematics Program at the U.S. Department of Energy. Sandia National Laboratories is a multiprogram laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

http://www.siam.org/journals/sisc/36-5/91421.html

†Sandia National Laboratories, Livermore, CA 94551 (tgkolda@sandia.gov, apinar@sandia.gov, tplante@sandia.gov, scomand@sandia.gov).

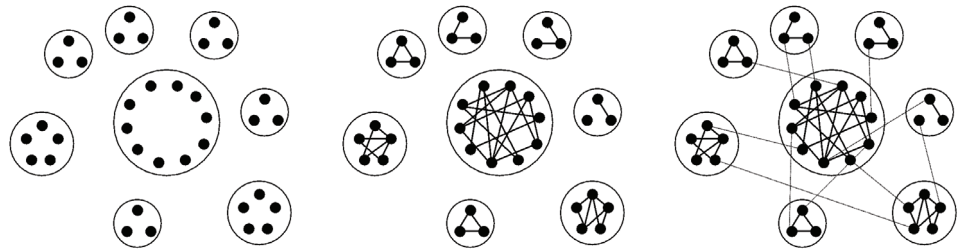
[50]. Roughly speaking, $c_i$ captures the social cohesion around node $i$; for instance, in a clique (i.e., a graph in which every node is adjacent to every other node), every node has $c_i = 1$.

In this paper, we consider how to reproduce two fundamental properties of graphs: the degree distribution and the clustering coefficients by degree [49]. Let $V_d = \{ i \mid d_i = d \}$ be the set of nodes of degree $d$, and define $n_d = |V_d|$. The degree distribution is specified by the sequence $\{ n_d \}_{d \in \mathbb{N}}$. In most real-world networks representing interaction data, there are a few nodes with high degrees and many nodes with low degrees, with a smooth transition between them. In other words, the degree distribution is heavy-tailed, and this feature has long been considered critical in distinguishing real networks from arbitrary sparse networks [2, 13, 43]. Let $c_d = \frac{1}{n_d} \sum_{i \in V_d} c_i$ be the average clustering coefficient for nodes of degree $d$. The clustering coefficients by degree are specified by the sequence $\{ c_d \}_{d \in \mathbb{N}}$. The clustering coefficients of real-world graphs are much higher than those of random graphs with the same degree distribution [18]. Nonetheless, most generative models fail to match clustering coefficients of real-world graphs [42].

In previous work by a subset of the authors, we introduce the block two-level Erdős–Rényi (BTER) model [44]. The BTER model can be tuned to capture both the degree distribution and degreewise clustering coefficients for real-world networks. The inputs to the BTER model are the sequences $\{ n_d \}_{d \in \mathbb{N}}$ and $\{ c_d \}_{d \in \mathbb{N}}$; our goal is to create a graph with similar behavior in these measures. To do this, BTER works as follows. The nodes are divided into *affinity blocks*; see Figure 1a. The fact that we can infer community structure from local clustering coefficient measurements is justified in the original BTER paper [44] and now has additional theoretical justification since it has recently been shown that triangle-rich graphs resemble unions of dense blocks [22]. Each node has an assigned degree in such a way that, in expectation, the degree distribution matches what it specified by $\{ n_d \}_{d \in \mathbb{N}}$. For each node, the links are divided between local links within the affinity block (Phase 1 in Figure 1b) and global links that may connect to any node (Phase 2 in Figure 1c). The proportion of local links depends on the clustering coefficients specified by $\{ c_d \}_{d \in \mathbb{N}}$. Edges are generated by choosing endpoints in a probabilistic way, as is explained in detail further on. The goal of this paper is to explain the specifics of the model and provide a scalable implementation. We also provide additional evidence of the model's usefulness.

**1.1. Contributions.** The BTER model [44] was previously introduced as a scalable model that can reproduce degree distributions and clustering coefficients. The scalability is based on *independent* edge generation; i.e., there is no knowledge of previously generated edges in deciding on the next edge. However, the original paper [44] did not specify implementation details and, moreover, recommended an ad hoc procedure for some of the parameter choices. In this paper, we make the following contributions:

- We provide a detailed reference implementation that clearly explains how to choose the BTER parameters to match the specified degree distribution and clustering coefficient profile. We note that there is no iterative optimization to fit BTER; the parameters of the model are directly calculated from the inputs. The edges are generated independently and in an arbitrary order, so the BTER generative model can potentially be used in streaming scenarios.
- We present efficient data structures for a scalable implementation, requiring only $O(d_{\max})$ storage and $O(\log d_{\max})$ operations per edge, where $d_{\max}$ is the maximum degree. Since our approach generates all edges independently, it can be easily parallelized. We provide examples demonstrating the scalability.



(a) Preprocessing: Distribution of nodes into affinity blocks.

(b) Phase 1: Local links within each affinity block.

(c) Phase 2: Global links across affinity blocks.

FIG. 1. *BTER model phases* [44].

• We demonstrate that BTER gives good approximations to the degree distributions and clustering coefficient behaviors of several large, heavy-tailed, real-world graphs. We consider examples from the Laboratory for Web Algorithms [30], including a graph with over 130 million nodes and 4.6 billion edges, the largest publicly available graph of which we are aware. We also compare BTER to competing methods on a pair of smaller graphs.

• We consider how BTER may be used for arbitrary benchmarking purposes when there is no target graph to match. Since the model requires a degree distribution and clustering coefficient profile as input, we focus on how to generate these. In particular, we recommend the generalized log-normal distribution for the degrees as an alternative to the standard power law.

For very large graphs, the inputs can be expensive to compute, especially the clustering coefficients. However, we have recently proposed a sampling method that scales to very large graphs [47, 25].

**1.2. Related work.** Since the goal of this paper is to focus on the implementation and scalability of BTER, we limit our discussion to the most salient related models. A more thorough discussion of related work can be found in the paper that originally proposed the BTER model [44].

The majority of graph models add edges one at a time in a way that each random edge influences the formation of future edges, making them inherently serial and therefore unscalable. The classic example is preferential attachment [2], but there are a variety of related models; see, e.g., [26, 29]. These models are more focused on capturing qualitative properties of graphs and typically are difficult to match to real-world data [42]. Perhaps the most relevant is [21], which creates a graph with power law degree distribution and then "rewires" it to improve the clustering coefficients. The musketeer model starts with a given graph and then does multilevel rewiring to attempt to preserve certain features of the original [23]. Another related model, the clustering model proposed by Newman [39], assigns "individuals" to "groups" (a bipartite graph with individual and group nodes) and then creates a graph of connections between individuals by assigning connection probabilities to each group; in other words, each group is modeled as an Erdős–Rényi graph.

A widely used model for modeling large-scale graphs is the stochastic Kronecker graph (SKG) model, also known as R-MAT [8, 28]. The generation process is easily parallelized and can scale to very large graphs. Notably, SKG has been selected as the generator for the Graph 500 Supercomputer Benchmark [20] and has been used in a variety of other studies [14, 32, 35, 34, 19, 15, 33, 24]. Unfortunately, SKG has some drawbacks: (1) It can be extremely expensive to fit to real data (using KronFit, the SKG parameter fitting algorithm proposed by the SKG inventors), and even then the fit is imperfect [28]; (2) it can generate only log-normal tails (after a suitable addition of random noise) [46, 45], limiting the degree distributions that it can capture; and (3) most importantly, it rarely closes wedges, so the clustering coefficients are much smaller than what is produced in real data [42, 25].

Another model of relevance is the Chung–Lu (CL) model [10, 11, 1]. It is very similar to the edge-configuration model of Newman, Watts, and Strogatz [38]. Let $d_i$ denote the *desired degree* for node $i$. In the CL model, the probability of an edge is proportional to the product of the degrees of its endpoints; i.e., the probability of edge $(i, j)$ is $\propto d_i d_j$. Edges can be generated independently by picking endpoints proportional to their desired degrees. If all degrees are the same, CL reduces to the well-known Erdős–Rényi model [16]. The CL model is often used as a null model; for example, it is the basis of the modularity metric [40]. Graphs generated by the CL model and the SKG model are, in fact, very similar [41]. The advantage of the CL model is that it can be better tuned to real-world degree distributions. The disadvantage of the model is that, like SKG, it rarely closes wedges. CL occurs as a special case of BTER when Phase 1 is skipped (see section 3). The CL construction is a very important part of BTER and will be explained in more detail in the next section.

**2. Notation and background.** Let $G = (V, E)$ be an undirected and unweighted graph, where $V$ denotes the set of vertices and $E$ denotes the set of edges. We let $n = |V|$ and $m = |E|$ denote the total number of vertices and edges, respectively. The set of node $i$'s neighbors and the degree of node $i$ are, respectively,

$$V_i \equiv \{\, j \mid (i, j) \in E \,\} \quad \text{and} \quad d_i = |V_i|.$$

The set of degree-$d$ nodes and the number of degree-$d$ nodes are, respectively,

$$V_d = \{\, i \mid d_i = d \,\} \quad \text{and} \quad n_d = |V_d|.$$

The set $\{\, n_d \,\}$ defines the degree distribution. Observe that the degree distribution specifies the total number of nodes and edges:

$$n = \sum_d n_d \quad \text{and} \quad m = \frac{1}{2} \sum_d d \cdot n_d. \tag{2.1}$$

For convenience, we use the notation $d_{\max} = \max \{\, d_i \mid i \in V \,\}$.

**2.1. Clustering coefficients.** We can discuss clustering coefficients in terms of wedges, closed wedges, and triangles. A wedge is a path of length 2. Figure 2 shows a *wedge* centered at node $j$; i.e., the path $i$-$j$-$k$ is a wedge. We say wedge $i$-$j$-$k$ is closed if $(i, k) \in E$; otherwise, the wedge is called open. A closed wedge forms a triangle, i.e., a three-cycle. The number of wedges centered at node $i$ is $\binom{d_i}{2} = d_i(d_i - 1)/2$. The clustering coefficient at node $i$ is

$$c_i = \frac{\#\text{ of closed wedges centered at node } i}{\#\text{ wedges centered at node } i} = \frac{|\,\{\, (j, k) \in E \mid j, k \in V(i) \,\}\,|}{\binom{d_i}{2}}.$$

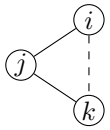



FIG. 2. *Wedge centered at node $j$. The wedge is closed if $(i,k) \in E$.*

This is the proportion of closed wedges divided by the total number of wedges. For BTER, we are specifically interested in *clustering coefficient per degree* [50], defined as

$$c_d = \frac{\#\text{ of closed wedges centered at a node of degree } d}{\#\text{ wedges centered at a node of degree } d} = \frac{1}{n_d} \sum_{i \in V_d} c_i.$$

To summarize an entire graph, it is often convenient to consider the *global clustering coefficient* (GCC) [50, 3], defined as

$$c = \frac{\#\text{ of closed wedges}}{\#\text{ wedges}}.$$

Note that this is *not* the mean of the $c_i$'s. Since every triangle corresponds to three closed wedges, the number of closed wedges is three times the number of triangles.

**2.2. Erdős–Rényi graphs.** As described in section 3, BTER uses Erdős–Rényi to model its affinity blocks. An Erdős–Rényi graph [16] on $n$ vertices with connection probability $\rho$ is a graph such that each pair of vertices is independently connected with probability $\rho$. We refer to $\rho$ as the *connectivity*. If $\rho$ is a constant, we call this a dense Erdős–Rényi graph; if $\rho = O(1/n)$, then we call this a sparse Erdős–Rényi graph.

**2.3. Chung-Lu graphs.** BTER uses a variation on CL for its global links that span across affinity blocks. Here we explain salient details that are relevant to the implementation discussion.

The CL model [10, 11, 1] approximates the probability that edge $(i,j)$ exists by

$$\Pr\left((i,j) \in E\right) = \frac{d_i d_j}{2m},$$

where $d_i$ indicates the desired degree of node $i$. We can perform an independent coin flip for each of the possible $n^2$ edges, but this is too expensive. We consider a "fast" version of the CL model that generates edges by independently choosing endpoints proportional to their degrees. Rather than doing an independent coin flip for each of $n^2$ possible edges, we can do $2m$ coin flips to pick two random endpoints per edge. Each endpoint is picked independently such that the probability of picking node $i$ is

$$\Pr\left(i\right) = d_i/2m.$$

Hence, the probability (or, more precisely, the expected value) of edge $(i,j)$ after picking $m$ edges is

$$\Pr\left((i,j) \in E\right) = m \cdot 2 \cdot \Pr\left(i\right) \cdot \Pr\left(j\right) = \frac{d_i d_j}{2m}.$$

The factor of 2 is because $(i,j)$ and $(j,i)$ are equivalent for an undirected graph.

Let $D_i$ correspond to the random variable that is the degree of node $i$ for a single graph generated according to this procedure. In the "fast" version, it is easy to see that the value of $D_i$ is Poisson distributed with mean $d_i$ [15], so $\mathbb{E}(D_i) = d_i$, i.e., the expected value of $D_i$ is $d_i$. Note that CL admits nonintegral desired degrees. If the desired degree is $d_i = 3.5$, then $\mathbb{E}(D_i) = 3.5$ even though every realization $D_i$ is integral.

Despite the variation in individual degrees, the degree distribution is usually a good match to the desired one because there is spill-over from adjacent degrees. For instance, some nodes that were expected to become degree 4 are instead degree 5 and vice versa. This will not be the case if there are gaps in the degree distribution (i.e., $n_3, n_5 \gg 0$ and $n_4 = 0$). Also, degree-1 nodes may pose a particular problem. According to the calculations in [15], approximately 36% of the pool of potential degree-1 nodes will not be selected (i.e., have degree zero) and another 28% will have degree 2 or larger. To counteract these problems, [15] proposes to increase the pool of potential degree-1 nodes while keeping the total probability of potential degree-1 nodes constant. Specifically, we "blow up" the set of degree-1 nodes by a factor $\beta \geq 1$. Hence, we add $(\beta - 1) \cdot n_1$ nodes with desired degree $d_1$, but we adjust the probability of picking any individual degree-1 node so that

$$\Pr(i) = \begin{cases} d_i/2m & \text{if } d_i \geq 2, \\ \beta^{-1}/2m & \text{if } d_i = 1. \end{cases}$$

The BTER algorithm has a similar issue with degree-1 nodes, so the reference implementation includes the option to specify a blowup factor, $\beta$.

Finally, we note that since each endpoint and each edge is picked independently, a graph generated according to fast CL may contain self-edges and repeat edges. We have ignored these details in the discussion above because the practical impact has been small in our experiments (we simply discard such edges).

**3. BTER generative graph model.** We give a high-level overview of BTER in section 3.1 and discuss the challenges of a scalable implementation in section 3.2. The remainder of the section addresses the proposed solutions to these challenges and presents the implementation.

**3.1. The BTER model.** BTER is based on the premise that a graph with a heavy-tailed degree distribution and high clustering coefficients must contain dense Erdős–Rényi blocks; moreover, the distribution of the sizes of those groups follows the same type of distribution of the degrees [44]. Therefore, BTER organizes nodes into affinity blocks such that nodes within the same affinity block have a much higher chance of being connected than nodes at random, but BTER also behaves like the CL model in that it is able to match an arbitrary degree distribution.

The BTER model requires two user-specified inputs: (1) the desired degree distribution, $\{ n_d \}_{d \in \mathbb{N}}$, and (2) the desired clustering coefficients by degree, $\{ c_d \}_{d \in \mathbb{N}}$. These quantities may be measurements from an existing graph or set arbitrarily, e.g., for benchmark purposes. (We note that the original description in [44] did not take the original clustering coefficients but rather a function to determine the connectivity.) The desired number of nodes and edges can be computed from the degree distribution per (2.1). There are three main steps to the graph generation, as described below. The steps are depicted in Figure 1.

*Preprocessing.* Imagine starting with $n$ isolated vertices. Each vertex is assigned a degree, based on the degree distribution $\{n_d\}$. So we arbitrarily assign $n_1$ vertices to

have degree 1, $n_2$ to have degree 2, etc. We then partition the $n$ vertices into affinity blocks. For reasons that will be explained in the details that follow, an affinity block ideally contains $d+1$ vertices that are assigned degree $d$. The idea is that each affinity block can potentially form a clique, in which case every node in that block has a clustering coefficient of 1. Note that there are many small blocks with vertices of low degree, and a few large blocks of high-degree vertices. At this stage, no edges have been added.

*Phase* 1. This phase adds edges *within* each affinity block. Each block is a dense Erdős–Rényi graph, where the density depends on the size of the block. For a block involving degree-$d$ vertices, the density is determined based on $c_d$. We show how to choose the connectivity within each block to ensure that each vertex achieves its desired clustering coefficient and is not incident to more edges than its desired degree (in expectation).

*Phase* 2. This phase adds edges between the blocks. Consider some vertex $i$ with an assigned degree $d_i$. Suppose it is already incident to $d'_i$ edges from Phase 1. We set $w_i = d_i - d'_i$ to be the *excess degree*[1] of $i$. We must create $e_i$ edges incident to vertex $i$ to satisfy its degree requirement. We construct a CL graph with degree sequence $\{ w_i \}_{i=1}^{n}$ to complete the graph construction.

**3.2. Developing a scalable implementation.** Our goal is to show that it is possible to have a highly scalable implementation of the BTER method. The main goal is to have *independent edge insertions* so that the edge generation can be parallelized.

As stated, Phase 2 edge insertions must happen after Phase 1, because we need to know the excess degrees. We parallelize this process by computing the *expected* excess degree. Given all the input parameters, we can precompute the expected excess degree for any vertex (this requires compact representations and data structures) during the preprocessing. From this, we can precompute the total number of Phase 1 and Phase 2 edges.

To perform a parallel edge insertion, we first decide randomly whether this should be a Phase 1 or Phase 2 edge. For a Phase 1 edge, we select a random affinity block (with the appropriate probability) and create an edge between two distinct randomly selected block members. For a Phase 2 edge, we perform a CL edge insertion based on expected excess degrees. Because every edge is generated independently, there will be duplicates, but these are discarded in the final graph.

Given the structure of parallel edge insertion, the main challenges in developing a scalable implementation are as follows:

- *Preprocessing data structures.* A naïve implementation of the preprocessing step would require $O(n)$ variables and storage by storing the Phase 1 and Phase 2 degrees of every node. We design compact representations and data structures for the affinity blocks. This contains all the relevant information with minimal storage.
- *Repeats in Phase* 1. Independent edge generation in Phase 1 leads to many repeated edges without enough distinct edges, and this affects the overall degree distribution when edge repeats are removed. We show how to determine the number of extra Phase 1 edges to be inserted to rectify this.

[1] This definition of excess degree should not be confused with the "excess degree distribution" of Newman [37].

- *The degree-1 problem of Phase 2.* A parallel implementation of Phase 2 results in numerous degree-1 vertices becoming isolated. We use a fix for this discussed in section 2.3, which is different than the one proposed in the original BTER paper [44].

Once these issues are addressed, we have an embarrassingly parallel edge generation algorithm that requires only $O(\log d_{\max})$ work per edge. The remainder of this section gives an in-depth but informal presentation of our implementation. Detailed algorithm specifications at pseudocode level are also provided.

**3.3. Preprocessing.** Let $d_i$ denote the (desired) total degree of vertex $i$ and $w_i$ denote its (desired) excess degree. For convenience, the nodes are indexed by increasing degree *except* for degree-1 nodes, which are indexed last. Hence, if $d_i, d_j \geq 2$ and $i < j$, then $d_i \leq d_j$. For an example, see the numbering in Figure 3.

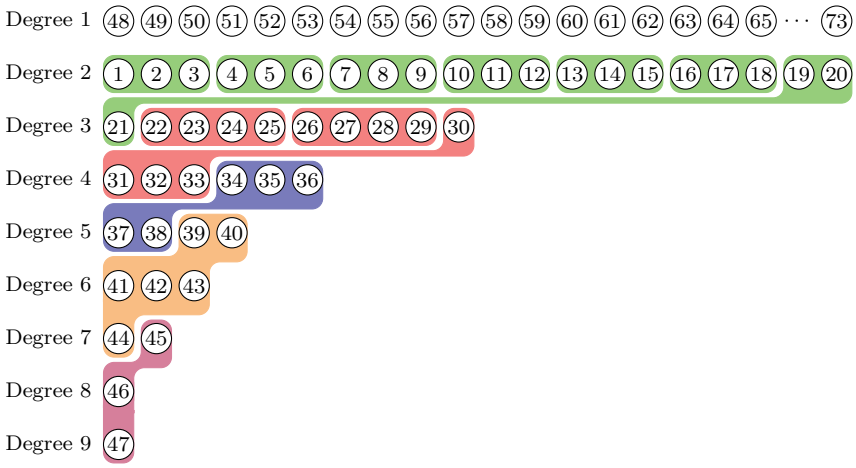



FIG. 3. *Example of affinity blocks and groups for a graph of* 73 *nodes. Connected nodes denote the same affinity block. The same color denotes the same group membership. (See online version for color.)*

In the preprocessing phase, we assign nodes to affinity blocks. We let $b_i$ denote the block assignment of node $i$. For the assignment to affinity blocks, degree-1 nodes are ignored. The remaining nodes are assigned to affinity blocks in order (of degree). A *homogeneous* affinity block has $d + 1$ nodes of degree $d$. In Figure 3, the blocks are denoted by colored ovals, and 4-5-6 is a homogeneous block. The vast majority of (low-degree) nodes are assigned to homogeneous affinity blocks. However, there are not always enough nodes of degree $d$ to fill in a homogeneous block; therefore, we also have a few (at most $d_{\max}$) *heterogeneous* affinity blocks with nodes of different degrees. For instance, in Figure 3, 19-20-21 is a heterogeneous block.

For Phase 1, we let $i_b$ denote the starting index of block $b$, $d_b$ denote the minimum degree of block $b$, $\rho_b$ denote its desired connectivity, $n_b$ denote the number of nodes, $m_b$ denote the desired number of edges in the affinity block, and $w_b$ denote the block *weight,* which is the number of edges to be inserted that accounts for expected repeats (see section 3.6). Not all this information needs to be saved. To generate Phase 1 edges, we need only $i_b$, $n_b$, and $w_b$ to know the nodes that are involved with each block and how many edges to insert. The process will insert $w_b$ edges which will, in expectation, produce $m_b$ unique edges so that each node within the block will have expected local (i.e., inside the affinity block) degree $m_b/\binom{n_b}{2}$. However, these

three values are not what are stored since we can store even less information for more efficiency.

We can compress the data further since all affinity blocks of the same size and minimum degree can be grouped together into an *affinity group*—all blocks in the same group share the same block size and weight. In Figure 3, all nodes with the same color are in the same affinity group, e.g., 1–21 are in the same affinity group, likewise nodes 22–33, etc. The information needed to store an affinity group boils down to four items of information: $i_g$, the starting index of the group; $b_g$, the number of blocks in the group; $n_g$, the size of each block; and $w_g$, the total weight of all blocks in the group. The maximum number of groups is bounded by $d_{\max}$, so we store at most $4 \cdot d_{\max}$ values.

Phase 2 needs to build a CL model using the excess degrees, $\{ w_i \}$. We note that these values are computed in advance and are fixed throughout the entire process. We can exploit the homogeneity of nodes of the same degree to save space for this calculation as well. In a block where all nodes have the same degree, we say the nodes are *bulk nodes*. In a block with nodes of differing degrees, all nodes with degree equal to the minimum degree are still bulk nodes. We refer to the remaining nodes as *filler nodes*. In Figure 3, nodes 1–20 are degree-2 bulk nodes, nodes 22–30 are degree-3 bulk nodes, nodes 34–36 are degree-4 bulk nodes, and so on. Node 21 is a degree-3 filler node, nodes 31–33 are degree-4 filler nodes, etc. It is possible to have either no bulk nodes or no filler nodes for a given degree. In Figure 3, there are no filler degree-2 nodes and no bulk degree-6 nodes. Observe that all bulk nodes of degree $d$ (for any $d$) are in blocks of the same size and connectivity (i.e., the same internal degree); therefore, they all have the same excess degree. The filler nodes of degree $d$ (for any $d$) participate in at most one block and so all have the same excess degree. This means that there are two possible values for excess degree for the set of nodes with desired degree $d$. Hence, to generate Phase 2 links we need just five values for degree $d$: $n_d^{\text{fill}}$ and $n_d^{\text{bulk}}$, the number of filler and bulk nodes; $w_d^{\text{fill}}$ and $w_d^{\text{bulk}}$, the total weight for filler nodes and for bulks nodes, which is exactly the total excess degree for each; and the starting index $i_d$. Technically, the starting index can be computed from $\{ n_d \}$, but it reduces the work to store these indices explicitly. However, we actually do not store $n_d^{\text{bulk}}$ since it can be computed using $n_d$ and $n_d^{\text{fill}}$. Additionally, because of how they are used, it is more convenient to store $w_d = w_d^{\text{fill}} + w_d^{\text{bulk}}$ and $r_d = w_d^{\text{fill}}/w_d$ than $w_d^{\text{fill}}$ and $w_d^{\text{bulk}}$. Hence, the total working storage for Phase 2 information is $5 \cdot d_{\max}$ values.

The total storage (including inputs) needed by the generation routine is $10 \cdot d_{\max}$ values (the sequence $\{ c_d \}$ is not retained). It is possible to modify the core data structures to store only the distinct degrees instead of maintaining a continuum of degrees through $d_{\max}$. This would change the storage requirement to $O(d_{\text{uniq}})$ instead of $O(d_{\max})$, where $d_{\text{uniq}}$ is the number of distinct degrees in the graph. However, we present our ideas based on $O(d_{\max})$ storage for clarity of presentation.

For convenience, notation is described in Table 1. The node- and block-level variables are not used in the algorithms.

**3.4. Preprocessing algorithm.** The BTER setup procedure is described in Algorithm 1. The inputs are the degree distribution, $\{ n_d \}$; the clustering coefficients per degree, $\{ c_d \}$; and the blowup factor for degree-1 nodes, $\beta$.

The method precomputes the index for the first node of each degree, $\{ i_d \}$, and the number of nodes with degree greater than the degree $d$, $\{ n'_d \}$. The latter is not saved after the preprocessing phase.

The degree-1 nodes are handled in a special way. All degree-1 nodes are designated

TABLE 1

*Description of variables. A ★ symbol in the second column indicates that this variable is explicitly computed and stored for use by the sampling procedure; a × symbol means the variable is not explicitly computed. The last column gives the formula to derive it from the stored values.*

| | | | |
|---|---|---|---|
| ***Scalars*** | | | |
| $n$ | | Total number of nodes | $n = \sum n_d$ |
| $m$ | | Total number of edges | $m = \frac{1}{2}\sum d \cdot n_d$ |
| $w$ | | Total number of edges to insert | $w = w^{(1)} + w^{(2)}$ |
| $d_{\max}$ | ★ | Largest (desired) degree | |
| $b_{\max}$ | × | Total number of affinity blocks | $b_{\max} = \sum_g b_g$ |
| $g_{\max}$ | ★ | Total number of affinity groups | $g_{\max} \le d_{\max}$ |
| $\beta$ | ★ | Blowup factor for degree-1 vertices | |
| ***Node level*** $i = 1, \ldots, n$ | | | |
| $d_i$ | × | Degree (desired) of node $i$ | |
| $b_i$ | × | Block id for degree $i$ | |
| $w_i$ | × | Excess degree of node $i$ | $w_i = \frac{1}{2}[d_i - (\rho_{b_i} \cdot d_{b_i})]$ |
| ***Block level*** $b = 1, \ldots, b_{\max}$ | | | |
| $d_b$ | × | Minimum degree in block $b$ | |
| $\rho_b$ | × | Connectivity of degree $b$ | $\rho_b = \sqrt[3]{c_{d_b}}$ |
| $n_b$ | × | Number of nodes in block $b$ | $n_b = d_b + 1$ |
| $m_b$ | × | Number of unique edges in block $b$ | $m_b = \rho_b \binom{n_b}{2}$ |
| $w_b$ | × | Weight of block $b$ | $w_b = \binom{n_b}{2} \ln(1/(1-\rho_b))$ |
| ***Degree level*** $d = 1, \ldots, d_{\max}$ | | | |
| $n_d$ | ★ | Number of nodes of degree $d$ | |
| $c_d$ | ★ | Mean clustering coefficient for nodes of degree $d$ | |
| $i_d$ | ★ | Index of first degree of degree $d$ | |
| $n'_d$ | | Number of nodes of degree greater than $d$ | $n'_d = \sum_{d'>d} n'_{d'}$ |
| $n_d^{\text{fill}}$ | ★ | Number of fill nodes of degree $d$ | |
| $n_d^{\text{bulk}}$ | | Number of bulk nodes of degree $d$ | $n_d^{\text{bulk}} = n_d - n_d^{\text{fill}}$ |
| $w_d$ | ★ | Excess degree of nodes of degree $d$ | |
| $r_d$ | ★ | Ratio of fill excess degree for degree $d$ | |
| $w_d^{\text{fill}}$ | | Excess degree of fill nodes of degree $d$ | $w_d^{\text{fill}} = r_d \cdot w_d$ |
| $w_d^{\text{bulk}}$ | | Excess degree of bulk nodes of degree $d$ | $w_d^{\text{bulk}} = w_d - w_d^{\text{fill}}$ |
| ***Group level*** $g = 1, \ldots, g_{\max}$ | | | |
| $i_g$ | ★ | Index of first node in group $g$ | |
| $b_g$ | ★ | Number of affinity blocks in group $g$ | |
| $n_g$ | ★ | Number of nodes per block in group $g$ | |
| $w_g$ | ★ | Weight of group $g$ (including duplicate edges) | |
| ***Phase level*** $k = 1, 2$ | | | |
| $w^{(k)}$ | | Weight of phase $k$ | $w^{(1)} = \sum_g w_g$<br>$w^{(2)} = \sum w_d$ |

as "fill" nodes. The number of candidate degree-1 nodes may be increased using the blowup factor, $\beta$. However, if the blowup is used, the majority of the (desired) degree-1 nodes will ultimately have degree 0 and can be removed in postprocessing.

The main loop walks through each degree, determining the information for Phases 1 and 2. It first allocates degree-$d$ nodes as fill nodes for the last incomplete block, if needed. The number of nodes necessary to complete the last incomplete block in the previous group is denoted by $n_*^{\text{fill}}$. The excess degree of any fill nodes depends on the internal degree of the last incomplete block, denoted by $d_*$. The excess degree is used to determine the weight of the degree-$d$ fill nodes for Phase 2, $w_d^{\text{fill}}$.

If more nodes of degree $d$ remain, these are allocated as bulk nodes, and a new group is formed. The number of bulk nodes of degree $d$ is denoted by $n_d^{\text{bulk}}$. For the new group, we determine the index of the first node, the number of blocks, and the size of each block. The very last block of the very last group is special because the

remaining nodes may not be enough to fill it. For simplicity, and because it is often the case for heavy-tailed networks, we assume the last group contains only one block. This simplifies the task of handling it as a special case. We compute excess degree for these bulk nodes and then the corresponding weight of degree-$d$ bulk nodes for Phase 2, $w_d^{\text{bulk}}$. We also compute the weight of the group, $w_g$, according to a special formula, as justified in section 3.6. Finally, we compute the number of nodes needed to fill out the final block in the group currently being processed, $n_*^{\text{fill}}$.

Rather than storing $w_d^{\text{fill}}$ and $w_d^{\text{bulk}}$ directly, it is easier (for the edge generation phase) to have their sum, $w_d$, and the ratio of fill nodes, $r_d$. Likewise, we do not return $n_d^{\text{bulk}}$ since it can be easily recomputed using $n_d$ and $n_d^{\text{fill}}$. We do return $i_d$, but this step can be omitted and recomputed if this is more efficient (e.g., reducing communication to workers in a parallel setting). Finally, we no longer need to keep $\{ c_d \}$ after the preprocessing is complete.

**3.5. Phase 1.** Phase 1 creates intrablock links. Each affinity block is modeled as an Erdős–Rényi graph. An overwhelming majority of the triangles are formed in this phase, and thus we pick the Erdős–Rényi constant, $\rho$, for the block to match the target clustering coefficient $c$. A vertex of degree $d$ and clustering coefficient $c$ is incident to $c \cdot \binom{d}{2}$ triangles. Assume this vertex is grouped with other vertices of degree $d$ into a block with $d+1$ vertices, which holds for all homogeneous blocks. If we build an Erdős–Rényi graph of this block with parameter $\rho$, then this vertex is expected to be incident to $\binom{d}{2}\rho^3$ triangles. Solving for $\rho$ yields $\rho = \sqrt[3]{c}$. Therefore, for block $b$, the connectivity is $\rho_b = \sqrt[3]{c_{d_b}}$, where $d_b$ denotes the minimum degree in the block (since most blocks are homogeneous, this choice works well). Note that the clustering coefficients of vertices will be higher if we consider only the affinity blocks. This is to compensate for the edges that will be added in Phase 2 to increase the number of wedges—likely without contributing any triangles.

The difficulty in Phase 1 is that we expect a preponderance of repeat edges because edges are generated independently. Consider affinity block $b$ with $n_b$ nodes and connectivity $\rho_b$, meaning that each node in block $b$ wants internal degree $\rho_b \cdot d_b$. BTER wants approximately $m_b = \rho_b \binom{n_b}{2}$ *distinct* edges in block $b$. Determining the number of draws with replacement to get a desired number of distinct items can be cast as a *coupon collector problem*. Specifically, the coupon collector problem is as follows: "Suppose we have a box with $x$ distinct coupons. We draw a coupon and return it to the box (sampling with replacement). How many draws do we need to find $y$ distinct coupons?" Our problem is slightly different than the standard problem since we want to draw $y$ distinct edges, where $y = m_b$ may not be integral; this is fine since the goal is to achieve $m_b$ distinct edges in expectation. We have developed a good approximation for the expected number of edges that need to be inserted:

$$w_b = \binom{n_b}{2} \ln(1/(1-\rho_b)). \tag{3.1}$$

The proof is provided in Appendix A.

We illustrate the utility of (3.1) by an example with $n_b = 10$ nodes and connectivity $\rho_b = 0.5$, corresponding to $m_b = 22.5$ edges, on average. In this case, the formula predicts that we need to do $w_b = 31.1916$ draws, in expectation, to see the desired number of unique edges, in expectation. We do 10,000 random experiments as follows. For $i = 1, \ldots, 10{,}000$, the random variable $X_i \sim \text{Poisson}(w_b)$ is the number of items *drawn* from the $\binom{n_b}{2} = 45$ possible edges, and $Y_i$ is the number of those items that are *unique*. A histogram of the $Y_i$ values is shown in Figure 4. The average number of unique items is exactly the desired value.

**Algorithm 1** BTER setup.

**procedure** BTER_SETUP($\{ n_d \}$, $\{ c_d \}$, $\beta$)

*Number nodes from least degree to greatest, except degree-1 nodes are last*

$i_2 \leftarrow 1$
**for** $d = 3, \ldots, d_{\max}$ **do**
  $i_d \leftarrow i_{d-1} + n_{d-1}$
**end for**
$i_1 \leftarrow i_{d_{\max}} + n_{d_{\max}}$

*Compute number of nodes with degree greater than d*

**for** $d = 1, \ldots, d_{\max}$ **do**
  $n'_d \leftarrow \sum_{d' > d} n'_{d'}$
**end for**

*Handle degree-1 nodes*

$n_1^{\text{fill}} \leftarrow \beta \cdot n_1$, $w_1 \leftarrow \frac{1}{2} n_1$, $r_1 \leftarrow 1$

*Main loop*

$g \leftarrow 0$, $n_*^{\text{fill}} \leftarrow 0$, $d_* \leftarrow 0$
**for** $d = 2, \ldots, d_{\max}$ **do**
  **if** $n_*^{\text{fill}} > 0$ **then** ▷ Try to fill incomplete block from current group
    $n_d^{\text{fill}} \leftarrow \min(n_*^{\text{fill}}, n_d)$
    $n_*^{\text{fill}} \leftarrow n_*^{\text{fill}} - n_d^{\text{fill}}$
    $w_d^{\text{fill}} \leftarrow \frac{1}{2} n_d^{\text{fill}} (d - d_*)$
  **else**
    $n_d^{\text{fill}} \leftarrow 0$, $w_d^{\text{fill}} \leftarrow 0$
  **end if**
  $n_d^{\text{bulk}} \leftarrow n_d - n_d^{\text{fill}}$
  **if** $n_d^{\text{bulk}} > 0$ **then** ▷ Create a new group for degree-$d$ bulk nodes
    $g \leftarrow g + 1$
    $i_g \leftarrow i_d + n_d^{\text{fill}}$
    $b_g \leftarrow \lceil n_d^{\text{bulk}} / (d+1) \rceil$
    $n_g \leftarrow d + 1$
    **if** $b_g \cdot (d+1) > (n'_d + n_d^{\text{bulk}})$ **then** ▷ Special handling of last group
      **if** $b_g \neq 1$ **then throw error**
      $n_g \leftarrow (n'_d + n_d^{\text{bulk}})$
    **end if**
    $\rho_* \leftarrow \sqrt[3]{c_d}$
    $d_* \leftarrow (n_g - 1) \cdot \rho_*$
    $w_d^{\text{bulk}} \leftarrow \frac{1}{2} n_d^{\text{bulk}} \cdot (d - d_*)$
    $w_g \leftarrow b_g \cdot \frac{1}{2} n_g (n_g - 1) \cdot \log(1/1 - \rho_*)$
    $n_*^{\text{fill}} \leftarrow (b_g \cdot n_g) - n_d^{\text{bulk}}$
  **else**
    $w_d^{\text{bulk}} \leftarrow 0$
  **end if**
  $w_d \leftarrow w_d^{\text{fill}} + w_d^{\text{bulk}}$, $r_d \leftarrow w_d^{\text{fill}} / w_d$
**end for**

**return** $\{ i_d \}, \{ w_d \}, \{ r_d \}, \{ n_d^{\text{fill}} \}, \{ w_g \}, \{ i_g \}, \{ b_g \}, \{ n_g \}$

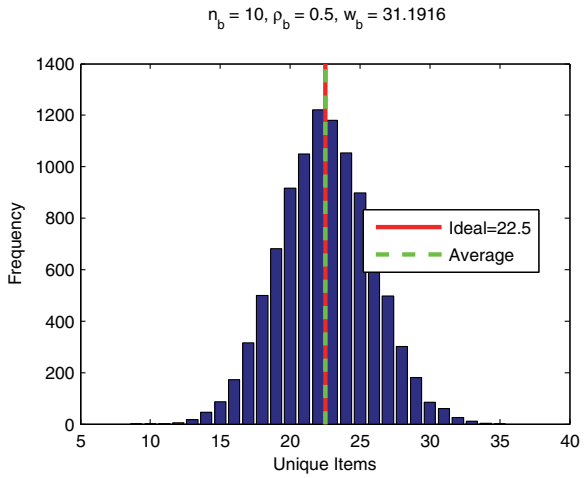



FIG. 4. *Distribution of the number of unique edges on* 10,000 *random trials.*

From (3.1), we can determine the number of extra edges needed in Phase 1. Specifically, we insert $w^{(1)} = \sum_b w_b$ edges to get a total of $m^{(1)} = \sum_b m_b$ unique Phase 1 edges. We let $w_g$ be the sum of the weights of all its constituent blocks. To generate a Phase 1 edge, the process for a single edge proceeds as follows:

1. Pick an affinity group randomly, where the probability of picking group $g$ is proportional to its weight, $w_g$.
2. Pick a block within the affinity group uniformly at random (all blocks within the same group have the same weight).
3. Pick two nodes uniformly at random *without replacement*[2] from the selected block—these two nodes form an edge.

The first step is a weighted sampling step and requires $O(\log g_{\max})$ work, where $g_{\max} \leq d_{\max}$ is the total number of affinity groups. The second and third steps are constant time operations.

**3.6. Phase 2.** Phase 2 is simply applying the CL model on the expected excess degrees. In creating an edge, we choose two nodes independently. Those nodes are chosen proportional to their excess degree. For node $i$ in group $b$, let $w_i = \frac{1}{2}[d_i - (\rho_b \cdot d_b)]$ denote half its excess degree. The total number of edges that should be inserted in Phase 2 is $w^{(2)} = \sum_i w_i$. Ignoring duplicate edges (which are fairly rare in our experiments), we have $m^{(2)} = w^{(2)}$ Phase 2 edges.

Let $n_d^{\text{fill}}$ and $n_d^{\text{bulk}}$ be the number of filler and bulk nodes of degree $d$, let $w_d = \sum_{i \in V_d} w_i$ be the weight of all degree-$d$ nodes, and let $r_d$ be the proportion that are filler nodes. Inserting a Phase 2 edge proceeds as follows:

1. Pick degree $d$, where the probability of picking degree $d$ is proportional to $w_d$.
2. Pick between filler and bulk nodes by selecting a uniform random number $x \in [0, 1]$ and choosing filler if $x < r_d$ and bulk otherwise.

[2]The terminology "without replacement" means that the first node is selected uniformly at random from the $n_b$ nodes in block $b$ and the second node is selected uniformly at random from the *remaining* $n_b - 1$ nodes in block $b$.

3. Pick a filler or bulk node (as determined in step 2) of degree $d$, uniformly at random.

The first step is a weighted sampling step and required $O(\log(d_{\max}))$ work, while the other two steps are constant time operations.

One complication in Phase 2 is that getting the correct number of degree-1 nodes poses a problem—approximately 36% of the pool of potential degree-1 nodes will not be selected and another 28% will have degree 2 or larger. A fix for this problem has been proposed in [15], which involves increasing the pool of degree-1 nodes, without changing the expected number of edges that will be connected to these vertices. This increase in the pool size is controlled by the "blowup factor," $\beta \geq 1$. This is included in the setup described in Algorithm 1 (line 10).

**3.7. Independent edge generation.** Lastly, we pull everything together to explain the independent edge generation. We insert a total of $w = w^{(1)} + w^{(2)}$ edges, flipping a weighted coin for each edge to determine if it is Phase 1 or Phase 2. We expect to generate a total of $m = m^{(1)} + m^{(2)}$ edges.

Generating the edges is extremely inexpensive: $O(\log(d_{\max}))$ per edge. The expensive step is deduplication, where extra copies of repeated edges are removed. The same difficulty exists for the current Graph 500 (SKG) benchmark. Some may argue that duplicate edges are a useful feature since real data also has duplicates, but it is not clear that the duplication rates are similar to those observed in real data.

**3.8. BTER sampling implementation.** BTER edge generation is shown in Algorithm 2. The procedure RANDOM_SAMPLE does a weighted sampling according to a specified discrete distribution. For $p$ bins, the cost is $O(\log(p))$. For each edge, we randomly select between the phases using a weighted coin. A Phase 1 edge requires one sample from a discrete distribution of size $g_{\max}$ and three additional random values drawn uniformly from $[0, 1]$. A Phase 2 edge requires two samples from a discrete distribution of size $d_{\max}$ and four additional random values drawn uniformly from $[0, 1]$. Since $g_{\max} \leq d_{\max}$, an upper bound on the cost per edge is the cost of one discrete random sample on a distribution of size $d_{\max}$ plus four random values drawn uniformly from $[0, 1]$.

In Algorithm 2, we generate each edge independently. It may also be possible to "bulk" the computations by first determining the total number of edges for each phase and perform the computation for each phase separately. Within each phase, the procedure itself can be easily vectorized to boost runtime performance, as in MATLAB.

**3.9. Edge deduplication.** Any method can be used for deduplication. In general, the simplest procedure is to hash the edges in such a way that $(i, j)$ and $(j, i)$ hash to the same key. Then it is easy enough to sort each bucket to remove duplicates. In a parallel environment, since we are hashing by edge and not vertex, there should not be load balancing problems. In fact, hashing by a single endpoint is not recommended because of the heavy-tailed nature of the graph.

**3.10. Implementations.** We have a reference implementation in MATLAB that is available at http://www.sandia.gov/~tgkolda/feastpack/. We have also implemented the method in Hadoop MapReduce and use this version in some of our experiments. Since the BTER algorithm generates edges independently, map tasks can perform all the work of creating edges, and reduce tasks simply remove duplicate edges; hence, the implementation runs as a single MapReduce job. Each map task is given the desired degree and clustering coefficient, which is sufficient to compute

**Algorithm 2** BTER sample.

**procedure** BTER_SAMPLE($\{n_d\}, \{i_d\}, \{w_d\}, \{r_d\}, \{n_d^{\text{fill}}\}, \{w_g\}, \{i_g\}, \{b_g\}, \{n_g\}$)
  $w^{(1)} \leftarrow \sum_g w_g$, $w^{(2)} \leftarrow \sum w_d$, $w \leftarrow w^{(1)} + w^{(2)}$
  $E \leftarrow \emptyset$
  **for** $j = 1, \ldots, w$ **do**
    $r \sim U[0,1]$
    **if** $r < w^{(1)}/w$ **then**
      $E \leftarrow E \cup$ BTER_SAMPLE_PHASE1($\{w_g\}, \{i_g\}, \{b_g\}, \{n_g\}$)
    **else**
      $E \leftarrow E \cup$ BTER_SAMPLE_PHASE2($\{w_d\}, \{r_d\}, \{n_d\}, \{n_d^{\text{fill}}\}, \{i_d\}$)
    **end if**
  **end for**
  **return** $E$

**procedure** BTER_SAMPLE_PHASE1($\{w_g\}, \{i_g\}, \{b_g\}, \{n_g\}$)
  $g \leftarrow$ RANDOM_SAMPLE($\{w_g\}$) ▷ Choose group
  $r_1 \sim U[0,1]$, $\delta = i_g + \lfloor r_1 \cdot b_g \rfloor \cdot n_g$ ▷ Choose block and compute its offset
  $r_2 \sim U[0,1]$, $i \leftarrow \lfloor r_2 \cdot n_g \rfloor + \delta$ ▷ Choose 1st node
  $r_3 \sim U[0,1]$, $j \leftarrow \lfloor r_3 \cdot (n_g - 1) \rfloor + \delta$ ▷ Choose 2nd node
  **if** $j \geq i$ **then**
    $j \leftarrow j + 1$
  **end if**
  **return** $(i, j)$

**procedure** BTER_SAMPLE_PHASE2($\{w_d\}, \{r_d\}, \{n_d\}, \{n_d^{\text{fill}}\}, \{i_d\}$)
  $i \leftarrow$ BTER_SAMPLE_PHASE2_NODE($\{w_d\}, \{r_d\}, \{n_d\}, \{n_d^{\text{fill}}\}, \{i_d\}$)
  $j \leftarrow$ BTER_SAMPLE_PHASE2_NODE($\{w_d\}, \{r_d\}, \{n_d\}, \{n_d^{\text{fill}}\}, \{i_d\}$)
  **return** $(i, j)$

**procedure** BTER_SAMPLE_PHASE2_NODE($\{w_d\}, \{r_d\}, \{n_d\}, \{n_d^{\text{fill}}\}, \{i_d\}$)
  $d \leftarrow$ RANDOM_SAMPLE($\{w_d\}$) ▷ Choose degree
  $r_1 \sim U[0,1]$, $r_2 \sim U[0,1]$
  **if** $r_1 < r_d$ **then**
    $i \leftarrow \lfloor r_2 \cdot n_d^{\text{fill}} \rfloor + i_d$ ▷ Fill node
  **else**
    $i \leftarrow \lfloor r_2 \cdot (n_d - n_d^{\text{fill}}) \rfloor + (i_d + n_d^{\text{fill}})$ ▷ Bulk node
  **end if**
  **return** $i$

affinity blocks and sampling probabilities. Each map task uses a different seed for random number generation used in creating edges. Map tasks are assigned a fixed number of edges to generate. The default is one million edges, so a graph of $w$ edges requires $w/10^6$ map tasks. There is no HDFS input file for the map tasks; instead, we wrote a subclass of ORG.APACHE.HADOOP.MAPREDUCE.INPUTFORMAT that causes a map task to generate a given number of records. For each edge, the map emits a key-value pair consisting of a hash value for the edge (based on the two endpoints) and the endpoints. Note that there is no assignment of specific nodes to specific map or reduce tasks. The reducer tasks collect edges with the same hash value and remove any duplicates before emitting the final list of all edges. We enable Hadoop compression between the map and reduce phases for faster performance.

**4. Numerical comparisons.** We consider the performance of BTER on various real-world data sets, including what is currently the largest publicly available graph

modeled in the sense of matching degree distribution. We provide additional plots in the supplementary online materials, specifically, log-binned data as well as cumulative degree distributions.

**4.1. Small data.** In Figure 5, we present comparisons of BTER with the state-of-the-art scalable generative models SKG [8, 28] and CL [1, 10, 11, 15] on two small data sets available from SNAP [48]. We use SKG because it is the current choice as the Graph 500 generator [20]. We omit details here and instead refer the reader to the references listed above for details of the method and to [46, 45] for an in-depth analysis and discussion of problems with SKG. The CL model has been described in section 2.3. We treat all edges as undirected and remove any duplicate edges and loops. The graph ca-AstroPh is a collaboration network based on 124 months of data from the astrophysics section of the arXiv preprint server; it has 9,987 nodes and 25,973 edges. The graph soc-Epinions1 is a who-trusts-whom online social (review) network from the Epinions website with 75,879 nodes and 405,740 edges.

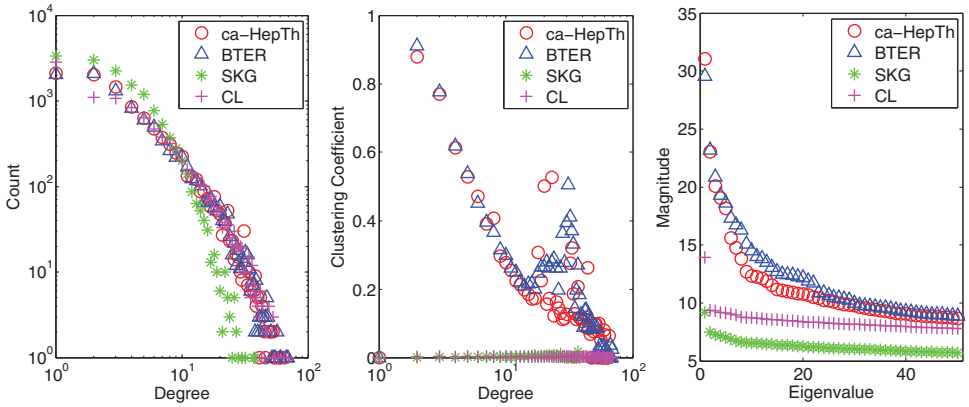
The parameters of SKG are from [28]. The input to CL is the degree distribution of the real graph and a blowup factor of $\beta = 10$ [15]. The inputs to BTER are the degree distribution and clustering coefficients per degree (computed exactly by counting all wedges and triangles) of the real graph and a blowup factor of $\beta = 10$. Timings are not reported as they are negligible for all three methods.

*Degree distribution.* The degree distributions for the original graphs and the models are shown in Figures 5a and 5d. SKG is known to have oscillations in the degree distribution [46, 45], and these oscillations are easily visible in Figure 5d. The oscillations are correctable with an appropriate addition of noise [46, 45] (not shown), but even then SKG tends to overestimate the low-degree nodes and miss the highest degree nodes. In contrast, both CL and BTER closely match the real data.

*Clustering coefficients.* The clustering coefficients per degree for the original graphs and the models are shown in Figures 5a and 5e. The SKG graph model has no inherent mechanism for closing triangles and creating a community structure. Although a few triangles may close at random, they are insufficient for the SKG-generated graph to match the clustering coefficients in the real data. The situation for CL is similar to that for SKG—there is no reason for wedges to close. BTER, on the other hand, provides a much closer match to the real data.

*Eigenvalues of adjacency matrix.* We show the top 50 leading eigenvalues (in magnitude) of the adjacency matrix in Figures 5c and 5f. BTER provides a much closer match to the real data—especially the first few eigenvalues. Under certain circumstances, matching the degree distribution should produce a match in eigenvalues [31]. However, based on the observation that the eigenvalues of a graph with community structure are larger than those of the CL graph with the same degree distribution, we conjecture that graphs with community structure require that the triangle structure also be matched to obtain a good fit for the eigenvalues.

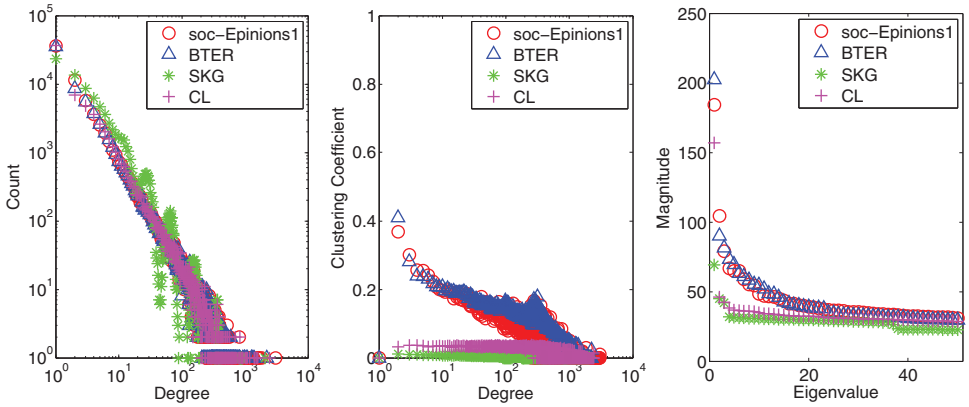
**4.2. Large data.** We demonstrate that BTER is able to fit large-scale real-world data. We do not compare BTER to SKG because it is not possible to fit the parameters for such large graphs. We do not compare BTER to the CL model because we can easily explain the performance: its match in terms of the degree distribution is nearly identical to that of BTER, and its clustering coefficients are close to zero for the small data. The data sets are described in Table 2(a). We treat all edges as undirected and remove any duplicate edges and loops. We obtained real-world graphs from the Laboratory for Web Algorithms [30], which compressed the graphs using LLP and WebGraph [7, 5]. Briefly, the networks are described as follows:




(a) Degree distribution for ca-HepTh.

(b) Clustering coefficients for ca-HepTh.

(c) Leading adjacency matrix eigenvalues for ca-HepTh.




(d) Degree distribution for soc-Epinions1.

(e) Clustering coefficients for soc-Epinions1.

(f) Leading adjacency matrix eigenvalues for soc-Epinions1.

FIG. 5. *Comparison of CL, SKG, and BTER on small graphs.*

- amazon-2008 [7, 5]: A graph describing similarity among books as reported by the Amazon store.
- ljournal-2008 [9, 7, 5]: Nodes represent users on LiveJournal. Node $x$ connects to node $y$ if $x$ registered $y$ as a friend.
- hollywood-2011 [7, 5]: This is a graph of actors. Two actors are joined by an edge whenever they appear in a movie together.
- twitter-2010 [27, 7, 5]: Nodes are Twitter users, and node $x$ links to node $y$ if $y$ follows $x$.
- uk-union-2006-06-2007-05 (shortened to uk-union) [6, 7, 5]: Links between Web pages on the .uk domain. We ignore the time labeling on the links.

To the best of our knowledge, uk-union is the largest publicly available graph.

The smaller graphs (amazon-2008, ljournal-2008, hollywood-2011) are those with up to roughly 100M edges. These can be easily processed using MATLAB on an SGI Altix UV 10 with 32 cores (4 Xeon 8-core 2.0GHz processors) and 512 GB DDR3 memory. None of the parallel capabilities of MATLAB are enabled for these studies.

TABLE 2
*Network characteristics of original and BTER-generated graphs.*

| Graph | $\|V\|$ | $\|E\|$ | $d_{\max}$ | $d_{\text{avg}}$ | **GCC** |
|---|---|---|---|---|---|
| amazon-2008 | 1M | 4M | 1,077 | 10 | 0.260 |
| ljournal-2008 | 5M | 50M | 19,432 | 18 | 0.124 |
| hollywood-2011 | 2M | 114M | 13,107 | 115 | 0.175 |
| twitter-2010 | 40M | 1,202M | 2,997,487 | 60 | 0.001 |
| uk-union | 122M | 4,659M | 6,366,528 | 76 | 0.007 |

(a) Large data set properties.

| Graph | $\|V\|$ | $\|E\|$ | $d_{\max}$ | $d_{\text{avg}}$ | **GCC** | **Gen.** | **Dedup.** |
|---|---|---|---|---|---|---|---|
| amazon-2008 | 1M | 4M | 1,052 | 10 | 0.253 | 2.27s | 9.25s |
| ljournal-2008 | 5M | 49M | 18,510 | 19 | 0.127 | 33.81s | 126.40s |
| hollywood-2011 | 2M | 114M | 11,676 | 115 | 0.180 | 88.54s | 362.25s |
| twitter-2010 | 38M | 1,135M | 1,635,823 | 59 | 0.004 | 230s | |
| uk-union | 120M | 4,405M | 1,497,950 | 73 | 0.111 | 1350s | |

(b) Properties of BTER-generated graphs, including generation and edge deduplication time.

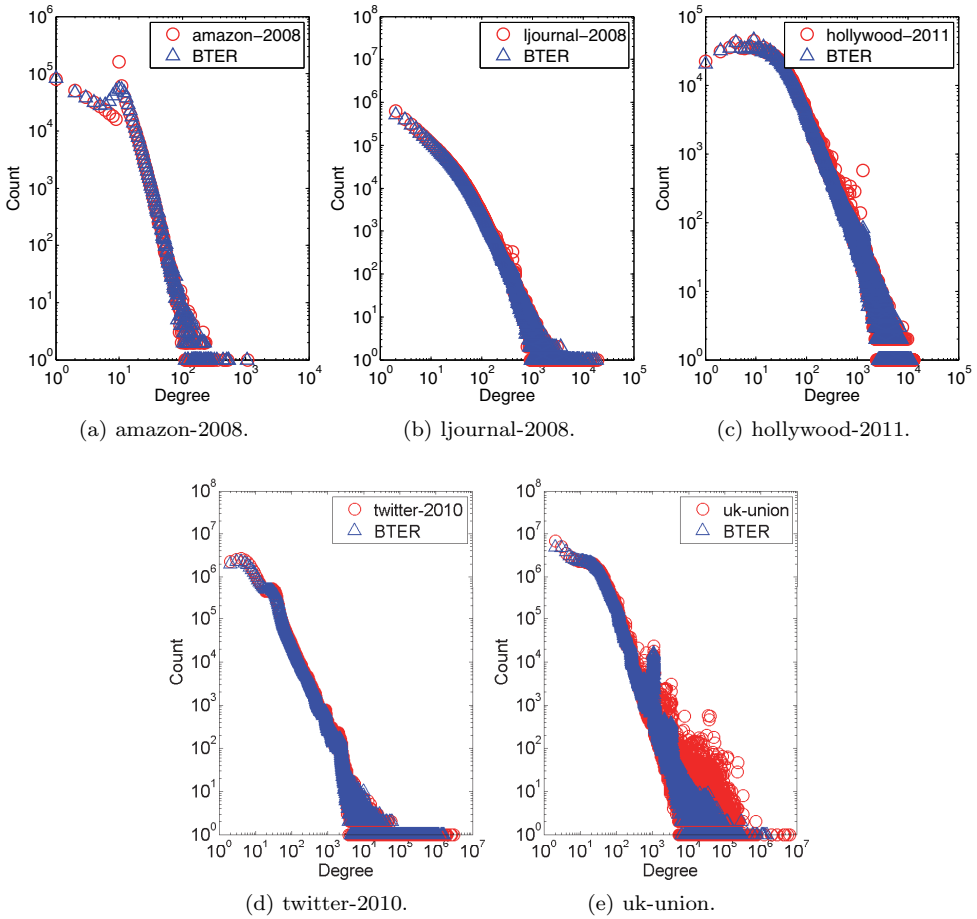
To give a sense of the memory requirements, storing the hollywood-2011 graph as a sparse matrix in MATLAB requires 3.4GB of storage. Each of the larger graphs (twitter-2010, uk-union) has over 1B edges. These are processed on a Hadoop cluster with 32 compute nodes. Each compute node has an Intel i7 930 CPU at 2.8GHz (four physical cores, HyperThreading enabled), 12 GB of memory, and 4 2TB SATA disks. All experiments were run using Apache Hadoop version 0.20.203.0.[3] The results in Table 2(b) were obtained using 132 map and 32 reduce tasks.

The inputs to BTER are the degree distribution and clustering coefficients by degree. (We used a blowup of $\beta = 1$ for the experiments reported here.) Computing the degree distribution is straightforward. However, for the clustering coefficients calculations, we used the sampling approach as implemented in [25] with 2000 samples per degree, so the expected error is $\epsilon = 0.05$ at a confidence level of 99.9%. Sampling was not required for the smaller graphs, but we used it in all experiments for consistency.

*BTER timing.* Table 2(b) shows the details and timings for the graphs produced by BTER. Observe the close match in the characteristics of the graphs in terms of number of nodes, number of edges, maximum degree, average degree, and global clustering coefficient. For the smaller graphs, we are able to separate the edge generation and deduplication time. The generation time is not strictly proportional to the number of desired edges because we have to generate extra edges for Phase 1 to account for possible duplicates (see section 3.5). The parallelism of Hadoop yields a large advantage in terms of time. The twitter-2010 graph has 10 times more edges than hollywood-2011, but it takes less than half the time to do the computation on the 32-node Hadoop cluster.

*Degree distribution.* Figure 6 illustrates the match between the real data and the BTER graph. BTER cannot easily match discontinuities in the degree distribution because of the randomness in creating edges. The issue is that nodes generally do not get *exactly* the desired degree; in the realization of the graph, observed degrees may deviate by one or two from the expected degree. For a smooth degree distribution, neighboring degrees cancel the effect of one another. For discontinuous distributions,

[3] http://hadoop.apache.org/




(a) amazon-2008. (b) ljournal-2008. (c) hollywood-2011.

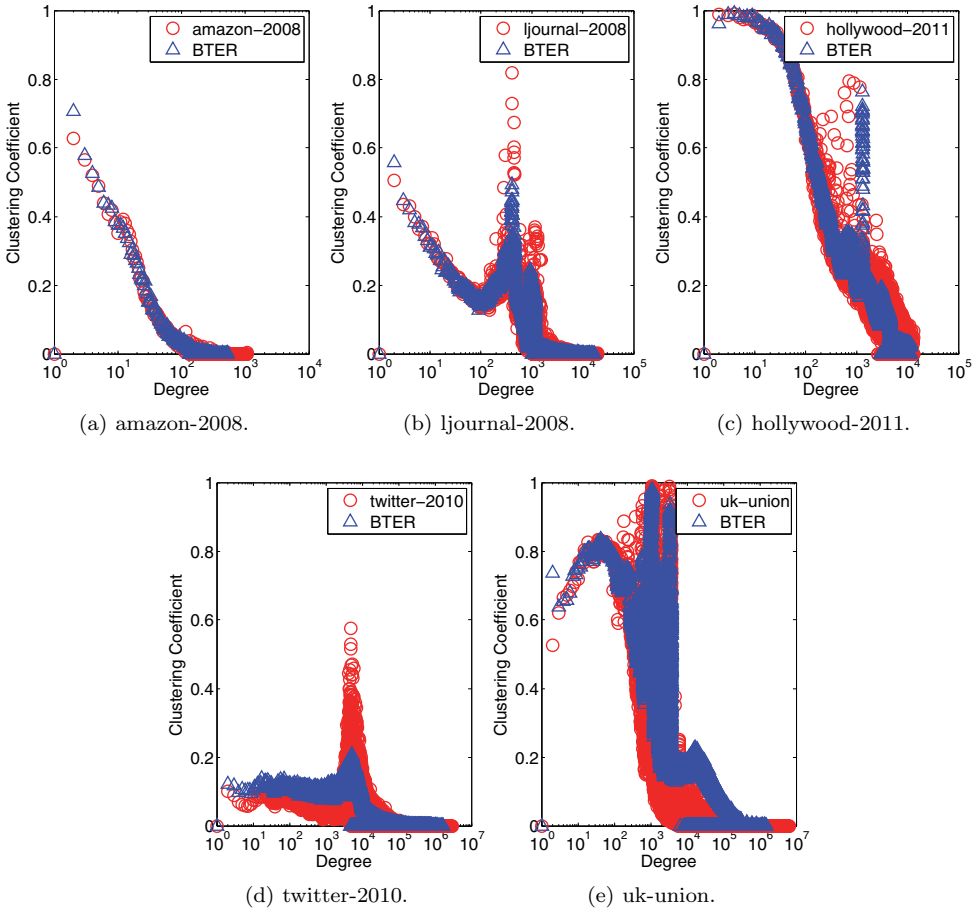
(d) twitter-2010. (e) uk-union.

FIG. 6. *Degree distributions of original and BTER-generated graphs.*

the BTER degree distribution is a "smoothed" version. This is evident, for instance, in the amazon-2008 data, where we can see a smoothing effect on the sharp discontinuity near degree 10.

*Clustering coefficients.* BTER's strength is its ability to match clustering coefficients and therefore create community structure. Most degree distributions are heavy-tailed and have a relatively consistent structure. The same is not true for clustering coefficients. Different profiles can potentially lead to graphs with fundamentally different structures. Figure 7 shows the clustering coefficients of the real data and the BTER-generated graphs. There is a very close match.

**5. Proposed benchmark parameters and scalability.** Thus far we have considered how BTER can be used to match real-world data. For benchmarking purposes, where there is no specific graph to be matched, "ideal" profiles for degree distribution and clustering coefficient by degree are required. In this section, we propose some possibilities, noting that these are tunable for various testing scenarios, i.e., specifying an average degree, a maximum clustering coefficient, etc. Generating an artificial degree distribution can be problematic. For instance, using a straight




(a) amazon-2008. (b) ljournal-2008. (c) hollywood-2011.

(d) twitter-2010. (e) uk-union.

FIG. 7. *Clustering coefficients of original and BTER-generated graphs.*

power law (PL) distribution can lead to impossible situations, such as choosing a degree greater than the number of nodes. This necessitates a discrete distribution, which engenders its own problems. After proposing methods for working around these problems, we use the proposed benchmark in a scalability study for the MapReduce implementation of BTER.

**5.1. Idealized degree distribution.** It has been hypothesized that degree distribution of real-world networks follows a PL degree distribution, i.e.,

$$n_d \propto d^{-\gamma},$$

for some parameter $\gamma$ [2]. However, our observation is that PL distributions are difficult to use as a model—a point that is discussed in more detail below. It has been suggested that PLs are not necessarily the best descriptors for real-world networks [43, 4]. Finally, proving (in a statistical sense) that a single observed degree distribution is PL is difficult [13].

For benchmarking purposes, our goal is to specify an ideal average degree, $\bar{d}$, and an absolute bound on maximum degree, $d^*$. Let $f(d)$ define the desired proportion-

ality of degree $d$, e.g., $f(d) = d^{-\gamma}$ for a PL distribution. We then create a *discrete* distribution on $d = 1, \ldots, d^*$ as

$$\Pr(D = d) = \frac{f(d)}{\sum_{d'=1}^{d^*} f(d')}.$$

Ideally, the average degree is equal to $\bar{d}$, and the probability of having degree $d^*$ is sufficiently small, i.e.,

$$\bar{d} = \sum_{d=1}^{d^*} d \cdot f(d) \quad \text{and} \quad \Pr(D = d^*) < \epsilon_{\text{tol}},$$

where $\epsilon_{\text{tol}}$ is small enough such that $n \cdot \epsilon_{\text{tol}} \ll 1$ (where $n$ is the number of nodes). For the power law distribution, it can be difficult to find a value for $\gamma$ that yields a high enough average degree and a low enough probability of choosing $d^*$. Hence, we propose instead a generalized log-normal (GLN) distribution, i.e.,

$$n_d \propto \exp\left[-\left(\frac{\log d}{\alpha}\right)^{\delta}\right],$$

for some parameters $\alpha$ and $\delta$. The supplementary material includes example degree distributions that vary $\alpha$ and $\delta$. The shape of the distribution is typical of the real-world graphs shown in section 4.

We consider two scenarios, both with $n = 10^7$ nodes. We do a parameter search on $\alpha$ and $\delta$ (`fminsearch` in MATLAB) to locate the optimal parameters. A function `degdist_param_search` that finds the optimal parameters for either discrete GLN (DGLN) or discrete PL (DPL) for user-specified values of $d_{\text{avg}}$ and $d_{\max}$ is included in the reference code to be released at a future date.

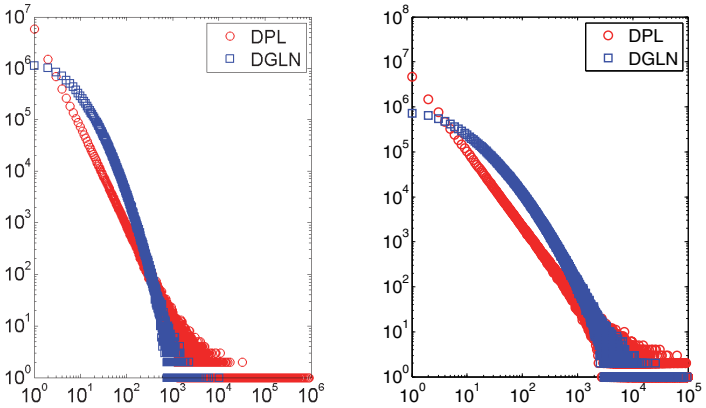
*Scenario* 1 *for degree distribution fitting.* In the first scenario, the targets are $\bar{d} = 16$ and $d^* = 10^6$. For DPL, the optimal parameter is $\gamma = 1.911$ with $d_{\text{avg}} = 16$ and $\Pr(D = d^*) = 1.97 \times 10^{-12}$. For DGLN, the optimal parameters are $\alpha = 1.988$ and $\delta = 2.079$ with $d_{\text{avg}} = 16$ and $\Pr(D = d^*) = 4.14 \times 10^{-26}$. Realizations of the two distributions are pictured in Figure 8a. For this scenario, both degree distributions are reasonable in that there is no sharp dropoff as we get close to the maximum allowable degree, $d^*$.

*Scenario* 2 *for degree distribution fitting.* In the second scenario, the targets are $\bar{d} = 64$ and $d^* = 10^5$. For PL, the optimal parameter is $\gamma = 1.668$ with $d_{\text{avg}} = 64$ but $\Pr(D = d^*) = 2.16 \times 10^{-9}$ (fairly large). For DGLN, the optimal parameters are $\alpha = 2.171$ and $\delta = 1.877$ with $d_{\text{avg}} = 64$ and $\Pr(D = d^*) = 8.35 \times 10^{-12}$. Realizations of the two distributions are pictured in Figure 8b. In this scenario, the problem with PL becomes apparent—near $d^*$, there are still many degrees with *multiple* nodes so that the cutoff is extremely abrupt. In comparison, DGLN fades more naturally to the desired maximum degree.

**5.2. Idealized clustering coefficients.** As there is no definitive structure to clustering coefficients, we propose a simple parameterized curve that has some similarity to real data observations.

Let $\{ n_d \}$ define the specified degree distribution, and let $d_{\max}$ be the maximum degree such that $n_d > 0$. We define $\bar{c}_d$, the mean value for $c_d$, as

$$\bar{c}_d = c_{\max} \exp(-(d-1) \cdot \xi) \quad \text{for} \quad d \geq 2,$$




(a) Scenario 1: $\bar{d} = 16$ and $d^* = 10^6$.

(b) Scenario 2: $\bar{d} = 64$ and $d^* = 10^5$.

FIG. 8. *Example degree distributions from DPL and DGLN for $n = 10^7$ nodes.*

where $c_{\max}$ and $\xi$ are parameters. If $c_{\max}$ is specified, then a simple parameter search can be used to fit $\xi$ to a target global clustering coefficient; code to fit the data is included in the reference code. The final values for $\{ c_d \}$ are selected as

$$c_d \sim \mathcal{N}(\bar{c}_d, \min\{10^{-2}, \bar{c}_d/2\}).$$

The randomness could, of course, be omitted.

**5.3. Example graphs.** We generate two example graphs per the scenarios below. Table 3 lists the network characteristics and Figure 9 shows the target and BTER-generated degree distributions and clustering coefficients.

TABLE 3
*Network characteristics of BTER-generated graphs for benchmarking.*

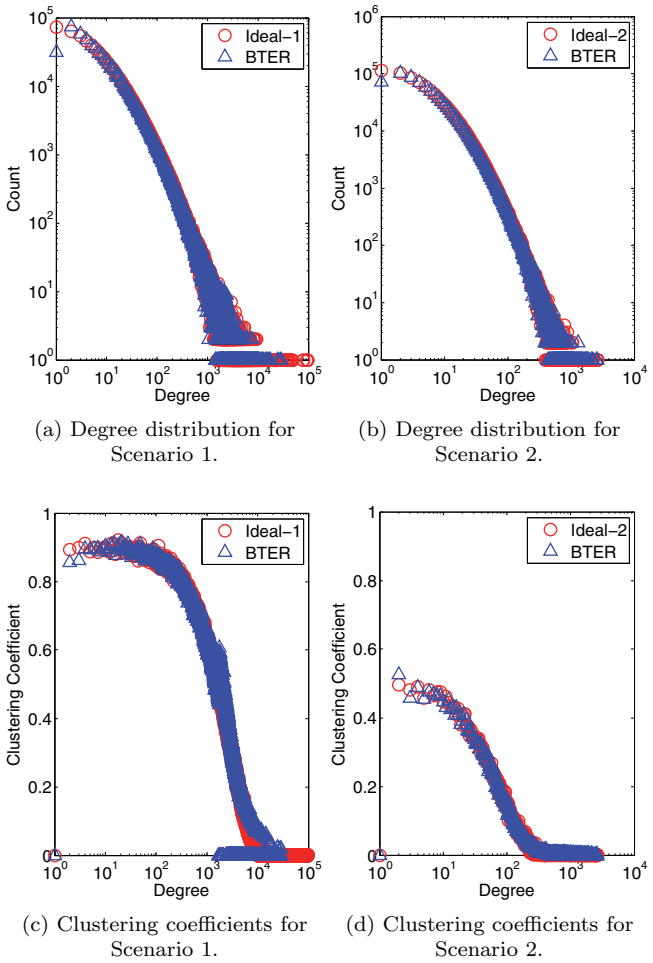
| Graph | $\lvert V \rvert$ | $\lvert E \rvert$ | $d_{\max}$ | $d_{\text{avg}}$ | **GCC** | **Gen.** | **Dedup.** |
|---|---|---|---|---|---|---|---|
| Scenario 1 | 1M | 35M | 28,643 | 72 | 0.406 | 35.11s | 117.18s |
| Scenario 2 | 1M | 8M | 2,594 | 17 | 0.104 | 5.07s | 20.66s |

*Scenario* 1. For the first setup, we selected $\bar{d} = 75$ and $d^* = 100{,}000$ to define the degree distribution. The parameter search selected $\alpha = 2.14$ and $\delta = 1.83$. For the clustering coefficients, we set $c_{\max} = 0.9$ and a target GCC of 0.15. The parameter search selected $\xi = 3.59 \times 10^{-4}$ for defining the clustering coefficient profile.

*Scenario* 2. For the second setup, we selected $\bar{d} = 16$ and $d^* = 10{,}000$ to define the degree distribution. The parameter search selected $\alpha = 1.98$ and $\delta = 2.08$. For the clustering coefficients, we set $c_{\max} = 0.5$ and a target GCC of 0.10. The parameter search selected $\xi = 0.01$ for defining the clustering coefficient profile.

**5.4. Scalability test.** We use the DGLN distribution to create target distributions that define a series of graphs of different sizes but similar community structure. From these, we generate example graphs with our Hadoop MapReduce implementation of BTER and demonstrate scalability of our code.

Table 4 describes characteristics of the test graphs. We selected $\bar{d} = 32$ for all graphs and maintained $d^*$ proportional to $\sqrt{|V|}$, a relation that we often observe in




(a) Degree distribution for Scenario 1.

(b) Degree distribution for Scenario 2.

(c) Clustering coefficients for Scenario 1.

(d) Clustering coefficients for Scenario 2.

FIG. 9. *Target distributions and results of BTER-generated graphs.*

social network graphs. The last two columns of the table list the number of generated (Phase 1 and 2) and unique edges measured from full graphs realized by BTER.

Target degree distributions were created using the formulas and functions presented in section 5.1. Optimal parameters $\alpha$ and $\delta$ were computed using MATLAB function `degdist_param_search` (part of the reference code). The function takes $\bar{d}$ and $d^*$ as inputs, plus the $\Pr(D = d^*)$ which we set to $0.001/|V|$. Then we computed a DGLN from $\alpha$, $\delta$, and the number of nodes $|V|$ listed in the second column of Table 4. We created clustering coefficient distributions as in section 5.2, setting $c_{\max} = 0.5$ and GCC $= 0.15$ for all graphs.

The MapReduce implementation of BTER was executed on a 32-node cluster running Apache Hadoop version 0.20.203.0. Each compute node contains a quad-core processor and four hard drives configured to run independently (no RAID striping). Hadoop was configured for a maximum of 128 simultaneous map tasks, effectively pairing each core with a hard drive for maximum I/O throughput. A MapReduce job can therefore launch up to 128 mappers in parallel. Hadoop was also configured for a

TABLE 4
*Graph characteristics for testing BTER scalability.*

| Graph | $\|V\|$ | $d^*$ | $d$ | **Gen** $\|E\|$ | **Unique** $\|E\|$ |
|---|---|---|---|---|---|
| 1 | 1M | 50k | 32 | 25M | 16M |
| 2 | 2M | 70k | 32 | 50M | 32M |
| 3 | 4M | 100k | 32 | 101M | 64M |
| 4 | 8M | 140k | 32 | 201M | 128M |
| 5 | 16M | 200k | 32 | 403M | 256M |
| 6 | 32M | 280k | 32 | 808M | 512M |
| 7 | 64M | 400k | 32 | 1,614M | 1,024M |
| 8 | 128M | 570k | 32 | 3,233M | 2,047M |
| 9 | 256M | 800k | 32 | 6,467M | 4,096M |

maximum of 128 simultaneous reduce tasks. BTER uses the reduce phase to remove duplicate edges, so we expect this to compete for machine resources with BTER map tasks that generate Phase 1 and 2 edges.

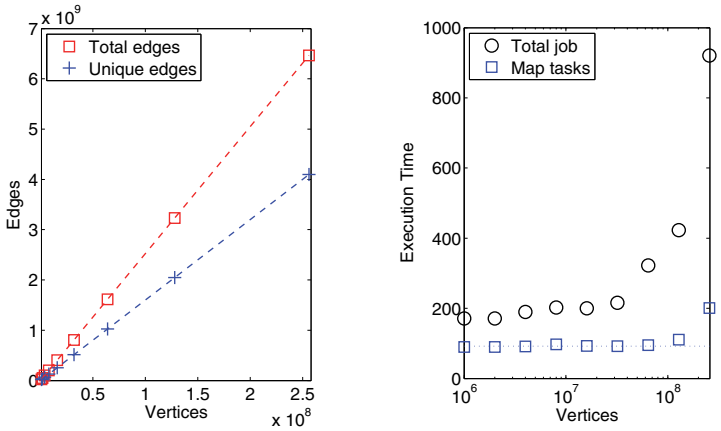
Figure 10a renders the data in Table 4, showing how the computational work varies across test graphs. To assess scalability in the weak sense, we set the workload of each map task in BTER to be 1 million edges; hence, Graph 1 executes with 1 map task and Graph 9 executes with 256. The number of reduce tasks was set to the number of map tasks, up to the configured Hadoop limit of 128.

Figure 10b plots BTER execution time (wall clock time) as a function of the number of vertices requested. Perfect scalability would appear as a horizontal line. We observe excellent scalability up to 32 million vertices (32 map tasks), and then a significant dropoff in parallel performance. However, map task scalability is excellent through 128 tasks, the full capacity of the cluster. Hence, BTER edge generation in the map phase is fully scalable to the available hardware. Removal of duplicate edges in the reduce phase is less scalable. Here the edges must be sorted on the map task node, shuffled across the network to reduce task nodes, and merged at the reducers. Close examination of Hadoop log files and network traffic suggests that the reduce phase suffers from bandwidth limitations during the shuffle and from spillover to disk during the merge. We conjecture that deduplication scalability could be improved on a cluster with larger network bandwidth and more physical memory per compute node.

We also show an example of the generated graph in Figure 11. This is for Graph 9 with 256M vertices and 4B edges. We give log-binned results. The BTER model is fairly close to the intended distributions for both the degree distribution and the clustering coefficient by degree.

**6. Conclusions and future work.** This paper demonstrates that the BTER generative model is useful for modeling massive networks, especially compared with other scalable models. We provide a detailed algorithm along with analysis explaining the workings of the method. The original paper on BTER [44] provided none of the implementation details and, in fact, did not directly use the clustering coefficient data but rather estimated it via a function. Here we give precise details on the implementation, which is nontrivial due to issues such as repeat edges. We are able to build a model of a graph with 120M nodes and 4.4B edges in less than 25 minutes on a 32-node Hadoop cluster.

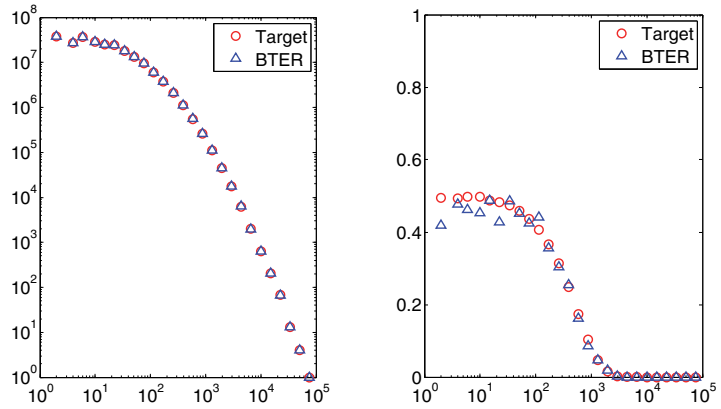
The development of a realistic graph model is an important step in developing effective "null" models that nonetheless share the properties of real-world networks. Such models will be useful in detecting anomalies, statistical sampling, and community




(a) Test graph sizes.

(b) BTER execution time in seconds.

FIG. 10. *Scalability of BTER MapReduce implementation.*




(a) Degree distribution (log-binned).

(b) Clustering coefficients (log-binned).

FIG. 11. *Example target and actual degree distributions for* 256*M nodes using proposed idealized distributions.*

detection. For example, the BTER model does not have larger communities beyond the affinity blocks, whereas we might expect that real-world graphs have a richer structure such as a hierarchy or other complex behavior.

The proposed BTER model, along with the proposed degree and clustering coefficient distributions, may also boost benchmarking efforts in graph processing. The proposed degree distributions capture the essence of degree distributions that we see in practice and generate realistic distributions even at large scales (whereas PL has a reputation of generating a few degrees that are much larger than observed in practice). Moreover, the proposed distribution allows us to modify both the average and the maximum degree, which is critical for benchmarking. The proposed clustering coefficient curves implicitly embed triangle structure into the graphs, which is a critical feature that distinguishes real graphs from arbitrary sparse graphs. Finally, the proposed generation algorithm scales to extremely large graphs because it generates edges in parallel.

Of course, we should list the limitations of the BTER model. First and foremost, we consider only simple graphs. More complex models would be needed for directed and/or weighted graphs. For instance, even capturing the degree distributions of a directed graph is a challenge [15]. We also ignore complex community structure, such as bipartite or near-bipartite community structure, as well as hierarchical structure, which has been observed in real-world applications [40, 12]. Such structure could potentially be incorporated by adjusting the way that affinity blocks are linked. Currently, affinity blocks link only within themselves and then randomly to other nodes in the graphs. To simulate bipartite structure, the affinity blocks could be paired. To generate hierarchical structure, the affinity blocks could be arranged in that way, with excess degree being biased towards blocks that are closer in the hierarchy. We do not incorporate node and edge types which would be relevant, for instance, in an e-commerce graph that represents users and items, connected via purchases and ratings of items by users. Finally, although we can generate edges in a streaming fashion, the BTER model has no real concept of evolving parameters in time. All of these issues are topics for future studies.

**Appendix A. Coupon collector derivation.** Consider a universe $U$ of objects/coupons, and suppose we pick objects uniformly at random with replacement from $U$. The following theorem proves the bound used in (3.1), when $U$ is the set of possible pairs in an affinity block (so $|U| = \binom{n_b}{2}$). This is a simple take on the standard coupon collector problem, where we wish to pick up all distinct coupons. (We follow the analysis of section 3.6.1 in [36].)

THEOREM A.1. *For a given $\rho \in (0,1)$, the expected number of independent draws required to select $\rho|U|$ distinct coupons from $U$ is $|U|\ln(1/(1-\rho)) + O(1)$.*

*Proof.* For convenience, we assume that $\rho|U|$ is an integer. Consider a sequence of draws. Let $X_i$ (for integer $0 \le i < \rho|U|$) be the random variable denoting the number of draws required to get one more (distinct) coupon after $i$ distinct coupons have been collected. Observe that the quantity of interest is $\mathbb{E}[\sum_{i<\rho|U|} X_i]$, which by linearity of expectation is $\sum_{i<\rho|U|} \mathbb{E}[X_i]$. (The usual coupon collector analyses consider this sum for $\rho = 1$.)

When $i$ distinct coupons have already been collected, the probability that a single draw gives a new coupon is exactly $1 - i/|U|$. Think of this as probability of "failure." The number of draws required for a success (new coupon) follows a geometric distribution (Chap VI.8 of [17]) and the mean of this is $1/(1-i/|U|) = |U|/(|U|-i)$. Using this bound, the expected total number of draws can be expressed as follows:

$$\begin{aligned}\sum_{i<\rho|U|} \mathbb{E}[X_i] &= \sum_{i<\rho|U|} \frac{|U|}{|U|-i} \\ &= |U|\left[\sum_{i\le|U|} \frac{1}{i} - \sum_{i\le(1-\rho)|U|} \frac{1}{i}\right] \\ &= |U|\Big[\ln|U| - \ln((1-\rho)|U|) + O(1/|U|)\Big] = |U|\ln(1/(1-\rho)) + O(1).\end{aligned}$$

(We use the standard bound for the Harmonic sum, $\sum_{i\le r} 1/i = \ln r + \gamma + O(1/r)$, where $\gamma$ is the Euler–Mascheroni constant.) ∎

## REFERENCES


[1] W. Aiello, F. Chung, and L. Lu, *A random graph model for power law graphs*, Exp. Math., 10 (2001), pp. 53–66, http://projecteuclid.org/euclid.em/999188420.

[2] A.-L. Barabási and R. Albert, *Emergence of scaling in random networks*, Science, 286 (1999), pp. 509–512, http://dx.doi.org/10.1126/science.286.5439.509.

[3] A. Barrat and M. Weigt, *On the properties of small-world network models*, Eur. Phys. J. B Condens. Matter, 13 (2000), pp. 547–560, http://dx.doi.org/10.1007/s100510050067.

[4] Z. Bi, C. Faloutsos, and F. Korn, *The "DGX" distribution for mining massive, skewed data*, in KDD '01: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, 2001, pp. 17–26, http://dx.doi.org/10.1145/502512.502521.

[5] P. Boldi, M. Rosa, M. Santini, and S. Vigna, *Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks*, in WWW'11: Proceedings of the 20th International World Wide Web Conference, ACM, New York, 2011, pp. 587–596, http://dx.doi.org/10.1145/1963405.1963488.

[6] P. Boldi, M. Santini, and S. Vigna, *A large time-aware graph*, ACM SIGIR Forum, 42 (2008), pp. 33–38, http://dx.doi.org/10.1145/1480506.1480511.

[7] P. Boldi and S. Vigna, *The webgraph framework* I*: Compression techniques*, in WWW'04: Proceedings of the 13th International World Wide Web Conference, ACM, New York, 2004, pp. 595–602, http://dx.doi.org/10.1145/988672.988752.

[8] D. Chakrabarti, Y. Zhan, and C. Faloutsos, *R-MAT: A recursive model for graph mining*, in SDM04: Proceedings of the 2004 SIAM International Conference on Data Mining, SIAM, Philadelphia, 2004, pp. 442–446, http://dx.doi.org/10.1137/1.9781611972740.43.

[9] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan, *On compressing social networks*, in KDD '09: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, 2009, pp. 219–228, http://dx.doi.org/10.1145/1557019.1557049.

[10] F. Chung and L. Lu, *The average distances in random graphs with given expected degrees*, Proc. Natl. Acad. Sci. USA, 99 (2002), pp. 15879–15882, http://dx.doi.org/10.1073/pnas.252631999.

[11] F. Chung and L. Lu, *Connected components in random graphs with given degree sequences*, Ann. Combin., 6 (2002), pp. 125–145, http://www.combinatorics.net/new/Annals/Abstract/6_2_125.aspx.

[12] A. Clauset, C. Moore, and M. Newman, *Hierarchical structure and the prediction of missing links in networks*, Nature, 453 (2008), pp. 98–101, http://dx.doi.org/10.1038/nature06830.

[13] A. Clauset, C. R. Shalizi, and M. E. J. Newman, *Power-law distributions in empirical data*, SIAM Rev., 51 (2009), pp. 661–703, http://dx.doi.org/10.1137/070710111.

[14] D. Dominguez-Sal, P. Urbón-Bayes, A. Giménez-Vanó, S. Gómez-Villamor, N. Martnez-Bazán, and J. Larriba-Pey, *Survey of graph database performance on the HPC scalable graph analysis benchmark*, in Web-Age Information Management, H. Shen, J. Pei, M. Özsu, L. Zou, J. Lu, T.-W. Ling, G. Yu, Y. Zhuang, and J. Shao, eds., Lecture Notes in Comput. Sci. 6185, Springer, Berlin, Heidelberg, 2010, pp. 37–48, http://dx.doi.org/10.1007/978-3-642-16720-1_4.

[15] N. Durak, T. G. Kolda, A. Pinar, and C. Seshadhri, *A scalable null model for directed graphs matching all degree distributions: In, out, and reciprocal*, in Proceedings of the 2nd IEEE Workshop on Network Science, IEEE Press, Piscataway, NJ, 2013, pp. 23–30, http://arxiv.org/abs/1210.5288.

[16] P. Erdös and A. Rényi, *On the evolution of random graphs*, Magyar Tud. Akad. Mat. Kutató Int. Közl, 5 (1960), pp. 17–61, http://www.math-inst.hu/~p_erdos/1960-10.pdf.

[17] W. Feller, *An Introduction to Probability Theory and Applications: Vol.* I, 3rd ed., John Wiley and Sons, New York, 1968.

[18] M. Girvan and M. E. J. Newman, *Community structure in social and biological networks*, Proc. Natl. Acad. Sci. USA, 99 (2002), pp. 7821–7826, http://dx.doi.org/10.1073/pnas.122653799.

[19] D. F. Gleich and A. B. Owen, *Moment-based estimation of stochastic Kronecker graph parameters*, Internet Math., 8 (2012), pp. 232–256, http://dx.doi.org/10.1080/15427951.2012.680824.

[20] *Graph* 500 *Benchmark*, http://www.graph500.org/specifications(accessed April 2, 2014).

[21] W. Guo and S. Kraines, *A random network generator with finely tunable clustering coefficient for small-world social networks*, in CASON '09: International Conference on Computational Aspects of Social Networks, IEEE Press, Piscataway, NJ, 2009, pp. 10–17, http://dx.doi.org/10.1109/CASoN.2009.13.

[22] R. Gupta, T. Roughgarden, and C. Seshadhri, *Decompositions of triangle-dense graphs*, in ITCS'14: Proceedings of the 5th Conference on Innovations in Theoretical Computer Science, ACM, New York, 2014, pp. 471–482, http://dx.doi.org/10.1145/2554797.2554840.

[23] A. Gutfraind, L. A. Meyers, and I. Safro, *Multiscale Network Generation*, preprint, http://arxiv.org/abs/1207.4266, 2012.

[24] J. Kepner, *The Kronecker theory of power law graphs*, in Graph Algorithms in the Language of Linear Algebra, J. Kepner and J. Gilbert, eds., SIAM, Philadelphia, 2011, pp. 205–240.

[25] T. G. Kolda, A. Pinar, T. Plantenga, C. Seshadhri, and C. Task, *Counting triangles in massive graphs with MapReduce*, SIAM J. Sci. Comput., to appear.

[26] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal, *Stochastic models for the web graph*, in Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, IEEE Press, Piscataway, NJ, 2000, pp. 57–65, http://dx.doi.org/10.1109/SFCS.2000.892065.

[27] H. Kwak, C. Lee, H. Park, and S. Moon, *What is Twitter, a social network or a news media?*, in WWW '10: Proceedings of the 19th International World Wide Web Conference, ACM, New York, 2010, pp. 591–600, http://dx.doi.org/10.1145/1772690.1772751.

[28] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, *Kronecker graphs: An approach to modeling networks*, J. Mach. Learn. Res., 11 (2010), pp. 985–1042, http://jmlr.csail.mit.edu/papers/v11/leskovec10a.html.

[29] J. Leskovec, J. Kleinberg, and C. Faloutsos, *Graph evolution: Densification and shrinking diameters*, ACM Trans. Knowledge Discovery Data, 1 (2007), 2, http://dx.doi.org/10.1145/1217299.1217301.

[30] *LWA: Laboratory for Web Algorithms*, http://law.di.unimi.it/datasets.php (accessed April 2, 2014).

[31] M. Mihail and C. Papadimitriou, *On the eigenvalue power law*, in RANDOM 2002: Proceedings of Randomization and Approximation Techniques in Computer Science, Lecture Notes in Comput. Sci. 2483, Springer, Berlin, Heidelberg, 2002, pp. 254–262, http://dx.doi.org/10.1007/3-540-45726-7_20.

[32] B. Miller, N. Bliss, and P. Wolfe, *Subgraph detection using eigenvector $L_1$ norms*, in NIPS 2010: Advances in Neural Information Processing Systems, 2010, pp. 1633–1641, http://books.nips.cc/papers/files/nips23/NIPS2010_0954.pdf.

[33] B. Miller, L. Stephens, and N. Bliss, *Goodness-of-fit statistics for anomaly detection in Chung-Lu random graphs*, in ICASSP 2012: IEEE International Conference on Acoustics, Speech and Signal Processing, IEEE Press, Piscataway, NJ, 2012, pp. 3265–3268, http://dx.doi.org/10.1109/ICASSP.2012.6288612.

[34] D. Mir and R. N. Wright, *A differentially private estimator for the stochastic Kronecker graph model*, in EDBT-ICDT '12: Proceedings of the 2012 Joint EDBT/ICDT Workshops, ACM, New York, 2012, pp. 167–176, http://dx.doi.org/10.1145/2320765.2320818.

[35] S. Moreno, S. Kirshner, J. Neville, and S. V. N. Vishwanathan, *Tied Kronecker product graph models to capture variance in network populations*, in Proceedings of the 48th Annual Allerton Conference on Communication, Control, and Computing, 2010, pp. 1137–1144, http://dx.doi.org/10.1109/ALLERTON.2010.5707038.

[36] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995.

[37] M. Newman, *Component sizes in networks with arbitrary degree distributions*, Phys. Rev. E, 76 (2007), 045101, http://dx.doi.org/10.1103/PhysRevE.76.045101.

[38] M. Newman, D. Watts, and S. Strogatz, *Random graph models of social networks*, Proc. Natl. Acad. Sci. USA, 99 (2002), pp. 2566–2572, http://www.pnas.org/content/99/suppl_1/2566.full.

[39] M. E. J. Newman, *Properties of highly clustered networks*, Phys. Rev. E, 68 (2003), 026121, http://dx.doi.org/10.1103/PhysRevE.68.026121.

[40] M. E. J. Newman, *Finding community structure in networks using the eigenvectors of matrices*, Phys. Rev. E, 74 (2006), 036104, http://dx.doi.org/10.1103/PhysRevE.74.036104.

[41] A. Pinar, C. Seshadhri, and T. G. Kolda, *The similarity between stochastic Kronecker and Chung-Lu graph models*, in Proceedings of the 2012 SIAM International Conference on Data Mining, SIAM, Philadelphia, 2012, pp. 1071–1082, http://dx.doi.org/10.1137/1.9781611972825.92.

[42] A. Sala, L. Cao, C. Wilson, R. Zablit, H. Zheng, and B. Y. Zhao, *Measurement-calibrated graph models for social network experiments*, in WWW '10: Proceedings of the 19th International World Wide Web Conference, 2010, pp. 861–870, http://dx.doi.org/10.1145/1772690.1772778.

[43] A. Sala, S. Gaito, G. P. Rossi, H. Zheng, and B. Y. Zhao, *Revisiting Degree Distribution Models for Social Graph Analysis*, preprint, arXiv:1108.0027, 2011.

[44] C. Seshadhri, T. G. Kolda, and A. Pinar, *Community structure and scale-free collections of Erdös-Rényi graphs*, Phys. Rev. E, 85 (2012), 056109, http://dx.doi.org/10.1103/PhysRevE.85.056109.

[45] C. Seshadhri, A. Pinar, and T. G. Kolda, *An in-depth study of stochastic Kronecker graphs*, in ICDM 2011: Proceedings of the 2011 IEEE International Conference on Data Mining, IEEE Press, Piscataway, NJ, 2011, pp. 587–596, http://dx.doi.org/10.1109/ICDM.2011.23.

[46] C. Seshadhri, A. Pinar, and T. G. Kolda, *An in-depth analysis of stochastic Kronecker graphs*, J. ACM, 60 (2013), 13, http://dx.doi.org/10.1145/2450142.2450149.

[47] C. Seshadhri, A. Pinar, and T. G. Kolda, *Triadic measures on graphs: The power of wedge sampling*, in SDM13: Proceedings of the 2013 SIAM International Conference on Data Mining, SIAM, Philadelphia, 2013, pp. 10–18, http://dx.doi.org/10.1137/1.9781611972832.2.

[48] *SNAP: Stanford Network Analysis Project*, http://snap.stanford.edu/ (accessed April 2, 2014).

[49] J. C. Vivar and D. Banks, *Models for networks: A cross-disciplinary science*, Wiley Interdisciplinary Reviews: Computational Statistics, 4 (2012), pp. 13–27, http://dx.doi.org/10.1002/wics.184.

[50] D. Watts and S. Strogatz, *Collective dynamics of 'small-world' networks*, Nature, 393 (1998), pp. 440–442, http://dx.doi.org/10.1038/30918.