# Smooth Kronecker: Solving the Combing Problem in Kronecker Graphs

Vaastav Anand*
University of British Columbia
vaastav@cs.ubc.ca

Puneet Mehrotra*
University of British Columbia
puneet89@cs.ubc.ca

Daniel Margo*†
Google
dmargo@eecs.harvard.edu

Margo Seltzer
University of British Columbia
mseltzer@cs.ubc.ca

## ABSTRACT

Graphs and graph-processing have become increasingly important. This has led to an explosion in the development of graph-processing systems, each of which is evaluated relative to its predecessors. In the absence of a large corpus of real-world graphs, synthetic generators provide an easy way to construct graphs of varying sizes. The most widely used generator is the Kronecker generator. Unfortunately, the Kronecker generator was not designed to produce graphs for benchmarking and when used in this fashion, it is problematic in two ways. First, the *fraction* of zero-degree vertices scales with the graph size, dramatically reducing the effective size of the connected graph. Second, the generator produces a vertex degree distribution not found in real world settings. We demonstrate these phenomena and present the Smooth Kronecker Generator, which remedies the vertex degree distribution problem *without* changing the statistical properties of the graph.

## CCS CONCEPTS

• **General and reference** → **Evaluation**; **Perform-ance**; • **Mathematics of computing** → **Graph algorithms**; *Random graphs*; • **Theory of computation** → **Graph algorithms analysis**.

*Equal contribution; order decided by official coin flip
†Work done as part of Ph.D. at Harvard University.


*GRADES-NDA'20, June 14, 2020, Portland, OR, USA*

ACM ISBN 978-1-4503-8021-8/20/06…$15.00
https://doi.org/10.1145/3398682.3399161

## KEYWORDS

Kronecker graphs, benchmarking

**ACM Reference Format:**
Vaastav Anand, Puneet Mehrotra, Daniel Margo, and Margo Seltzer. 2020. Smooth Kronecker: Solving the Combing Problem in Kronecker Graphs. In *3rd Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA) (GRADES-NDA'20), June 14, 2020, Portland, OR, USA*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3398682.3399161

## 1 INTRODUCTION

Graph processing has become sufficiently critical that we now have a large number of graph processing systems, many of which claim to provide a performance improvement relative to other systems ( [8, 11, 17, 25, 26]).

There is a dearth of real world large scale graphs, so when evaluating systems scalability characteristics, researchers turn to synthetically generated graphs, most frequently, those produced by the Graph500 [20] and LUBM [7] generators. The Graph500 generator is an implementation of the Kronecker [13] network model, which is a refinement of R-MAT graphs. The Graph500 generator produces an unlabeled topology meant to *statistically* resemble a real-world network. There are other open source R-MAT generators [9, 16] that have also been used in practice for generating large graphs, and some researchers implement their own R-MAT generators [25]. The LUBM generator produces a labeled topology such that its *semantic* constraints, e.g., students per class, resemble a real-world RDF dataset. LUBM is tightly bound to a particular benchmark query set, because the queries have to make sense with respect to the semantic constraints, while the Graph500 generator is used with general-purpose benchmarks that do not impose such constraints, such as PageRank, connected component, and breadth first search.

A study of five years of graph systems papers showed that R-MAT and Kronecker generators are the third most

widely used data source for graph system performance evaluations [19]. Unfortunately, the more general purpose of these graphs, the Kronecker graphs, present two challenges that must be addressed to produce meaningful results: 1) an unrealistic number of zero-degree vertices and 2) a combed degree distribution. We discuss these challenges in more detail in Section 3.

Section 2 provides an overview of R-MAT and Kronecker graph generation. Section 3 illustrates the challenges arising when benchmarking with these graphs and why these graphs are not entirely representative of real graphs. Section 4 introduces Smooth Kronecker, which addresses the most vexing issue of the generators. Our goals are twofolds: First, we want the community to adopt the Smooth Kronecker model to avoid the combed degree distribution, and Second, we want the community to be cognizant of, and address in scalability studies, the fact that the number of zero-degree vertices grows faster than the total number of vertices when scaling Kronecker graphs.

## 2 THE KRONECKER GRAPHS

Generative graph models, such as the Erdös Rényi random graph model [6] and Barabasi's preferential attachment model [2], occupy prestigious roles in the history of graph theory. These models are theoretically important but are not typically used for *representative* benchmarks. Chakrabarti et al.'s "R-MAT: A Recursive Model for Graph Mining [3]" represents knowledge mined from real graphs by fitting them to a statistical model's parameters. The Kronecker graphs, our object of study, are a refinement of R-MAT graphs, so we begin with R-MAT graph generation.

An R-MAT or a Kronecker Graph is usually specified by 3 parameters: i) a scale factor $S$, where $2^S$ is the desired number of nodes in the graph, ii) an edge factor $e$, where $e \times 2^S$ is the desired number of edges in the graph, and iii) a seed matrix $[a\ b; c\ d]$, where the sum of the elements of the matrix is 1. Consider a graph $G$ with $N$ (a power of 2) nodes, and the $N * N$ adjacency matrix, $A$, where $A(i, j) = 1$ indicates the existence of an edge between nodes $i$ and $j$. Let's divide $A$ into its four quadrants and introduce a $2 \times 2$ *seed* that is a probability distribution over the four quadrants. R-MAT generates edges in the graph by recursively sampling from the seed distribution, according to the following procedure:

```
seed = 2 x 2 seed
A = an N x N adjacency matrix

for each desired edge in G
    Q = A
    do
        # select a quadrant of A, according
        # to the seed distribution
        Q = sample(Q, seed)
    while |Q| > 1

    # Q is now a single cell in A
    place edge at Q
```

R-MAT is widely used because it is simple: the distribution is intuitively meaningful, and it's easy to publish parameters. Furthermore, when the distribution is derived from real graph data (as its authors intended and demonstrated), researchers can claim that the synthetic graphs distribution are "like" a real graph and therefore valid experimental data. In particular, this provides a credible method to vary the size of graphs in an experiment and maintain their inter-comparability. DARPA's HPCS supercomputing group chose R-MAT as the input generator for its SSCA#2 scalable graph analysis benchmark specification in 2005[1], which evolved into the "HPC Scalable Graph Analysis Benchmark" in 2010[5], and then the Graph500 supercomputing benchmark[20].

The Kronecker model of Leskovec, Chakrabarti, and Kleinberg [12] characterized R-MAT's recursive step as the Kronecker product of the distribution matrix with itself and the recursive series as a Kronecker exponentiation process. The distribution of graphs generated by this process are the Kronecker graphs. These extensions provide for the use of distributions with dimensions other than $2 \times 2$ and also give a more rigorous procedure to fit the model's parameters to real graph data.

## 3 BENCHMARKING CHALLENGES

There are two issues of concern using Kronecker graphs for benchmarking graph systems.

- **Zero-degree vertices:** The Kronecker model generates zero-degree vertices (i.e., vertices isolated from the rest of the graph) in proportion to a complex function of its parameters, which means the real vertex count does not actually grow exponentially with the scale parameter.
- **Degree distribution:** The model produces a degree distribution that is dramatically unlike any real dataset, with implications for benchmarks that depend on the degree, such as triangle counting (Figure 9).

In principle, a $2 \times 2$ Kronecker seed produces a graph with $|V| = 2^{scale}$ vertices and $|E|$ edges, where $|E| = O(|V|)$ for very large, sparse graphs. For example, the Graph500 specifies that the scale varies and the edge count covaries by a constant factor $2^{scale} * edgefactor$. Naively, one might assume that using *scale* as an axis would produce a log-scale plot. However, one would be wrong – isolated (i.e., zero-degree) vertices reduce the effective size of the graph. That is,

for nearly every system and every algorithm, computing on a one million node graph where half its vertices are isolated is comparable to computing on a graph with 500,000 vertices and is not comparable to computing on a graph with one million non-zero degree vertices.

Seshadhri et. al. [24] show that: 1) *in Kronecker graphs, a large fraction of the vertices are isolated in expectation*, and 2) the fraction of isolated vertices *grows with increasing scale and a fixed edge factor*. Using Graph500 settings, as the scale parameter increases from 26 to 42, there is a corresponding increase in the percentage of isolated vertices, from 51% to 74% [24]. While the scale parameter is intended to indicate the logarithm base 2 of the number of vertices, in reality, in plots of Graph500 graphs with the *scale* parameter on the x-axis, *the real vertex count is sub-exponential on the x-axis, the edge count is exponential, and the average degree is super-exponential*. This is a mistake that past researchers have made whilst doing scalability analysis of their graph processing systems [18, 23]. It is particularly problematic as synthetic graphs are most frequently used to demonstrate scalability, because publicly released real world data sets are sufficiently small to not pose any scalability challenges.

Figure 1 shows how the true number of non-zero degree vertices falls remarkably short of the expected number of non-zero degree vertices for any given scale factor value for Graph500 parameters. For Graph500 parameters, to obtain the "expected" vertex count of $2^{scale}$, one must usually request two to four times as many vertices (i.e., increase scale by one to two) and decrease the edgefactor proportionally. This is not an error in the Kronecker model so much as a misunderstanding between the graph modeling and benchmarking communities. The graph modeling community regards densification "with growth" as a key feature of power law models, and Leskovec et al. present it as one of the their model's strengths [13]. However, this definition of scale gives a complex x-axis with a non-trivial relationship to simple benchmark parameters, such as the vertex count.

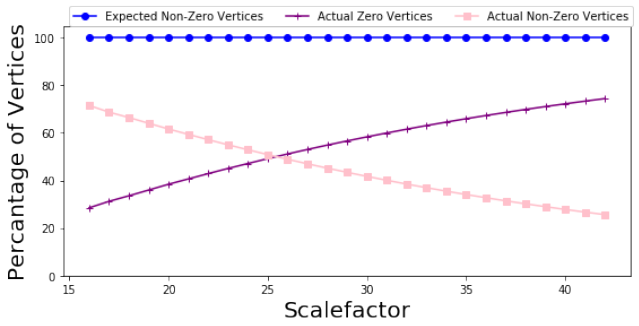



Figure 1: **The real number of non-zero degree vertices falls remarkably short of the expected number of non-zero degree vertices as scale factor increases.**

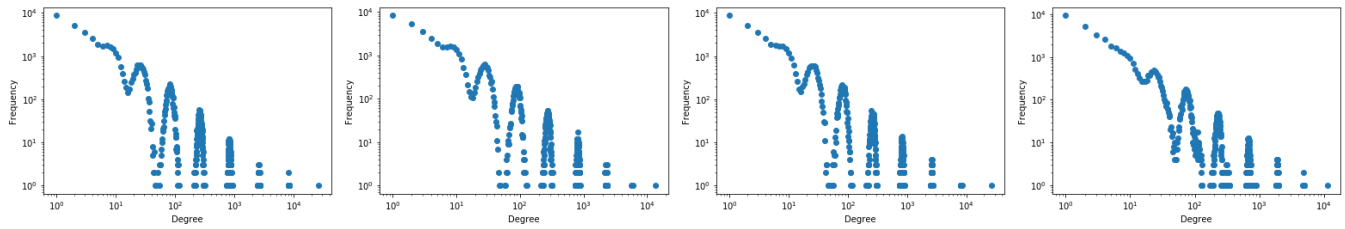
If a benchmark algorithm depends on the vertex count then its relationship with this x-axis is similarly complicated.

Isolated vertices in the Kronecker model are a symptom of a more serious problem. Figure 2 plots the frequency distribution of vertex degrees in a Kronecker graph drawn from the Graph500's default parameters. The graph's distribution is "combed" at regular geometric intervals, between which vertices with a given degree are highly improbable. This distribution matches no real world data of which we are aware and clearly may affect an algorithm whose expected runtime depends on the degree distribution, such as triangle counting. Moreover, the same combing appears in other fundamental parameters, such as the k-core distribution. This combing problem is so pervasive that it calls into question whether Kronecker graphs ought be treated as representative datasets for benchmarking purposes.

Seshadri proposes to smooth the distribution by independently blurring each Kronecker iteration with uniform random noise [24], i.e., "Noisy Kronecker". With the application of random noise, there are just as many expected degrees as there are vertices, and the degree distribution is less clustered, although in principle it still varies around the same combs. This implies that noise must be applied once per graph and not once per edge draw, otherwise the model converges back to the combed distribution. Moreover, in the model where noise is picked once per edge draw, there are $k$ additional random variables that determine the degree distribution. It is crucial that $k$ is not large, otherwise the noisy model converges back to the same combs as the original. When using these graphs for benchmarking, it is essential to control for these random variables by running the benchmarks on a large number of such graphs.

It is unclear how to introduce enough noise to get rid of the combing without dramatically changing other key features of the model, such as the diameter, number of triangles, eigenvalue distribution, etc. Currently, eliminating that combing requires a great deal of noise relative to the seed distribution, which changes these other key features. The Graph500 reference implementation includes a "noise" parameter, but *it is not part of the specification and is compiled out of the reference code*. When compiled in, it fixes the noise draws for each level at uniform intervals along a range of the noise parameters, $[-n, n]$ and uses an altered noise equation, for which it is not immediately obvious that Seshadri's proof on noise leading to a log-normal tail holds. This choice of using uniform random noise from a noise range $[-n, n]$ causes problems by altering the planted edge-cuts in the generated graph.

In a series of $k$ random draws from the uniform noise range $[-n, n]$, the minimum draw in expectation is $-n+2n/k$, while the Graph500 generator guarantees the minimum draw is always $-n$. The $ith$ iteration of the Kronecker generation




(a) (b) (c) (d)

Figure 2: **The Combing Effect in the Degree-Frequency plot of Kronecker & R-MAT Graphs with Graph500 parameters S=16, e=16, a=0.57, b=0.19, c=0.19 generated by (a) Graph500 [20] (b) Snap Kronecker [16], (c) Ligra's R-MAT generator [25] (d) TrillionG's generator [22] with Noise=0.0**

algorithm with minimum noise generates the $ith$ bit of each vertex. If we sort the vertices by this bit, this gives a bi-partitioning of the graph in which the anti-diagonal is weakened by the minimum noise. If the noise magnitude is significant, this will substantially reduce the expected edge-cut of the any $2^{k/2}$-way partitioning in expectation, which realistically covers all practical partitionings for any large graph. So, adding noise has both topological and spectral ramifications in proportion to the size of the noise. The topological ramifications are particularly troubling: Synthetic graphs are most frequently used in benchmarking when demonstrating the scalability of a graph processing system to multiple machines. These systems therefore require partitioning the graph. So, evaluators are left with two bad choices: use a Kronecker generator without noise and an unrealistic degree distribution or use noisy Kronecker, which will exaggerate the efficacy of a distributed system, because the resulting graph is more partitionable.

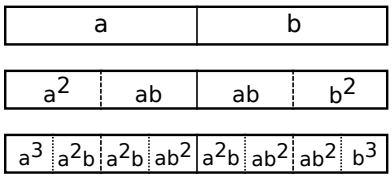



Figure 3: **A one-dimensional Kronecker series with two distribution parameters, $a$ and $b$. At iteration $k$ the number of samples is $2^k$ but the number of unique sample probabilities is only $k+1$.**

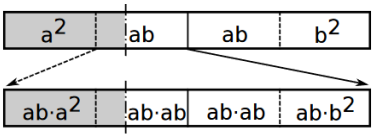



Figure 4: **Sampling the leftmost third of the one-dimensional, two-parameter Kronecker series as it approaches infinity. To sample the rightmost third swap $a$ and $b$, and to sample the middle subtract the left and right from the unit.**

## 4 SMOOTH KRONECKER

Intuitively, both combing and isolated vertices are the result of the same process. Figure 3 illustrates, in the one-dimensional case, that after $k$ draws from the Kronecker series, the number of *unique* sample probabilities grows as $k+1$, not $2^k$. For a seed with two parameters, $a$ and $b$, the first Kronecker product of the seed with itself is $[a^2, ab, ab, b^2]$. Thus, each sample's probability is a sum of its binary representation, which is why there are only $k+1$ unique sample probabilities. Fundamentally, this means that there are exponentially more vertices in a Kronecker model than there are expected degrees.

The Smooth Kronecker algorithm smooths the degree distribution by resampling the seed distribution's parameters into two seeds whose dimensions are relatively prime to one another, e.g., $2 \times 2$ and $3 \times 3$. By randomly substituting one seed for the other, we smooth the Kronecker generator in the same manner that adding noise would blur it. Crucially, we derive the additional seed from the original seed distribution, so unlike graphs produced by noisy Kronecker, Smooth Kronecker graphs preserve the Kronecker graph's statistical properties (e.g., diameter, triangles, etc.). The key idea is to create one or more alternate $d_1 \times d_1$ seeds, sampled from the distribution of the initial seed's Kronecker product as it goes to infinity, but with a dimension such that $d_1$ is not a power of $d$. Given a scale factor of $k$, we use the $d_1 \times d_1$ seed instead of the $d \times d$ seed one out of $k$ times.

Once again, consider the one-dimensional case with a two parameter seed $[a, b]$. From the infinite Kronecker product series of this seed, we want to downsample to obtain a three parameter seed $[x, y, z]$. Figure 4 shows that $x$ is just the geometric series whose initial term is $aa$ and ratio is $ab$; this corresponds to the geometric series $\frac{1}{3} = \frac{1}{4} + \frac{1}{16} + \frac{1}{64} + \ldots$. $y$ and $z$ follow similar geometric series. In general, any down-sampled distribution can be explicitly solved as a geometric series of $d$-ary fractions. The two dimensional case is just the bilinear extension of the one dimensional case.

The alternative $d_1 \times d_1$ seed acts as a smoothing filter that pushes the finite Kronecker product towards a better approximation of the infinite Kronecker product. At any step in edge generation, there is a $1/k$ probability that the distribution is locally smoothed by an alternative, but representative,

sample from the infinite series. The finite Kronecker product gives an undersampled binomial degree distribution, but the infinite Kronecker product gives a continuous log normal degree distribution, so improving this approximation reduces combing. Resampling and randomly substituting when we use the resampled seed are just clever tools to efficiently approximate the infinite series without explicitly materializing a large distribution. Algorithm 1 presents an algorithm for using a $2 \times 2$ input seed with a $3 \times 3$ seed, which covers all the popular Kronecker benchmark parameters.

The single $d_1 \times d_1$ seed increases the model's vertex count, so graphs drawn from this model are not directly comparable to graphs drawn from a pure $d \times d$ model, because the explicit scale is different. Fortunately, it is trivial to extend this algorithm to arbitrary mixtures of $d_1 \times d_1$ and $d \times d$ seeds. For example, $3^5$ and $2^8$ are approximately equal, so one can substitute five $3 \times 3$ seeds for eight $2 \times 2$ seeds and preserve the approximate scale. In fact, this gives our scale parameters more degrees of freedom than the pure $d \times d$ model, because we are no longer restricted to purely logarithmic scales.

Algorithm 1 Smooth Kronecker Algorithm with a $2 \times 2$ input seed. Essentially, when we substitute the $3 \times 3$ seed we generate a trinary bit. This method generalizes to arbitrary mixtures of $d$-ary seeds.

```
function SMOOTH KRONECKER(A, B, C, D, k, e)
    2 × 2_seed ← [A, B, C, D]
    3 × 3_seed ← 3x3 RESAMPLE(A, B, C, D)
    for e in edges do
        source ← 0
        target ← 0
        base ← 1
        r ← RANDOM INT(0, k)
        for all i ∈ RANGE(0, k) do
            if i ≠ r then
                cell ← RANDOM CHOICE(2 × 2_seed)
                source ← source + (cell/2) * base
                target ← target + (cell%2) * base
                base ← base * 2
            else
                cell ← RANDOM CHOICE(3 × 3_seed)
                source ← source + (cell/3) * base
                target ← target + (cell%3) * base
                base ← base * 3
            end if
        end for
        Yield source, target
    end for
end function
```

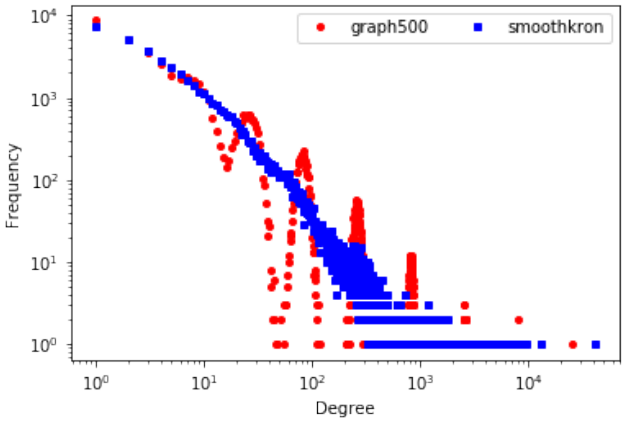



Figure 5: **Degree-frequency plot of a Smooth Kronecker graph superimposed over the traditional Kronecker graph generated using Graph500 reference from Figure 2a.**

Figure 5 compares the degree-frequency of a traditional Kronecker graph, previously shown in Figure 2a, to an equivalent graph that substitutes five resampled $3 \times 3$ seeds for eight of the original $2 \times 2$ seeds. The combing is almost entirely corrected by the addition of the alternative seeds.

Note that combing is visible in the plots only when the graph's scale is relatively small. Increasing the scale visually pushes the combs closer together in a fixed-width log-scale plot, because the comb count is equal to the scale plus one. Thus, at large scales the combs *appear* to blend together and vanish, but this is strictly a visual illusion, because the plot width does not increase in proportion to the scale. In reality combing is *more* exaggerated at large scales, because

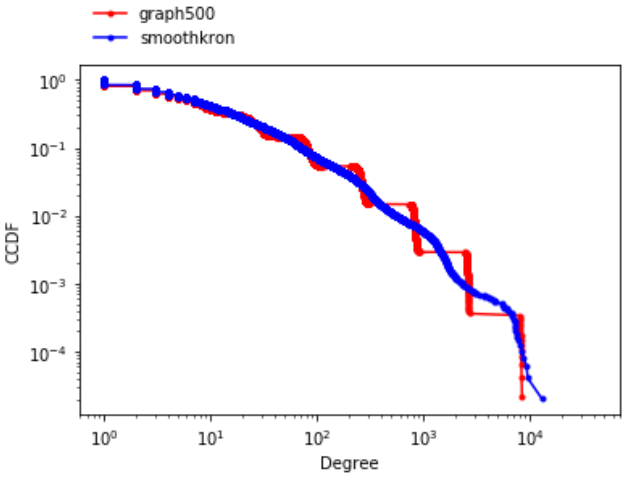
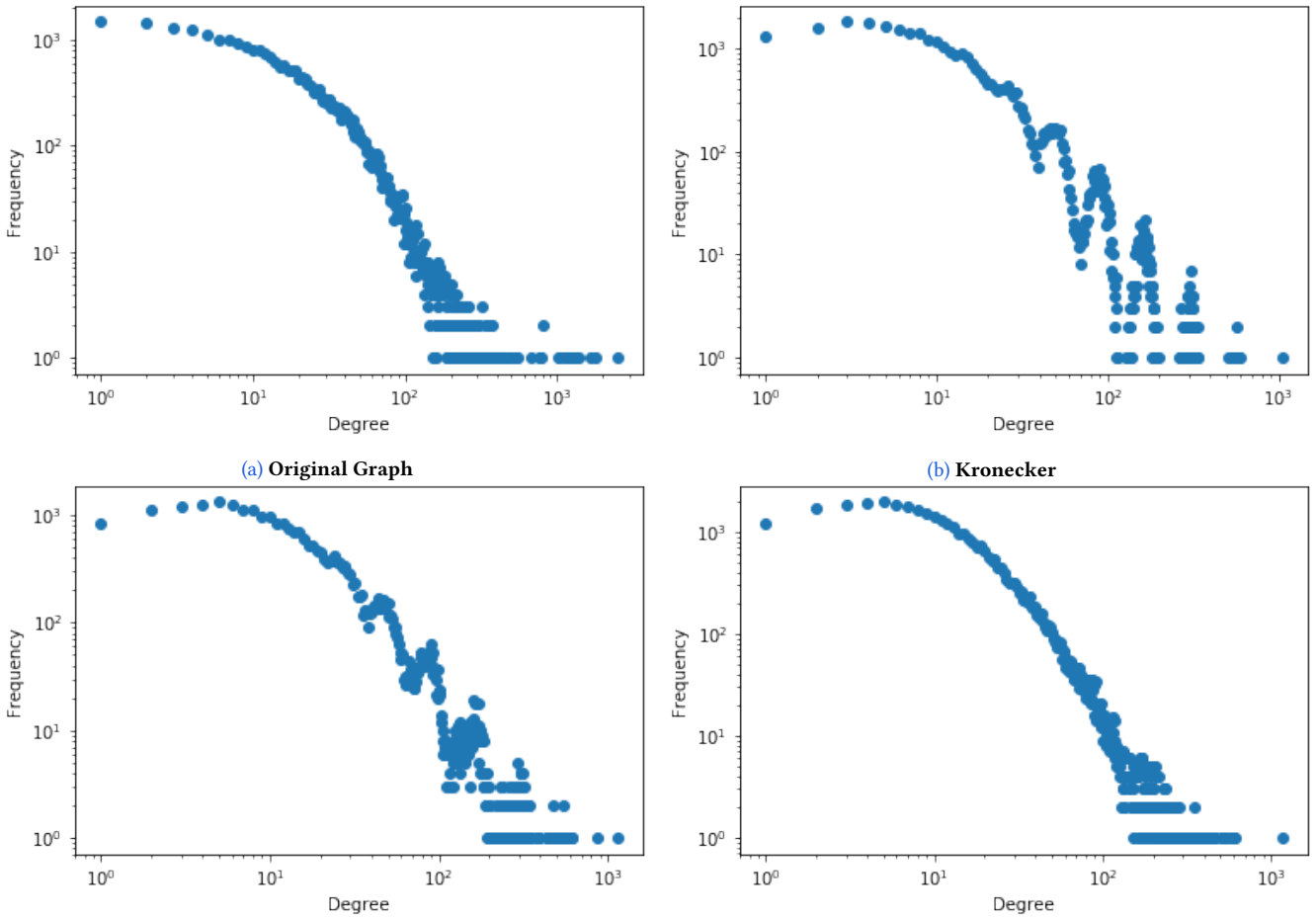



Figure 6: **CCDF of the degree of a smoothed Kronecker graph superimposed over the traditional Kronecker graph from Figure 2.**




(a) **Original Graph**

(b) **Kronecker**

(c) **Noisy Kronecker**

(d) **Smooth Kronecker**

Figure 7: **Degree-Frequency plots of the cit-HepTh graph [4] and equivalent Kronecker graphs with seed parameters a=0.4307, b=0.23808, and c=0.1859.**

each new comb is separated from its adjacent combs by a geometric factor.

It is much easier to see details in the degree's complementary cumulative distribution (CCDF); in this representation the combs appear as "steps." Figure 6 compares the CCDF of the same traditional and smoothed Kronecker graphs. Here we see that the smoothed graph actually follows the same combing pattern as the traditional graph, but it is dampened by orders of magnitude (note that the y-axis is also log scale). This is further evidence that the $3 \times 3$ resampled seed acts as a smoothing filter over the pure $2 \times 2$ series.

The original Kronecker generator and the Noisy Kronecker generators do not generate graphs that have degree distributions that match that of real graphs. For example, Figure 7 shows the degree distribution of the High-Energy Physics Theory dataset [4] from the SNAP website [15] along with the degree distributions of three kronecker graphs: one without noise, one with noise, and one smooth kronecker graph that are generated to approximate the original graphs, with parameters extracted from the KronFit algorithm [13]. To generate the smooth version, we used twelve 2x2 seeds and two 3x3 seeds instead of fifteen 2x2 seeds. The graph generated using the Smooth Kronecker has a degree distribution that matches that of the real graph, unlike the graph generated from a noisy kronecker model.

The combing effect in Kronecker models is not limited to degree frequency, but is also present in other parameters that broadly measure "centrality," such as k-cores. Figure 8 depicts the k-core CCDF of a Kronecker graph and an equivalent smoothed Kronecker graph.

Varying the degree distribution can lead to unexpected speedups or slowdowns in benchmarks. Figure 9 shows the runtime of Galois' two different algorithms [21], NodeIterator and EdgeIterator, for the triangle counting benchmark, on original kronecker graphs from graph500 and snap, a noisy kronecker graph from PaRMAT, and our smooth

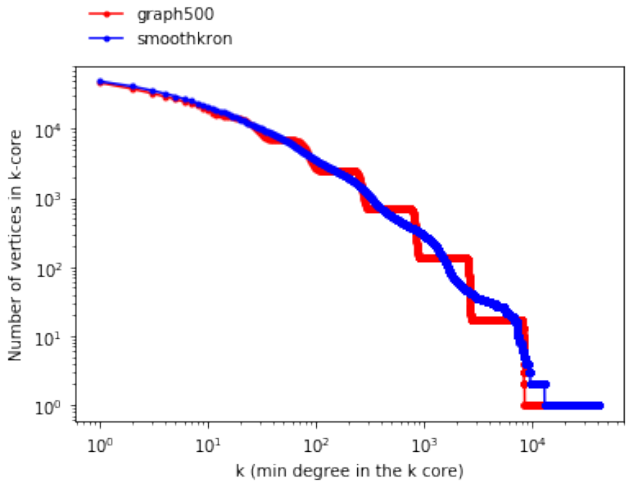



Figure 8: **k-core CCDF of a Kronecker graph from graph500 and a smooth Kronecker graph with the parameters from Figure 2a.**

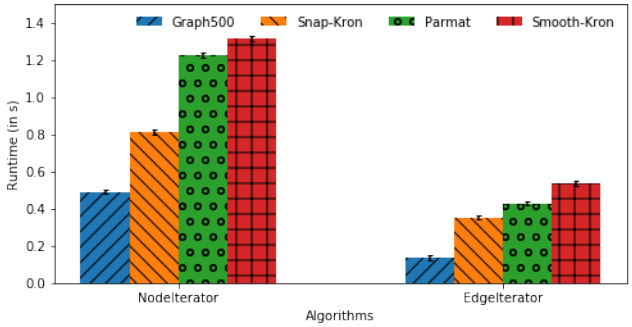



Figure 9: **Triangle Counting runtime on Galois' two triangle counting algorithms for two noise-free Kronecker graphs (Graph500 and Snap-Kron), a noisy Kronecker graph (Parmat), and a smooth Kronecker graph with the parameters from Figure 2a.**

kronecker graph. The benchmark was run 25 times with a hot cache for each graph on a 32GB Intel i5 3.1GHz Linux desktop. The triangle counting in Galois first does a degree sort on the graph, and therefore explicitly eliminates variation that could arise from vertex order randomization. The runtimes we see appear to be, for the most part, related to the differences in degree distribution; the more combed the distribution, the faster triangle counting runs. Thus, benchmark results using either Kronecker or noisy Kronecker might over predict system performance. The difference between Kronecker and Smooth Kronecker is between a factor of two and three, depending on which algorithm is used; this is more than the typical difference between systems evaluated in most performance studies. The noisy Kronecker graphs do better, but still produce optimistic results.

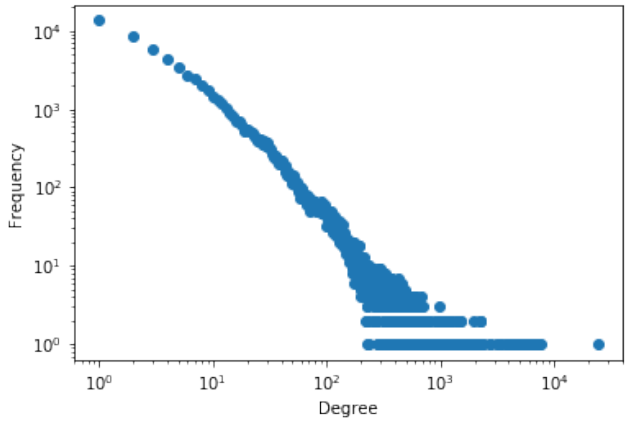



Figure 10: **A smoothed Kronecker graph with fifteen $2 \times 2$ seeds and only one resampled $3 \times 3$ seed.**

As shown in Figure 10, one resampled seed is enough to produce a much smoother degree-frequency curve. Broadly this means that many more vertex scales are possible in the mixed-seed model; in principle one could realize any vertex scale by factoring the desired scale and generating appropriate seeds. However, recall that one motivation for the seed-based Kronecker model is to avoid materializing large distributions. Therefore it is not efficient to realize vertex scales that are not the products of small factors. And of course, one must avoid powers of primes, else the model degenerates to the original Kronecker model. Like noisy Kronecker, Smooth Kronecker does not significantly change the isolated vertex count, because it intentionally follows the degree distribution of the original Kronecker model, in which many vertices have zero degree in expectation.

## 5 RELATED WORK

It has been well established in literature that the generative graphs as created by the R-MAT [3] and the Kronecker models[14] have many deficiencies as benchmarking data. Seshadri et. al. [24] study the properties of the Kronecker graphs and provide a thorough mathematical analysis that demonstrates how the Kronecker model cannot generate a power-law or a log-normal graph. As mentioned earlier, they propose introducing noise to smooth the degree distribution and provide analytical proofs for the same. The noise parameter introduced in their work is not widely used, because there is no consensus on the amount of noise to add nor the impact such noise has on the topological and spectral properties of the graph. Most R-MAT and Kronecker generators we studied in this work, such as Ligra[25], SNAP[16], and Graph500[20], employ different techniques for introducing noise into the seed matrices, and therefore, they lead to unpredictable results.

There have been many alternative generative graph models proposed, such as the Multiplicative Attribute Graph (MAG) model [10], a generalization of the Kronecker generative model. The MAG model can produce log-normal and power law graphs, but it remains widely unused for generating large synthetic graphs for evaluation of graph processing systems. Moreover, these recursive models are not efficient in their space and time complexities and often fail to generate massive graphs on commodity machines.

There have been other graph generators that address scalability challenges in the R-MAT and Kronecker generators. TrillionG[22] is one such synthetic graph generator that can generate trillion-scale graphs on a cluster of commodity machines. TrillionG uses a scope-based generation model of which the R-MAT and Kronecker models are extreme cases. They also define a recursive vector model, which while supporting the extreme case equivalence with Kronecker and R-MAT models, is also able to generate graphs of massive scale. TrillionG does not, however, solve the problem of combing in degree-distribution and relies on adding random noise.

## 6 CONCLUSIONS

We have demonstrated that the use of synthetic Kronecker graphs to evaluate graph systems is problematic as existing Kronecker graph generators produce degree distributions unlike any real graphs. We then introduced the Smooth Kronecker generator to address the unusual degree distribution produced by current Kronecker and R-MAT generators. Our approach significantly improves over the previously proposed fix of Seshadhri et. al. In particular, smooth kronecker smooths the degree distribution *without* changing key statistical properties of the generated graphs.

Our use of these graph models is fundamentally different from that of the authors of the original R-MAT and Kronecker models. As originally proposed, R-MAT and Kronecker are data mining models whose "parameters" are really metrics given by a best-fit of the model to a given graph. In this context, combing and isolated vertices are not really flaws, but rather features of the model that one must account for in the fitting process. Our goal is to address the use of the Kronecker model as a synthetic benchmark model, as it has become *de facto* practice to use it as such. In this context, the model is not merely a mining abstraction, but instead takes on a life of its own; real systems are evaluated and presented on the basis of algorithm performance on Kronecker graphs. The Kronecker model enjoys this use not simply because of its "realism," but also because of practical features such as its trivial operations, trivial parallelism, small data structures, and small parameters that are easy to publish, reproduce, and compare. However, using these kronecker graph generators for benchmarking presents a difficult dilemma. Using pure Kronecker graphs means that the combed degree distribution will produce some benchmark results quite different from what we expect on real world graphs. Using noisy Kronecker graphs requires determining, per-benchmark, how much noise to add. If the noise is fixed, this translates into a new modeling parameter; if the noise is truly random, producing accurate results requires averaging across a collection of random graphs. Smooth Kronecker is intended for this ecosystem. It preserves all the desirable properties of the Kronecker graphs, remedies the combed distribution problem, and enables generation of graphs with scales other than $2^N$.

## 7 AVAILABILITY

The Smooth Kronecker source code is available at: https://github.com/dmargo/smooth_kron_gen.

## REFERENCES


[1] David A Bader and Kamesh Madduri. 2005. Design and implementation of the HPCS graph analysis benchmark on symmetric multiprocessors. In *International Conference on High-Performance Computing*. Springer, 465–476.

[2] Albert-László Barabási and Réka Albert. 1999. Emergence of scaling in random networks. *science* 286, 5439 (1999), 509–512.

[3] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. 2004. R-MAT: A Recursive Model for Graph Mining.. In *SDM*, Vol. 4. SIAM, 442–446.

[4] SNAP: Stanford Large Network Dataset Collection. [n.d.]. High-energy physics theory citation network. https://snap.stanford.edu/data/cit-HepTh.html.

[5] David Dominguez-Sal, P Urbón-Bayes, Aleix Giménez-Vanó, Sergio Gómez-Villamor, Norbert Martínez-Bazan, and Josep-Lluis Larriba-Pey. 2010. Survey of graph database performance on the hpc scalable graph analysis benchmark. In *International Conference on Web-Age Information Management*. Springer, 37–48.

[6] Paul Erdös and Alfréd Rényi. 1960. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci* 5, 17-61 (1960), 43.

[7] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. 2005. LUBM: A benchmark for OWL knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web* 3, 2 (2005), 158–182.

[8] Minyang Han and Khuzaima Daudjee. 2015. Giraph unchained: barrierless asynchronous parallel execution in pregel-like graph processing systems. *Proceedings of the VLDB Endowment* 8, 9 (2015), 950–961.

[9] Farzad Khorasani, Rajiv Gupta, and Laxmi N. Bhuyan. 2015. Scalable SIMD-Efficient Graph Processing on GPUs. In *Proceedings of the 24th International Conference on Parallel Architectures and Compilation Techniques (PACT '15)*. 39–50.

[10] Myunghwan Kim and Jure Leskovec. 2012. Multiplicative attribute graph model of real-world networks. *Internet Mathematics* 8, 1-2 (2012), 113–160.

[11] Aapo Kyrola, Guy E Blelloch, and Carlos Guestrin. 2012. GraphChi: Large-Scale Graph Computation on Just a PC.. In *Operating Systems Design and Implementation*, Vol. 12. 31–46.

[12] Jurij Leskovec, Deepayan Chakrabarti, Jon Kleinberg, and Christos Faloutsos. 2005. Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication. In *European Conference*

*on Principles of Data Mining and Knowledge Discovery*. Springer, 133–145.

[13] Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. 2010. Kronecker graphs: An approach to modeling networks. *Journal of Machine Learning Research* 11, Feb (2010), 985–1042.

[14] Jure Leskovec and Christos Faloutsos. 2007. Scalable modeling of real graphs using kronecker multiplication. In *Proceedings of the 24th international conference on Machine learning*. ACM, 497–504.

[15] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[16] Jure Leskovec and Rok Sosič. 2016. SNAP: A General-Purpose Network Analysis and Graph-Mining Library. *ACM Transactions on Intelligent Systems and Technology (TIST)* 8, 1 (2016), 1.

[17] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M Hellerstein. 2012. Distributed GraphLab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment* 5, 8 (2012), 716–727.

[18] Peter Macko, Virendra J Marathe, Daniel W Margo, and Margo I Seltzer. 2015. LLAMA: Efficient Graph Analytics Using Large Multiversioned Arrays. In *International Conference on Data Engineering*. IEEE.

[19] Daniel Wyatt Margo. 2017. Sorting Shapes the Performance of Graph-Structured Systems. Ph.D. Dissertation, Harvard John A. Paulson School of Engineering and Applied Sciences, http://nrs.harvard.edu/urn-3:HUL.InstRepos:40046459.

[20] Richard C Murphy, Kyle B Wheeler, Brian W Barrett, and James A Ang. 2010. Introducing the graph 500. *Cray User's Group (CUG)* (2010).

[21] Donald Nguyen, Andrew Lenharth, and Keshav Pingali. 2013. A lightweight infrastructure for graph analytics. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 456–471.

[22] Himchan Park and Min-Soo Kim. 2017. TrillionG: A trillion-scale synthetic graph generator using a recursive vector model. In *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 913–928.

[23] Amitabha Roy, Ivo Mihailovic, and Willy Zwaenepoel. 2013. X-stream: Edge-centric graph processing using streaming partitions. In *24th ACM Symposium on Operating Systems Principles*. ACM, 472–488.

[24] Comandur Seshadhri, Ali Pinar, and Tamara G Kolda. 2011. An in-depth study of stochastic Kronecker graphs. In *2011 IEEE 11th International Conference on Data Mining*. IEEE, 587–596.

[25] Julian Shun and Guy E Blelloch. 2013. Ligra: a lightweight graph processing framework for shared memory. In *ACM SIGPLAN Notices*, Vol. 48. ACM, 135–146.

[26] Guozhang Wang, Wenlei Xie, Alan J Demers, and Johannes Gehrke. 2013. Asynchronous Large-Scale Graph Processing Made Easy.. In *CIDR*, Vol. 13. 3–6.

## A KRONECKER SEED RESAMPLING

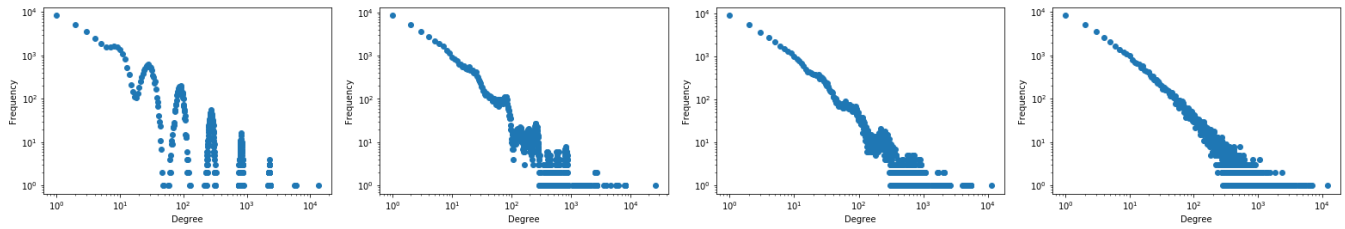
Based on the discussion in Section 2, Algorithm 2 shows how a given 2x2 seed matrix with parameters $A, B, C, D$ are resampled into a 3x3 seed matrix that is used by our Smooth Kronecker Generator.

## B NOISY KRONECKER GRAPHS

Figure 11 shows the effect of adding different amounts of noise to the Kronecker graph generation. PaRMAT [9], another open-source R-MAT generator, adds a small amount of noise, but as shown in Figure 11b, the default amount of noise used is not enough to completely rectify the combing effect present in the Kronecker graphs. Snap's R-MAT generator [16] achieves smoothing of the degree-frequency distribution, but does so at the expense of violating the Kronecker model assumption that the sampling probabilities in the seed sum to 1. Their R-MAT generator adds an unrestricted random amount of noise to the sampling probabilities. Similarly, the graph generator presented in TrillionG [22] uses a Noisy Stochastic Kronecker Generator and therefore also suffers from the combed degree-distribution.

Algorithm 2 Function to resample a Kronecker 2x2 seed matrix into a 3x3 matrix

**function** 1DThird($A, B$)
  Yield $A \times A(1 - A \times B)$
**end function**
**function** 2DNinth($A, B, C, D$)
  $Right \leftarrow A \times B \times 1DThird(A + C, B + D)$
  $Bottom \leftarrow A \times C \times 1DThird(A + B, C + D)$
  $Initial \leftarrow A \times A + Right + Bottom$
  Yield $Initial(1 - A \times D)$
**end function**
**function** 3x3Resample($A, B, C, D$)
  $a \leftarrow 2DNinth(A, B, C, D)$
  $b \leftarrow 2DNinth(B, A, D, C)$
  $c \leftarrow 2DNinth(C, A, D, B)$
  $d \leftarrow 2DNinth(D, B, C, A)$
  $ab \leftarrow 1DThird(A + B, C + D) - a - b$
  $ac \leftarrow 1DThird(A + C, B + D) - a - c$
  $bd \leftarrow 1DThird(B + D, A + C) - b - d$
  $cd \leftarrow 1DThird(C + D, A + B) - c - d$
  $x \leftarrow 1 - a - b - c - d - ab - ac - bd - cd$
  Yield $a, ab, b, ac, x, bd, c, cd, d$
**end function**




(a) (b) (c) (d)

Figure 11: **Degree-Frequency plots showing Kronecker Graphs with parameters S=16, e=16, a=0.57, b=0.19, c=0.19 generated with varying levels of noise using (a) Snap's R-MAT generator [16] with no noise,(b) PaRMAT [9] with small noise (default behavior), (c) TrillionG with noise=0.1, (d) Snap's R-MAT generator [16] with more noise (default behaviour)**